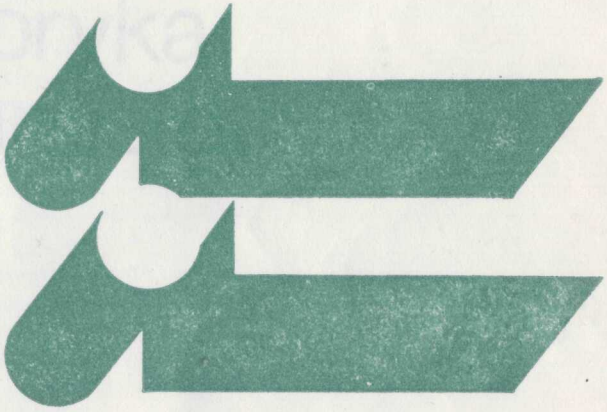


Moim Rodzicom

Witold Lipski

Kombinatoryka dla programistów

Wydanie drugie

Wydawnictwa Naukowo-Techniczne
Warszawa 1989

Biblioteka
Inżynierii Oprogramowania

Redaktor serii
BARBARA OSUCHOWSKA

sześćdziesiąta piąta pozycja
serii ukazującej się
od roku 1987

Wykaz dotychczas wydanych tomów Biblioteki zamieszczono na końcu książki

WITOLD LIPSKI

ISBN 83-204-1023-1

English summary, see p. 181

Kombinatoryka dla programistów
Combinatorics for Programmers

Русское резюме, см. стр. 182

Комбинаторика для программистов

W książce przedstawiono wybrane zagadnienia kombinatoryki, teorii grafów i algorytmów kombinatorycznych. Szczególny nacisk położono na algorytmiczne podejście do problemów kombinatorycznych. Rozważanym problemom towarzyszą zwykle szczegółowe algorytmy ich rozwiązania i analiza złożoności obliczeniowej tych algorytmów. Algorytmy zamieszczone w pracy to nieco skondensowane nieformalne wersje programów napisanych w Pascalu. Każdy z rozdziałów zawiera na końcu zbiór zadań.
Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz dla studentów kierunków informatycznych.

519:681.3

Redaktor wydania pierwszego
KRYSTYNA REGULSKA

Okładkę i układ typograficzny serii projektował
TADEUSZ PIETRZYK

Redaktor techniczny
KRYSTYNA ORŁOŚ

Tytuł dotowany przez Ministra Edukacji Narodowej

Spis treści

d Autora do wydania pierwszego

Trudno byłoby chyba wymienić dział informatyki teoretycznej, w którym w ciągu ostatniego dziesięciolecia poczyniono by większe postępy niż w projektowaniu i analizie algorytmów kombinatorycznych. Z jednej strony znaleziono wiele nowych, efektywniejszych metod rozwiązywania problemów kombinatorycznych za pomocą komputerów, z drugiej zaś uzyskano wyniki teoretyczne świadczące coraz wyraźniej o nieistnieniu „rozsądnie efektywnych" algorytmów dla szerokiej klasy zagadnień. Efektywne algorytmy kombinatoryczne znajdują zastosowania w wielu dziedzinach nienumerycznego przetwarzania informacji, zwłaszcza w optymalizacji dyskretnej i badaniach operacyjnych.

W pracy przedstawiono wybrane działy kombinatoryki, szczególny nacisk kładąc na konstruktywne, algorytmiczne podejście – omawianym problemom kombinatorycznym towarzyszą z reguły algorytmy ich rozwiązywania wraz z analizą ich złożoności obliczeniowej. Algorytmy te są nieco skondensowanymi wersjami programów uruchomionych w języku Pascal. Na wybór omawianych zagadnień – w dużej mierze wyrywkowy, ze względu na założoną objętość pracy z jednej strony i obszerność dziedziny z drugiej – wpłynęły zarówno zainteresowania autora, jak i chęć dopełnienia raczej niż dublowania dwóch innych książek o pokrewnej tematyce wydanych w serii Biblioteki Inżynierii Oprogramowania (poz. [2]) oraz w serii Informatyka (poz. [76]).*

Pierwszy, najobszerniejszy rozdział zawiera wykład najbardziej klasycznych działów kombinatoryki (permutacje, podziały zbioru i liczby, współczynnik dwumienny, funkcje tworzące itp.) wraz z wieloma – niekoniecznie klasycznymi – algorytmami generowania omawianych obiektów kombinatorycznych. W rozdziale 2 przedstawiono podstawowe techniki używane przy projektowaniu algorytmów grafowych, w szczególności metody systematycznego przeszukiwania grafów. Tematyka związana z grafami jest kontynuowana przez dwa następne rozdziały; w pierwszym z nich omówiono metody znajdowania najkrótszych dróg w grafach, których krawędziom przypisano dowolne „długości", w drugim skoncentrowano się na problemie znajdowania maksymalnego przepływu w sieci

* Zob. także L. Banachowski, A. Kreczmar, W. Rytter: *Analiza algorytmów i struktur danych*. Wyd. 1. Warszawa, WNT 1987 (wznowienie w druku). – *Przyp. red. do wyd.* 2.

(tzn. w grafie o określonych „przepustowościach" krawędzi). W ostatnim rozdziale omówiono zastosowanie kombinatorycznego pojęcia matroidu do rozwiązywania pewnej klasy zagadnień optymalizacyjnych.

Książka jest przeznaczona dla programistów pragnących wzbogacić swą wiedzę na temat algorytmów kombinatorycznych oraz podbudować teoretycznie swą wiedzę praktyczną. Od Czytelnika wymaga się jedynie elementarnych wiadomości z matematyki, znajomości języka programowania Pascal (p. poz. [36] i [75]) oraz pewnego doświadczenia w programowaniu w języku wyższego poziomu.

Pragnę podziękować doc. dr. hab. Wiktorowi Markowi za wiele cennych uwag, które pozwoliły usunąć usterki z pierwotnej wersji tej książki.

WITOLD LIPSKI

Warszawa, w grudniu 1980

Wprowadzenie do kombinatoryki 1

Pojęcia wstępne 1.1

W punkcie tym są zebrane podstawowe definicje i oznaczenia dotyczące używanych pojęć logicznych i teoriomnogościowych oraz przedstawionych w dalszym ciągu algorytmów.

Zaczniemy od pojęć logicznych i teoriomnogościowych (Czytelnika zainteresowanego w głębszym poznaniu tych pojęć odsyłamy do pozycji [49] i [57]). Używać będziemy spójników logicznych $\vee$ (lub), $\wedge$ (i), $\neg$ (nie), $\Rightarrow$ (jeśli ..., to ...), $\Leftrightarrow$ (wtedy i tylko wtedy, gdy). Fakt, że x jest elementem zbioru X zapisujemy jako $x \in X$, jego zaprzeczenie przez $x \notin X$. Zbiór tych elementów zbioru X, które spełniają warunek Φ oznaczamy przez $\{x \in X: \Phi\}$ (lub $\{x: \Phi\}$, jeśli wiadomo, o jaki zbiór X chodzi), natomiast $\{a_1, \ldots, a_n\}$ oznacza zbiór, którego jedynymi elementami są $a_1, \ldots, a_n$ (w szczególności jedynym elementem zbioru $\{a\}$ jest a). Operacje teoriomnogościowe sumy, przecięcia i różnicy oznaczamy odpowiednio przez $\cup$, $\cap$ i $\setminus$, zbiór pusty natomiast (nie zawierający żadnego elementu) przez $\emptyset$. Fakt, że zbiór A jest zawarty w zbiorze B (tzn. A jest podzbiorem zbioru B) oznaczamy przez $A \subseteq B$ lub $B \supseteq A$ (mamy zawsze $\emptyset \subseteq A$, $A \subseteq A$); symbol „$\subset$" jest zarezerwowany dla przypadku zawierania wykluczającego przypadek $A = B$ (mówimy wtedy, że A jest *podzbiorem właściwym* zbioru B). Zbiór wszystkich podzbiorów zbioru X oznaczamy przez $\mathscr{P}(X)$, a liczność zbioru X (tzn. liczbę jego elementów) przez $|X|$.

Ciąg długości n o wyrazach $a_1, \ldots, a_n$ oznaczamy przez $\langle a_1, \ldots, a_n \rangle$ albo po prostu przez $a_1, \ldots, a_n$ lub $a_1 \ldots a_n$. Ciąg $\langle a, b \rangle$ długości dwa nazywamy *parą uporządkowaną*. *Iloczyn kartezjański* $A \times B$ zbiorów A, B definiujemy jako zbiór wszystkich par $\langle a, b \rangle$ gdzie $a \in A$, $b \in B$. Przez *relację binarną* (o *lewej dziedzinie* A i *prawej dziedzinie* B) rozumiemy dowolny podzbiór $R \subseteq A \times B$. Jeśli $A = B$, to mówimy o relacji binarnej na zbiorze A. Zamiast $\langle a, b \rangle \in R$ piszemy często aRb.

O relacji R na zbiorze X mówimy, że jest:

(a) *zwrotna*, jeśli xRx dla każdego $x \in X$,

(b) *przechodnia*, jeśli $(xRy \wedge yRz) \Rightarrow xRz$ dla dowolnych $x, y, z \in X$,

(c) *symetryczna*, jeśli $xRy \Rightarrow yRx$ dla dowolnych $x, y \in X$,

(d) *antysymetryczna*, jeśli $(xRy \wedge yRx) \Rightarrow x = y$ dla dowolnych $x, y \in X$.

Dowolną relację binarną, która jest zwrotna, przechodnia i symetryczna nazywamy *relacją równoważności*, natomiast relację zwrotną, przechodnią i antysymetryczną nazywamy *częściowym porządkiem*. Relację częściowego porządku oznaczamy zwykle przez „$\leqslant$", a parę $\langle X, \leqslant\rangle$ nazywamy *zbiorem częściowo uporządkowanym*. Stosować będziemy oczywiste oznaczenia, takie jak $x \geqslant y$ dla $y \leqslant x$, $x < y$ dla $x \leqslant y \wedge x \neq y$ itp. Przykładem zbioru częściowo uporządkowanego jest zbiór liczb całkowitych z relacją podzielności, zbiór liczb całkowitych (lub rzeczywistych) ze zwykłą relacją mniejszości lub równości „$\leqslant$", jak również zbiór $\mathcal{P}(X)$ z relacją zawierania $\subseteq$.

Jeśli funkcja (odwzorowanie) f przyporządkowuje każdemu elementowi $x \in X$ element $f(x) \in Y$, to piszemy $f\colon X \to Y$ (funkcję taką możemy traktować jako relację $R \subseteq X \times Y$ o tej własności, że dla każdego $x \in X$ istnieje w R dokładnie jedna para postaci $\langle x, y\rangle$, $y \in Y$; dla naszych celów wystarczy jednak intuicyjne pojęcie funkcji). Dla dowolnych $A \subseteq X$, $B \subseteq Y$ definiujemy

$$f(A) = \{y \in Y\colon \text{istnieje } x \in A, \text{ takie że } y = f(x)\}$$
$$f^{-1}(B) = \{x \in X\colon f(x) \in B\}$$

(zamiast $f^{-1}(\{b\})$ piszemy po prostu $f^{-1}(b)$).
Jeśli $f(X) = Y$, to mówimy o funkcji z X *na* Y. Funkcja $f\colon X \to Y$ jest *różnowartościowa* (*wzajemnie jednoznaczna*), jeśli dla dowolnych $a, b \in X$

$$a \neq b \Rightarrow f(a) \neq f(b)$$

Pojęciem często przez nas używanym będzie pojęcie grafu (por. [9], [31]). Przez *graf niezorientowany* (lub krótko: *graf*) rozumiemy dowolną parę $G = \langle V, E\rangle$, taką że

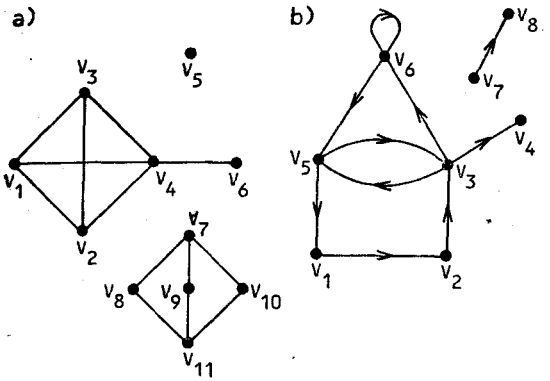
$$E \subseteq \{\{u, v\}\colon u, v \in V \wedge u \neq v\}$$

Grafem zorientowanym nazywamy dowolną parę $G = \langle V, E\rangle$, taką że $E \subseteq V \times V$. W obu przypadkach zbiory V i E nazywamy odpowiednio zbiorem *wierzchołków* i zbiorem *krawędzi* grafu G.

Graf reprezentujemy zwykle na płaszczyźnie w postaci zbioru punktów odpowiadających wierzchołkom i łączących je linii odpowiadających krawędziom. Linia odpowiadająca krawędzi $\{u, v\}$ lub $\langle u, v\rangle$ łączy punkty odpowiadające wierzchołkom u, v, przy czym w tym drugim przypadku jest zaopatrzona w strzałkę określającą kierunek z u do v (rys. 1.1).

W kontekście ustalonego grafu $G = \langle V, E\rangle$ będziemy często używać oznaczeń $u - v$, $u \to v$ zamiast odpowiednio $\{u, v\} \in E$ i $\langle u, v\rangle \in E$. Jeśli krawędź e jest postaci $\{u, v\}$ lub $\langle u, v\rangle$, to mówimy, że krawędź e jest *incydentna* z wierzchołkami u i v, oraz że wierzchołki u i v *sąsiadują* ze sobą. *Stopień wierzchołka* określamy jako liczbę krawędzi incydentnych z nim. Wierzchołek stopnia 0 nazywamy *izolowanym* (np. wierzchołek v_5 na rys. 1.1a). *Drogą* w grafie $G = \langle V, E\rangle$ nazywamy ciąg wierzchołków $v_0, v_1, \ldots, v_k$, taki że $k \geqslant 0$ i $v_i - v_{i+1}$ (lub $v_i \to v_{i+1}$, jeśli G jest grafem zorientowanym), $i = 1, \ldots, k-1$. Wierzchoł-

ki v_1 i v_k nazywamy odpowiednio *początkiem* i *końcem* drogi, liczbę k natomiast jej *długością*. Drogę, której początek pokrywa się z końcem, nazywamy *cyklem*. Jeśli wszystkie wierzchołki drogi $v_1, \ldots, v_k$ są różne, to mówimy o drodze *elementarnej*. Podobnie, cykl $v_1, \ldots, v_k$ $(v_1 = v_k)$ nazywamy *elementarnym*, jeśli wierzchołki $v_1, \ldots, v_{k-1}$ są różne. *Podgrafem* grafu $G = \langle V, E \rangle$ nazywamy dowolny graf $G' = \langle V', E' \rangle$, taki że $V' \subseteq V$ i $E' \subseteq E$.



Rys. 1.1 (a) Graf niezorientowany (b) Graf zorientowany

Niech $G = \langle V, E \rangle$ będzie dowolnym grafem niezorientowanym i niech $v \in V$. Niech A będzie zbiorem tych wierzchołków $u \in V$, do których istnieje droga z v. Zbiór A wraz z krawędziami grafu G, incydentnymi z wierzchołkami z A, określa pewien podgraf zwany *składową spójną* grafu G. Oczywiście zbiory wierzchołków składowych spójnych dowolnego grafu są parami rozłączne. Na przykład dla grafu z rysunku 1.1a są to zbiory $V_1 = \{v_1, v_2, v_3, v_4, v_6\}$, $V_2 = \{v_5\}$ i $V_3 = \{v_7, v_8, v_9, v_{10}, v_{11}\}$.

Mówimy, że grafy $G = \langle V, E \rangle$ i $G' = \langle V', E' \rangle$ są *izomorficzne*, jeśli istnieje wzajemnie jednoznaczne odwzorowanie f z V na V', takie że dla dowolnych $u, v \in V$ mamy $\{u, v\} \in E \Leftrightarrow \{f(u), f(v)\} \in E'$ ($\langle u, v \rangle \in E \Leftrightarrow \langle f(u), f(v) \rangle \in E'$ w przypadku grafów zorientowanych). Zwykle nie będziemy rozróżniać między grafami izomorficznymi.

Dla dowolnej liczby rzeczywistej x będziemy używać oznaczeń $\lfloor x \rfloor$ i $\lceil x \rceil$, odpowiednio dla największej liczby całkowitej nie większej niż x i najmniejszej liczby całkowitej nie mniejszej od x, np. $\lfloor 3.5 \rfloor = 3$, $\lceil 3.5 \rceil = 4$, $\lfloor -3.5 \rfloor = -4$, $\lceil -3.5 \rceil = -3$.

Przechodzimy teraz do pojęć związanych z algorytmami. Będziemy je zwykle zapisywać w języku programowania będącym nieformalną wersją Pascala [36], [75]. Jeśli realizacja jakiegoś fragmentu programu jest oczywista, lecz żmudna i zaciemniająca ideę algorytmu, to fragment taki będziemy czasami zastępować opisem w języku naturalnym. Stosujemy też nieformalne konstrukcje, takie jak na przykład pętle **for** $x \in X$ **do** P (wykonaj instrukcję P dla wszystkich elementów x zbioru X – w dowolnej kolejności), STOS $\Leftarrow x$ (połóż zawartość zmiennej x na stosie), $x \Leftarrow$ STOS (zdejmij szczytowy element stosu i podstaw go pod zmien-

ną x), KOLEJKA $\Leftarrow x$ (dodaj x do kolejki jako ostatni element), $x \Leftarrow$ KOLEJKA (usuń pierwszy element z kolejki i podstaw go pod zmienną x) itp. Będziemy zwykle opuszczać deklaracje typów i zmiennych (czasami dla uniknięcia nieporozumień umieszczamy odpowiednie wyjaśnienie w komentarzu). Zmienną pojawiającą się w procedurze uważamy za lokalną w tej procedurze, chyba że w sposób jawny jest powiedziane – w komentarzu – coś innego. Wiersze programu numerujemy tak, by umożliwić powoływanie się na „pętlę 17", „blok 9" itp.

Podstawowym parametrem algorytmu, który będzie nas interesować, jest jego *złożoność obliczeniowa* (lub krótko: *złożoność*), tzn. liczba kroków wykonywanych – w najgorszym przypadku – przez algorytm, jako funkcja wymiaru problemu reprezentowanego przez dane wejściowe. Dla przykładu, jeśli algorytm akceptuje jako dane dowolny graf $G = \langle V, E \rangle$, to przez wymiar problemu możemy rozumieć $|V|$. Złożoność naszego algorytmu jest określona wtedy jako funkcja f, taka że $f(n)$ jest równe maksymalnej liczbie kroków wykonywanych przez algorytm dla dowolnego grafu o n wierzchołkach. Możemy też uważać za wymiar problemu parę $\langle |V|, |E| \rangle$ – wtedy złożoność jest funkcją dwóch zmiennych i $f(n, m)$ jest równe maksymalnej liczbie kroków wykonywanych przez algorytm dla dowolnego grafu o n wierzchołkach i m krawędziach. Pozostaje jeszcze wyjaśnić nieco dokładniej, co rozumiemy przez „krok algorytmu". Otóż przyjmujemy, że nasze programy są tłumaczone na język maszynowy typowego komputera, posiadającego w repertuarze swoich instrukcji rozkazy przenoszenia słowa z pamięci do akumulatora i odwrotnie, operacje arytmetyczne dodawania, odejmowania, mnożenia i dzielenia, skoki warunkowe, operacje wejścia-wyjścia oraz wyposażonego w mechanizm adresowania pośredniego (tzn. określania argumentu operacji przez adres komórki pamięci zawierającej adres tego argumentu). Wykonanie dowolnej spośród powyższych instrukcji uważamy właśnie za krok algorytmu. Oczywiście, przy tak określonym „kroku", złożoność algorytmu zależy od konkretnego sposobu tłumaczenia naszego programu, jak również od konkretnej postaci rozkazów maszynowych. Nie będzie nas jednak nigdy interesować dokładna złożoność algorytmu, lecz jedynie asymptotyczna szybkość wzrostu liczby kroków algorytmu, gdy wymiar problemu rośnie nieograniczenie (aby o takiej szybkości wzrostu można było mówić zakładamy, że obszar pamięci naszego komputera jest nieograniczony oraz że każda komórka pamięci może zawierać dowolnie dużą liczbę całkowitą). Jest jasne, że przy dowolnych dwu „rozsądnych" sposobach tłumaczenia odpowiednie złożoności różnią się co najwyżej o stałą multiplikatywną, a więc ich szybkość wzrostu jest taka sama. Czytelnika pragnącego uściślić powyższe, bardzo nieformalne rozważania odsyłam do pozycji [1] i [2].

Przy porównywaniu szybkości wzrostu dwóch funkcji $f(n)$ i $g(n)$ (o wartościach nieujemnych) bardzo wygodne są następujące oznaczenia:

$$f(n) = O(g(n)) \Leftrightarrow \text{istnieją stałe } C, N > 0 \text{ takie, że } f(n) \leqslant C\,g(n) \text{ dla wszystkich } n \geqslant N$$

$f(n) = \Omega(g(n)) \Leftrightarrow$ istnieją stałe $C, N > 0$ takie, że $f(n) \geqslant Cg(n)$ dla wszystkich $n \geqslant N$

Oczywiście $f(n) = \Omega(g(n))$ wtedy i tylko wtedy, gdy $g(n) = O(f(n))$. Symbole $O(g(n))$ i $\Omega(g(n))$ czytamy odpowiednio: „rzędu co najwyżej $g(n)$" oraz „rzędu co najmniej $g(n)$". Jeśli złożoność pewnego algorytmu jest $O(g(n))$, to mówimy też, że algorytm ten „działa w czasie $O(g(n))$".* W podobny sposób definiujemy symbole $O(g(n_1, \ldots, n_k))$ i $O(g(n_1, \ldots, n_k))$ dla funkcji wielu zmiennych, np.

$f(n_1, \ldots, n_k) = O(g(n_1, \ldots, n_k)) \Leftrightarrow$ istnieją stałe $C, N \geqslant 0$ takie, że $f(n_1, \ldots, n_k) \leqslant Cg(n_1, \ldots, n_k)$ dla wszystkich $n_1, \ldots, n_k \geqslant N$

Rozważaną powyżej złożoność nazywamy czasem *złożonością czasową*, w odróżnieniu od *złożoności pamięciowej*, określającej wielkość obszaru pamięci używanego przez algorytm jako funkcję wymiaru problemu.

Funkcje i rozmieszczenia 1.2

Klasycznym problemem kombinatorycznym jest pytanie, na ile sposobów można rozmieścić pewne obiekty w pewnej liczbie „pudełek", tak by były spełnione zadane ograniczenia. Problem ten można sformułować nieco bardziej formalnie w następujący sposób. Dane są zbiory X, Y, przy czym $|X| = n$, $|Y| = m$. Ile jest funkcji $f: X \to Y$ spełniających dane ograniczenia? Elementy zbioru X odpowiadają obiektom, elementy zbioru Y pudełkom, każda funkcja $f: X \to Y$ określa natomiast pewne rozmieszczenie przez podanie dla każdego obiektu $x \in X$ pudełka $f(x) \in Y$, w którym ten obiekt się znajduje. Inną, tradycyjną interpretację otrzymamy traktując Y jako zbiór „kolorów", a $f(x)$ jako „kolor obiektu x". Nasz problem jest więc równoważny pytaniu, na ile sposobów możemy pokolorować obiekty, tak by były spełnione zadane ograniczenia.

Zauważmy, że bez zmniejszenia ogólności możemy zawsze przyjąć, że $X = \{1, \ldots, n\}$ i $Y = \{1, \ldots, m\}$. Każdą funkcję f możemy wtedy identyfikować z ciągiem $\langle f(1), \ldots, f(n)\rangle$.

Problem nasz jest najprostszy, gdy nie nakładamy żadnych ograniczeń na rozmieszczenia. Mamy bowiem następujące twierdzenie:

TWIERDZENIE 1.1
Jeśli $|X| = n$, $|Y| = m$, to liczba wszystkich funkcji $f: X \to Y$ jest równa m^n.

DOWÓD
Przyjmując $X = \{1, \ldots, n\}$, sprowadzamy zagadnienie do pytania o liczbę wszystkich ciągów $\langle y_1, \ldots, y_n\rangle$ o wyrazach ze zbioru m-elementowego Y. Każdy wyraz y_i możemy wybrać na m sposobów, co daje m^n możliwości wyboru ciągu $\langle y_1, \ldots, y_n\rangle$.

* Zapisu symbolicznego $f(n) = O(g(n))$ nie należy traktować jako równości; na przykład z $f(n) = O(g(n))$ i $h(n) = O(g(n))$ oczywiście nie wynika $f(n) = h(n)$.

Łatwo również znaleźć liczbę rozmieszczeń, dla których każde pudełko zawiera co najwyżej jeden obiekt – takie rozmieszczenia odpowiadają funkcjom różnowartościowym. Oznaczamy przez $[m]_n$ liczbę wszystkich funkcji różnowartościowych ze zbioru n-elementowego w zbiór m-elementowy.

TWIERDZENIE 1.2

Jeśli $|X| = n$, $|Y| = m$, to liczba wszystkich funkcji różnowartościowych $f: X \to Y$ jest równa

$$[m]_n = m(m-1)\ldots(m-n+1) \tag{1.1}$$

(przyjmujemy $[m]_0 = 1$).

DOWÓD

Poszukujemy tym razem liczby ciągów różnowartościowych $\langle y_1, \ldots, y_n\rangle$ o wyrazach ze zbioru Y. Wyraz y_1 takiego ciągu możemy wybrać na m sposobów, wyraz y_2 na $m-1$ sposobów, ogólnie: przy ustalonych wyrazach $y_1, \ldots, y_{i-1}$ możemy jako y_i wybrać dowolny spośród $m-i+1$ elementów zbioru $Y \setminus \{y_1, \ldots, y_{i-1}\}$ (zakładamy $n \leqslant m$; jeśli $n > m$, to oczywiście zarówno $[m]_n$, jak i szukana liczba funkcji jest równa zeru). Daje to $m(m-1)\ldots(m-h+1)$ możliwości wyboru ciągu różnowartościowego $\langle y_1, \ldots, y_n\rangle$. ■

Oto dla przykładu $[4]_3 = 24$ ciągi różnowartościowe długości 3 o elementach ze zbioru $X = \{1, 2, 3, 4\}$:

⟨1, 2, 3⟩	⟨2, 1, 3⟩	⟨3, 1, 2⟩	⟨4, 1, 2⟩
⟨1, 2, 4⟩	⟨2, 1, 4⟩	⟨3, 1, 4⟩	⟨4, 1, 3⟩
⟨1, 3, 2⟩	⟨2, 3, 1⟩	⟨3, 2, 1⟩	⟨4, 2, 1⟩
⟨1, 3, 4⟩	⟨2, 3, 4⟩	⟨3, 2, 4⟩	⟨4, 2, 3⟩
⟨1, 4, 2⟩	⟨2, 4, 1⟩	⟨3, 4, 1⟩	⟨4, 3, 1⟩
⟨1, 4, 3⟩	⟨2, 4, 3⟩	⟨3, 4, 2⟩	⟨4, 3, 2⟩

Jeśli $m = n$, to każda funkcja różnowartościowa $f: X \to Y$ jest wzajemnie jednoznacznym odwzorowaniem zbioru X na zbiór Y. W takim przypadku $[n]_n = n(n-1)\ldots 1$ oznaczamy przez $n!$ (*n silnia*). Każde wzajemnie jednoznaczne odwzorowanie $f: X \to X$ nazywamy *permutacją* zbioru X. Jako szczególny przypadek twierdzenia 1.2 otrzymujemy

TWIERDZENIE 1.3

Liczba permutacji zbioru n-elementowego jest równa $n!$. ■

Permutacje rozważać będziemy w następnych punktach, teraz natomiast zajmiemy się jeszcze innym typem rozmieszczenia obiektów w pudełkach. Załóżmy, że rozmieszczamy n obiektów w m pudełkach, przy czym każde pudełko zawiera teraz ciąg, a nie jak poprzednio zbiór umieszczonych w nim obiektów. Dwa rozmieszczenia uważamy za równe, jeśli w każdym pudełku zawierają taki sam ciąg obiektów. Rozmieszczenia tego typu będziemy nazywać *rozmieszczeniami uporządkowanymi n obiektów w m pudełkach*. Oznaczamy liczbę takich uporządkowań przez $[m]^n$.

TWIERDZENIE 1.4

Liczba rozmieszczeń uporządkowanych n obiektów w m pudełkach jest równa

$$[m]^n = m(m+1)\ldots(m+n-1) \tag{1.2}$$

(przyjmujemy $[m]^0 = 1$).

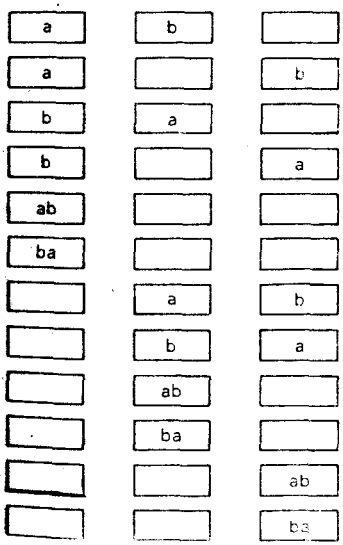
DOWÓD

Będziemy konstruować rozmieszczenie uporządkowane przez kolejne dodawanie nowych obiektów. Pierwszy obiekt możemy rozmieścić na m sposobów, drugi na $m+1$ sposobów, gdyż da się umieścić w jednym z $m-1$ pustych pudełek lub w pudełku zawierającym pierwszy obiekt przed lub za tym obiektem. Ogólnie, załóżmy, że rozmieściliśmy już $i-1$ obiektów, przy czym dla $k = 1, 2, \ldots, m$ w k-tym pudełku znajduje się r_k obiektów. Wtedy i-ty obiekt możemy dodać na r_k+1 sposobów do k-tego pudełka, co w sumie daje

$$(r_1+1)+\ldots+(r_m+1) = (r_1+\ldots+r_m)+m = m+i-1$$

możliwości. Wszystkich rozmieszczeń uporządkowanych jest więc $m(m+1)\ldots$ $\ldots(m+n-1)$. ■

Na rysunku 1.2 przedstawiono wszystkie $[3]^2 = 12$ rozmieszczeń uporządkowanych elementów a, b w trzech pudełkach.



RYS. 1.2 Rozmieszczenia uporządkowane elementów a, b w trzech pudełkach

Odnotujmy na zakończenie następujące proste zależności:

$$[m]_n = (m-n+1)[m]_{n-1} \tag{1.3}$$

$$[m]_n = m!/n! \tag{1.4}$$

$$[m]^n = [m+n-1]_n \tag{1.5}$$

Permutacje: rozkład na cykle, znak permutacji 1.

Przypomnijmy, że permutacją zbioru n-elementowego X nazywamy dowolną wzajemnie jednoznaczną funkcję $f: X \to X$. Zwykle permutację określamy za pomocą tablicy o dwóch wierszach, z których każdy zawiera wszystkie elementy zbioru X, przy czym element $f(x)$ występuje pod elementem x. Dla przykładu permutację f zbioru $\{a, b, c, d\}$, taką że

$$f(a) = d; \qquad f(b) = a; \qquad f(c) = c; \qquad f(d) = b$$

zapisujemy następująco:

$$f = \begin{pmatrix} a & b & c & d \\ d & a & c & b \end{pmatrix}$$

Jeśli ustalimy porządek elementów w górnym wierszu, to każdej permutacji odpowiada jednoznacznie ciąg zawarty w dolnym wierszu, np. dla permutacji f jest to $\langle d, a, c, b\rangle$. Dlatego też permutacją zbioru n-elementowego X będziemy niekiedy nazywać dowolny ciąg różnowartościowy długości n o elementach ze zbioru X.

W naszych rozważaniach natura elementów zbioru X będzie nieistotna – przyjmujemy dla prostoty $X = \{1, \ldots, n\}$. Oznaczmy zbiór wszystkich permutacji tego zbioru przez S_n. Dowolną permutację $f \in S_n$ będziemy zwykle identyfikować z ciągiem $\langle a_1, \ldots, a_n\rangle$, gdzie $a_i = f(i)$. Przez *złożenie* permutacji f, g rozumiemy permutację fg zdefiniowaną następująco:

$$fg(i) = f(g(i))$$

Zauważmy, że aby wyznaczyć złożenie dwóch permutacji, powiedzmy

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix} \qquad g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix}$$

wystarczy zmienić porządek kolumn w permutacji f tak, by w pierwszym wierszu uzyskać ciąg występujący w drugim wierszu permutacji g – drugi wiersz permutacji f określa wtedy złożenie fg. W naszym przypadku

$$g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix}$$

$$f = \begin{pmatrix} 2 & 5 & 3 & 1 & 4 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix}$$

$$fg = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix}$$

Permutację

$$e = \begin{pmatrix} 1 & 2 & \ldots & n \\ 1 & 2 & \ldots & n \end{pmatrix}$$

nazywamy *permutacją identycznościową*. Oczywiście $ef = fe = f$ dla dowolnej permutacji $f \in S_n$. Łatwo też zauważyć, że każda permutacja $f \in S_n$ wyznacza jednoznacznie permutację f^{-1}, taką że $ff^{-1} = f^{-1}f = e$. Nazywamy ją *permutacją odwrotną* do f. Aby ją wyznaczyć, wystarczy zamienić miejscami wiersze w przedstawieniu permutacji f. Na przykład dla

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$$

otrzymujemy

$$f^{-1} = \begin{pmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{pmatrix}$$

Z naszych rozważań wynika, że dla dowolnych permutacji $f, g, h \in S_n$ są spełnione warunki

$$(fg)h = f(gh) \tag{1.6}$$

$$fe = ef = f \tag{1.7}$$

$$f^{-1}f = ff^{-1} = e \tag{1.8}$$

Fakt ten wyrażamy mówiąc, że S_n tworzy *grupę* ze względu na działanie złożenia. Grupę tę nazywamy *grupą symetryczną stopnia n*. Dowolny podzbiór $G \subseteq S_n$ spełniający warunki

$$f, g \in G \Rightarrow fg \in G$$

$$f \in G \Rightarrow f^{-1} \in G$$

nazywamy *grupą permutacji stopnia n*.

Każdą permutację $f \in S_n$ możemy reprezentować graficznie za pomocą grafu zorientowanego o zbiorze wierzchołków $X = \{1, \ldots, n\}$, w którym $x \to y$ wtedy i tylko wtedy, gdy $f(x) = y$. Od każdego wierzchołka x tego grafu odchodzi dokładnie jedna krawędź, mianowicie $\langle x, f(x)\rangle$. Podobnie jedyną krawędzią dochodzącą do wierzchołka x jest $\langle f^{-1}(x), x\rangle$. Łatwo zauważyć, że startując z dowolnego wierzchołka x_0 i rozważając kolejno wierzchołki $x_1 = f(x_0), x_2 = f(x_1), \ldots$, dochodzimy po skończonej liczbie kroków do wierzchołka x_0, tzn. $x_l = f(x_{l-1}) = x_0$ dla pewnego $l \geqslant 1$. Stąd wniosek, że nasz graf składa się z pewnej liczby cykli elementarnych o rozłącznych zbiorach wierzchołków dających w sumie cały zbiór X. Załóżmy, że w rozkładzie tym występuje k cykli

$$a_0^{(i)} \to a_1^{(i)} \to \ldots \to a_{n_i-1}^{(i)} \to a_0^{(i)} \qquad i = 1, \ldots, k$$

Każdemu takiemu cyklowi odpowiada permutacja

$$f_i = [a_0^{(i)} a_1^{(i)} \ldots a_{n_i-1}^{(i)}]$$

zwana również *cyklem* (*długości* n_i), zdefiniowana następująco:

$$f_i(a_0^{(i)}) = a_1^{(i)}, \qquad f_i(a_1^{(i)}) = a_2^{(i)}, \ldots, \qquad f_i(a_{n_i-1}^{(i)}) = a_0^{(i)}$$

$$f_i(x) = x \text{ dla } x \in X \setminus \{a_0^{(i)}, \ldots, a_{n_i-1}^{(i)}\}$$

Naszą permutację możemy przedstawić w postaci złożenia cykli

$$f = [a_0^{(1)} a_1^{(1)} \dots a_{n_1-1}^{(1)}] [a_0^{(2)} a_1^{(2)} \dots a_{n_2-1}^{(2)}] \dots [a_0^{(k)} a_1^{(k)} \dots a_{n_k-1}^{(k)}]$$

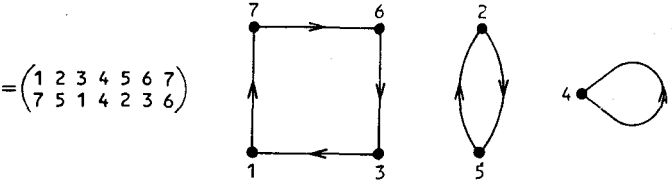
Takie przedstawienie permutacji nazywamy *rozkładem na cykle*. Powiemy, że permutacja f jest typu $\langle \lambda_1, \dots, \lambda_n \rangle$, jeśli zawiera ona w rozkładzie na cykle dokładnie λ_i cykli o długości i, dla $i = 1, 2, \dots, n$. Typ $\langle \lambda_1, \dots, \lambda_n \rangle$ zapisujemy zwykle symbolicznie $1^{\lambda_1} \dots n^{\lambda_n}$ (pomijając i^{λ_i}, jeśli $\lambda_i = 0$). Dla przykładu permutacja

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 1 & 4 & 2 & 3 & 6 \end{pmatrix}$$

ma następujący rozkład na cykle:

$$f = [1\ 7\ 6\ 3]\,[2\ 5]\,[4]$$

a więc jest ona typu $1^1 2^1 4^1$. Zilustrowano to na rys. 1.3.



Rys. 1.3 Rozkład permutacji na cykle

Parę $\langle a_i, a_j \rangle$, $i < j$ nazywamy *inwersją* premutacji $\langle a_i, \dots, a_n \rangle$, jeśli $a_i > a_j$. Dla dowolnej permutacji $f \in S_n$ oznaczamy przez $I(f)$ liczbę jej inwersji oraz definiujemy znak tej permutacji w następujący sposób:

$$sgn\,(f) = (-1)^{I(f)}$$

Permutację f nazywamy *parzystą*, jeśli $sgn\,(f) = 1$, oraz *nieparzystą*, jeśli $sgn\,(f) = -1$. Zilustrujemy te pojęcia na paru przykładach. Permutacja identycznościowa $\langle 1, \dots, n \rangle$ nie zawiera żadnych inwersji, a więc $I(e) = 0$, i jest to dla każdego n permutacja parzysta. Permutacja $\langle n, n-1, \dots, 1 \rangle$ ma $n(n-1)/2$ inwersji (liczba wszystkich par $\langle i, j \rangle$, $i \neq j$ jest równa $[n]_2 = n(n-1)$, przy czym każdej parze $\langle i, j \rangle$, $i < j$ odpowiada para $\langle j, i \rangle$, $j > i$). Nasza permutacja jest więc parzysta dla n postaci $4k$ lub $4k+1$ oraz nieparzysta w pozostałych przypadkach. Permutacja $\langle 2, 4, 3, 5, 1 \rangle$ ma następujące inwersje: $\langle 2, 1 \rangle$, $\langle 4, 3 \rangle$, $\langle 4, 1 \rangle$, $\langle 3, 1 \rangle$, $\langle 5, 1 \rangle$; jest więc nieparzysta.

Znak permutacji możemy wyznaczyć za pomocą bezpośredniego zliczenia wszystkich inwersji, jednakże algorytm taki wymaga na ogół liczby kroków rzędu co najmniej liczby inwersji, a więc $\Omega(n^2)$. Pokażemy teraz algorytm o złożoności $O(n)$. Będziemy do tego celu potrzebować kilku lematów.

Dowolną permutację będącą cyklem o długości 2 nazywamy *transpozycją*. Ważną rolę w dalszych rozważaniach odgrywać będą transpozycje sąsiednich elementów, tzn. transpozycje postaci $[i\ i+1]$.

LEMAT 1.5

Dowolną permutację $f \in S_n$ można przedstawić w postaci złożenia $I(f)$ transpozycji sąsiednich elementów.

DOWÓD

Zauważmy przede wszystkim, że jeśli f jest postaci $\langle a_1, \ldots, a_n\rangle$, a $t = [i\ i+1]$, to złożenie ft jest postaci $\langle a_1, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n\rangle$. Oznaczamy przez r_i liczbę inwersji wprowadzonych przez element i:

$$r_i = |\{j\colon j < i \wedge a_j > a_i\}|$$

Łatwo zauważyć, że w $\langle a_1, \ldots, a_n\rangle$ możemy sprowadzić element $1 = a_{r_1+1}$ na pierwszą pozycję, dokonując r_1 transpozycji sąsiednich elementów, następnie element 2 sprowadzić na drugą pozycję dokonując r_2 transpozycji sąsiednich elementów itd. Ostatecznie, po $r_1+\ldots+r_n = I(f)$ krokach otrzymujemy ciąg $\langle 1, \ldots, n\rangle$. Oznacza to, że $ft_1 \ldots t_{I(f)} = e$, gdzie $t_1, \ldots, t_{I(f)}$ są pewnymi transpozycjami sąsiednich elementów. A zatem

$$f = (t_1 \ldots t_{I(f)})^{-1} = t_{I(f)}^{-1} \ldots t_1^{-1}$$

co kończy dowód lematu, gdyż $t^{-1} = t$ dla dowolnej transpozycji t. ■

LEMAT 1.6

Dla dowolnych permutacji $f, g \in S_n$

$$sgn(fg) = sgn(f)\, sgn(g)$$

DOWÓD

Załóżmy najpierw, że g jest transpozycją postaci $t = [i\ i+1]$. Jeśli $f = \langle a_1, \ldots, a_n\rangle$, to

$$ft = \langle a_1, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n\rangle$$

i oczywiście

$$I(ft) = \begin{cases} I(f)+1 & \text{jeśli } a_i < a_{i+1} \\ I(f)-1 & \text{jeśli } a_i > a_{i+1} \end{cases}$$

W obu przypadkach $sgn(ft) = -(-1)^{I(f)} = -sgn(f)$. Dowolną permutację g możemy na mocy poprzedniego lematu przedstawić w postaci $t_1 \ldots t_k$, gdzie $t_1, \ldots, t_k$ są transpozycjami sąsiednich elementów i $k = I(g)$. Mamy

$$\begin{aligned} sgn(fg) &= sgn(ft_1 \ldots t_k) = -sgn(ft_1 \ldots t_{k-1}) = \ldots \\ &= (-1)^k sgn(f) = sgn(g)\, sgn(f) \end{aligned}$$

■

LEMAT 1.7

Każda transpozycja jest permutacją nieparzystą. Ogólniej, znak dowolnego cyklu o długości k jest równy $(-1)^{k-1}$.

DOWÓD

Pierwsza część lematu wynika z faktu, iż ciąg

$$\langle 1, \ldots, i-1, j, i+1, \ldots, j-1, i, j+1, \ldots, n\rangle$$

można sprowadzić do ciągu $\langle 1, \ldots, n\rangle$ dokonując najpierw $j-i$ transpozycji

$$[j-1\ \ j], [j-2\ \ j-1], \ldots, [i\ \ i+1]$$

a następnie $(j-i)-1$ transpozycji

$$[i+1\ \ i+2], [i+2\ \ i+3], \ldots, [j-1\ \ j]$$

Oznacza to, że transpozycja $[i\ j]$ może być przedstawiona w postaci złożenia $(j-i)+(j-i)-1 = 2(j-i)-1$ transpozycji sąsiednich pozycji. Zgodnie z poprzednimi lematami znak naszej transpozycji $[i\ j]$ jest równy $(-1)^{2(j-i)-1} = -1$.

Druga część lematu jest wnioskiem z pierwszej i z faktu, iż dowolny cykl $[a_1 \ldots a_k]$ jest złożeniem $k-1$ transpozycji:

$$[a_1 \ldots a_k] = [a_1\ a_2]\,[a_2\ a_3] \ldots [a_{k-1}\ a_k] \qquad \blacksquare$$

LEMAT 1.8

Znak dowolnej permutacji f typu $1^{\lambda_1} \ldots n^{\lambda_n}$ wyraża się wzorem

$$sgn(f) = (-1)^{\sum_{j=1}^{\lfloor n/2 \rfloor} \lambda_{2j}}$$

DOWÓD

Jeśli f ma w rozkładzie na cykle dokładnie λ_i cykli o długości i, to wobec poprzednich lematów

$$sgn(f) = \prod_{i=1}^{n} [(-1)^{i-1}]^{\lambda_i} = \prod_{j=1}^{\lfloor n/2 \rfloor} (-1)^{\lambda_{2j}} = (-1)^{\lambda_2+\lambda_4+\cdots} \qquad \blacksquare$$

Zauważmy, że o liczbie inwersji permutacji zbioru X możemy mówić jedynie w przypadku $X = \{1, \ldots n\}$, ogólniej: gdy na zbiorze X jest określony liniowy porządek. Jednakże znak permutacji zależy jedynie od jej typu, a więc jest niezależny od tego, jak uporządkujemy zbiór X.

Zapowiedziana efektywna metoda znajdowania znaku permutacji polega na znajdowaniu typu tej permutacji i korzystaniu z lematu 1.8.

ALGORYTM 1.9 (Znajdowanie znaku permutacji)

Dane: Dowolna permutacja $f \in S_n$ dana w postaci ciągu $P[1], \ldots, P[n]$ ($P[i] = f(i)$).

Wyniki: Po zakończeniu działania algorytmu $s = sgn(f)$.

```
1  begin
2    s := 1;
3    for i := 1 to n do NOWY[i] := true;
4    for i := 1 to n do
5      if NOWY[i] then (* znajdź cykl zawierający i *)
6        begin j := P[i];
7          while j ≠ i do
8            begin NOWY[j] := false;  s := -s;  j := P[j]
9            end
10       end
11 end
```

Algorytm ten przegląda kolejno pozycje 1, ..., n permutacji (pętla 4) i za każdym razem gdy element $P[i]$ nie był jeszcze analizowany ($NOWY[i]$ = **true**) wyznacza cykl, do którego ten element należy (blok 6). Zauważmy, że jeśli cykl ten jest długości k, to pętla 7 jest wykonywana $k-1$ razy. W konsekwencji wykonanie tej pętli zmienia znak zmiennej s na przeciwny wtedy i tylko wtedy, gdy k jest parzyste. Wartość początkowa zmiennej s jest równa jedności, a więc po znalezieniu wszystkich cykli $s = (-1)^p$, gdzie p jest liczbą cykli parzystej długości. Wobec lematu 1.8 mamy $s = sgn(f)$.

Łatwo się też przekonać, że liczba kroków algorytmu jest rzędu n dla dowolnej permutacji $f \in S_n$. Aby to wykazać, zauważmy, że sumaryczna liczba kroków wykonywanych w pętli 4, nie licząc kroków w wewnętrznym bloku 6, jest $O(n)$. Sumaryczna liczba kroków wykonywanych w bloku 6 w trakcie działania algorytmu jest rzędu sumy długości wszystkich cykli, zatem również $O(n)$. Daje to całkowitą złożoność $O(n)$.

Generowanie permutacji 1.4

Zajmiemy się teraz algorytmami generowania wszystkich $n!$ permutacji zbioru n-elementowego. Zagadnienie to ma bogatą literaturę (por. np. artykuł [60]) i jeszcze dłuższą historię, której źródeł można się doszukać w początkach XVII wieku, kiedy to w Anglii powstała specjalna sztuka bicia w dzwony polegająca – w pewnym uproszczeniu – na wybiciu na n różnych dzwonach wszystkich $n!$ permutacji [14], [74]. Premutacje te należało wybijać „z pamięci", co spowodowało, iż adepci tej sztuki znaleźli pewne proste metody systematycznego generowania wszystkich permutacji (bez powtórzeń). Niektóre z tych metod zostały niezależnie odkryte współcześnie w kontekście maszyn cyfrowych. Sztuka ta – choć mało znana w Polsce – przetrwała po nasze czasy, skoro słynna *Księga Rekordów Guinnesa* [29] zawiera wzmiankę o wybiciu wszystkich 8! = 40320 permutacji na 8 dzwonach w 1963 roku – ustanowienie tego rekordu trwało 17 godzin 58 $\frac{1}{2}$ minuty! Oczywyście użycie maszyny cyfrowej pozwala generować per-

mutacje znacznie szybciej, jednak różnica nie jest aż tak duża, jak by się to mogło wydawać – w ciągu 18 godzin nawet bardzo szybka maszyna nie wygeneruje wszystkich permutacji zbioru n-elementowego, jeśli $n > 13$ (z drugiej strony permutacje zbioru 8-elementowego można otrzymać w ciągu ułamka sekundy). Jest to po prostu konsekwencja faktu, iż $n!$ rośnie bardzo szybko ze wzrostem n.

W punkcie tym opiszemy trzy różne metody generowania ciągu wszystkich $n!$ permutacji zbioru n-elementowego. Zakładać będziemy, że elementy naszego zbioru są pamiętane w tablicy $P[1], \ldots, P[n]$. We wszystkich trzech metodach elementarną operacją, której poddaje się tablicę P, jest pojedyncza transpozycja, tzn. wymiana zawartości zmiennych $P[i]$ oraz $P[j]$, gdzie $1 \leqslant i, j \leqslant n$. Operację tę oznaczać będziemy przez $P[i] :=: P[j]$. Oczywiście jest ona równoważna ciągowi instrukcji

$$pom := P[i]; \qquad P[i] := P[j]; \qquad P[j] := pom;$$

gdzie *pom* jest pewną zmienną pomocniczą.

Pierwszą z metod, które opiszemy, najłatwiej zrozumieć, jeśli permutowanymi elementami są liczby $1, 2, \ldots, n$. Na zbiorze wszystkich permutacji — ogólniej, na zbiorze wszystkich ciągów długości n o elementach ze zbioru $X = \{1, \ldots, n\}$ – jest określony *porządek leksykograficzny*:

$\langle x_1, \ldots, x_n \rangle < \langle y_1, \ldots, y_n \rangle$

$\Leftrightarrow$ istnieje $k \geqslant 1$, takie że $x_k < y_k$ oraz $x_l = y_l$ dla każdego $l < k$

Zauważmy, że jeśli zamiast liczb $1, \ldots, n$ rozważać litery $a, b, \ldots$ z naturalnym porządkiem $a < b < c < \ldots < z$, to porządek leksykograficzny określa standardową kolejność, w jakiej wyrazy o długości n pojawiają się w słowniku. *Porządek antyleksykograficzny*, oznaczany przez $<'$, definiujemy podobnie, z tym że zarówno kolejność pozycji w ciągu, jak i uporządkowanie elementów zbioru X ulega odwróceniu:

$\langle x_1, \ldots, x_n \rangle <' \langle y_1, \ldots, y_n \rangle$

$\Leftrightarrow$ istnieje $k \leqslant n$, takie że $x_k > y_k$ oraz $x_l = y_l$, dla $l > k$

Dla przykładu podajemy teraz permutacje zbioru $X = \{1, 2, 3\}$ w porządku leksykograficznym (a) oraz antyleksykograficznym (b):

(a)	(b)
1 2 3	1 2 3
1 3 2	2 1 3
2 1 3	1 3 2
2 3 1	3 1 2
3 1 2	2 3 1
3 2 1	3 2 1

Nieco wygodniej jest sformułować algorytm generowania permutacji w porządku antyleksykograficznym. Zauważmy w tym celu, że ciąg permutacji zbioru $\{1, \ldots, n\}$ w tym porządku ma następujące własności, wynikające bezpośrednio z definicji:

(A1) W pierwszej permutacji elementy występują w rosnącej kolejności, a w ostatniej w malejącej; innymi słowy, ostatnia permutacja jest odwróceniem pierwszej.

(A2) Nasz ciąg można podzielić na n bloków długości $(n-1)!$ odpowiadających malejącym wartościom elementu w ostatniej pozycji. Pierwsze $n-1$ pozycji bloku zawierającego element p w ostatniej pozycji określa ciąg premutacji zbioru $\{1, \ldots, n\} \setminus \{p\}$ w porządku antyleksykograficznym.

Własności te sugerują następujący prosty algorytm rekurencyjny:

ALGORYTM 1.10 (Generowanie wszystkich permutacji w porządku antyleksykograficznym)

Dane: n

Wyniki: Ciąg permutacji zbioru $\{1, \ldots, n\}$ w porządku antyleksykograficznym.

```
1  procedure REVERSE(m); (* odwrócenie ciągu P[1],..., P[m];
   tablica P jest globalna *)
2  begin i := 1; j := m;
3    while i < j do
4      begin P[i] :=: P[j]; i := i+1; j := j-1
5      end
6  end; (* REVERSE *)
7  procedure ANTYLEX(m);
   (* tablica P jest globalna *)
8  begin
9    if m = 1 then (* P[1], ..., P[n] zawiera nową permutację *)
10     write (P[1], ..., P[n])
11   else
12     for i := 1 to m do
13       begin ANTYLEX(m-1);
14         if i < m then
15           begin P[i] :=: P[m]; REVERSE(m-1)
16           end
17       end
18 end; (* ANTYLEX *)
19 begin (* program główny *)
20   for i := 1 to n do P[i] :=: i;
21   ANTYLEX(n)
22 end
```

Aby zrozumieć działanie tego algorytmu zauważmy przede wszystkim, że wykonanie procedury *REVERSE*(m) powoduje odwrócenie kolejności ciągu elementów w $P[1], \ldots, P[m]$. Aby udowodnić poprawność algorytmu wystarczy pokazać, przez indukcję względem m, że jeśli $P[1] < \ldots < P[m]$, to wywołanie *ANTYLEX*(m) powoduje wygenerowanie wszystkich permutacji zbioru $\{P[1], \ldots P[m]\}$ w porządku antyleksykograficznym (przy nie zmienionych wartościach $P[m+1], \ldots, P[n]$). Załóżmy $P[i] = a_i$, $1 \leqslant i \leqslant m$, $a_1 < \ldots < a_m$ oraz rozważmy pętlę 12. Efekt wykonania pierwszej iteracji tej pętli jest następujący:

$P[1]$	$P[2]$	...	$P[m-2]$	$P[m-1]$	$P[m]$	
a_1	a_2		a_{m-2}	a_{m-1}	a_m	
a_{m-1}	a_{m-2}		a_2	a_1	a_m	(po wykonaniu *ANTYLEX*$(m-1)$)
a_m	a_{m-2}		a_2	a_1	a_{m-1}	(po transpozycji $P[1] :=: P[m]$)
a_1	a_2		a_{m-2}	a_m	a_{m-1}	(po wykonaniu *REVERSE*$(m-1)$)

(Skorzystaliśmy tu z założenia indukcyjnego, że *ANTYLEX*$(m-1)$ poprawnie generuje wszystkie permutacje elementów $a_1, \ldots, a_{m-1}$ w porządku antyleksykograficznym). W analogiczny sposób, przez indukcję względem i dowodzimy, że i-ta iteracja pętli 12 powoduje wygenerowanie wszystkich permutacji elementów $a_1, \ldots, a_{m-i}, a_{m-i+2}, \ldots, a_m$ przy $P[m] = a_{m-i+1}$. Wobec własności A2 oznacza to, że *ANTYLEX*(m) generuje wszystkie permutacje elementów $a_1, \ldots, a_m$ w porządku antyleksykograficznym.

Warto zauważyć, że fakt, iż warunek $P[i] = i$, $1 \leqslant i \leqslant n$ na początku działania programu (p. wiersz 20) był istotny jedynie dla łatwiejszego zrozumienia działania algorytmu. Sam algorytm sformułowany jest całkowicie w terminach pozycji, których wartości ulegają wymianie, i nie korzysta w żadnym stopniu z zawartości zmiennych $P[1], \ldots, P[n]$.

Zastanówmy się teraz, ile transpozycji wykonuje algorytm 1.10 przy wygenerowaniu każdej następnej permutacji. Łatwo zauważyć, że liczba ta jest zmienna: co druga permutacja generowana jest kosztem jednej transpozycji $P[1] :=: P[2]$, lecz są i takie, które wymagają $\lfloor (n-1)/2 \rfloor + 1 = \lfloor (n+1)/2 \rfloor$ transpozycji. Choć średnia liczba transpozycji przypadających na każdą permutację jest niewielka (por. zad. 1.13), to jednak w niektórych zastosowaniach lepszy byłby algorytm, w którym każda następna permutacja powstaje z poprzedniej przez dokonanie jednej tylko transpozycji. Może to być istotne w sytuacji, gdy z każdą permutacją związane są pewne obliczenia, i gdy istnieje możliwość korzystania z częściowych wyników uzyskanych dla poprzedniej permutacji, jeśli kolejne permutacje mało się od siebie różnią.

Pokażemy teraz, że algorytm taki jest rzeczywiście możliwy. Jego zasadniczy schemat można opisać za pomocą następującej procedury rekurencyjnej:

```
1  procedure PERM(m);
   (* tablica P jest globalna *)
2  begin
3    if m = 1 then (* P[1], ..., P[n] zawiera nową permutację *)
4      write (P[1], ..., P[n])
5    else
6      for i := 1 to m do
7        begin PERM(m-1);
8          if i < m then P[B[m, i]] :=: P[m]
9        end
10 end
```

Zadaniem tej procedury jest generowanie wszystkich permutacji elementów $P[1], \ldots, P[n]$ przez kolejne generowanie wszystkich permutacji elementów $P[1], \ldots, P[n-1]$ i zamianę elementu $P[n]$ z jednym z elementów $P[1], \ldots, P[n-1]$ wyznaczonym przez tablicę $B[m, i]$, $1 \leqslant i < m \leqslant n$. Oczywiście, aby ten schemat działał poprawnie, musimy określić tablicę B tak, by zagwarantować, że każda transpozycja $P[B[m, i]] :=: P[m]$ w wierszu 8 wprowadza inny element do $P[m]$. Podamy teraz algorytm, w którym wartość $B[m, i]$ obliczana jest dynamicznie jako funkcja m oraz i [44].

ALGORYTM 1.11 (Generowanie wszystkich permutacji przez minimalną liczbę transpozycji)

Dane: n

Wyniki: Ciąg permutacji zbioru $\{1, \ldots, n\}$, w którym każda następna powstaje z poprzedniej przez dokonanie pojedynczej transpozycji.

```
1  procedure B(m, i);
2  begin
3    if (m mod 2 = 0) and (m > 2) then
4      if i < m-1 then B := i
5      else B := m-2
6    else B := m-1
7  end; (* B *)
8  procedure PERM(m);
   (* tablica P jest globalna *)
9  begin
10   if m = 1 then (* P[1], ..., P[n] jest nową permutacją *)
11     write (P[1], ..., P[n])
12   else
13     for i := 1 to m do
14       begin PERM(m-1);
15         if i < m then P[B(m, i)] :=: P[m]
16       end
```

```
17  end; (* PERM *)
18  begin (* program główny *)
19    for i := 1 to n do P[i] :=: i;
20    PERM(n)
21  end
```

Zauważmy, że dla nieparzystego m transpozycja w wierszu 15 sprowadza się do $P[m-1] :=: P[m]$, dla każdego $i < m$, dla parzystego m natomiast zawartość $P[m]$ wymieniana jest kolejno z zawartością $P[1], P[2], \ldots, P[m-3], P[m-2], P[m-2]$ ($P[1]$ dla $m = 2$).

Dla każdego $m \geqslant 1$ określimy permutację φ_m następująco:
$\varphi_m(i)$ = indeks j taki, że $P[j]$ zawiera początkową wartość zmiennej $P[i]$ po wykonaniu $PERM(m)$

Dla przykładu, jeśli początkowo zmienne $P[1], \ldots, P[4]$ zawierają ciąg 1 2 3 4, to łatwo się przekonać, że po wykonaniu $PERM(4)$ ciąg ten zmienia się na 4 1 2 3. Oznacza to, że φ_4 jest cyklem $[1\ 2\ 3\ 4]$.

Pokażemy teraz poprawność algorytmu 1.11; dokładniej, udowodnimy dla każdego $m \geqslant 1$ następujący warunek:

Warunek W_m: Wykonanie $PERM(m)$ powoduje wygenerowanie wszystkich permutacji elementów $P[1], \ldots, P[m]$, przy czym φ_m jest transpozycją $[m\ m-1]$ jeśli m jest nieparzyste ($m > 1$) oraz cyklem $[1\ 2 \ldots m]$ jeśli m parzyste.

Dowód przebiega przez indukcję względem m, jak zwykle w przypadku dowodu poprawności algorytmów rekurencyjnych. Łatwo się bezpośrednio przekonać, że warunki W_1, W_2 są prawdziwe niech więc $m \geqslant 3$. Wykażemy W_m przy założeniu prawdziwości W_{m-1}. Dowód rozbijemy na dwa przypadki, w zależności od tego, czy m jest parzyste czy nieparzyste.

m nieparzyste. Na mocy założenia indukcyjnego wykonanie $PERM(m-1)$ w wierszu 14 powoduje za każdym razem przesunięcie zawartości $P[1], \ldots, P[m-1]$ wzdłuż cyklu $[1\ 2 \ldots m-1]$. Transpozycja $P[m] :=: P[m-1]$ w wierszu 15 wybiera więc za każdym razem inny element do $P[m]$. Jeśli początkowo $P[i] = a_i$, $1 \leqslant i \leqslant m$, to w $P[m]$ są umieszczane kolejno elementy $a_m, a_{m-2}, a_{m-3}, \ldots, a_1, a_{m-1}$. $PERM(m)$ generuje więc wszystkie permutacje elementów $P[1], \ldots, P[m]$.
Zauważmy, że przesunięcie $P[1], \ldots, P[m-1]$ wzdłuż cyklu $[1\ 2 \ldots m-1]$, a następnie dokonanie transpozycji $P[m] :=: P[m-1]$ jest równoważne przesunięciu $P[1], \ldots, P[m]$ wzdłuż cyklu $[1\ 2 \ldots m-3\ m-2\ m\ m-1]$ długości m. Gdyby transpozycja w wierszu 15 była dokonywana dla każdego $i \leqslant m$, to wykonanie pętli 13 spowodowałoby sprowadzenie wszystkich elementów na swoje początkowe miejsca. W rzeczywistości natomiast ostatnia transpozycja, dla $i = n$, nie jest wykonywana. Zatem $\varphi_m = [m\ m-1]$.

m parzyste. Prześledźmy zawartość tablicy P w czasie wykonywania $PERM(m)$. Zmiany, jakim podlega ta zawartość, przedstawiono w tablicy 1.1.

TABLICA 1.1 Zmiana zawartości zmiennych $P[1], \ldots, P[m]$ w czasie wykonywania procedury $PERM(m)$, m parzyste.

Liczba wykonanych iteracji pętli 13	$P[1]$	$P[2]$	...	$P[m-3]$	$P[m-2]$	$P[m-1]$	$P[m]$
0	a_1	a_2		a_{m-3}	a_{m-2}	a_{m-1}	a_m
1	a_m	a_2		a_{m-3}	a_{m-1}	a_{m-2}	a_1
2	a_m	a_1		a_{m-3}	a_{m-2}	a_{m-1}	a_2
⋮							
$m-3$	a_m	a_1		a_{m-4}	a_{m-1}	a_{m-2}	a_{m-3}
$m-2$	a_m	a_1		a_{m-4}	a_{m-3}	a_{m-1}	a_{m-2}
$m-1$	a_m	a_1		a_{m-4}	a_{m-2}	a_{m-3}	a_{m-3}
m	a_m	a_1		a_{m-4}	a_{m-3}	a_{m-2}	a_{m-1}

Z tablicy tej wynika, że każdy element pojawia się w $P[m]$ – a więc $PERM(m)$ generuje wszystkie permutacje – oraz że φ_m jest cyklem $[1\,2 \ldots m]$. Dowód poprawności algorytmu jest tym samym zakończony.

Algorytm 1.11 ma pewną własność, która może być bardzo pożyteczna w niektórych zastosowaniach. Wyobraźmy sobie, że poszukujemy permutacji spełniającej określone warunki. W sytuacji takiej, jeśli dla pewnego m wartości $P[m+1], \ldots, P[n]$ nie spełniają żądanych ograniczeń, nie ma potrzeby wywoływania $PERM(m)$ i przechodzenia przez wszystkie $m!$ permutacji elementów $P[1], \ldots, P[m]$. Permutacje te możemy opuścić przez dokonanie permutacji elementów $P[1], \ldots, P[m]$ określonej przez φ_m. Zauważmy, że permutacje φ_m mają w naszym przypadku bardzo prostą postać.

Ostatni algorytm generowania permutacji, który przedstawimy, konstruuje ciąg, w którym różnice między kolejnymi permutacjami są jeszcze mniejsze: każda następna powstaje z poprzedniej przez pojedynczą transpozycję sąsiednich elementów. Algorytm ten jest przypisywany zwykle Johnsonowi [38] i Trotterowi [68]. Jego ideę najlepiej zilustrować na przykładzie. Załóżmy, że skonstruowaliśmy już ciąg permutacji elementów $2, 3, \ldots, n$ o tej własności, np. 2 3, 3 2 dla $n = 3$. Wtedy żądany ciąg permutacji elementów $1, 2, \ldots, n$ otrzymujemy wstawiając element 1 na wszystkie możliwe sposoby do każdej permutacji elementów $2, 3, \ldots, n$. W naszym przypadku otrzymujemy

1 2 3
2 1 3
2 3 1
3 2 1
3 1 2
1 3 2

Ogólnie element 1 przesuwany jest między pierwszą a ostatnią pozycją, na przemian w przód i w tył $(n-1)!$ razy.

Na podstawie tej konstrukcji możemy łatwo otrzymać algorytm rekurencyjny generujący żądany ciąg permutacji dla dowolnego n. Jednakże metoda ta zastosowana bezpośrednio ma tę wadę, że konstruuje ciąg permutacji „w całości", i dopiero po zakończeniu całej konstrukcji można go odczytać. Oczywiście rozwiązanie tego typu wymagałoby na ogół olbrzymiej zajętości pamięci. Dlatego też pokażemy teraz wersję nierekurencyjną tego algorytmu. W wersji tej dla każdego i, $1 \leqslant i < n$, zmienna boolowska $PR[i]$ zawiera informację o tym, czy element i jest przesuwany do przodu ($PR[i]$ = **true**), czy też do tyłu ($PR[i]$ = = **false**), zmienna $C[i]$ wskazuje natomiast, którą z możliwych $n-i+1$ pozycji element i zajmuje względem elementów $i+1, \ldots, n$ na swej drodze w przód lub w tył. Pozycję elementu i w tablicy P wyznaczamy na podstawie jego pozycji w bloku zawierającym $i, i+1, \ldots, n$, oraz liczby elementów spośród $1, 2, \ldots i-1$, które znajdują się na lewo od tego bloku. Liczba ta, będąca wartością zmiennej x, obliczana jest jako liczba elementów $j < i$, które poruszając się do tyłu osiągnęły swe skrajne lewe położenie ($C[j] = n-j+1, PR[j]$ = **false**). Każda nowa permutacja powstaje przez transpozycję najmniejszego spośród elementów j, które nie znajdują się w skrajnym położeniu (tzn. $C[j] < n-j+1$) z jego lewym lub prawym sąsiadem. Realizuje to poniższy algorytm – dokładny dowód jego poprawności pozostawiamy Czytelnikowi.

ALGORYTM 1.12 (Generowanie wszystkich permutacji przez minimalną liczbę transpozycji sąsiednich elementów)

Dane: n

Wyniki: Ciąg permutacji zbioru $\{1, \ldots, n\}$, w którym każda następna powstaje przez dokonanie pojedynczej transpozycji sąsiednich elementów.

```
 1  begin
 2    for i := 1 to n do
 3      begin P[i] := i; C[i] := 1; PR[i] := true
 4      end;
 5    C[n] := 0; (* tak, by C[i] ≠ n-i+1 w wierszu 10 dla i = n *)
 6    write (P[1], ..., P[n]);
 7    i := 1;
 8    while i < n do
 9      begin i := 1; x := 0;
10        while C[i] = n-i+1 do
11          begin PR[i] := not PR[i]; C[i] := 1;
12            if PR[i] then x := x+1;
13            i := i+1
14          end,
```

```
15          if i <> n then
16            begin (* dokonanie transpozycji *)
17              if PR[i] then k := C[i]+x
18              else k := n-i+1-C[i]+x;
19              P[k] := : P[k+1];
20              write (P[1], ..., P[n]);
21              C[i] := C[i]+1
22            end
23      end
24  end
```

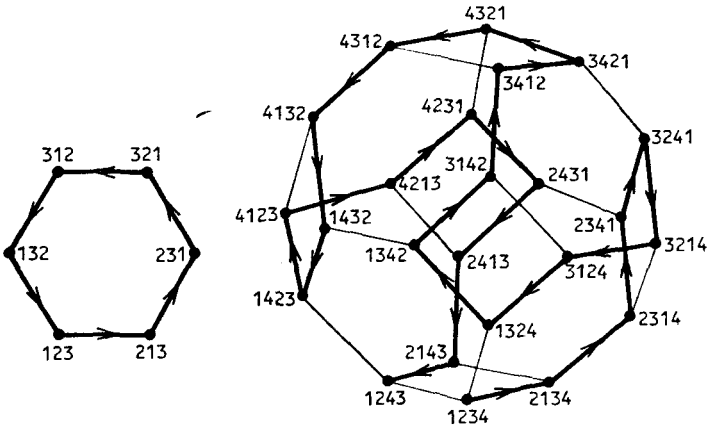
Zauważmy, że algorytm ten, podobnie jak oba poprzednie, nie korzysta w żadnym stopniu z zawartości zmiennych $P[1], \ldots, P[n]$. Jest to oczywiste, gdyż zmienne te występują jedynie w wierszu 19 (jeśli nie liczyć inicjalizacji w wierszu 3 i wyprowadzania wyników).

Na rysunku 1.4 przedstawiono ciągi permutacji wygenerowane dla $n = 4$ przez algorytmy 1.10, 1.11 i 1.12.

a)	b)	c)
1 2 3 4	1 2 3 4	1 2 3 4
2 1 3 4	2 1 3 4	2 1 3 4
1 3 2 4	2 3 1 4	2 3 1 4
3 1 2 4	3 2 1 4	2 3 4 1
2 3 1 4	3 1 2 4	3 2 4 1
3 2 1 4	1 3 2 4	3 2 1 4
1 2 4 3	4 3 2 1	3 1 2 4
2 1 4 3	3 4 2 1	1 3 2 4
1 4 2 3	3 2 4 1	1 3 4 2
4 1 2 3	2 3 4 1	3 1 4 2
2 4 1 3	2 4 3 1	3 4 1 2
4 2 1 3	4 2 3 1	3 4 2 1
1 3 4 2	4 1 3 2	4 3 2 1
3 1 4 2	1 4 3 2	4 3 1 2
1 4 3 2	1 3 4 2	4 1 3 2
4 1 3 2	3 1 4 2	1 4 3 2
3 4 1 2	3 4 1 2	1 4 2 3
4 3 1 2	4 3 1 2	4 1 2 3
2 3 4 1	4 2 1 3	4 2 1 3
3 2 4 1	2 4 1 3	4 2 3 1
2 4 3 1	2 1 4 3	2 4 3 1
4 2 3 1	1 2 4 3	2 4 1 3
3 4 2 1	1 4 2 3	2 1 4 3
4 3 2 1	4 1 2 3	1 2 4 3

Rys. 1.4 Ciągi permutacji wygenerowane przez
(a) algorytm 1.10
(b) algorytm 1.11
(c) algorytm 1.12

Pokażemy jeszcze na zakończenie pewną ciekawą interpretację ciągu wygenerowanego przez algorytm 1.12. Rozważmy w tym celu graf G_n, którego wierzchołki odpowiadają wszystkim permutacjom zbioru $\{1, \ldots, n\}$, i w którym dwa wierzchołki odpowiadające permutacjom f, g połączone są krawędzią wtedy i tylko wtedy, gdy g powstaje z f przez pojedynczą transpozycję sąsiednich elementów



Rys. 1.5 Interpretacja ciągu permutacji wygenerowanego przez algorytm 1.12

(tak więc każdy wierzchołek połączony jest z dokładnie $n-1$ innymi wierzchołkami). Nietrudno zauważyć, że ciąg permutacji wygenerowany przez algorytm 1.12 odpowiada *drodze Hamiltona* w G_n, tzn. drodze zawierającej każdy wierzchołek grafu dokładnie raz. Przedstawiono to, dla $n = 3$ i $n = 4$, na rysunku 1.5.

Podzbiory zbioru, zbiory z powtórzeniami, generowanie podzbiorów zbioru 1.

Każdy zbiór n-elementowy $X = \{x_1, \ldots, x_n\}$ ma dokładnie 2^n podzbiorów. Aby się o tym przekonać wystarczy każdemu podzbiorowi $Y \subseteq X$ przyporządkować ciąg binarny (tzn. o wyrazach 0 lub 1) $b_1 b_2 \ldots b_n$ określony następująco:

$$b_i = \begin{cases} 0 & \text{jeśli } x_i \notin Y \\ 1 & \text{jeśli } x_i \in Y \end{cases}$$

Otrzymujemy w ten sposób wzajemnie jednoznaczną odpowiedniość pomiędzy elementami zbioru $\mathscr{P}(X)$ wszystkich podzbiorów zbioru X a wszystkimi ciągami binarnymi długości n. Ciągów takich jest oczywiście 2^n, a więc, na mocy naszej odpowiedniości, również $|\mathscr{P}(X)| = 2^n$.

Określony przez nas ciąg $b_1 b_2 \ldots b_n$ stanowi również wygodną reprezentację maszynową podzbioru Y, szczególnie w sytuacji, gdy liczność zbioru X jest niewielka i ciąg $b_1 b_2 \ldots b_n$ może być zakodowany w jednym słowie maszynowym. Reprezentacja taka sugeruje prostą metodę generowania wszystkich podzbiorów. Wystarczy zauważyć, że każdy ciąg binarny $b_{n-1} b_{n-2} \ldots b_0$ odpowiada wzajemnie jednoznacznie pewnej liczbie z przedziału $0 \leqslant r \leqslant 2^n - 1$, a mianowicie liczbie $r = \sum_{i=0}^{n-1} b_i 2^i$, której $b_{n-1} b_{n-2} \ldots b_0$ jest reprezentacją binarną. Liczbę tę oznaczać będziemy przez $[b_{n-1} b_{n-2} \ldots b_0]$. Tak więc możemy wygenerować kolejno

wszystkie liczby r z przedziału $0 \leqslant r \leqslant 2^n - 1$ (na przykład zaczynając od 0 i dodając 1 w każdym kroku), a ich reprezentacje binarne określą wszystkie podzbiory zbioru n-elementowego. Metoda ta jest szczególnie wygodna dla realizacji w języku wewnętrznym maszyny.

W niektórych sytuacjach zależy nam na tym, by każdy następny wygenerowany podzbiór jak najmniej różnił się od poprzedniego. Motywacja przydatności algorytmu o tego typu własności jest analogiczna jak w przypadku algorytmów 1.11 i 1.12 generowania permutacji: przy dokonywaniu pewnych obliczeń związanych z każdym wygenerowanym podzbiorem istnieje możliwość wykorzystania wyników częściowych uzyskanych dla poprzedniego podzbioru. Zauważmy, że w przypadku opisanego przez nas algorytmu opartego na reprezentacji binarnej liczby, kolejne wygenerowane podzbiory mogą się znacznie od siebie różnić: na przykład po zbiorze $(n-1)$-elementowym odpowiadającym 011 ... 1 następuje jednoelementowy zbiór odpowiadający 100 ... 0.

Opiszemy teraz inną metodę, w której każdy następny podzbiór powstaje z poprzedniego przez dodanie lub odjęcie jednego elementu. Opiera się on na następującej prostej obserwacji. Jeśli ciąg $C_1, C_2, \ldots, C_m$ zawiera wszystkie $m = 2^k$ ciągów binarnych długości k, przy czym C_i różni się od C_{i+1} na dokładnie jednej współrzędnej $(i = 1, \ldots, m-1)$, to ciąg

$$C_1\,0, C_2\,0, \ldots, C_m\,0, C_m\,1, C_{m-1}\,1, \ldots, C_1\,1$$

zawiera wszystkie ciągi binarne długości $k+1$, przy czym każde dwa sąsiednie ciągi różnią się na dokładnie jednej współrzędnej. Określa to bezpośrednio pewien algorytm rekurencyjny konstruowania ciągu wszystkich podzbiorów. Otrzymany w ten sposób ciąg $C_1, \ldots, C_{2n}$, nazywamy *binarnym kodem Grey'a rzędu n*. Podamy teraz wersję nierekurencyjną tego algorytmu.

ALGORYTM 1.13 (Generowanie wszystkich podzbiorów zbioru n-elementowego)

Dane: n

Wyniki: Ciąg wszystkich podzbiorów zbioru n-elementowego, w którym każdy następny podzbiór powstaje z poprzedniego przez dodanie lub odjęcie pojedynczego elementu. Każdy podzbiór reprezentowany jest przez ciąg binarny $B[1], \ldots, B[n]$.

```
1  begin
2    for i := 1 to n do B[i] := 0; (* zbiór pusty *)
3    i := 0; (* i = liczba dotychczas wygenerowanych podzbiorów *)
4    repeat
5      write (B[1], ..., B[n]);
6      i := i+1; p := 1; j := i;
7      while j mod 2 = 0 do
8        begin (* j2^(p-1) = i *)
9          j := j/2; p := p+1
10       end; (* p = Q(i)+1 *)
```

```
11      if p ⩽ n then B[p] := 1−B[p] (* zmiana na pozycji p *)
12    until p > n
13  end
```

Udowodnimy teraz, że opisany algorytm istotnie generuje wszystkie ciągi binarne długości n. Niech $Q(i)$ oznacza najwyższą potęgę dwójki, która dzieli i. Jeśli przedstawimy i w postaci binarnej jako $i = [b_n b_{n-1} \ldots b_1]$, to mamy oczywiście $Q(i)+1 = min\{j : b_j = 1\}$. Pokażemy najpierw, że po wyjściu z wewnętrznej pętli 7 mamy $p = Q(i)+1$. W tym celu zauważmy, że przy wejściu do pętli $j2^{p-1} = i$ (gdyż $p = 1, j = i$) oraz każda iteracja pętli nie narusza tej równości (jako że $(j/2)\,2^{(p+1)-1} = j2^{p-1}$). Przy wyjściu z pętli j jest nieparzyste, a więc $Q(i) = p-1$, czyli istotnie $p = Q(i)+1$. Załóżmy teraz, że $k < n$ oraz że w pierwszych 2^k iteracjach pętli 3 wygenerowane zostały wszystkie 2^k ciągi binarne $b_1\, b_2 \ldots b_n$ takie, że $b_{k+1} = \ldots = b_n = 0$ (jest to oczywiście prawdą dla $k = 0$). W ostatniej spośród tych 2^k iteracji zmienna i przybiera wartość 2^k (wiersz 7). Daje to $p = Q(2^k)+1 = k+1$, i w konsekwencji następuje zmiana wartości zmiennej $B[k+1]$ z 0 na 1. Przyjrzyjmy się teraz ciągowi

$$Q(2^k+1)+1,\ Q(2^k+2)+1,\ \ldots,\ Q(2^{k+1})+1 \tag{1.9}$$

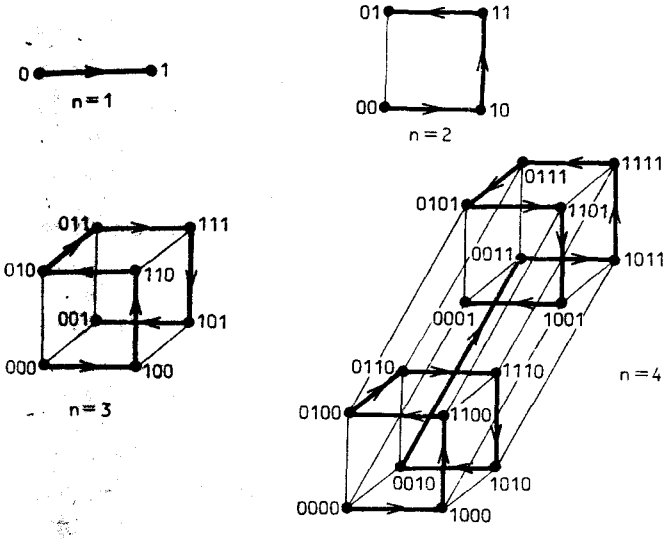
wartości zmiennej p wygenerowanych w następnych 2^k iteracjach pętli 3. Zauważmy, że $Q(2^k+m) = Q(2^k-m)$ dla $0 \leqslant m \leqslant 2^k-1$ (fakt ten jest szczególnie jasno widoczny, jeśli rozważyć dodawanie liczb 2^k+m i 2^k-m zapisanych w postaci binarnej). Ciąg (1.9) jest więc lustrzanym odbiciem ciągu wartości zmiennej p generowanych przez pierwsze 2^k iteracji pętli 3. Stąd wynika, że również ciągi $B[1], B[2], \ldots, B[k]$ uzyskiwane w pierwszych 2^k iteracjach pojawiają się, w odwrotnej kolejności, w następnych 2^k iteracjach. Daje to wszystkie ciągi binarne $b_1\, b_2 \ldots b_n$ takie, że $b_{k+1} = \ldots = b_n = 0$, wygenerowane w pierwszych 2^{k+1} iteracjach. Teraz jasne już jest, że nasz algorytm generuje wszystkie ciągi binarne długości n w takiej samej kolejności jak opisany poprzednio algorytm rekurencyjny.

Ciąg podzbiorów generowanych przez algorytm dla $n = 4$ przedstawiono na rys. 1.6.

```
0 0 0 0
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 0
1 1 1 0
1 0 1 0
0 0 1 0
0 0 1 1
1 0 1 1
1 1 1 1
0 1 1 1
0 1 0 1
1 1 0 1
1 0 0 1
0 0 0 1
```

Rys. 1.6 Ciąg podzbiorów wygenerowany przez algorytm 1.13 dla $n = 4$

Można pokazać, że średnia liczba kroków potrzebna do wygenerowania każdego następnego podzbioru (nie licząc wypisania tego podzbioru) jest ograniczona przez stałą niezależną od n (por. zad. 1.21). Ciąg podzbiorów generowany przez algorytm 1.13 można – podobnie jak to uczyniliśmy dla ciągu permutacji generowanego przez algorytm 1.12 – zilustrować na grafie, którego wierzchołki odpowiadają wszystkim ciągom binarnym długości n, a dwa wierzchołki połączone są krawędzią, jeśli odpowiadające ciągi różnią się na dokładnie jednej pozycji. Graf taki nazywamy (*binarną*) *kostką n-wymiarową*. Oczywiście ciąg wygenerowany przez nasz algorytm odpowiada drodze Hamiltona w tym grafie. Przedstawiono to dla $n = 1, 2, 3, 4$ na rys. 1.7.



Rys. 1.7 Drogi Hamiltona w kostkach n-wymiarowych

W niektórych zastosowaniach pojęciem naturalniejszym od zbioru jest *zbiór z powtórzeniami*. Każdy element zbioru z powtórzeniami może występować w tym zbiorze kilkakrotnie, przy czym liczba tych wystąpień, zwana *krotnością* elementu w zbiorze, jest istotna. Zbiór z powtórzeniami zawierający, dla przykładu, element a o krotności 2, element b o krotności 3 oraz element c o krotności 1 oznaczać będziemy przez (a, a, b, b, b, c), lub $(2*a, 3*b, 1*c)$. Kolejność elementów jest nieistotna:

$$(a, a, b, b, b, c) = (b, a, b, a, c, b) = \ldots$$

ważna jest jedynie krotność każdego elementu – mamy

$$(a, a, b, b, b, c) \neq (a, b, c)$$

w odróżnieniu od równości zbiorów

$$\{a, a, b, b, b, c\} = \{a, b, c\}$$

Jeśli krotność każdego elementu w pewnym zbiorze z powtórzeniami jest równa jedności, to oczywiście możemy go identyfikować ze zwykłym zbiorem. Niech A, B będą dwoma zbiorami z powtórzeniami. Powiemy, że A jest *podzbiorem* B (co oznaczamy $A \subseteq B$), jeśli krotność każdego elementu w A jest nie większa od krotności tego elementu w B. Niech X będzie zbiorem z powtórzeniami zawierającym r różnych elementów $x_1, \ldots, x_r$ o krotnościach odpowiednio $k_1, \ldots, k_r$. Liczbę $|X| = k_1 + \ldots + k_r$ nazywamy *licznością* X. Każdemu podzbiorowi $A \subseteq X$ odpowiada jednoznacznie ciąg

$$\langle m_1, \ldots, m_r \rangle, \qquad 0 \leqslant m_1 \leqslant k_1, \ldots, 0 \leqslant m_r \leqslant k_r \tag{1.10}$$

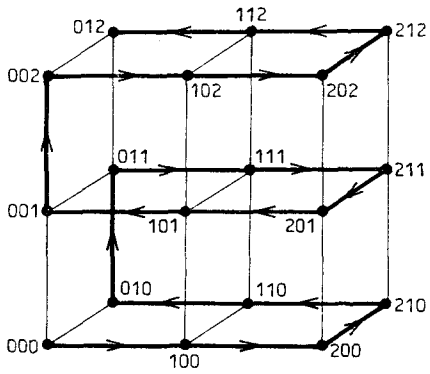
gdzie m_i oznacza krotność elementu x_i w A. Stąd natychmiastowy wniosek, że wszystkich podzbiorów $A \subseteq X$ jest

$$(k_1+1)(k_2+1)\ldots(k_r+1) \tag{1.11}$$

Podzbiory te – ściślej, odpowiadające im ciągi (1.10) – można wygenerować w sposób podobny do użytego w algorytmie 1.13. Wystarczy w tym celu zauważyć, że jeśli ciąg $C_1, C_2, \ldots, C_m$ zawiera wszystkie $p = (k_1+1)\ldots(k_s+1)$ ciągów $m_1, \ldots, m_s$, $0 \leqslant m_1 \leqslant k_1, \ldots, 0 \leqslant m_s \leqslant k_s$, to ciąg

$$C_1\,0,\ C_2\,0,\ \ldots,\ C_p\,0,\ C_p\,1,\ C_{p-1}\,1,\ \ldots,\ C_1\,1,\ C_1\,2,\ C_2\,2,\ \leqslant,\ C_p\,2,\ C_p\,3,\ \ldots \tag{1.12}$$

długości $p(k_{s+1}+1)$ zawiera wszystkie ciągi $\langle m_1, \ldots, m_{s+1} \rangle$, $0 \leqslant m_1 \leqslant k_1, \ldots, 0 \leqslant m_{s+1} \leqslant k_{s+1}$. Oczywiście ciąg podzbiorów uzyskany za pomocą tej konstrukcji ma tę własność, że każdy następny podzbiór powstaje z poprzedniego przez dodanie lub odjęcie jednego elementu. Pozostawiamy Czytelnikowi wyeliminowanie rekursji z tego algorytmu (zad. 1.22), zilustrujemy go jedynie na rys. 1.8 przez drogę Hamiltona w pewnym grafie. Interpretacja tego rysunku jest taka sama jak rysunku 1.7.



Rys. 1.8 Droga Hamiltona w grafie odpowiadającym podzbiorom zbioru z powtórzeniami $(x_1, x_1, x_2, x_3, x_3)$

Podzbiory k-elementowe, współczynnik dwumienny 1.6

Liczbę wszystkich podzbiorów k-elementowych zbioru n-elementowego oznaczać będziemy przez $\binom{n}{k}$. Symbol $\binom{n}{k}$ zwany jest *współczynnikiem dwumiennym* ze względu na następujący wzór na n-tą potęgę dwumianu $x+y$:

$$(x+y)^n = \sum_{i=1}^{n} \binom{n}{k} x^i y^{n-i} \tag{1.13}$$

Aby się przekonać o prawdziwości tego wzoru wystarczy zauważyć, że współczynnik przy $x^i y^{n-i}$ jest równy liczbie sposobów, w jaki spośród n czynników iloczynu $(x+y) \dots (x+y)$ można wybrać k czynników jako te, z których wybieramy składnik x.

Oczywiście $\binom{n}{k} = 0$ dla $k>n$. Warto wspomnieć, że podzbiory k-elementowe zbioru n-elementowego nazywa się też w starszej literaturze kombinatorycznej *kombinacjami k-wyrazowymi ze zbioru n-elementowego bez powtórzeń.*

Ze współczynnikami dwumiennymi związanych jest wiele ciekawych tożsamości. Oto niektóre z nich.

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n \tag{1.14}$$

Wzór ten wynika stąd, że suma po lewej stronie wyraża liczbę wszystkich podzbiorów zbioru n-elementowego.

$$\sum_{i=0}^{n} \binom{n}{i} i = n2^{n-1} \tag{1.15}$$

Dla dowodu utwórzmy listę zawierającą kolejno wszystkie podzbiory zbioru n-elementowego X. Lista ta zawiera $\binom{n}{i}$ podzbiorów i-elementowych dla $i = 1, \dots, n$, tak więc jej długość (tzn. liczba wystąpień elementów zbioru X) wyraża się sumą po lewej stronie wzoru (1.15). Z drugiej strony każdy element $x \in X$ występuje na liście 2^{n-1} razy, gdyż taka jest liczba podzbiorów zawierających x (jest ich tyle, ile wszystkich podzbiorów zbioru $X \setminus \{x\}$). Długość naszej listy jest więc równa $n2^{n-1}$. Dowód jest tym samym zakończony.

$$\binom{n}{k} = \binom{n}{n-k} \tag{1.16}$$

Jest to bezpośrednia konsekwencja faktu, iż każdemu podzbiorowi k-elemen-

towemu $Y \subseteq X$ odpowiada wzajemnie jednoznacznie $(n-k)$-elementowy podzbiór $X \setminus Y$ zbioru X.

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \tag{1.17}$$

Wzór ten otrzymujemy przez następujące rozumowanie. Ustalmy pewien element x zbioru n-elementowego X. Zbiór wszystkich podzbiorów k-elementowych zbioru X rozpada się na dwie rozłączne klasy: tych podzbiorów, które nie zawierają elementu x, oraz tych, które x zawierają. Liczność pierwszej klasy wynosi $\binom{n-1}{k}$, drugiej zaś $\binom{n-1}{k-1}$, tyle ile jest podzbiorów $(k-1)$-elementowych zbioru $X \setminus \{x\}$.

Tożsamość (1.17) wiąże się ściśle z następującym *trójkątem Pascala*:

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1 \\
1 & 5 & & 10 & & 10 & & 5 & 1 \\
\vdots & & & & & & & &
\end{array}$$

Jeśli ponumerować kolejne wiersze tego trójkąta liczbami 0, 1, 2, ..., to i-ty wiersz zawiera liczby

$$\binom{i}{0}\binom{i}{1}\cdots\binom{i}{i}$$

Na mocy wzoru (1.17) każdą z tych liczb – oprócz skrajnych, równych jedności – można otrzymać jako sumę występujących nad nią liczb w poprzednim wierszu. Daje to prostą metodę generowania trójkąta Pascala, a tym samym znajdowania współczynników rozwinięcia $(x+y)^i$ – dane są one przez i-ty wiersz trójkąta (por. (1.13)).

Udowodnimy jeszcze jedną tożsamość związaną ze współczynnikami dwumiennymi, a mianowicie następującą *tożsamość Cauchy'ego*:

$$\binom{m+n}{k} = \sum_{s=0}^{k} \binom{m}{s}\binom{n}{k-s} \tag{1.18}$$

W tym celu wyobraźmy sobie, że z grupy złożonej z m mężczyzn i n kobiet chcemy wybrać k osób. Możemy to uczynić na $\binom{m+n}{k}$ sposobów (lewa strona). Wszystkie podzbiory k-elementowe naszego zbioru osób możemy sklasyfikować ze względu na liczbę mężczyzn w podzbiorze. Podzbiór k-elementowy zawierający s mężczyzn możemy otrzymać wybierając najpierw s mężczyzn – na jeden z $\binom{m}{s}$ możliwych

sposobów — a następnie $k-s$ kobiet, na jeden spośród $\binom{n}{k-s}$ sposobów. Wszystkich możliwych podzbiorów k osób, w tym s mężczyzn, jest zatem $\binom{m}{s}\binom{n}{k-s}$. Bezpośrednio stąd wynika już wzór (1.18).

Zauważmy, że dla $m = k = n$ wzór ten przybiera postać

$$\binom{2n}{n} = \sum_{s=0}^{n} \binom{n}{s}^2 \tag{1.19}$$

Ostatnią tożsamością, którą udowodnimy jest

$$\binom{n}{k}\binom{k}{m} = \binom{n}{m}\binom{n-m}{n-k} \tag{1.20}$$

Dla dowodu policzmy liczbę par $\langle K, M\rangle$ podzbiorów zbioru n-elementowego X, takich że $K \subseteq M$, $|K| = k$, $|M| = m$. Z jednej strony podzbiór K możemy wybrać na $\binom{n}{k}$ sposobów, a podzbiór $M \subseteq K$ na $\binom{k}{m}$ sposobów, co daje $\binom{n}{k}\binom{k}{m}$ możliwych par $\langle K, M\rangle$ (lewa strona (1.20)). Z drugiej strony, dla każdego z $\binom{n}{m}$ podzbiorów $M \subseteq X$ możemy wybrać na $\binom{n-m}{n-k}$ sposobów podzbiór $K \subseteq M$, gdyż taki podzbiór K wyznaczony jest jednoznacznie przez podzbiór $(n-k)$-elementowy $X \setminus K$ zbioru $(n-m)$-elementowego $X \setminus M$. Szukana liczba par $\langle K, M\rangle$ jest więc równa $\binom{n}{m}\binom{n-m}{n-k}$ (prawa strona (1.20)).

Podamy teraz bezpośredni wzór pozwalający na obliczanie $\binom{n}{k}$.

TWIERDZENIE 1.14

$$\binom{n}{k} = \frac{[n]_k}{k!} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\ldots(n-k+1)}{1\cdot 2\cdot \ldots \cdot k} \tag{1.21}$$

DOWÓD

Każdy spośród $[n]_k$ ciągów różnowartościowych długości k wyznacza podzbiór k-elementowy złożony z elementów występujących w tym ciągu, przy czym ten sam podzbiór S otrzymujemy z dokładnie $k!$ ciągów, odpowiadających wszystkim permutacjom zbioru S. Stąd $\binom{n}{k} = [n]_k/k!$. Wystarczy teraz skorzystać z twierdzenia 1.2. ∎

Zauważmy, że twierdzenie to pozwala łatwo udowodnić wszystkie tożsamości rozważane w tym punkcie. Na przykład dla (1.17) mamy

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{k!(n-k)!}\left(\frac{n-k}{n} + \frac{k}{n}\right) =$$

$$= \frac{(n-1)!}{k!(n-1-k)!} + \frac{(n-1)!}{(k-1)!\,(n-k)!} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Jednakże tego typu „algebraiczny" dowód polegający na mechanicznym prze-

kształcaniu wyrażeń jest znacznie mniej przejrzysty od dowodu „kombinatorycznego" pokazanego poprzednio.

Z drugiej strony wzór (1.21) pozwala łatwo udowodnić następującą własność współczynników dwumiennych:

$$\binom{n}{0} < \binom{n}{1} < \ldots < \binom{n}{\lfloor n/2 \rfloor} = \binom{n}{\lceil n/2 \rceil} > \binom{n}{\lceil n/2 \rceil + 1} > \ldots > \binom{n}{n} \tag{1.22}$$

($n > 1$). Wystarczy zauważyć, że

$$\binom{n}{i+1} = \frac{n(n-1)\ldots(n-i)}{1\cdot 2\cdot \ldots \cdot i(i+1)} = \frac{n-i}{i+1} \cdot \frac{n(n-1)\ldots(n-i+1)}{1\cdot 2\cdot \ldots \cdot i} =$$
$$= \frac{n-i}{i+1}\binom{n}{i}$$

Mamy $(n-i)/(i+1) > 1$ dla $i < (n-1)/2$, tzn. dla $i < \lfloor n/2 \rfloor$, podobnie $(n-i)/(i+1) < 1$ dla $i > (n-1)/2$, tzn. dla $i \geqslant \lceil n/2 \rceil$, oraz $(n-\lfloor n/2 \rfloor)/(\lfloor n/2 \rfloor + 1) = \lceil n/2 \rceil / \lceil n/2 \rceil = 1$, jeśli n nieparzyste (dla parzystego n mamy $\lfloor n/2 \rfloor = \lceil n/2 \rceil$ i środkowa równość w (1.22) oczywiście zachodzi).

Zajmiemy się teraz znalezieniem liczby k-elementowych podzbiorów zbioru z powtórzeniami $(k*x_1, \ldots, k*x_n)$. Pytamy, innymi słowy, na ile sposobów można utworzyć k-elementowy zbiór z powtórzeniami mając do dyspozycji n różnych elementów, powiedzmy $1, \ldots, n$, z których każdy możemy użyć dowolną ilość razy. Warto zauważyć, że liczba ta jest równa liczbie ciągów niemalejących długości k o elementach ze zbioru $\{1, \ldots, n\}$. Wynika to z faktu, że elementy każdego podzbioru dają się jednoznacznie ustawić w ciąg niemalejący. Oczywiście podzbiorem bez powtórzeń odpowiadają ciągi rosnące.

TWIERDZENIE 1.15

Liczba k-elementowych podzbiorów zbioru $(k*x_1, \ldots, k*x_n)$ jest równa

$$\binom{n+k-1}{k} \tag{1.23}$$

DOWÓD

Mamy wyznaczyć liczbę ciągów niemalejących $\langle a_1, \ldots, a_k \rangle$ o elementach ze zbioru $\{1, \ldots, n\}$. Każdy taki ciąg można podzielić na n sekcji, gdzie i-ta sekcja zawiera pewną — być może zerową — liczbę następujących po sobie elementów i. Jeśli sekcje te oddzielimy od siebie $n-1$ zerami użytymi w charakterze separatorów, to otrzymamy ciąg długości $k+n-1$. Na przykład dla $n = 5, k = 4$ ciągowi $\langle 2, 2, 4, 5 \rangle$ odpowiada ciąg $\langle 0, 2, 2, 0, 0, 4, 0, 5 \rangle$. Ten ostatni ciąg jest jednoznacznie wyznaczony przez rozmieszczenie występujących w nim $n-1$ zer. Istotnie, możemy go odtworzyć wypełniając sekcję do pierwszego zera wystąpieniami elementu 1, sekcję od pierwszego do drugiego zera wystąpieniami elementu 2 itd. Lecz liczba rozmieszczeń $n-1$ na $k+n-1$ pozycjach jest równa

$$\binom{k+n-1}{n-1} = \binom{n+k-1}{(n+k-1)-n-1} = \binom{n+k-1}{k}. \qquad \blacksquare$$

Znalezioną przez nas liczbę podzbiorów z powtórzeniami można zapisać nieco inaczej:

$$\binom{n+k-1}{k} = \frac{n(n+1)\ldots(n+k-1)}{k!} = \frac{[n]^k}{k!} \tag{1.24}$$

Pokażemy interpretację kombinatoryczną tej równości. Przypomnijmy w tym celu, że $[n]^k$ jest liczbą rozmieszczeń uporządkowanych k elementów $x_1, \ldots, x_k$ w n pudełkach. Zauważmy, że każdemu takiemu rozmieszczeniu zawierającemu r_i elementów w i-tym pudełku odpowiada k-elementowy podzbiór $(r_1 * y_1, \ldots, r_n * y_n)$ zbioru z powtórzeniami $(k*y_1, \ldots, k*y_n)$. W tym przyporządkowaniu istotna jest jedynie liczba elementów w każdym z pudełek. A więc na przykład rozmieszczeniom

pudełko 1	pudełko 2	pudełko 3
$x_3\ x_2$		$x_5\ x_1\ x_4$
$x_4\ x_1$		$x_2\ x_3\ x_5$

odpowiada ten sam podzbiór $(2*y_1, 3*y_3)$. Jasne jest, że każde dwa rozmieszczenia odpowiadające temu samemu podzbiorowi różnią się permutacją obiektów $x_1, \ldots, x_k$. W konsekwencji liczba wszystkich k-elementowych podzbiorów zbioru z powtórzeniami $(k*y_1, \ldots, k*y_n)$ jest równa $[n]^k/k!$. Oczywiście rozumowanie to wraz ze wzorem (1.24) stanowi inny dowód twierdzenia 1.15.

Generowanie podzbiorów k-elementowych 1.7

Pokażemy teraz dwa algorytmy generowania wszystkich k-elementowych podzbiorów zbioru n-elementowego X. Bez zmniejszenia ogólności możemy przyjąć, że $X = \{1, \ldots, n\}$. Każdemu k-elementowemu podzbiorowi odpowiada wtedy wzajemnie jednoznacznie ciąg rosnący długości k o elementach z X: na przykład podzbiorowi $\{3, 5, 1\}$ odpowiada ciąg $\langle 1, 3, 5\rangle$.

Można łatwo pokazać algorytm generujący wszystkie takie ciągi w porządku leksykograficznym. Wystarczy w tym celu zauważyć, że w porządku tym, ciągiem bezpośrednio następującym po $\langle a_1, \ldots, a_k\rangle$ jest

$$\langle b_1, \ldots, b_k\rangle = \langle a_1, \ldots, a_{p-1}, a_p+1, a_p+2, \ldots, a_p+k-p+1\rangle$$

gdzie

$$p = max\,\{i: a_i < n-k+1\}$$

Co więcej, ciągiem bezpośrednio następującym po $\langle b_1, \ldots, b_k\rangle$ jest

$$\langle c_1, \ldots, c_k\rangle = \langle b_1, \ldots, b_{p'-1}, b_{p'}+1, b_{p'}+2, \ldots, b_{p'}+k-p'+1\rangle$$

gdzie

$$p' = \begin{cases} p-1 & \text{jeśli } b_k = n \\ k & \text{jeśli } b_k < n \end{cases}$$

(zakładamy, że ciągi $\langle a_1, \ldots, a_k \rangle$ i $\langle b_1, \ldots, b_k \rangle$ są różne od $\langle n-k+1, \ldots, n \rangle$, ostatniego ciągu w naszym porządku). Prowadzi to do następującego prostego algorytmu:

ALGORYTM 1.16 (Generowanie wszystkich k-elementowych podzbiorów zbioru $\{1, \ldots, n\}$ w porządku leksykograficznvm)

Dane: n, k.

Wyniki: Ciąg wszystkich podzbiorów k-elementowych zbioru $\{1, \ldots, n\}$ w porządku leksykograficznym.

```
1  begin
2    for i := 1 to k do A[i] := i; (* pierwszy podzbiór *)
3    p := k;
4    while p ⩽ 1 do
5      begin write (A[1], ..., A[k]);
6        if A[k] = n then p := p-1 else p := k;
7        if p ⩾ 1 then
8          for i := k downto p do A[i] := A[p]+i-p+1
9      end
10 end
```

Ciąg wszystkich podzbiorów 4-elementowych zbioru $\{1, \ldots, 6\}$ wygenerowany przez ten algorytm przedstawiono na rys. 1.9.

```
1 2 3 4
1 2 3 5
1 2 3 6
1 2 4 5
1 2 4 6
1 2 5 6
1 3 4 5
1 3 4 6
1 3 5 6
1 4 5 6
2 3 4 5
2 3 4 6
2 3 5 6
2 4 5 6
3 4 5 6
```

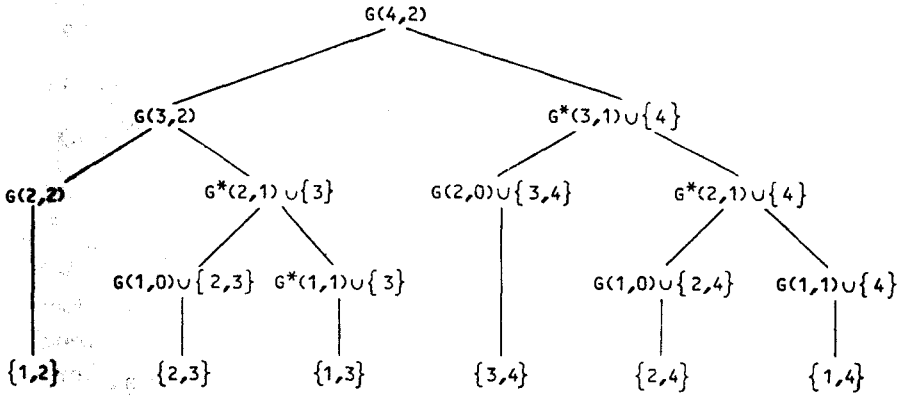
Rys. 1.9 Ciąg podzbiorów 4-elementowych zbioru $\{1, \ldots, 6\}$ skonstruowany przez algorytm 1.16

Drugi algorytm, który teraz opiszemy generuje wszystkie k-elementowe podzbiory w taki sposób, że każdy następny podzbiór powstaje z poprzedniego przez usunięcie pewnego elementu i dodanie innego. Algorytm ten przedstawimy w postaci rekurencyjnej. Oznaczmy w tym celu przez $G(n, k)$ listę (tzn. ciąg) zawierającą wszystkie k-elementowe podzbiory zbioru $\{1, \ldots, n\}$, której pierwszym podzbiorem jest $\{1, \ldots, k\}$, ostatnim $\{1, 2, \ldots, k-1, n\}$ i każdy następny podzbiór powstaje z poprzedniego przez usunięcie pewnego elementu i dodanie

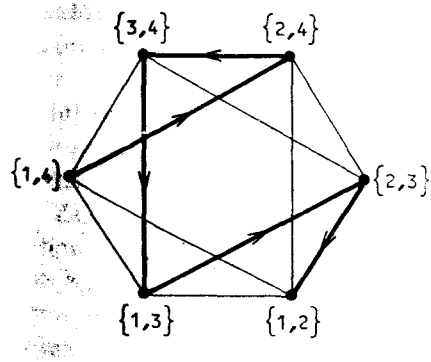
...ego. Zauważmy, że jeśli skonstruowaliśmy już $G(n-1, k)$ i $G(n-1, k-1)$, to $G(n, k)$ możemy określić następująco:

$$G(n, k) = G(n-1, k), G^*(n-1, k-1) \cup \{n\} \tag{1.25}$$

gdzie $G^*(n-1, k-1) \cup \{n\}$ oznacza listę powstałą z $G(n-1, k-1)$ przez odwrócenie jej kolejności, a następnie dodanie elementu n do każdego zbioru. Istotnie $G(n-1, k)$ zawiera wszystkie podzbiory k-elementowe zbioru $\{1, \ldots, n\}$ nie zawierające n, $G^*(n-1, k-1) \cup \{n\}$ wszystkie podzbiory k-elementowe zawierające n, przy czym ostatnim podzbiorem na liście $G(n-1, k)$ jest $\{1, 2, \ldots, k-1, n-1\}$, natomiast pierwszym podzbiorem na liście $G^*(n-1, k-1) \cup \{n\}$ jest $\{1, 2, \ldots, k-1, n\}$. Na rysunku 1.10 pokazano konstrukcję listy $G(4, 2)$.



Rys. 1.10 Konstrukcja listy $G(4, 2)$



Rys. 1.11 Droga Hamiltona w grafie odpowiadającym wszystkim podzbiorom dwuelementowym zbioru $\{1, 2, 3, 4\}$

Liście $G(n, k)$ możemy – podobnie jak w przypadku generowania wszystkich podzbiorów – przyporządkować pewną drogę Hamiltona w grafie, którego wierzchołki odpowiadają podzbiorom dwuelementowym zbioru $\{1, 2, 3, 4\}$, przy czym wierzchołki odpowiadające podzbiorom A, B są połączone krawędzią wtedy i tylko wtedy, gdy $|A \cap B| = k-1 = 1$. Przedstawiono to na rysunku 1.11.

Pozostawiamy Czytelnikowi pozbycie się rekursji w opisanym przez nas algorytmie (por. zad. 1.29).

Podziały zbioru

Przez *podział* zbioru n-elementowego X na k bloków rozumieć będziemy dowolną rodzinę $\pi = \{B_1, \ldots, B_k\}$ taką, że $B_1 \cup \ldots \cup B_k = X$, $B_i \cap B_j = \emptyset$ dla $1 \leqslant i < j \leqslant k$, oraz $B_i \neq \emptyset$ dla $1 \leqslant i \leqslant k$. Podzbiory $B_1, \ldots, B_k$ nazywamy *blokami* podziału π. Zbiór wszystkich podziałów zbioru X na k bloków oznaczać będziemy przez $\Pi_k(X)$, natomiast zbiór wszystkich podziałów przez $\Pi(X)$. Oczywiście $\Pi(X) = \Pi_1(X) \cup \ldots \cup \Pi_n(X)$ (co więcej, $\{\Pi_1(X), \ldots, \Pi_n(X)\}$ jest podziałem zbioru $\Pi(X)$).

Ściśle związane z podziałami jest pojęcie relacji równoważności. Każdemu podziałowi π możemy przyporządkować relację równoważności

$$E(\pi) = \bigcup_{B \in \pi} (B \times B) \tag{1.26}$$

(elementy $x, y \in X$ są w relacji $E(\pi)$ wtedy i tylko wtedy, gdy należą do tego samego bloku podziału π). Na odwrót, każdej relacji równoważności E na zbiorze X możemy przyporządkować podział

$$X/E = \{x/E : x \in X\} \tag{1.27}$$

gdzie x/E oznacza *klasę abstrakcji* elementu x, tzn. zbiór wszystkich elementów będących w relacji E z x:

$$x/E = \{y \in X : xEy\} \tag{1.28}$$

Nietrudno zauważyć, że wzory (1.26) i (1.27) określają wzajemnie jednoznaczną odpowiedniość między podziałami a relacjami równoważności na zbiorze X.

Jeśli $\pi, \sigma \in \Pi(X)$ i każdy blok $B \in \sigma$ jest sumą pewnej liczby bloków podziału π, to mówimy, że π jest *rozdrobnieniem* podziału σ i piszemy $\pi \leqslant \sigma$. Dla przykładu

$$\{\{1\}, \{2, 5\}, \{4, 6\}, \{3\}\} \leqslant \{\{1, 2, 3, 5\}, \{4, 6\}\}$$

Łatwo sprawdzić, że $\pi \leqslant \sigma$ wtedy i tylko wtedy, gdy dla relacji równoważności odpowiadających tym podziałom zachodzi $E(\pi) \subseteq E(\sigma)$. Oczywiście tak określona relacja $\leqslant$ jest częściowym porządkiem na zbiorze $\Pi(X)$. Niektóre własności zbioru $\Pi(X)$ uporządkowanego przez relację rozdrobnienia przypominają analogiczne własności zbioru $\mathscr{P}(X)$ uporządkowanego przez zawieranie. Pokażemy teraz jedną z takich własności.

Zbiór częściowo uporządkowany $\langle A, \leqslant \rangle$ nazywamy *kratą*, jeśli dla dowolnych elementów $x, y \in A$ istnieją elementy $a, b \in A$ takie, że

(a) $a \leqslant x$, $a \leqslant y$ oraz dla dowolnego elementu c takiego, że $c \leqslant x$, $c \leqslant y$ mamy $c \leqslant a$,

(b) $b \geqslant x$, $b \geqslant y$ oraz dla dowolnego elementu c takiego, że $c \geqslant x$, $c \geqslant y$ mamy $c \geqslant b$.

Elementy takie – jeśli istnieją – są jednoznacznie wyznaczone przez x i y. Istotnie, dla dowolnych elementów a_1, a_2 spełniających warunek (a) mamy $a_2 \leqslant a_1$

1.8 oraz $a_1 \leqslant a_2$, czyli $a_1 = a_2$, podobnie dla (b). Element a nazywamy *kresem dolnym* elementów x, y i oznaczamy przez $x \wedge y$, element b natomiast nazywamy *kresem górnym* i oznaczamy przez $x \vee y$.

Przykładem kraty jest $\langle \mathscr{P}(X), \subseteq \rangle$. W tym przypadku mamy oczywiście $A \wedge B = A \cap B$ oraz $A \vee B = A \cup B$.

TWIERDZENIE 1.17

Zbiór $\Pi(X)$ uporządkowany przez relację rozdrobnienia $\leqslant$ tworzy kratę, przy czym

$$\pi \wedge \sigma = \{A \cap B : (A \in \pi) \wedge (B \in \sigma) \wedge (A \cap B \neq \emptyset)\} \tag{1.29}$$

tzn.

$$E(\pi \wedge \sigma) = E(\pi) \cap E(\sigma) \tag{1.30}$$

podział $\pi \vee \sigma$ jest natomiast określony następująco:

$$\langle x, y \rangle \in E(\pi \vee \sigma) \Leftrightarrow \text{istnieje } k \geqslant 1 \text{ oraz ciag } x_1, \ldots, x_k \text{ taki, że } x = x_1, y = x_k \text{ oraz } \langle x_l, x_{l+1} \rangle \in E(\pi) \text{ lub } \langle x_l, x_{l+1} \rangle \in E(\sigma) \text{ dla } i = 1, 2, \ldots, k-1. \tag{1.31}$$

DOWÓD

Zauważmy przede wszystkim, że prostym wnioskiem z definicji relacji równoważności jest fakt, iż przecięcie dwóch relacji równoważności jest relacją równoważności. Tak więc $E(\pi) \cap E(\sigma)$ jest relacją równoważności, która z kolei wyznacza jednoznacznie pewien podział α taki, że $E(\pi) \cap E(\sigma) = E(\alpha)$ (por. (1.27)). Oczywiście $E(\alpha) \subseteq E(\pi)$, $E(\alpha) \subseteq E(\sigma)$, a więc $\alpha \leqslant \pi$, $\alpha \leqslant \sigma$. Co więcej, dla każdego podziału α' takiego, że $\alpha' \leqslant \pi$, $\alpha' \leqslant \sigma$ mamy $E(\alpha') \subseteq E(\pi)$, $E(\alpha') \subseteq E(\sigma)$ i w konsekwencji $E(\alpha') \subseteq E(\pi) \cap E(\sigma) = E(\alpha)$, czyli $\alpha' \leqslant \alpha$. A zatem $\alpha = \pi \wedge \sigma$, co dowodzi wzoru (1.30). Zauważmy, że $x, y \in E(\pi \wedge \sigma) = E(\pi) \cap E(\sigma)$ wtedy i tylko wtedy, gdy x, y należą do tego samego bloku podziału π i tego samego bloku podziału σ, tzn. $x, y \in A \cap B$, gdzie $A \in \pi$, $B \in \sigma$. Dowodzi to wzoru (1.29).

Załóżmy teraz, że R jest relacją binarną określoną przez prawą stronę równoważności (1.31). Nietrudno sprawdzić, że R jest relacją równoważności. Istotnie, xRx (wystarczy przyjąć $k = 1$), xRy pociąga za sobą yRx (wystarczy rozważyć ciąg $x_k, x_{k-1}, \ldots, x_1$) oraz xRy, yRz pociąga za sobą xRz (wystarczy rozważyć konkatenację odpowiednich ciągów). Relacja R wyznacza jednoznacznie podział β taki, że $E(\beta) = R$. Oczywiście $E(\pi) \leqslant R = E(\beta)$ i $E(\sigma) \subseteq R = E(\beta)$, tzn. $\pi \leqslant \beta$ oraz $\sigma \leqslant \beta$. Załóżmy teraz, że β' jest dowolnym podziałem takim, że $\pi \leqslant \beta'$, $\sigma \leqslant \beta'$, oraz niech $\langle x, y \rangle \in E(\beta)$. Istnieje wtedy ciąg $x = x_1, x_2, \ldots, x_k = y$ spełniający prawą stronę równoważności (1.31), i w konsekwencji $\langle x_i, x_{i+1} \rangle \in E(\pi) \cup E(\sigma) \subseteq E(\beta')$, $1 \leqslant i \leqslant k$. Wobec przechodniości relacji $E(\beta')$ mamy $\langle x, y \rangle \in E(\beta')$. Udowodniliśmy w ten sposób, że $\beta \leqslant \beta'$, a więc $\beta = \pi \vee \sigma$. ■

Liczby Stirlinga drugiego i pierwszego rodzaju

Liczbę Stirlinga drugiego rodzaju $S(n, k)$ definiujemy jako liczbę podziałów zbioru n-elementowego na k bloków:

$$S(n, k) = |\Pi_k(X)| \quad \text{gdzie } |X| = n \tag{1.32}$$

Dla przykładu $S(4, 2) = 7$, gdyż istnieje dokładnie 7 podziałów zbioru $\{1, ..., 4\}$ na dwa bloki:

$\{\{1, 2, 3\}, \{4\}\}$
$\{\{1, 2, 4\}, \{3\}\}$
$\{\{1, 3, 4\}, \{2\}\}$
$\{\{1, 2\}, \{3, 4\}\}$
$\{\{1, 3\}, \{2, 4\}\}$
$\{\{1, 4\}, \{2, 3\}\}$
$\{\{1\}, \{2, 3, 4\}\}$

Oczywiście $S(n, k) = 0$ dla $k > n$. Przyjmujemy również $S(0, 0) = 1$, gdyż pusta rodzina bloków jest – zgodnie z definicją – podziałem zbioru pustego. Z liczbami Stirlinga drugiego rodzaju związane jest – podobnie jak ze współczynnikami dwumiennymi – wiele ciekawych tożsamości. Pokażemy najpierw tożsamość przypominającą nieco wzór (1.17) związany z trójkątem Pascala:

$$S(n, k) = S(n-1, k-1) + kS(n-1, k) \quad \text{dla } 0 < k < n \tag{1.33}$$

$$S(n, n) = 1 \quad \text{dla } n \geqslant 0 \tag{1.34}$$

$$S(n, 0) = 0 \quad \text{dla } n > 0 \tag{1.35}$$

Wzory (1.34) i (1.35) są oczywiste. Dla dowodu wzoru (1.33) rozważmy zbiór wszystkich podziałów zbioru $\{1, ..., n\}$ na k bloków. Zbiór ten rozpada się na dwie rozłączne klasy: tych podziałów, które zawierają blok jednoelementowy $\{n\}$, oraz tych podziałów, dla których n jest elementem większego (co najmniej dwuelementowego) bloku. Liczność pierwszej klasy jest równa $S(n-1, k-1)$, tyle ile jest podziałów zbioru $\{1, ..., n-1\}$ na $k-1$ bloków. Liczność drugiej klasy wynosi $kS(n-1, k)$, gdyż każdemu podziałowi zbioru $\{1, ..., h-1\}$ na k bloków odpowiada w tej klasie dokładnie k podziałów powstałych przez dodanie elementu n kolejno do każdego z bloków.

Wzory (1.33), (1.34) i (1.35) umożliwiają łatwe wyznaczanie wartości $S(n, k)$, nawet dla dużych wartości n i k. Liczby $S(n, k)$ dla $0 \leqslant n, k \leqslant 10$ przedstawiono w tabl. 1.2.

Zauważmy, że tablicę tę można traktować jako „trójkąt Stirlinga", w którym każdą wartość – oprócz skrajnych, równych jedności – można otrzymać jako sumę liczb występujących nad nią, dokładniej liczby występującej dokładnie nad nią pomnożonej przez k, oraz liczby nad nią po lewej stronie.

1.9 ...CA 1.2 Liczby Stirlinga drugiego rodzaju $S(n, k)$

n \ k	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0	0
7	0	1	63	301	350	140	21	1	0	0	0
8	0	1	127	966	1701	1050	266	28	1	0	0
9	0	1	255	3025	7770	6951	2646	462	36	1	0
10	0	1	511	9330	34105	42525	22827	5880	750	45	1

Oto przykład innej zależności rekurencyjnej związanej z liczbami Stirlinga drugiego rodzaju:

$$S(n, k) = \sum_{i=k-1}^{n-1} \binom{n-1}{i} S(i, k-1) \qquad \text{dla } k \geqslant 2 \tag{1.36}$$

Dla dowodu rozważmy zbiór wszystkich $S(n, k)$ podziałów zbioru $X = \{1, \ldots, n\}$. Zbiór ten rozpada się na rozłączne klasy odpowiadające różnym podzbiorom zbioru X będącym blokiem zawierającym element n. Zauważmy, że dla każdego b-elementowego podzbioru $B \subseteq X$ zawierającego element n istnieje dokładnie $S(n-b, k-1)$ podziałów zbioru X na k bloków zawierających B w charakterze bloku. Istotnie, każdy taki podział odpowiada jednoznacznie podziałowi zbioru $X \setminus B$ na $k-1$ bloków. Zbiór b-elementowy $B \subseteq X$ zawierający element n można wybrać na $\binom{n-1}{b-1}$ sposobów, tak więc

$$S(n, k) = \sum_{b=1}^{n-(k-1)} \binom{n-1}{b-1} S(n-b, k-1) = \sum_{b=1}^{n-(k-1)} \binom{n-1}{n-b} S(n-b, k-1) =$$

$$= \sum_{i=k-1}^{n-1} \binom{n-1}{i} S(i, k-1)$$

Liczbę Bella B_n definiujemy jako liczbę wszystkich podziałów zbioru n-elementowego

$$B_n = |\Pi(X)|, \quad \text{gdzie } |X| = n \tag{1.37}$$

Innymi słowy

$$B_n = \sum_{k=0}^{n} S(n, k) \tag{1.38}$$

Udowodnimy teraz następującą prostą zależność rekurencyjną związaną z liczbami Bella:

$$B_{n+1} = \sum_{i=0}^{n} \binom{n}{i} B_i \tag{1.39}$$

(przyjmujemy $B_0 = 1$). Dowód przebiega podobnie jak w przypadku zależności (1.36). Zbiór wszystkich podziałów zbioru $X = \{1, ..., n+1\}$ możemy rozłożyć na rozłączne klasy w zależności od bloku B zawierającego element $n+1$, lub – równoważnie – w zależności od zbioru $X \setminus B$. Dla zbioru $X \setminus B \subseteq \{1, ..., n\}$ istnieje dokładnie $|\Pi(X \setminus B)| = B_{X \setminus B|}$ podziałów zbioru X zawierających B w charakterze bloku. Grupując nasze klasy w zależności od liczności zbioru $X \setminus B$ otrzymujemy wzór (1.39).

Liczby B_n dla $0 \leqslant n \leqslant 20$ przedstawiono w tabl. 1.3.

TABLICA 1.3 Liczby Bella B_n

n	B_n
0	1
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4140
9	21147
10	115975
11	678570
12	4213597
13	27644437
14	190899322
15	1382958545
16	10480142147
17	82864869804
18	682076806159
19	5832742205057
20	51724158235372

Istnieje ścisły związek między liczbami $S(n, k)$ a liczbą wszystkich funkcji ze zbioru n-elementowego na zbiór k-elementowy, tzn. funkcji $f: X \to Y, f(X) = Y$ dla $|X| = n, |Y| = k$. Każdej takiej funkcji f możemy przyporządkować następujący podział zbioru X na k boków:

$$N(f) = \{f^{-1}(y): y \in Y\} \tag{1.40}$$

zwany *jądrem* funkcji f (warunek $f(X) = Y$ gwarantuje, że podzbiory $f^{-1}(y)$ są niepuste). Z drugiej strony nietrudno zauważyć, że każdemu podziałowi $\pi \in \in \Pi_k(X)$ odpowiada dokładnie $k!$ funkcji z X na Y takich, że $N(f) = \pi$. Każda taka funkcja odpowiada wzajemnie jednoznacznie przyporządkowaniu blokom podziału π elementów zbioru Y. Oznaczając przez $s_{n,k}$ liczbę funkcji z X na Y otrzymujemy więc

$$s_{n,k} = k!\, S(n, k) \tag{1.41}$$

Korzystając z tego wzoru możemy udowodnić jeszcze jedną własność liczb Stirlinga drugiego rodzaju, która dotyczy związku między wielomianami x^k a wielomianami

$$[x]_k = x(x-1)\ldots(x-k+1) \tag{1.42}$$

Dowolny wielomian $P(x)$ zmiennej x stopnia n możemy jednoznacznie przedstawić jako $P(x) = \sum_{k=0}^{n} a_k [x]_k$. Jest to szczególny przypadek oczywistego faktu, iż istnieje jednoznaczny rozkład $P(x) = \sum_{k=0}^{n} a_k p_k(x)$ dla dowolnego ciągu wielomianów $p_0(x), p_1(x), \ldots$ takiego, że $p_k(x)$ jest stopnia k dla każdego $k \geqslant 0$. Innymi słowy każdy taki ciąg stanowi *bazę* w przestrzeni liniowej wielomianów. Okazuje się, że liczby Stirlinga drugiego rodzaju są równe dokładnie współczynnikom przejścia z bazy $1, x, x^2, \ldots$ do bazy $1, [x]_1, [x]_2, \ldots$

TWIERDZENIE 1.18

Dla każdego $n \geqslant 0$

$$x^n = \sum_{k=0}^{n} S(n,k)\,[x]_k \tag{1.43}$$

DOWÓD

Załóżmy najpierw, że x jest nieujemną liczbą całkowitą. Policzymy na dwa sposoby liczbę wszystkich funkcji $f\colon A \to B$, gdzie $|A| = n$, $|B| = x$. Z jednej strony jest ona równa x^n (p. twierdzenie 1.1). Z drugiej strony zbiór takich funkcji f możemy sklasyfikować ze względu na zbiór $f(A)$. Oczywiście każda funkcja f jest odwzorowaniem zbioru A na zbiór $f(A)$, tak więc dla dowolnego podzbioru $Y \subseteq B$, gdzie $|Y| = k$, liczba wszystkich funkcji $f\colon A \to B$ takich, że $f(A) = Y$ jest równa $s_{n,k}$, czyli, zgodnie ze wzorem (1.41), $k!\,S(n,k)$. Biorąc pod uwagę fakt, że podzbiór Y o liczności k możemy wybrać na $\binom{n}{k}$ sposobów otrzymujemy ostatecznie

$$x^n = \sum_{k=0}^{x} \binom{x}{k} k!\,S(n,k) = \sum_{k=0}^{n} [x]_k\,S(n,k) \tag{1.44}$$

(górny wskaźnik sumowania mogliśmy zamienić z x na n, gdyż $S(n,k) = 0$ dla $k > n$, oraz $[x]_k = 0$ dla $k > x$). Skoro równość wielomianów zachodzi dla dowolnego całkowitego $x \geqslant 0$, wielomiany te są tożsamościowo równe (gdyż ich różnica jest albo tożsamościowo równa zeru, albo ma skończoną liczbę zer). ■

Liczby Stirlinga pierwszego rodzaju $s(n,k)$ definiujemy jako współczynniki przy kolejnych potęgach zmiennej x w wielomianie $[x]_k$:

$$[x]_n = \sum_{k=0}^{n} s(n,k)\,x^k \tag{1.45}$$

Innymi słowy liczby $s(n, k)$ spełniają rolę odwrotną w stosunku do liczb $S(n, k)$ — pozwalają przejść z bazy $1, [x]_1, [x]_2, \ldots$ do bazy $1, x, x^2, \ldots$ Oczywiście $s(n, k) = 0$ dla $k > n$. Liczby $s(n, k)$ wygodnie jest wyznaczać korzystając z następujących zależności rekurencyjnych:

$$s(n, k) = s(n-1, k-1) - (n-1)\, s(n-1, k) \quad \text{dla } 0 < k < n \tag{1.46}$$

$$s(n, n) = 1 \quad \text{dla } n \geqslant 0 \tag{1.47}$$

$$s(n, 0) = 0 \quad \text{dla } n > 0 \tag{1.48}$$

Wzory (1.47) i (1.48) są oczywiste, natomiast (1.46) otrzymujemy porównując współczynniki przy x^k po obu stronach równości

$$[x]_n = [x]_{n-1}\,(x-n+1) \tag{1.49}$$

Mamy mianowicie

$$\sum_{k=0}^{n} s(n, k)\, x^k = (x-n+1) \sum_{k=0}^{n-1} s(n-1, k)\, x^k =$$

$$= \sum_{k=0}^{n-1} s(n-1, k)\, x^{k+1} - (n-1) \sum_{k=0}^{n-1} s(n-1, k)\, x^k =$$

$$= \sum_{k=1}^{n-1} \big(s(n-1, k-1) - (n-1)\, s(n-1, k)\big)\, x^k +$$

$$+ s(n-1, n-1)\, x^n - (n-1)\, s(n-1, 0)$$

W tablicy 1.4 przedstawiono liczby $s(n, k)$ dla $0 \leqslant n, k \leqslant 10$.

TABLICA 1.4 Liczby Stirlinga pierwszego rodzaju $s(n, k)$

n \ k	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	-1	1	0	0	0	0	0	0	0
3	0	2	-3	1	0	0	0	0	0	0
4	0	-6	11	-6	1	0	0	0	0	0
5	0	24	-50	35	-10	1	0	0	0	0
6	0	-120	274	-225	85	-15	1	0	0	0
7	0	720	-1764	1624	-735	175	-21	1	0	0
8	0	-5040	13068	-13132	6769	-1960	322	-28	1	0
9	0	40320	-109584	118124	-67284	22449	-4536	546	-36	1
10	0	-362880	1026576	-1172700	723680	-269325	63273	-9450	870	-45

Generowanie podziałów zbioru

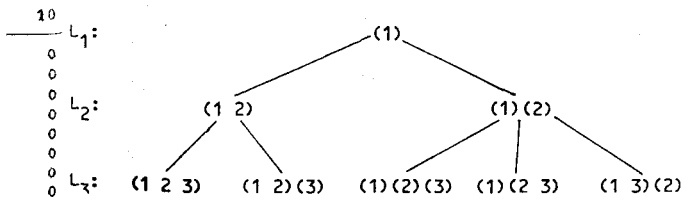
Opiszemy teraz algorytm generowania wszystkich podziałów zbioru. Ideę tego algorytmu najłatwiej wyjaśnić formułując go najpierw w postaci rekurencyjnej. Zauważmy najpierw, że każdy podział π zbioru $\{1, \ldots, n\}$ wyznacza jednoznacznie podział π_{n-1} zbioru $\{1, \ldots, n-1\}$ powstały z π przez usunięcie elementu n z odpowiedniego bloku (i usunięcie powstałego pustego bloku, jeśli element n two-

rzył blok jednoelementowy). Na odwrót, mając dany podział $\sigma = \{B_1, \ldots, B_k\}$ zbioru $\{1, \ldots, n-1\}$ łatwo znaleźć wszystkie podziały π zbioru $\{1, \ldots, n\}$ takie, że $\pi_{n-1} = \sigma$. Są to mianowicie podziały

$$\begin{aligned} &B_1 \cup \{n\}, B_2, \ldots, B_k \\ &B_1, B_2 \cup \{n\}, \ldots, B_k \\ &\vdots \\ &B_1, B_2, \ldots, B_k \cup \{n\} \end{aligned} \tag{1.50}$$

$$B_1, B_2, \ldots, B_k, \{n\} \tag{1.51}$$

Sugeruje to następującą prostą metodę rekurencyjną generowania wszystkich podziałów: Mając daną listę L_{n-1} wszystkich podziałów zbioru $\{1, \ldots, n-1\}$ tworzymy listę L_n wszystkich podziałów zbioru $\{1, \ldots, n\}$ przez zamianę każdego podziału σ na liście L_{n-1} odpowiadającym mu ciągiem (1.50). Zauważmy, że możemy przy tym zagwarantować, by kolejne podziały mało się od siebie różniły – dokładniej mówiąc możemy zakładać, że każdy następny podział na liście powstaje z poprzedniego przez usunięcie pewnego elementu z pewnego bloku (może to spowodować usunięcie bloku jednoelementowego) i dodanie go do innego bloku lub też utworzenie z niego bloku jednoelementowego. Istotnie, kolejne podziały ciągu (1.50) spełniają ten warunek. Jeśli odwrócić kolejność ciągu (1.50) dla co drugiego podziału listy L_{n-1}, to element n porusza się na zmianę do przodu i do tyłu, i podziały „na styku" ciągów utworzonych z kolejnych podziałów listy L_{n-1} mało się od siebie różnią (przy założeniu, że kolejne podziały listy L_{n-1} mało się od siebie różnią). Na rysunku 1.12 pokazano konstrukcję listy L_n dla $n = 1, 2, 3$ (podziały przedstawiono w nieco uproszczonym zapisie, np. (1 2) (3) oznacza $\{\{1, 2\}, \{3\}\}$).



Rys. 1.12 Konstrukcja list L_1, L_2, L_3

Podamy teraz realizację nierekurencyjną tego algorytmu (jest to zmodyfikowana wersja algorytmu opisanego w [39]). Podział zbioru $\{1, \ldots, n\}$ reprezentować będziemy przez ciąg bloków uporządkowany według rosnącego najmniejszego elementu w bloku. Ten najmniejszy element bloku nazywać będziemy *numerem* bloku. Zauważmy, że numery kolejnych bloków nie są na ogół kolejnymi liczbami naturalnymi. W algorytmie używać będziemy zmiennych $POPRZ[i]$, $NAST[i]$, $1 \leqslant i \leqslant n$ zawierających odpowiednio numer poprzedniego i następnego bloku dla bloku o numerze i ($NAST[i] = 0$, jeśli blok numer i jest ostatnim

blokiem podziału). Dla każdego elementu i, $1 \leqslant i \leqslant n$ numer bloku zawierającego element i pamiętany będzie w zmiennej $BLOK[i]$, kierunek w jakim „porusza" się element i zakodowany będzie natomiast w zmiennej boolowskiej $PR[i]$ ($PR[i]$ = **true**, jeśli i porusza się do przodu; zauważmy pewne podobieństwo do algorytmu 1.12).

ALGORYTM 1.19 (Generowanie wszystkich podziałów zbioru $\{1, \ldots, n\}$)
Dane: n
Wyniki: Ciąg wszystkich podziałów zbioru $\{1, \ldots, n\}$, w którym każdy następny podział powstaje z poprzedniego przez przeniesienie pojedynczego elementu do innego bloku.

```
1   begin
2     for i := 1 to n do (* umieść i w pierwszym bloku *)
3       begin BLOK[i] := 1; PR[i] := true
4       end;
5     NAST[1] := 0;
6     wypisz podział;
7     j := n; (* j = element aktywny *)
8     while j > 1 do
9       begin k := BLOK[j];
10        if PR[j] then (* j porusza się do przodu *)
11          begin
12            if NAST[k] = 0 then (* k jest ostatnim blokiem *)
13              begin NAST[k] := j; POPRZ[j] :=k; NAST[j] := 0
14              end;
15            if NAST[k] > j then (* j utworzy nowy blok *)
16              begin POPRZ[j] := k; NAST[j] := NAST[k]
17                POPRZ[NAST[j]] := j; NAST[k] := j
18              end;
19            BLOK[j] := NAST[k]
20          end
21        else (* j porusza się do tyłu *)
22          begin BLOK[j] := POPRZ[k];
23            if k = j then (* j tworzył blok jednoelementowy *)
24              if NAST[k] = 0 then NAST[POPRZ[k]] := 0
25              else begin NAST[POPRZ[k]] := NAST[k];
26                   POPRZ[NAST[k]] := POPRZ[k]
27                   end
28          end;
```

```
29          wypisz podział;
30          j := n;
31          while (j > 1) end
32           ((PR[j] and (BLOK[j] = j)) or (not PR[j] and (BLOK[j] = 1))) do
33             begin  PR[j] := not PR[j]; j := j-1
34             end
35      end
36  end
```

Algorytm ten konstruuje najpierw podział $\{\{1, ..., n\}\}$ (pętla 2) – zauważmy, że jest to pierwszy podział na liście L_n utworzonej przez opisaną przez nas konstrukcję rekurencyjną. Zadaniem głównej pętli 8 jest przesunięcie „aktywnego" elementu j do sąsiedniego bloku – poprzedniego lub następnego (w tym ostatnim przypadku może zajść potrzeba utworzenia nowego bloku postaci $\{j\}$), a następnie wyznaczenie aktywnego elementu w nowo powstałym podziale. Z opisanej konstrukcji rekurencyjnej wynika, że dany element przesuwany jest tylko wtedy, gdy wszystkie elementy od niego większe osiągają swe skrajne lewe lub prawe położenie; dokładniej, element aktywny j^* jest najmniejszym elementem takim, że dla każdego większego elementu j jest spełniony jeden z następujących dwu warunków:

(1) $PR[j]$ and $(BLOK[j] = j)$, tzn. element j porusza się do przodu i osiągnął swe skrajne położenie na prawo (oczywiście j nie może być elementem bloku o najmniejszym elemencie większym niż j),

(2) not $PR[j]$ and $(BLOK[j] = 1)$, tzn. element j porusza się do tyłu i osiągnął swe skrajne położenie na lewo (w bloku pierwszym).

Zasada ta jest zrealizowana w wierszach 30÷34. Przy okazji jest zmieniany kierunek poruszania się elementów $j > j^*$. Dodatkowym warunkiem w pętli 31 jest $j > 1$, gdyż z samej reprezentacji podziału wynika, że $j = 1$ nie może być elementem aktywnym (element 1 jest oczywiście zawsze elementem bloku numer 1). Jeśli każdy z elementów $j > 1$ spełnia warunek (1) lub (2), to łatwo się przekonać, że wszystkie podziały zostały już wygenerowane. W takim przypadku po wyjściu z pętli 31 mamy $j = 1$ i następuje wyjście z głównej pętli 8, tzn. zakończenie działania algorytmu. Z konstrukcji rekurencyjnej wynika też łatwo, że elementem aktywnym dla pierwszego podziału listy L_n, tzn. dla $\{\{1, ..., n\}\}$ jest element n. Taka też wartość przypisywana jest zmiennej j przed wejściem do pętli 8 (wiersz 7).

Przeanalizujmy teraz proces przesuwania elementu aktywnego (wiersze 9÷28). Najpierw znajduje się numer k bloku zawierającego element aktywny. Jeśli element ten porusza się do przodu, to wystarczy przesunąć go do bloku o numerze $NAST[k]$ (p. wiersz 19), lecz w dwóch pozostałych przypadkach zmienną $NAST[k]$ należy najpierw zmodyfikować. Pierwszy przypadek zachodzi gdy $NAST[k] = 0$, tzn. gdy k jest numerem ostatniego bloku podziału. Wtedy j

utworzy blok jednoelementowy – wystarczy przyjąć $NAST[k] := j$ i odpowiednio zaktualizować wartość zmiennych $NAST[j]$ i $POPRZ[j]$ (p. wiersz 13). Podobny jest drugi przypadek, w którym $NAST[k] > j$. Warunek ten oznacza, że wszystkie bloki na prawo od bloku numer k zawierają elementy większe od j (wszystkie te elementy zajmują swe skrajne prawe położenie, w przeciwnym razie j nie byłoby elementem aktywnym). Z konstrukcji rekurencyjnej łatwo wynika, że w przypadku tym należy utworzyć blok jednoelementowy zawierający j. Dokonywane jest to w bloku 16 (jedyną różnicą w stosunku do przypadku pierwszego jest to, że tym razem utworzony blok nie jest ostatnim blokiem podziału).

W sytuacji gdy element j porusza się do tyłu (p. wiersz 21) wystarczy przesunąć go do poprzedniego bloku (p. wiersz 22) i dokonać odpowiedniej aktualizacji tablic *POPRZ* i *NAST*, jeśli j tworzył blok jednoelementowy – ma to miejsce dokładnie wtedy, gdy $BLOK[j] = k = j$, gdyż każdy element $m > j$ bloku numer j zostałby wybrany elementem aktywnym w pętli 31.

```
( 1 2 3 4 )
( 1 2 3 )( 4 )
( 1 2 )( 3 )( 4 )
( 1 2 )( 3 4 )
( 1 2 4 )( 3 )
( 1 4 )( 2 )( 3 )
( 1 )( 2 4 )( 3 )
( 1 )( 2 )( 3 4 )
( 1 )( 2 )( 3 )( 4 )
( 1 )( 2 3 )( 4 )
( 1 )( 2 3 4 )
( 1 4 )( 2 3 )
( 1 3 4 )( 2 )
( 1 3 )( 2 4 )
( 1 3 )( 2 )( 4 )
```

Rys. 1.13 Ciąg podziałów zbioru $\{1, 2, 3, 4\}$ wygenerowany przez algorytm 1.19

Na rysunku 1.13 przedstawiono wszystkie podziały zbioru $\{1, ..., 4\}$ skonstruowane przez algorytm. Można pokazać, że średnia liczba kroków potrzebna do skonstruowania każdego następnego podziału jest ograniczona przez stałą niezależną od n (por. zad. 1.38; nie liczymy tu oczywiście liczby kroków potrzebnych do wypisania podziału).

Podziały liczby 1.

Zajmiemy się teraz następującym problemem: Na ile sposobów można daną liczbę naturalną n zapisać w postaci sumy

$$n = b_1 + \ldots + b_k \tag{1.51}$$

gdzie $k, b_1, \ldots, b_p > 0$. Będziemy przy tym uważali sumy (1.51) za równoważne, jeśli różnią się jedynie kolejnością składników. Klasę równoważnych sum postaci (1.51) możemy jednoznacznie reprezentować przez ciąg $a_1, \ldots, a_k$, gdzie $a_1 \geqslant \ldots \geqslant a_k$ oraz liczby $a_1, \ldots, a_k$ są liczbami $b_1, \ldots, b_k$ posortowanymi według

nierosnącej wielkości (w podobny sposób reprezentowaliśmy podzbiory zbioru przez ciągi rosnące i podzbiory zbioru z powtórzeniami przez ciągi niemalejące). Każdy taki ciąg $a_1, \ldots, a_k$ będziemy nazywać *podziałem liczby* n na k składników. Liczbę podziałów liczby n na k składników będziemy oznaczać przez $P(n, k)$, natomiast liczbę wszystkich podziałów liczby n (na dowolną liczbę składników) przez $P(n)$. Oczywiście

$$P(n) = \sum_{k=1}^{n} P(n, k), \qquad n > 0 \tag{1.52}$$

(przyjmujemy $P(0, 0) = P(0) = 1$). Dla przakładu $P(7) = 15$ i wszystkie podziały liczby 7 przedstawiono na rys. 1.14.

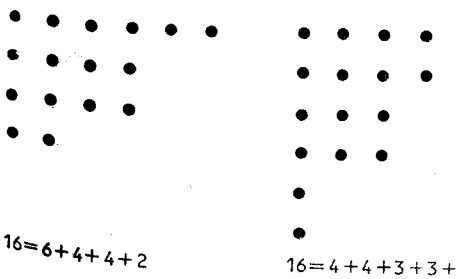
```
7
6 1
5 2
5 1 1
4 3
4 2 1
4 1 1 1
3 3 1
3 2 2
3 2 1 1
3 1 1 1 1
2 2 2 1
2 2 1 1 1
2 1 1 1 1 1
1 1 1 1 1 1 1
```

Rys. 1.14 Ciąg podziałów liczby 7 skonstruowany przez algorytm 1.22

Wiele ciekawych własności liczb $P(n)$ można udowodnić używając bardzo przejrzystego sposobu reprezentowania podziału liczby zwanego *diagramem Ferrersa*. Diagram Ferrersa dla podziału $n = a_1 + \ldots + a_k$ składa się z k wierszy odpowiadających składnikom podziału, przy czym i-ty wiersz zawiera ciąg a punktów (rys. 1.15).

Każdemu podziałowi liczby n odpowiada jednoznacznie *podział sprzężony* tej liczby, który otrzymujemy przez transpozycję (zamianę roli wierszy i kolumn) diagramu Ferrersa (por. zad. 1.15). Łatwo zauważyć, że transpozycja diagramu

11



Rys. 1.15 Diagram Ferrersa dla podziału liczby i podziału do niego sprężonego

Ferrersa określa wzajemnie jednoznaczną odpowiedniość między podziałami liczby n na k składników a podziałami liczby n o największym składniku równym k. Odnotujmy ten fakt:

TWIERDZENIE 1.20

Liczba podziałów liczby n na k składników jest równa liczbie podziałów liczby n o największym składniku równym k. ■

Udowodnimy jeszcze jedno twierdzenie tego typu.

TWIERDZENIE 1.21

Liczba podziałów liczby n na parami różne składniki jest równa liczbie podziałów liczby n na nieparzyste składniki.

DOWÓD

Pokażemy wzajemnie jednoznaczną odpowiedniość między podziałami, o których mowa w twierdzeniu. Rozważmy podział liczby n na nieparzyste składniki $b_1, \ldots, b_p$, gdzie składnik b_i występuje w podziale r_i razy, $1 \leqslant i \leqslant p$. Niech

$$r_i = 2^{q_1} + 2^{q_2} + \ldots \qquad (q_1 > q_2 > \ldots)$$

będzie przedstawieniem binarnym liczby r_i. Dokonujemy teraz zamiany r_i składników b_i na parami różne składniki

$$b_i 2^{q_1}, b_i 2^{q_2}, \ldots$$

(zamiana ta zachowuje oczywiście sumę składników podziału). Powtarzając te operacje dla każdego i, $1 \leqslant i \leqslant p$ i porządkując składniki według nierosnącej wielkości otrzymujemy w rezultacie podział liczby n na parami różne składniki. Wynika to z faktu, iż każdą liczbę naturalną można jednoznacznie przedstawić w postaci iloczynu liczby nieparzystej przez potęgę dwójki. Dla przykładu pokażemy opisaną transformację dla podziału $26 = 7+5+5+3+3+1+1+1$:

$$7+5+5+3+3+1+1+1 = 7\cdot 2^0 + 5\cdot 2^1 + 3\cdot 2^1 + 1\,(2^1+2^0) =$$
$$= 7+10+6+2+1 = 10+7+6+2+1$$

Łatwo zauważyć, że transformacji odwrotnej można dokonać, dla dowolnego podziału na parami różne składniki, przedstawiając każdy składnik jako $p2^q$, gdzie p nieparzyste, grupując następnie składniki według „czynnika nieparzystego" p i każdą taką grupę $p2^{q_1}, p2^{q_2}, \ldots$ zamieniając na $r = 2^{q_1} + 2^{q_2} + \ldots$ składników równych p. Tak więc opisana transformacja określa wzajemnie jednoznaczną odpowiedniość pomiędzy podziałami na składniki nieparzyste a podziałami na składniki parami różne. ■

Inne tego typu twierdzenia sformułowano w zadaniach 1.39÷1.41.

Pokażemy teraz prosty algorytm generowania wszystkich podziałów liczby. Będzie on generować podziały w porządku odwrotnym do leksykograficznego, tzn. podział

$$n = c_1 + \ldots + c_t \tag{1.53}$$

będzie wygenerowany – niekoniecznie bezpośrednio – po podziale

$$n = a_1 + \ldots + a_k \tag{1.54}$$

wtedy i tylko wtedy, gdy istnieje wskaźnik $p \leqslant min\,(k, l)$ taki, że $c_p < a_p$ oraz $c_m = a_m$ dla $1 \leqslant m < p$ (por. rys. 1.14). Oczywiście pierwszym podziałem w tym uporządkowaniu jest podział na jeden składnik równy n, ostatnim natomiast podział na n składników równych jedności. Zastanówmy się, jak wygląda podział bezpośrednio następujący po podziale (1.54). Poszukujemy podziału, który ma jak największą liczbę początkowych składników równych początkowym składnikom podziału (1.54) – oznaczmy te składniki przez $a_1, \ldots, a_{t-1}$ – i którego pozostałe składniki określone są przez podział bezpośrednio następujący po podziale $s = a_t + a_{t+1} + \ldots + a_k$. Łatwo zauważyć, że warunki te określają jednoznacznie

$$t = max\,\{i\colon\ a_i > 1\}$$

Problem sprowadza się więc do znalezienia bezpośredniego następnika podziału

$$s = a_t + \underbrace{1 + \ldots + 1}_{k-t \text{ razy}}, \quad a_t > 1$$

Nietrudno zauważyć, że podział taki ma postać

$$s = \underbrace{l + \ldots + l}_{\lfloor s/l \rfloor \text{ razy}} + (s \bmod l)$$

gdzie $l = a_t - 1$.

W algorytmie, który teraz opiszemy podział reprezentowany będzie przez zmienne $S[1], \ldots, S[d]$ zawierające parami różne składniki podziału ($S[1] > \ldots > S[d]$) oraz przez zmienne $R[i], \ldots, R[d]$, gdzie $R[i]$ zawiera informację, ile razy składnik $S[i]$ występuje w podziale ($R[i] > 0$). Pozwala to na znalezienie każdego następnego podziału w liczbie kroków ograniczonej przez pewną stałą niezależną od n.

ALGORYTM 1.22 (Znajdowanie wszystkich podziałów liczby)

Dane: n

Wyniki: Ciąg podziałów liczby n w porządku odwrotnym do leksykograficznego

```
1  begin
2    S[1] := n; R[1] := 1; d := 1; (* pierwszy podział *)
3    wypisz podział;
4    while S[1] > 1 do (* znajdź następny podział *)
5      begin sum := 0; (* sum = suma usuniętych składników *)
6        if S[d] = 1 then (* usuń składniki równe jedności *)
7          begin sum := sum + R[d]; d := d - 1
8          end;
```

```
 9          sum := sum+S[d]; R[d] := R[d]-1; l := S[d]-1;
10          if R[d] > 0 then d := d+1;
11          S[d] := l; R[d] := sum div l;
12          l := sum mod l;
13          if l ≠ 0 then (* dodaj ostatni składnik równy l *)
14            begin d := d+1; S[d] := l; R[d] := 1
15            end;
16          wypisz podział
17        end
18    end
```

Funkcje tworzące 1.1

W zagadnieniach kombinatorycznych polegających na policzeniu liczby obiektów spełniających pewne ograniczenia szukanym rozwiązaniem jest często ciąg a_0, $a_1, a_2, \ldots$, gdzie a_k jest liczbą poszukiwanych obiektów „wymiaru" k. Dla przykładu, jeśli poszukujemy liczby podziałów liczby, to możemy przyjąć $a_k = P(k)$, jeśli poszukujemy liczby podzbiorów pewnego ustalonego zbioru n-elementowego, to $a_k = \binom{n}{k}$ itd. W takim przypadku wygodnie jest przyporządkować ciągowi $a_0, a_1, a_2, \ldots$ szereg formalny

$$A(x) = \sum_{k=0}^{\infty} a_k x^k \tag{1.55}$$

zwany *funkcją tworzącą* dla tego ciągu. Nazwa „szereg formalny" oznacza, że (1.55) traktujemy jedynie jako wygodny zapis naszego ciągu – nie jest tu istotne dla jakich rzeczywistych (lub zespolonych) wartości zmiennej x jest on zbieżny. Nie będziemy bowiem nigdy obliczać wartości takiego szeregu dla konkretnej wartości zmiennej x, będziemy jedynie dokonywać na takich szeregach pewnych operacji, a następnie wyznaczać współczynniki przy poszczególnych potęgach zmiennej x. Dla dowolnych szeregów $A(x) = \sum_{k=0}^{\infty} a_k x^k$, $B(x) = \sum_{k=0}^{\infty} b_k x^k$ definiujemy operację *dodawania*:

$$A(x)+B(x) = \sum_{k=0}^{\infty} (a_k+b_k) x^k \tag{1.56}$$

operację *mnożenia przez liczbę* (rzeczywistą lub zespoloną):

$$pA(x) = \sum_{k=0}^{\infty} pa_k x^k \tag{1.57}$$

oraz *iloczynu Cauchy'ego* (krótko: *iloczynu*):

$$A(x)\cdot B(x) = \sum_{k=0}^{\infty} c_k x^k \tag{1.58}$$

gdzie

$$c_k = a_0 b_k + a_1 b_{k-1} + \ldots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i} \tag{1.59}$$

Jeśli $a_k = 0$ dla $k > n$, to szereg (1.55) utożsamiać będziemy z wielomianem $a_n x^n + \ldots + a_0$.

Z analizy matematycznej wiadomo (por. np. [41]), że jeśli szereg (1.55) jest zbieżny w pewnym otoczeniu zera, to jego suma $A(x)$ jest funkcją analityczną w tym otoczeniu oraz

$$a_k = A^{(k)}(0)/k!, \qquad k = 0, 1, 2, \ldots \tag{1.60}$$

($A^{(k)}(0)$ oznacza wartość k-tej pochodnej funkcji $A(x)$ dla $x = 0$; szereg (1.55) to nic innego jak szereg Maclaurina funkcji $A(x)$). Co więcej, jeśli $A(x)$, $B(x)$ są funkcjami analitycznymi w otoczeniu zera, to wzory (1.56)÷(1.59) pozostają w mocy, jeśli $A(x)$ i $B(x)$ traktować jako wartości funkcji A i B w punkcie x, a szeregi rozumieć w zwykłym sensie, tzn. tak jak w analizie matematycznej. Ta zachowująca działania, wzajemnie jednoznaczna odpowiedniość między szeregami zbieżnymi w otoczeniu zera i funkcjami analitycznymi w otoczeniu zera pozwala na utożsamienie szeregu formalnego (1.55) z definiowaną przez niego funkcją analityczną, w przypadku szeregów zbieżnych w otoczeniu zera (mimo że szeregi traktować będziemy zawsze jako szeregi formalne, tzn. jako jedynie formalny zapis ich współczynników). Tak więc pisać będziemy na przykład

$$\sum_{k=0}^{\infty} x^k = (1-x)^{-1}$$

$$\sum_{k=0}^{\infty} \frac{1}{k!} x^k = \mathrm{e}^x$$

itd.

Znajdziemy teraz funkcje tworzące dla kilku prostych ciągów. Zgodnie z (1.13)

$$\sum_{k=0}^{\infty} \binom{n}{k} x^k = \sum_{k=0}^{n} \binom{n}{k} x^k = (1+x)^n \tag{1.61}$$

a więc ustalonego n funkcją tworzącą ciągu $\binom{n}{0}, \binom{n}{1}, \binom{n}{2}, \ldots$ jest $(1+x)^n$.

Mamy

$$\sum_{k=0}^{\infty} 2^k x^k = \sum_{k=0}^{\infty} (2x)^k = (1-2x)^{-1}$$

a więc funkcją tworzącą ciągu $1, 2, 4, 8, \ldots, 2^k, \ldots$ jest $(1-2x)^{-1}$. Korzystając z faktu, że funkcję analityczną można różniczkować wyraz za wyrazem możemy napisać

$$\sum_{k=0}^{\infty} kx^k = x \sum_{k=0}^{\infty} kx^{k-1} = x \frac{d}{dx} \sum_{k=0}^{\infty} x^k = x \frac{d}{dx}(1-x)^{-1} = x(1-x)^{-2}$$

a więc funkcją tworzącą dla ciągu $0, 1, 2, 3, \ldots$ jest $x(1-x)^{-2}$.

Przyjrzyjmy się bliżej wzorowi (1.61). Każdy z czynników $(1+x)$ możemy traktować jako odpowiadający pewnemu elementowi a_i zbioru $X = \{a_1, \ldots, a_n\}$ i reprezentujący dwie możliwe liczby wystąpień tego elementu w podzbiorze: zero razy (składnik $x^0 = 1$) oraz jeden raz (składnik $x^1 = x$). Oczywiście każdy podzbiór zbioru X określony jest jednoznacznie przez podanie liczby wystąpień w nim każdego z elementów, tzn. wybranie jednego ze składników z każdego czynnika iloczynu $(1+x) \ldots (1+x)$. Tak wyznaczony składnik rozwinięcia tego iloczynu ma swój udział równy jedności we współczynniku przy x^k, gdzie k jest licznością naszego podzbioru. Rozumowanie to przenosi się w naturalny sposób na sytuację, w której liczba wystąpień elementu może być większa od jedności, tzn. na zbiory z powtórzeniami. Załóżmy na przykład, że $X = (2*a_1, 3*a_2, 1*a_3, 4*a_4)$ i oznaczmy przez c_k liczbę podzbiorów k-elementowych tego zbioru z powtórzeniami. Na mocy analogicznego rozumowania jak poprzednio, funkcja tworząca dla ciągu $c_0, c_1, c_2, \ldots$ jest równa

$$\sum_{k=0}^{\infty} c_k x^k = (1+x+x^2)(1+x+x^2+x^3)(1+x)(1+x+x^2+x^3+x^4) =$$
$$= 1+4x+9x^2+15x^3+20x^4+22x^5+20x^6+15x^7+9x^8+4x^9+x^{10}$$

Na liczbę wystąpień elementu a_i można nakładać dowolne ograniczenia przez odpowiedni dobór i-tego czynnika. Na przykład czynnik ten ma postać $1+x^3+x^7$, jeśli i-ty element może występować 0, 3 lub 7 razy, $1+x^2+x^4+\ldots = (1-x^2)^{-1}$, jeśli element ten może występować dowolną parzystą liczbę razy itd. W szczególności jeśli nie nakładamy żadnych ograniczeń na liczbę wystąpień elementu a_i, $1 \leqslant i \leqslant n$, to funkcja tworząca ma postać

$$(1+x+x^2+\ldots)\ldots(1+x+x^2+\ldots) = (1-x)^{-1}\ldots(1-x)^{-1} = (1-x)^{-n}$$

Jeśli zauważymy, że k-ta pochodna funkcji $(1-x)^{-n}$ jest równa

$$\frac{d^k}{dx^k}(1-x)^{-n} = (-n)(-n-1)\ldots(-n-k+1)(1-x)^{-n-k}(-1)^k =$$
$$= [n]^k(1-x)^{-n-k}$$

i rozwiniemy $(1-x)^{-n}$ w szereg Maclaurina, to otrzymamy

$$(1-x)^{-n} = \sum_{k=0}^{\infty} \frac{[n]^k}{k!} x^k = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$$

co stanowi jeszcze jeden niezależny dowód twierdzenia 1.15.

Funkcje tworzące są również wygodnym narzędziem dla dowodu tożsamości związanych ze współczynnikami dwumiennymi. Na przykład (1.18) otrzymujemy przez porównanie współczynników po obu stronach równości

$$\sum_{k=0}^{m+n}\binom{m+n}{k}x^k = (1+x)^{m+n} = (1+x)^m(1+x)^n =$$

$$= \sum_{i=0}^{m}\binom{m}{i}x^i \cdot \sum_{j=0}^{n}\binom{n}{j}x^j = \sum_{k=0}^{m+n}\sum_{s=0}^{k}\binom{m}{s}\binom{n}{k-s}x^k$$

(skorzystaliśmy tu ze wzoru na współczynniki iloczynu Cauchy'ego dwóch szeregów, a właściwie w tym przypadku wielomianów).

Pokażemy teraz jak funkcje tworzące można zastosować w pewnych zagadnieniach związanych z podziałami liczby. Zauważmy najpierw, że każdy podział $n = a_1 + \ldots + a_k$ można reprezentować jednoznacznie przez ciąg $\langle\lambda_1, \ldots, \lambda_n\rangle$, gdzie λ_i oznacza liczbę składników równych i, tzn. $\lambda_i = |\{j: a_j = i \wedge 1 \leqslant j \leqslant k\}|$. Mamy oczywiście

$$\sum_{i=1}^{n} i\lambda_i = n$$

oraz każdy ciąg $\langle\lambda_1, \ldots, \lambda_n\rangle$ $(\lambda_1, \ldots, \lambda_n \geqslant 0)$ spełniający ten warunek określa jednoznacznie podział liczby n zawierający λ_i składników równych i, $1 \leqslant i \leqslant n$.

Oznaczmy przez $P_h(n)$ liczbę podziałów liczby n na składniki nie przekraczające h.

TWIERDZENIE 1.23

Funkcja tworząca dla ciągu $P_h(0), P_h(1), P_h(2), \ldots$ jest równa

$$(1+x+x^2+x^3+\ldots)(1+x^2+x^4+x^6+\ldots)\ldots(1+x^h+x^{2h}+x^{3h}+\ldots) =$$
$$= (1-x)^{-1}(1-x^2)^{-1}\ldots(1-x^h)^{-1}$$

DOWÓD

Zgodnie z definicją iloczyn h szeregów po prawej stronie jest sumą składników postaci $x^{\lambda_1}x^{2\lambda_2}\ldots x^{h\lambda_h}$ (λ_i jest numerem wyrazu wybranego z i-tego szeregu). Współczynnik przy x^n jest więc równy liczbie ciągów $\langle\lambda_1, \ldots, \lambda_h\rangle$ takich, że $\sum_{i=0}^{h} i\lambda_i = n$, a więc, zgodnie z naszymi poprzednimi uwagami, jest on równy $P_h(n)$. ■

W analogiczny sposób dowodzimy następującego twierdzenia:

TWIERDZENIE 1.24

Funkcja tworząca dla ciągu $P(0), P(1), P(2), \ldots$ jest równa

$$(1+x+x^2+x^3+\ldots)(1+x^2+x^4+x^6+\ldots)\ldots(1+x^h+x^{2h}+x^{3h}+\ldots)\ldots =$$
$$= \prod_{h=1}^{\infty}(1-x^h)^{-1} \tag{1.62}$$

■

Uwaga: W tym miejscu należałoby ściśle zdefiniować szereg określony przez nieskończony iloczyn szeregów $F_1(x)\cdot F_2(x)\cdot \ldots$. Oznaczmy przez $I_n(x)$ „iloczyn częściowy" $F_1(x)\cdot F_2(x)\cdot \ldots \cdot F_n(x)$, oraz przez p_n najmniejszą niezerową potęgę o niezerowym współczynniku w $F_n(x)$ (dla iloczynu (1.62) mamy $p_n = n$). Załóżmy, że współczynnik przy zerowej potędze w każdym z szeregów $F_n(x)$ jest równy jedności, oraz że $\lim_{n\to\infty} p_n = \infty$ (oba te założenia są spełnione dla iloczynu (1.62)). Współczynnik przy x^n w $I_m(x)$ jest wtedy taki sam dla wszystkich m większych od pewnej liczby m_n (takiej, że $p_m > n$ dla $m > m_n$). Ten właśnie współczynnik przyjmujemy jako współczynnik przy x^n w szeregu określonym przez iloczyn nieskończony $F_1(x)\cdot F_2(x)\cdot \ldots$.

Technika funkcji tworzących pozwala na prosty dowód niektórych własności podziałów liczby. Podamy dla przykładu inny dowód twierdzenia 1.21. Zauważmy w tym celu, że funkcja tworząca dla podziałów na parami różne składniki (tzn. dla ciągu $r_1, r_2, \ldots$, gdzie r_n jest liczbą podziałów liczby n na parami różne składniki) jest równa

$$R(x) = (1+x)(1+x^2)(1+x^3)\ldots(1+x^k)\ldots$$

natomiast funkcja tworząca dla podziałów na nieparzyste składniki jest równa

$$N(x) = (1-x)^{-1}(1-x^3)^{-1}\ldots(1-x^{2k-1})^{-1}\ldots$$

Korzystając z zależności $1+x^k = (1-x^{2k})/(1-x^k)$ mamy

$$R(x) = \frac{1-x^2}{1-x}\cdot\frac{1-x^4}{1-x^2}\cdot\frac{1-x^6}{1-x^3}\cdot\ldots = \frac{1}{1-x}\cdot\frac{1}{1-x^3}\cdot\frac{1}{1-x^5}\cdot\ldots = N(x)$$

Podamy jeszcze dwa przykłady zastosowania funkcji tworzących. Pierwszy z nich dotyczy tzw. *liczb Fibonacciego* F_n. Liczby te są rozwiązaniem równania rekurencyjnego

$$F_{n+1} = F_n + F_{n-1}, \quad n \geqslant 1 \tag{1.63}$$

z warunkami początkowymi

$$F_0 = F_1 = 1 \tag{1.64}$$

Funkcja tworząca $F(x)$ dla ciągu $F_0, F_1, F_2, \ldots$ spełnia równanie

$$F(x) = \sum_{k=0}^{\infty} F_k x^k = 1 + x + \sum_{k=2}^{\infty}(F_{k-2}+F_{k-1})x^k =$$

$$= 1 + x + x^2\sum_{k=2}^{\infty} F_{k-2}x^{k-2} + x\sum_{k=2}^{\infty} F_{k-1}x^{k-1} =$$

$$= 1 + x + x^2F(x) + x(F(x)-1) = 1 + (x+x^2)F(x)$$

Stąd otrzymujemy funkcję tworzącą dla liczb Fibonacciego

$$F(x) = (1-x-x^2)^{-1}$$

Znajdując pierwiastki równania $1-x-x^2 = 0$ otrzymujemy rozkład $1-x-x^2 = (1-ax)(1-bx)$ gdzie

$$a = (1+\sqrt{5})/2, \qquad b = (1-\sqrt{5})/2$$

Zastosujemy teraz metodę współczynników nieoznaczonych, aby znaleźć przedstawienie

$$\frac{1}{(1-ax)(1-bx)} = \frac{A}{1-ax} + \frac{B}{1-bx} = \frac{A+B-(Ab+Ba)x}{(1-ax)(1-bx)}$$

Porównując współczynniki w liczniku otrzymujemy $A+B = 1$, tzn. $B = 1-A$, a zatem $Ab+(1-A)\,a = 0$, co daje $A = a/(a-b)$, $B = -b/(a-b)$. Tak więc

$$F(x) = A(1-ax)^{-1} + B(1-bx)^{-1} =$$

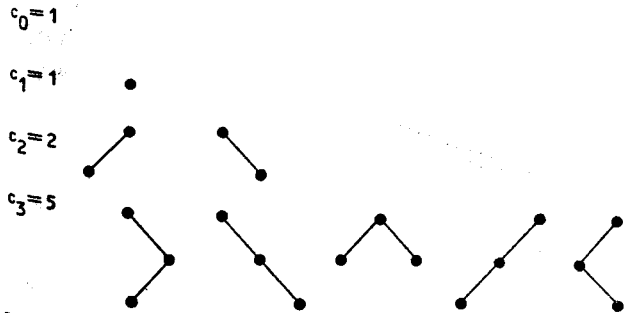
$$= A\sum_{k=0}^{\infty} a^k x^k + B\sum_{k=0}^{\infty} b^k x^k = \sum_{k=0}^{\infty} \frac{a^{k+1}-b^{k+1}}{a-b}\, x^k$$

i ostatecznie

$$F_k = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{k+1}\right] \tag{1.65}$$

Metoda ta przenosi się na dowolne liniowe równania rekurencyjne o stałych współczynnikach (por. zad. 1.46).

Jako ostatni przykład zastosowania funkcji tworzących wyznaczamy liczbę drzew binarnych o n wierzchołkach. Przez *drzewo binarne* o n wierzchołkach rozumiemy drzewo puste $T = \emptyset$, jeśli $n = 0$ lub trójkę $T = \langle L, r, P\rangle$, gdzie r jest wierzchołkiem zwanym *korzeniem drzewa*, L (*lewe poddrzewo*) jest drzewem binarnym o l wierzchołkach, P (*prawe poddrzewo*) jest drzewem binarnym o p wierzchołkach i $l+p+1 = n$. Mówimy, że drzewa binarne T_1 i T_2 są *izomorficzne*, co oznaczamy $T_1 \approx T_2$, jeśli $T_1 = T_2 = \emptyset$, lub $T_1 = \langle L_1, r_1, P_1\rangle$, $T_2 = \langle L_2, r_2, P_2\rangle$ gdzie $L_1 \approx L_2$ i $P_1 \approx P_2$. Oznaczmy przez c_k liczbę nieizomorficznych drzew binarnych o k wierzchołkach (por. rys. 1.16). Z podanych defi-



Rys. 1.16 Drzewa binarne o k wierzchołkach, $k = 1, 2, 3$

nicji rekurencyjnych wynika, że $c_0 = 1$ oraz jeśli $0 \leqslant s \leqslant k$, to istnieje dokładnie $c_s\, c_{k-1-s}$ nieizormoficznych drzew binarnych postaci $\langle L, r, P\rangle$, gdzie L jest drzewem binarnym o s wierzchołkach. Liczba s może przybierać każdą z wartości między 0 a $k-1$, a więc

$$c_k = c_0\, c_{k-1} + c_1\, c_{k-2} + \ldots + c_{k-1}\, c_0, \qquad k > 0 \tag{1.66}$$

Nieizomorficzne drzewa binarne dla $k = 0, 1, 2, 3$ przedstawiono na rys. 1.16.

Rozważmy funkcję tworzącą $C(x) = \sum_{k=0}^{\infty} c_k\, x^k$. Równość (1.66) przypomina bardzo wzór na współczynniki iloczynu Cauchy'ego $C(x)\, C(x) = C^2(x)$, dokładniej mówiąc spełnione jest równanie

$$C(x) = xC^2(x) + 1$$

czyli

$$xC^2(x) - C(x) + 1 = 0 \tag{1.67}$$

Pokażemy teraz, że istnieje funkcja $C(x)$ analityczna w otoczeniu zera spełniająca to równanie. Na mocy wspomnianej już zachowującej działania wzajemnie jednoznacznej odpowiedniości między szeregami a funkcjami analitycznymi współczynniki tego rozwiązania określą szereg formalny spełniający równanie (1.67). Traktując (1.67) jako równanie kwadratowe o niewiadomej $C(x)$ (wartości poszukiwanej funkcji analitycznej w punkcie x) otrzymujemy, dla $x \neq 0$,

$$C(x) = \frac{1 \pm \sqrt{1-4x}}{2x} \tag{1.68}$$

Rozwińmy $\sqrt{1-4x} = (1-4x)^{\frac{1}{2}}$ w szereg Maclaurina. Dla $k > 0$ mamy

$$\frac{d^k}{dx^k}(1-4x)^{\frac{1}{2}} = \frac{1}{2} \cdot \left(\frac{1}{2} - 1\right)\left(\frac{1}{2} - 2\right) \ldots \left(\frac{1}{2} - k + 1\right)(1-4x)^{\frac{1}{2}-k}(-4)^k =$$

$$= 2^k(-1) \cdot 1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k-3)\,(1-4x)^{\frac{1}{2}-k}$$

a więc

$$\sqrt{1-4x} = 1 - \sum_{k=1}^{\infty} \frac{2^k \cdot 1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k-3)}{k!}\, x^k =$$

$$= 1 - \sum_{k=1}^{\infty} \frac{2^k \cdot (2k-2)!}{k! \cdot 2 \cdot 4 \cdot \ldots \cdot (2k-2)}\, x^k = 1 - 2\sum_{k=1}^{\infty} \frac{(2k-2)!}{k!(k-1)!}\, x^k =$$

$$= 1 - 2\sum_{k=1}^{\infty} \frac{1}{k}\binom{2k-2}{k-1} x^k$$

Widać stąd, że aby otrzymać rozwiązanie o dodatnich współczynnikach należy wybrać znak minus w (1.68). Mamy więc

$$C(x) = \frac{1-\sqrt{1-4x}}{2x} = \sum_{k=1}^{\infty} \frac{1}{k}\binom{2k-2}{k-1}x^{k-1} = \sum_{k=0}^{\infty} \frac{1}{k+1}\binom{2k}{k}x^k$$

Otrzymujemy stąd ostatecznie

$$c_k = \frac{1}{k+1}\binom{2k}{k} \tag{1.69}$$

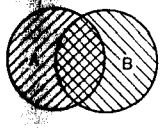
Liczby c_k zwane są *liczbami Catalana* i występują w kontekście szeregu innych problemów kombinatorycznych (por. zad. 1.47, 1.48).

1.13 Zasada włączania-wyłączania

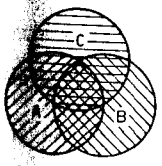
Podstawowe twierdzenie, które udowodnimy w tym paragrafie jest uogólnieniem oczywistego wzoru

$$|A \cup B| = |A| + |B| - |A \cap B|$$

który zachodzi dla dowolnych zbiorów A, B (rys. 1.17).



$$|A \cup B| = |A| + |B| - |A \cap B|$$



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

Rys. 1.17 Proste przypadki szczególne zasady włączania-wyłączania

Załóżmy, że dane są podzbiory $A_1, \ldots, A_n$ (niekoniecznie rozłączne) pewnego zbioru skończonego X, i poszukujemy liczności ich sumy $A_1 \cup \ldots \cup A_n$. Jako pierwsze „przybliżenie" tej liczności możemy przyjąć

$$|A_1| + \ldots + |A_n| \tag{1.70}$$

jednakże liczba ta jest na ogół za duża, gdyż jeśli $A_i \cap A_j \neq \emptyset$, to elementy przecięcia $A_i \cup A_j$ liczone są podwójnie. Spróbujmy sytuację poprawić odejmując od (1.70) sumę

$$\sum_{1 \leq i < j \leq n} |A_i \cap A_j| \tag{1.71}$$

Wtedy jednak otrzymamy liczbę na ogół za małą, gdyż jeśli $A_i \cap A_j \cap A_k \neq \emptyset$, to elementy przecięcia $A_i \cap A_j \cap A_k$ liczone są w (1.71) trzykrotnie. Kolejnym krokiem może być dodanie sumy $\sum_{1 \leqslant i < j < k \leqslant n} |A_i \cap A_j \cap A_k|$, lecz dla przyczyn podobnych jak poprzednio otrzymana liczba będzie na ogół za duża. Okazuje się jednak, że po n krokach otrzymamy zawsze poprawny wynik. Zachodzi mianowicie następujące twierdzenie:

TWIERDZENIE 1.25 (*Zasada włączania-wyłączania*)

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| + \sum_{1 \leqslant i < j < k \leqslant n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1} |A_1 \cap \ldots \cap A_n|$$

DOWÓD

Zastosujemy indukcję względem n. Dla $n = 1$ twierdzenie jest oczywiście prawdziwe. Załóżmy, że dla dowolnych podzbiorów $A_1, \ldots, A_{n-1}$

$$\left|\bigcup_{i=1}^{n-1} A_i\right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1 \leqslant i < j \leqslant n-1} |A_i \cap A_j| + \ldots + (-1)^{n-2} |A_1 \cap \ldots \cap A_{n-1}|$$

Stosując ten wzór do sumy

$$(A_1 \cup \ldots \cup A_{n-1}) \cap A_n = \bigcup_{i=1}^{n-1} (A_i \cap A_n)$$

otrzymujemy

$$\left|\bigcup_{i=1}^{n-1} A_i \cap A_n\right| = \sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1 \leqslant i < j \leqslant n-1} |A_i \cap A_j \cap A_n| + \ldots + (-1)^{n-2} |A_1 \cap \ldots \cap A_n|$$

a stąd

$$\left|\bigcup_{i=1}^{n} A_i\right| = \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n\right| = \left|\bigcup_{i=1}^{n-1} A_i\right| + |A_n| - \left|\bigcup_{i=1}^{n-1} A_i \cap A_n\right| =$$

$$= \sum_{i=1}^{n} |A_n| - \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| + \ldots + (-1)^{n-1} |A_1 \cap \ldots \cap A_n| \quad \blacksquare$$

Pokażemy teraz parę zastosowań zasady włączania-wyłączania.

TWIERDZENIE 1.26

Jeśli $|X| = n$, $|Y| = m$, to liczba wszystkich funkcji z X na Y (tzn. funkcji $f: X \to Y$ takich, że $f(X) = Y$) jest równa

$$s_{n,m} = \sum_{i=0}^{m-1} (-1)^i \binom{m}{i} (m-i)^n \tag{1.72}$$

DOWÓD

Niech $Y = \{y_1, \ldots, y_m\}$. Oznaczmy przez A_i zbiór tych funkcji $f: X \to Y$, dla których $y_i \notin f(X)$. Wtedy oczywiście

$$f(X) \neq Y \Leftrightarrow f \in \bigcup_{i=1}^{m} A_i$$

Wiemy, że wszystkich funkcji $f: X \to Y$ jest m^n (twierdzenie 1.1), wystarczy więc wyznaczyć liczność sumy $A_1 \cup \ldots \cup A_m$. Zauważmy najpierw, że dla dowolnego ciągu $1 \leqslant p_1 < \ldots < p_i \leqslant m$ przecięcie $A_{p_1} \cap \ldots \cap A_{p_i}$ jest zbiorem wszystkich funkcji $f: X \to Y$ takich, że $y_{p_1}, \ldots, y_{p_i} \notin f(X)$, a więc liczność tego przecięcia wynosi $(m-p)^n$, tyle ile jest funkcji $f: X \to Y \setminus \{y_{p_1}, \ldots, y_{p_i}\}$. Zbiór i-elementowy $\{y_{p_1}, \ldots, y_{p_i}\} \subseteq Y$ możemy wybrać na $\binom{m}{i}$ sposobów, a więc zgodnie z zasadą włączania-wyłączania

$$s_{n,m} = m^n - \Big|\bigcup_{i=0}^{m-1} A_i\Big| = m^n - \sum_{i=1}^{m-1} (-1)^{i-1} \binom{m}{i} (m-i)^n =$$

$$= \sum_{i=0}^{m-1} (-1)^i \binom{m}{i} (m-i)^n \qquad \blacksquare$$

Warto zauważyć, że na mocy wzoru (1.41) twierdzenie to dostarcza prostego wzoru na liczby Stirlinga drugiego rodzaju:

$$S(n,k) = \frac{1}{k!} s_{n,k} = \frac{1}{k!} \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^n \tag{1.73}$$

Innym zastosowaniem zasady włączania-wyłączania jest wyznaczenie liczby podzbiorów k-elementowych zbioru z powtórzeniami $X = (k_1 * a_1, \ldots, k_n * a_n)$ (na razie wiemy tylko, że podzbiorów tych jest $\binom{n}{k}$ gdy $k_1 = \ldots = k_n = 1$, oraz $\binom{n+k-1}{k}$ gdy $k_1, \ldots, k_n \geqslant k$). Zauważmy, że problem ten jest równoważny znalezieniu liczby rozwiązań całkowitych układu

$$x_1 + \ldots + x_n = k$$
$$0 \leqslant x_i \leqslant k_i \quad \text{dla } i = 1, \ldots, n$$

Metodę rozwiązania tego problemu zilustrujemy na przykładzie. Niech $X = (4*a_1, 3*a_2, 7*a_3)$, $k = 11$. Utwórzmy zbiory

A = zbiór wszystkich podzbiorów k-elementowych zbioru z powtórzeniami $(k*a_1, k*a_2, k*a_3)$,

A_1 = zbiór tych $Y \in A$, które zawierają więcej niż 4 wystąpienia elementu a_1,

A_2 = zbiór tych $Y \in A$, które zawierają więcej niż 3 wystąpienia elementu a_2,

A_3 = zbiór tych $Y \in A$, które zawierają więcej niż 7 wystąpień elementu a_3.

Mamy $|A| = \binom{n+k-1}{k} = \binom{3+11-1}{11} = \binom{13}{11} = 78$, oraz jest oczywiste, że $|A_1|$ wynosi tyle, ile jest 6-elementowych ($k-5=6$) podzbiorów zbioru z powtórzeniami $(11*a_1,\ 11*a_2,\ 11*a_3)$, tzn. $\binom{3+6-1}{6} = \binom{8}{6} = 28$, podobnie $|A_2| = \binom{3+7-1}{7} = \binom{9}{7} = 36$, $|A_3| = \binom{3+3-1}{3} = \binom{5}{3} = 10$. Łatwo również zauważyć, że $|A_1 \cap A_2| = \binom{3+2-1}{2} = \binom{4}{2} = 6$, tyle ile 2-elementowych ($k-5-4=2$) podzbiorów zbioru z powtórzeniami $(11*a_1,\ 11*a_2,\ 11*a_3)$, oraz $A_2 \cap A_3 = A_1 \cap A_3 = \emptyset$.

Podzbiór $Y \in A$ jest podzbiorem zbioru z powtórzeniami X wtedy i tylko wtedy, gdy nie należy do A_1, A_2 ani A_3, a zatem, zgodnie z zasadą włączania-wyłączania, szukana liczba podzbiorów k-elementowych zbioru z powtórzeniami X jest równa

$$|A|-|A_1|-|A_2|-|A_3|+|A_1 \cap A_2|+|A_2 \cap A_3|+|A_1 \cap A_3|-|A_1 \cap A_2 \cap A_3| = \\ = 78-28-36-10+6+0+0-0 = 10$$

Ostatnim zastosowaniem zasady włączania-wyłączania, które pokażemy, jest wyznaczenie liczby nieporządków na zbiorze n-elementowym. Przez *nieporządek* na zbiorze $\{1, \ldots, n\}$ rozumiemy dowolną permutację f tego zbioru taką, że $f(i) \neq i$ dla $1 \leqslant i \leqslant n$. Oznaczmy przez D_n zbiór wszystkich nieporządków na zbiorze $\{1, \ldots, n\}$ oraz

$$A_i = \{f \in S_n: \quad f(i) = i\}, \qquad i = 1, \ldots, n$$

(przypomnijmy, że S_n oznacza zbiór wszystkich permutacji zbioru $\{1, \ldots, n\}$). Permutacja f jest nieporządkiem wtedy i tylko wtedy, gdy nie należy do żadnego ze zbiorów A_i, a zatem, w myśl zasady włączania-wyłączania

$$|D_n| = |S_n| - \sum_{i=1}^{n} |A_i| + \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| - \ldots + (-1)^{n-1} |A_1 \cap \ldots \cap A_n|$$

Lecz dla dowolnego ciągu $1 \leqslant p_1 < \ldots < p_i \leqslant n$ przecięcie $A_{p_1} \cap \ldots \cap A_{p_i}$ jest zbiorem tych permutacji f, dla których $f(p_j) = p_j$ dla $1 \leqslant j \leqslant i$, a więc $|A_{p_1} \cap \ldots \cap A_{p_i}| = (n-i)!$. Zważywszy, że ciąg $1 \leqslant p_1 < \ldots < p_i \leqslant n$ możemy wybrać na $\binom{n}{i}$ sposobów, otrzymujemy ostatecznie

$$|D_n| = \sum_{i=0}^{n} (-1)^i \binom{n}{i} (n-i)! = \sum_{i=0}^{n} (-1)^i \frac{n!}{i!} = \\ = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^n \frac{1}{n!}\right)$$

Warto zauważyć, że suma w nawiasie równa jest początkowym składnikom szeregu $e^{-1} = \sum_{i=0}^{\infty} (-1)^i \frac{1}{i!}$. Oznacza to, że nieporządki stanowią asymptotycznie $e^{-1} = 0.36788 \ldots$ wszystkich permutacji.

Zadania 1.14

1.9 W zawodach bierze udział 8 zawodników. Na ile sposobów mogą zostać rozdzielone medale (złoty, srebrny i brązowy)?

Udowodnij, że liczba sposobów w jaki można rozsadzić n spośród m osób przy okrągłym stole jest równa $[m]_n/n$. (Rozsadzenia różniące się jedynie cyklicznym przesunięciem osób wokół stołu uważamy za równe).

1.10 Ile spośród liczb pomiędzy 1000 i 10000 składa się z nieparzystych cyfr, ile z różnych cyfr?

1.11 Ile jest możliwych rezultatów, którymi mogą się zakończyć zawody, w których startuje 10 osób w trzech konkurencjach, jeśli każda osoba startuje w jednej, dowolnie przez siebie wybranej konkurencji? (Przez rezultat zawodów rozumiemy zestawienie kolejności wszystkich zawodników startujących w każdej z konkurencji).

1.12 Ile palindromów długości n można utworzyć używając 26 liter alfabetu? (*Palindromem* nazywamy dowolny wyraz, który brzmi tak samo czytany w przód jak i wspak, np. anna).

1.13 Udowodnij, że rząd (tzn. liczność) dowolnej grupy $G \subseteq S_n$ jest dzielnikiem rzędu S_n. (Wskazówka: Pokazać, że zbiory postaci $fG = \{fg : g \in G, f \in S_n\}$, stanowią podział grupy S_n na rozłączne bloki $|G|$-elementowe).

1.14 Udowodnij, że liczba permutacji $f \in S_n$ typu $1^{\lambda_1}2^{\lambda_2} \ldots n^{\lambda_n}$ równa jest $n!/(1^{\lambda_1} \ldots n^{\lambda_n} \lambda_1! \ldots \lambda_n!)$.

1.15 Udowodnij, że permutacje $f, g \in S_n$ są tego samego typu wtedy i tylko wtedy, gdy istnieje permutacja $h \in S_n$ taka, że $g = hfh^{-1}$.

1.16 Permutację $f \in S_n$ nazywamy *inwolucją*, jeśli $ff = e$. Udowodnij, że $f \in S_n$ jest inwolucją wtedy i tylko wtedy gdy jest typu $1^{\lambda_1}2^{\lambda_2}$ $(\lambda_1 + 2\lambda_2 = n)$ oraz że dowolna permutacja jest złożeniem dwóch inwolucji.

1.17 Dla permutacji $\langle a_1, \ldots, a_n \rangle$ definiujemy *wektor inwersyjny* jako $\langle d_1, \ldots, d_n \rangle$, gdzie $d_i = |\{j < i : a_j > a_i\}|$. Udowodnij, że wektor inwersyjny wyznacza permutację jednoznacznie.

1.18 Udowodnij, że zbiór wszystkich permutacji parzystych tworzy grupę.

1.19 Udowodnij, że algorytm 1.10 generuje wszystkie permutacje (w nieco innej kolejności), jeśli usunąć instrukcję $P[i] :=: P[m]$ w wierszu 15. Niech $f_1, f_2, \ldots, f_{n!}$ będzie ciągiem permutacji generowanym przez tak zmodyfikowany algorytm. Udowodnij, że ciąg $n!$ elementów $f_1^{-1}(i), f_2^{-1}(i), \ldots$ $\ldots, f_{n!}^{-1}(i)$ jest dla każdego i $(1 \leqslant i \leqslant n)$ taki sam z dokładnością do cyklicznego przesunięcia (por. [45]).

1.20 Udowodnij, że asymptotyczna wartość (dla $n \to \infty$) średniej liczby transpozycji przypadających na każdą permutację wygenerowaną przez algorytm 1.10 jest równa $cosh\ 1 = 1.543 \ldots$ $(cosh\ x = \sum_{k=0}^{\infty} x^{2k}/(2k)!)$, w przypadku

algorytmu zmodyfikowanego tak jak w poprzednim zadaniu wartość ta wynosi natomiast $sinh\ 1 = 1.175\ldots$ ($sinh\ x = \sum_{k=0}^{\infty} x^{2k+1}/(2k+1)!$).

1.14 Udowodnij, że każdą z liczb $0, \ldots, n!-1$ można jednoznacznie przedstawić w postaci $j = \sum_{k=1}^{n-1} d_k\, k!$, gdzie $0 \leqslant d_i \leqslant i$, przy czym ciągi $d_{k-1}, \ldots, d_1$ odpowiadające kolejnym liczbom pojawiają się w porządku leksykograficznym. Podaj algorytm konstruowania ciągu $\langle d_{k-1}, \ldots, d_1\rangle$ odpowiadającego liczbie j.

1.15 Wyeliminuj rekursję z algorytmu 1.11. Opisz taką realizację, w której liczba kroków potrzebnych do skonstruowania każdej następnej permutacji (nie tylko średnia wartość liczby kroków!) jest ograniczona przez stałą niezależną od n. (Wskazówka: Rozważmy uporządkowaną leksykograficznie listę L_n wszystkich ciągów $\langle c_n, \ldots, c_2\rangle$, gdzie $1 \leqslant c_i \leqslant i$. Udowodnij, że j-ta transpozycja dokonana przez algorytm 1.11 ma postać $P[B[p, c_p]] := : P[p]$, gdzie $p = min\,\{i: c_i < i\}$, a $\langle c_n, \ldots, c_2\rangle$ jest j-tym ciągiem na liście L_n. Kolejne ciągi na liście L_n można generować efektywnie reprezentując ciąg $\langle c_n, \ldots, c_2\rangle$ przez ciąg $\langle \bar{c}_n, \ldots, \bar{c}_2\rangle$, gdzie $\bar{c}_i = 1$, jeśli $c_i = i$ a $\bar{c}_i = c_i$ w przeciwnym przypadku, oraz używając stosu, którego aktualną zawartością jest zawsze ciąg $a_1, b_1, a_2, b_2, \ldots, a_s, b_s$ taki, że

$$\bigcup_{q=1}^{s} \{i: a_q \leqslant i \leqslant b_q\} = \{i: c_i < i\}$$

Szczytowy element stosu określa wspomniany poprzednio wskaźnik p.

1.16 Napisz program realizujący „zwyczajną" wersję nierekurencyjną algorytmu 1.11 oraz wersję sugerowaną w poprzednim zadaniu. Sprawdź empirycznie przez wykonanie programu, czy ta ostatnia jest szybsza, jeśli idzie o średni czas potrzebny do skonstruowania każdej następnej permutacji.

1.17 Załóżmy, że dla każdego nieparzystego m mamy $B[m, 1] = \ldots = B[m, m-1] = b^{(m)}$, oraz że dla dowolnego parzystego $m \geqslant 4$ ciąg $B[m, 1], \ldots, B[m, m-1]$ spełnia jeden z następujących dwóch warunków:
(i) Każdy spośród elementów $1, \ldots, m-1$ występuje w nim dokładnie raz, przy czym odległość między pozycjami zawierającymi $m-1$ i $b^{(m-1)}$ jest parzysta (odległość między pozycjami i, j rozumiemy jako $|i-j|$).
(ii) Każdy spośród elementów zbioru $\{1, \ldots, m-1\} \setminus \{m-1, b^{(m-1)}\}$ występuje w nim dokładnie raz oraz element $m-1$ lub $b^{(m-1)}$ występuje dwukrotnie, przy czym odległość między tymi wystąpieniami jest nieparzysta.

Udowodnij, że procedura $PERM(n)$ generuje wtedy wszystkie permutacje elementów $P[1], \ldots, P[n]$, przy czym φ_n (p. dowód poprawności algorytmu 1.11) jest transpozycją dla każdego nieparzystego $n \geqslant 3$ i cyklem długości n dla każdego parzystego n (por. poz. [44]).

1.18 Korzystając z poprzedniego zadania udowodnij, że następujące wersje procedury $B(m, i)$ w algorytmie 1.11 prowadzą do poprawnego algorytmu generowania wszystkich permutacji:

```
(1) if (m mod 2 = 0) and (m > 2) then
      if i > 1 then B := m-i
      else B := m-2
    else B := m-1
(2) if m mod 2 = 0 then
      if (i = 1) or (i = m-1) then B := m-i
      else B := i
    else B := 1
(3) if (m mod 2 = 0) and (i > 2) then B := m-i
    else B := m-1 (*Wells [71]*)
(4) if m mod 2 = 0 then B := i
    else B := 1 (*Heap [32]*)
(5) if m mod 2 = 0 then B := i
    else B := m-2
```

Jaką postać ma permutacja φ_m w każdym z tych przypadków? Podaj inne możliwe wersje tej procedury.

1.19 Udowodnij, że średnia liczba kroków potrzebna do wygenerowania każdej następnej permutacji w algorytmie 1.12 jest ograniczona przez stałą niezależną od n. Podaj wersję algorytmu, w której jest to prawdą również dla liczby kroków dla każdej konkretnej wygenerowanej permutacji (por. zad. 1.15).

1.20 Algorytmy 1.11 i 1.12 generują ciągi zawierające na przemian permutacje parzyste i nieparzyste (dlaczego?). Czy jest to prawdą dla algorytmu 1.10?

1.21 Udowodnij, że średnia liczba kroków potrzebna do wygenerowania każdego następnego podzbioru przez algorytm 1.13 jest ograniczona przez stałą niezależną od n. Podaj wersję algorytmu, w której jest to prawdą również dla każdego konkretnego podzbioru (por. zad. 1.15).

1.22 Podaj nierekurencyjną wersję algorytmu generującego wszystkie podzbiory zbioru z powtórzeniami.

1.23 Zastosuj algorytm 1.13 do rozwiązania następującego problemu: Mając danych $n+1$ wektorów d-wymiarowych $a_1, \ldots, a_n, s$ o współrzędnych całkowitych sprawdzić czy istnieje podzbiór $J \subseteq \{1, \ldots, n\}$ taki, że $\sum_{j \in J} a_j = s$.

1.24 Udowodnij tożsamości

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n-1}{k-1} + \binom{n-2}{k-2} + \ldots + \binom{n-k}{0}$$

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n-1}{k} + \binom{n-2}{k} + \ldots + \binom{k}{k}$$

dwoma sposobami: przez wielekrotne stosowanie wzoru (1.17), oraz przez nadanie prawej stronie pewnej interpretacji kombinatorycznej (np. dla pierwszej tożsamości $\binom{n-j}{k-j}$ jest liczbą podzbiorów k-elementowych $A \subseteq \{0, \ldots, n\}$ takich, że $min\ \{i : i \notin A\} = j$).

1.25 Udowodnij tożsamość

$$n\binom{n-1}{k-1} = k\binom{n}{k}$$

przez zliczenie na dwa sposoby par $\langle a, K\rangle$, gdzie $a \in K$ i K jest k-elementowym podzbiorem ustalonego zbioru n-elementowego X. Użyj tej tożsamości do wyprowadzenia wzoru (1.21).

1.26 Udowodnij, że iloczyn dowolnych kolejnych k liczb naturalnych dzieli się przez $k!$ $\left(\text{Wskazówka: Rozważyć współczynnik dwumienny } \binom{n+k}{k}\right)$.

1.27 Udowodnij, że dla dowolnych liczb naturalnych p, q, n

$$[p+q]_n = \sum_{k=0}^{n} \binom{n}{k} [p]_k [q]_{n-k}$$

$$[p+q]^n = \sum_{k=0}^{n} \binom{n}{k} [p]^k [q]^{n-k}$$

(Wskazówka: W przypadku pierwszej tożsamości policz na dwa sposoby liczbę wszystkich funkcji różnowartościowych f: $X \to P \cup Q$, gdzie $|X| = n$, $|P| = p$, $|Q| = q$, $P \cap Q = \emptyset$. Dla drugiej tożsamości znajdź podobną interpretację w terminach rozmieszczeń uporządkowanych w $p+q$ pudełkach).

1.28 Udowodnij, że

$$(x_1 + x_2 + \ldots + x_p)^n = \sum_{n_1 + \ldots + n_p = n} \binom{n}{n_1\ n_2 \ldots n_p} x_1^{n_1} \ldots x_p^{n_p}$$

gdzie $\binom{n}{n_1\ n_2 \ldots n_p} = n!/(n_1!\, n_2! \ldots n_p!)$. Podać interpretację kombinatoryczną współczynników. Czemu jest równe $\binom{n}{kn-k}$?

1.29 Podaj nierekurencyjną wersję generowania podzbiorów k-elementowych opartą na wzorze (1.25). Pokaż, że zależność rekurencyjna

$$G(n, k) = G(n-1, k),\ G(n-2, k-1) \cup \{n-1, n\},\ G^*(n-2, k-1) \cup \{n\}$$

określa również pewną listę podzbiorów k-elementowych, w której sąsiednie podzbiory mało się od siebie różnią. Podaj algorytm nierekurencyjny oparty na tej zależności.

1.30 Pokaż, że krata podziałów nie jest rozdzielna, tzn. że na ogół $\pi \wedge (\sigma \vee \mu) \neq (\pi \wedge \sigma) \vee (\pi \wedge \mu)$. Wskaż odpowiednie podziały π, σ, μ.

1.31 Udowodnij, że liczba ciągów długości n o elementach ze zbioru k-elementowego, zawierających każdy element tego zbioru co najmniej raz jest równa $k!\,S(n,k)$.

1.32 Podział π zbioru n-elementowego X jest *typu* $1^{\lambda_1}2^{\lambda_2}\dots n^{\lambda_n}$ (gdzie $\lambda_1+2\lambda_2+\dots+n\lambda_n=n$), jeśli zawiera on λ_i bloków i-elementowych, dla $i=1,\dots,n$. Udowodnij, że liczba podziałów typu $1^{\lambda_1}2^{\lambda_2}\dots n^{\lambda_n}$ jest równa $n!/(\lambda_1!\dots\lambda_n!(1!)^{\lambda_1}\dots(n!)^{\lambda_n})$.

1.33 Oznaczmy dla każdego $k\geqslant 0$

$$g^{(k)}=\frac{d^k}{dx^k}g(x),\qquad f^{(k)}=\frac{d^k}{dy^k}f(y)\Big|_{y=g(x)}$$

Udowodnij, że

$$\frac{d_n}{dx^n}f(g(x))=\sum_{j=0}^{n}\sum_{\substack{k_1+k_2+\dots+k_n=j\\ k_1+2k_2+\dots+nk_n=n\\ k_1,k_2,\dots,k_n\geqslant 0}} f^{(j)}\frac{n!\,(g^{(1)})^{k_1}\dots(g^{(n)})^{k_n}}{k_1!\,(1!)^{k_1}\dots k_n!\,(n!)^{k_n}}$$

Pokaż, że suma współczynników przy wszystkich składnikach postaci $f^{(j)}(g^{(1)})^{k_1}\dots(g^{(n)})^{k_n}$ jest równa liczbie Bella B_n.

1.34 Udowodnij, że dla każdego $m\geqslant 0$ macierz liczb Stirlinga pierwszego rodzaju $[s(n,k)]_{0\leqslant n,k\leqslant m}$ jest odwrotnością macierzy liczb Stirlinga drugiego rodzaju $[S(n,k)]_{0\leqslant n,k\leqslant m}$.

1.35 Udowodnij, że

$$[x]^n=\sum_{k=0}^{n}|s(n,k)|\,x^k$$

1.36 Udowodnij, że $|s(n,k)|$ jest liczbą tych permutacji zbioru n-elementowego, które mają w rozkładzie na cykle dokładnie k cykli.

1.37 Rozważmy następujący program:

```
min := ∞ ;
for i := 1 to n do
  if P[i] < min then min := P[i];
```

Udowodnij, że prawdopodobieństwo, iż dla losowo wybranej permutacji $P[1],\dots,P[n]$ instrukcja $min := P[i]$ będzie wykonywana dokładnie k razy wynosi $|s(n,k)|/n!$ (Wskazówka: Skorzystaj z poprzedniego zadania oraz znajdź wzajemnie jednoznaczną odpowiedniość między permutacjami mającymi k cykli w rozkładzie kanonicznym a permutacjami, dla których instrukcja ta jest wykonywana k razy). Udowodnij, że średnia liczba wykonań powyżej instrukcji dla losowo wybranej permutacji jest $O(\log n)$, dokładnie wynosi $\sum_{k=1}^{n}\frac{1}{k}$ (Wskazówka: Policz liczbę cykli we wszystkich $n!$ permutacjach).

1.38 Udowodnij, że średnia liczba kroków potrzebna do skonstruowania każdego następnego podziału przez algorytm 1.19 jest ograniczona przez stałą niezależną od n. Podaj modyfikację algorytmu gwarantującą, by liczba ta była ograniczona przez stałą dla każdego wygenerowanego podziału (por. zad. 1.15).

1.39 Udowodnij, że liczba podziałów liczby n, w których żaden ze składników nie przekracza k, jest równa liczbie podziałów liczby $n+k$ na dokładnie k składników, tzn. $P(n+k, k)$.

1.40 Udowodnij, że liczba podziałów *samosprzężonych* liczby n (tzn. równych podziałowi sprzężonemu) równa jest liczbie podziałów liczby n na parami różne składniki nieparzyste.

1.41 Oznaczamy przez $E(n)$ i $O(n)$ odpowiednio liczbę podziałów liczby n na parami różne składniki parzyste oraz na parami różne składniki nieparzyste. Udowodnij, że

$$E(n)-O(n) = \begin{cases}(-1)^k & \text{jeśli } n \text{ jest postaci } (3k^2 \pm k)/2 \\ 0 & \text{w przeciwnym przypadku}\end{cases}$$

1.42 Podaj algorytm generowania wszystkich $P(n, k)$ podziałów liczby n na k składników w porządku odwrotnym do leksykograficznego, taki, że liczba kroków potrzebna do skonstruowania każdego następnego podziału jest ograniczona przez stałą niezależną od n.

1.43 Szereg

$$\sum_{k=0} \frac{c_k}{k!} x^k$$

nazywamy *eksponencjalną funkcją tworzącą* dla ciągu $c_0, c_1, \ldots$. Niech c_k będzie liczbą różnych ciągów długości k o elementach ze zbioru $Y = \{y_1, \ldots, y_n\}$, w których liczba wystąpień elementu y_i należy do zbioru $\{m_{i1}, m_{i2}, \ldots\}$ $(m_{i1} < m_{i2} < \ldots)$. Udowodnij, że eksponencjalna funkcja tworząca dla ciągu $c_0, c_1, c_2, \ldots$ jest równa

$$C(x) = \left(\frac{x^{m_{11}}}{m_{11}!} + \frac{x^{m_{12}}}{m_{12}!} + \ldots\right) \times \left(\frac{x^{m_{21}}}{m_{21}!} + \frac{x^{m_{22}}}{m_{22}!} + \ldots\right) \ldots \left(\frac{x^{m_{n1}}}{m_{n1}!} + \frac{x^{m_{n2}}}{m_{n2}!} + \ldots\right)$$

1.44 Udowodnij, że

$$F_{n+1} = \sum_{k=0}^{n} \binom{n-k+1}{k} = \sum_{k=0}^{\lfloor (n+1)/2 \rfloor} \binom{n-k+1}{k}$$

1.45 Udowodnij, że liczba ciągów binarnych długości n nie zawierających wystąpień jedynki na żadnych dwóch sąsiednich pozycjach jest równa liczbie Fibonacciego F_{n+1}.

1.46 Opisz metodę rozwiązywania równań rekurencyjnych postaci

$$C_0 a_n + C_1 a_{n-1} + \ldots + C_r a_{n-r} = 0$$

(z warunkami początkowymi określającymi wartości $a_0, a_1, \ldots, a_{r-1}$) za pomocą funkcji tworzących, analogiczną do metody użytej do znalezienia wzoru (1.65) określającego liczby Fibonacciego.

1.47 Niech p_n będzie liczbą możliwych rozmieszczeń nawiasów w iloczynie $x_0 \ldots x_n$ (np. $p_2 = 2$, gdyż mamy następujące 2 sposoby rozmieszczenia nawiasów: $((x_0 x_1)x_2)$, $(x_0(x_1 x_2))$). Udowodnij, że p_n równa się liczbie Catalana c_n.

1.48 Udowodnij, że liczba sposobów, w jaki $(n+2)$-kąt wypukły na płaszczyźnie można podzielić na rozłączne trójkąty za pomocą $n-1$ przekątnych nieprzecinających się wewnątrz tego $(n+2)$-kąta jest równa liczbie Catalana c_n.

1.49 Oznaczmy przez $\pi(x)$ ilość liczb pierwszych nie przekraczających x. Udowodnij, że

$$\pi(n) - \pi(\sqrt{n}) = n - 1 - \sum_{1 \leqslant i \leqslant k} \left\lfloor \frac{n}{p_i} \right\rfloor + \sum_{1 \leqslant i < j \leqslant k} \left\lfloor \frac{n}{p_i p_j} \right\rfloor - \ldots + (-1^k) \left\lfloor \frac{n}{p_1 p_2 \cdots p_k} \right\rfloor$$

gdzie $k = \pi(\sqrt{n})$.

1.50 Oznaczmy przez $\varphi(n)$ ilość liczb naturalnych nie przekraczających n i względnie pierwszych z n (tzn. nie mających wspólnego dzielnika większego od jedności). Udowodnij, że

$$\varphi(n) = n\left(1 - \sum_{1 \leqslant i \leqslant m} \frac{1}{q_i} + \sum_{1 \leqslant i < j \leqslant m} \frac{1}{q_i q_j} - \ldots + (-1)^m \frac{1}{q_1 q_2 \cdots q_m}\right) = n \prod_{i=1}^{m} \left(1 - \frac{1}{q_i}\right)$$

gdzie $q_1, \ldots, q_m$ są wszystkimi różnymi dzielnikami pierwszymi liczby n.

Algorytmy grafowe

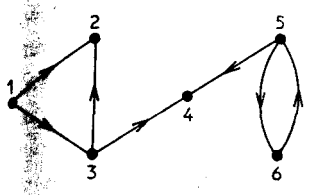
Reprezentacja maszynowa grafu

Oczywiście sposób reprezentacji grafu najbardziej przejrzysty i użyteczny dla człowieka – tzn. przez rysunek punktów i łączących je linii na płaszczyźnie – jest całkowicie bezużyteczny, jeśli chcemy problemy związane z grafami rozwiązywać za pomocą komputera. Wybór odpowiedniej struktury danych do reprezentacji grafu ma zasadniczy wpływ na efektywność algorytmów, i dlatego zajmiemy się teraz tym zagadnieniem nieco bliżej. Pokażemy kilka różnych sposobów reprezentacji i omówimy pokrótce ich podstawowe wady i zalety.

Będziemy rozważać zarówno grafy zorientowane, jak i niezorientowane. Graf oznaczać będziemy zawsze przez $G = \langle V, E \rangle$, gdzie V jest zbiorem wierzchołków, a E zbiorem krawędzi, przy czym $E \subseteq V \times V$ dla grafu zorientowanego i $E \subseteq \{\{x, y\} : x, y \in V \wedge x \neq y\}$ dla grafu niezorientowanego. Będziemy również konsekwentnie używać oznaczeń $|V| = n$ i $|E| = m$.

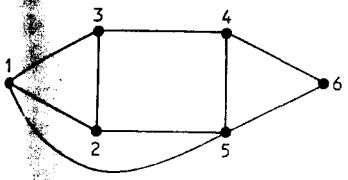
W teorii grafów klasycznym sposobem reprezentacji grafu jest *macierz incydencji*. Jest to macierz A o n wierszach odpowiadających wierzchołkom i m kolumnach odpowiadających krawędziom. Dla grafu zorientowanego kolumna odpowiadająca krawędzi $\langle x, y \rangle \in E$ zawiera -1 w wierszu odpowiadającym wierzchołkowi x, 1 w wierszu odpowiadającym wierzchołkowi y i zera we wszystkich pozostałych wierszach (*pętle*, tzn. krawędzie postaci $\langle x, x \rangle$ wygodnie jest reprezentować przez inną wartość, np. 2, w wierszu x). W przypadku grafu niezorientowanego kolumna odpowiadająca krawędzi $\{x, y\}$ zawiera 1 w wierszach odpowiadających x i y oraz zera w pozostałych wierszach. Zilustrowano to na rys. 2.1. Z algorytmicznego punktu widzenia macierz incydencji jest bardzo nieoszczędnym sposobem reprezentacji grafu. Po pierwsze wymaga aż nm miejsc pamięci, przy czym olbrzymia większość tych miejsc jest na ogół wypełniona zerami. Również dostęp do informacji jest bardzo niewygodny. Odpowiedź na elementarne pytania typu „czy istnieje krawędź $\langle x, y \rangle$?", „do jakich wierzchołków prowadzą krawędzie z x?" wymaga w najgorszym przypadku przeszukiwania wszystkich kolumn macierzy, a więc m kroków.

Bardziej efektywnym sposobem reprezentacji grafu jest *macierz sąsiedztwa wierzchołków*, określona jako macierz $B = [b_{ij}]$ wymiaru $n \times n$ gdzie $b_{ij} = 1$,



	⟨1,2⟩	⟨1,3⟩	⟨3,2⟩	⟨3,4⟩	⟨5,4⟩	⟨5,6⟩	⟨6,5⟩
1	-1	-1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	-1	-1	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	-1	-1	1
6	0	0	0	0	0	1	-1

b)



	{1,2}	{1,3}	{1,5}	{2,3}	{2,5}	{3,4}	{4,5}	{4,6}	{5,6}
1	1	1	1	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	0	0	0	0	1	1	1	0
5	0	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	0	1	1

Rys. 2.1 (a) Graf zorientowany i jego macierz incydencji
(b) Graf niezorientowany i jego macierz incydencji

jeśli istnieje krawędź prowadząca od i-tego do j-tego wierzchołka, oraz $b_{ij} = 0$ w przeciwnym przypadku. Rozumiemy tu, że krawędź $\{x, y\}$ grafu niezorientowanego prowadzi zarówno od x do y, jak i od y do x, tak że macierz sąsiedztwa wierzchołków takiego grafu jest zawsze symetryczna ($b_{ij} = b_{ji}$). Zilustrowano to na rys. 2.2.

a)

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

b)

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	1
6	0	0	0	1	1	0

Rys. 2.2 Macierze sąsiedztwa wierzchołków dla grafów z rys. 2.1

Główną zaletą macierzy sąsiedztwa wierzchołków jest fakt, iż możemy w jednym kroku otrzymać odpowiedź na pytanie „czy istnieje krawędź od x do y?". Wadą jest jednak fakt, że niezależnie od liczby krawędzi, zajętość pamięci wynosi n^2. W praktyce efekt tej niedogodności można czasami zmniejszyć przez pamiętanie całego wiersza (kolumny) macierzy w jednym słowie maszynowym — jest to możliwe dla małych n.

Jako argument przeciwko korzystaniu z macierzy sąsiedztwa wierzchołków przytoczymy jeszcze pewne twierdzenie dotyczące liczby kroków, jaką musi wykonać algorytm testujący pewną własność grafu na podstawie macierzy sąsiedztwa wierzchołków.

Niech P oznacza pewną własność grafów ($P(G) = 0$ lub $P(G) = 1$ w zależności od tego czy G ma, czy też nie ma naszej własności). Załóżmy, że cecha P spełnia następujące trzy warunki:

(a) $P(G) = P(G')$, jeśli grafy G, G' są izomorficzne;

(b) $P(G) = 0$ dla dowolnego grafu pustego $\langle V, \emptyset \rangle$ oraz $P(G) = 1$ dla dowolnego grafu pełnego $\langle V, P_2(V) \rangle$ o dostatecznie dużej liczbie wierzchołków;

(c) dodawanie krawędzi nie narusza własności P, tzn. $P(G) \leqslant P(G')$ dla dowolnych grafów $G = \langle V, E \rangle$, $G' = \langle V, E' \rangle$ takich, że $E \subseteq E'$.

Przykładem takiej własności P jest istnienie cyklu (w grafie o co najmniej trzech wierzchołkach).

TWIERDZENIE 2.1

Jeśli P jest własnością grafu spełniającą warunki (a), (b), (c), to każdy algorytm testujący własność P (tzn. obliczający wartość $P(G)$ dla danego grafu G) na podstawie macierzy sąsiedztwa wierzchołków wykonuje, w najgorszym przypadku, $\Omega(n^2)$ kroków, gdzie n jest liczbą wierzchołków grafu. ■

Twierdzenie to pozostaje prawdziwe również dla grafów zorientowanych i własności niezależnych od pętli grafu, tzn. spełniających dodatkowo warunek

(d) $P(G) = P(G')$ dla dowolnych grafów zorientowanych $G = \langle V, E \rangle$, $G' = \langle V, E \cup \langle v, v \rangle \rangle$, $v \in V$.

Dowód twierdzenia Czytelnik może znaleźć w pracach [5] i [59].

Oszczędniejsza jeśli idzie o zajętość pamięci – szczególnie w przypadku grafów „rzadkich", gdy m jest dużo mniejsze od n^2 – jest metoda reprezentacji grafu przez listę par odpowiadających jego krawędziom. Para $\langle x, y \rangle$ odpowiada krawędzi $\langle x, y \rangle$, jeśli graf jest zorientowany i krawędzi $\{x, y\}$ w przypadku grafu niezorientowanego (rys. 2.3). Zajętość pamięci wynosi w tym przypadku oczywiście $2m$. Niedogodnością jest duża liczba kroków – rzędu m w najgorszym przy-

a)

1	2
1	3
3	2
3	4
5	4
5	6
6	5

b)

1	2
1	3
1	5
2	3
2	5
3	4
4	5
4	6
5	6

RYS. 2.3 Listy krawędzi odpowiadające grafom z rys 2.1

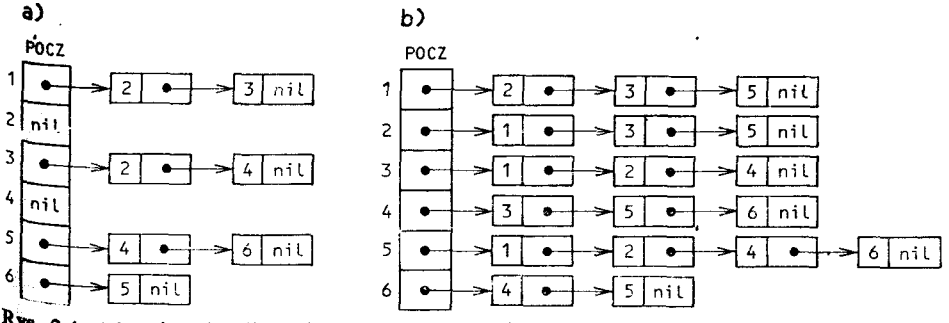
padku – potrzebna do uzyskania zbioru wierzchołków, do którego prowadzą krawędzie z danego wierzchołka.

Sytuację można znacznie poprawić przez uporządkowanie zbioru par leksykograficznie i stosowanie przeszukiwania binarnego, lecz najlepszym rozwiązaniem w wielu przypadkach okazuje się struktura danych, którą nazywać będziemy *listami incydencji*. Zawiera ona dla każdego wierzchołka $v \in V$ listę wierzchołków u takich, że $v \to u$ (lub $v-u$ w przypadku grafu niezorientowanego). Dokładniej, każdy element takiej listy jest rekordem r zawierającym wierzchołek $r.wierzch$ i wskaźnik $r.nast$ do następnego rekordu na liście ($r.nast = nil$ dla ostatniego rekordu listy). Początek każdej z list pamiętany jest w tablicy $POCZ$; dokładniej, $POCZ[v]$ jest wskaźnikiem do początku listy zawierającej wierzchołki ze zbioru $\{u: v \to u\}$ ($\{u: v-u\}$ dla grafu niezorientowanego). Całą taką listę będziemy zwykle nieformalnie oznaczać przez $ZA[v]$, a pętlę wykonującą pewną akcję dla każdego elementu u tej listy – w dowolnej, ale deterministycznie ustalonej kolejności, odpowiadającej kolejności elementów na liście – będziemy zapisywać jako „**for** $u \in ZA[v]$ **do** ...".

Zauważmy, że w przypadku niezorientowanym każda krawędź $\{u, v\}$ jest reprezentowana dwukrotnie: przez wierzchołek v na liście $ZA[u]$ i przez wierzchołek u na liście $ZA[v]$. W wielu algorytmach struktura grafu jest dynamicznie modyfikowana przez dodawanie i usuwanie krawędzi. W takich przypadkach zakładamy, że w naszych listach incydencji element listy $ZA[u]$ zawierający wierzchołek v jest zaopatrzony we wskaźnik do elementu listy $ZA[v]$ zawierającego wierzchołek u, oraz że każdy element listy zawiera wskaźnik nie tylko do następnego, lecz również do poprzedniego elementu listy. Wtedy usuwając pewien element z listy możemy łatwo – w liczbie kroków ograniczonej przez stałą – usunąć drugi element reprezentujący tę samą krawędź, bez konieczności przeszukiwania listy zawierającej ten element (por. zad. 2.3).

W analogiczny sposób określamy, dla każdego wierzchołka v grafu niezorientowanego, listę $PRZED[v]$ zawierającą wierzchołki ze zbioru $\{u: u \to v\}$.

Liczba miejsc pamięci potrzebna do reprezentacji grafu przez listy incydencji jest oczywiście rzędu $m+n$. Listy incydencji odpowiadające grafom z rys. 2.1 przedstawiono na rys. 2.4.



Rys. 2.4 Listy incydencji $ZA[v]$, $v \in V$ odpowiadające grafom z rys. 2.1

Przeszukiwanie grafu w głąb

Istnieje wiele algorytmów grafowych, których ogólny szkielet polega na systematycznym przeszukiwaniu wierzchołków grafu tak, by każdy wierzchołek był odwiedzony dokładnie raz. Dlatego też ważnym problemem jest znalezienie dobrych metod przeszukiwania grafu. Mówiąc najogólniej, metoda przeszukiwania jest „dobra" jeśli

(a) pozwala na łatwe „zanurzenie" w niej algorytmu rozwiązywania interesującego nas problemu oraz

(b) każda krawędź grafu analizowana jest co najwyżej raz (lub, co nie zmienia istotnie sytuacji, liczbę razy ograniczoną przez stałą).

Opiszemy teraz tego typu metodę przeszukiwania grafu niezorientowanego, będącą jedną z fundamentalnych technik projektowania algorytmów grafowych. Metodę tę nazywamy *przeszukiwaniem w głąb* (ang. depth first search [66]) dla przyczyn, które – mamy nadzieję – wkrótce staną się jasne.

Ogólna idea tej metody jest następująca. Przeszukiwanie zaczynamy od pewnego ustalonego wierzchołka v_0. Następnie wybieramy dowolny wierzchołek u sąsiadujący z v_0 i rozpoczynamy nasz proces od u. Ogólnie, przypuśćmy, że znajdujemy się przy wierzchołku v. Jeśli istnieje *nowy* (jeszcze nie odwiedzony) wierzchołek u, $v-u$, to odwiedzamy ten wierzchołek (przestaje on być nowy), i zaczynając od niego kontynuujemy przeszukiwanie. Jeśli natomiast nie istnieje żaden nowy wierzchołek sąsiadujący z v, to mówimy, że wierzchołek v został *wykorzystany*, cofamy się do wierzchołka, z którego doszliśmy do v, i proces kontynuujemy (jeśli $v = v_0$, to przeszukiwanie jest zakończone). Innymi słowy, przeszukiwanie w głąb z wierzchołka v polega na przeszukiwaniu w głąb ze wszystkich nowych wierzchołków sąsiadujących z v.

Można to łatwo zapisać za pomocą następującej procedury rekurencyjnej:

```
1  procedure WG(v);
   (* przeszukiwanie w głąb z wierzchołka v;
      zmienne NOWY, ZA są globalne *)
2  begin odwiedź v; NOWY[v] := false;
3     for u ∈ ZA[v] do
4      if NOWY[u] then WG(u)
5  end (* wierzchołek v został wykorzystany *)
```

Przeszukiwanie w głąb dowolnego, niekoniecznie spójnego grafu odbywa się zgodnie z następującym algorytmem:

```
1  begin
2    for v ∈ V do NOWY[v] := true; (* inicjalizacja *)
3    for v ∈ V do
4        if NOWY[v] then WG(v)
5  end
```

2.2 Pokażemy teraz, że algorytm ten odwiedza każdy wierzchołek dokładnie raz, a jego złożoność jest $O(n+m)$. Zauważmy najpierw, że wywołanie $WG(v)$ powoduje odwiedzenie wszystkich wierzchołków składowej spójnej grafu zawierającej v (jeśli $NOWY[u]$ = **true** dla każdego wierzchołka u tej składowej). Wynika to bezpośrednio ze struktury procedury WG: po odwiedzeniu wierzchołka (wiersz 2) następuje wywołanie procedury WG dla wszystkich jego nowych sąsiadów (wiersz 3), a tym samym odwiedzenie wszystkich poprzednio nieodwiedzonych sąsiadów. Zauważmy jeszcze, że każdy wierzchołek jest odwiedzany co najwyżej raz, gdyż odwiedzony może być jedynie wierzchołek v, dla którego $NOWY[v]$ = **true**, natomiast natychmiast po odwiedzeniu tego wierzchołka wykonywana jest instrukcja $NOWY[v]$:= **false** (wiersz 2).

Nasz algorytm rozpoczyna przeszukiwanie kolejno od każdego jeszcze nieodwiedzonego wierzchołka, a zatem odwiedzane są wszystkie wierzchołki grafu (niekoniecznie spójnego).

Aby oszacować złożoność obliczeniową algorytmu zauważmy najpierw, że liczba kroków w obu pętlach (wiersze 2 i 3) jest rzędu n, nie licząc kroków wykonywanych przez wywołania procedury WG. Procedura ta wywoływana jest co najwyżej n razy w drugiej pętli, oraz po odwiedzeniu każdego z wierzchołków, dla każdego z jego nowych sąsiadów, w sumie $O(n+m)$ razy. Całkowita liczba kroków wykonywanych przez pętlę w wierszu 3 procedury WG (nie licząc kroków wykonywanych przez $WG(u)$), dla wszystkich wywołań tej procedury, jest rzędu m, liczby krawędzi. Daje to całkowitą złożoność algorytmu $O(n+m)$.

Zauważmy, że algorytm przeszukiwania w głąb dowolnego grafu można łatwo zmodyfikować tak, by wyznaczał on składowe spójne tego grafu. Odnotujmy ten fakt:

WNIOSEK 2.2

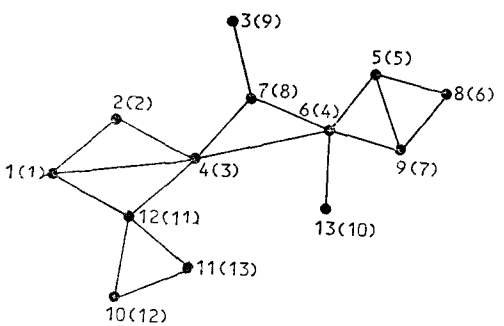
Składowe spójne dowolnego grafu reprezentowanego przez listy incydencji-można wyznaczyć w czasie $O(n+m)$. ■

Ze względu na dużą rolę przeszukiwania w głąb w projektowaniu algorytmów grafowych podamy jeszcze nierekurencyjną wersję procedury WG. Rekursję eliminujemy w standardowy sposób, przez użycie stosu (por. np. [1]). Każdy odwiedzany wierzchołek jest kładziony na stos oraz zdejmowany ze stosu, gdy zostaje wykorzystany.

```
1  procedure WG1(v);
     (* przeszukiwanie grafu w głąb z wierzchołka v – wersja nierekurencyjna
        procedury WG; zakładamy, że na początku procesu przeszukiwania
        P[u] = wskaźnik do pierwszego rekordu listy ZA[u] dla każdego wierz-
        chołka u; tablice P, NOWY są globalne *)
2  begin STOS := Ø; STOS ⇐ v; odwiedź v; NOWY[v] := false;
3    while STOS ≠ Ø do
```

```
4       begin t := top(STOS); (* t = szczytowy element stosu *)
          (* znajdź pierwszy nowy wierzchołek na liście ZA[t] *)
5         if P[t] = nil then b := false
6         else b := not NOWY[P[t]↑.wierzch];
7         while b do
8           begin P[t] := P[t]↑.nast;
9             if P[t] = nil then b := false
10            else b := not NOWY[P[t]↑.wierzch]
11          end;
12        if P[t] ≠ nil then (* nowy wierzchołek znaleziony *)
13            begin t := P[t]↑.wierzch; STOS ⇐ t;
14            odwiedź t; NOWY[t] := false
15            end
16        else (* wierzchołek t został wykorzystany *)
17          t ⇐ STOS (* zdejmij szczytowy element stosu *)
18        end
19  end
```

Poprawność tej procedury można pokazać przez łatwą modyfikację analizy procedury WG – pozostawiamy to Czytelnikowi. Na rysunku 2.5 pokazano graf, którego wierzchołki ponumerowano zgodnie z kolejnością, w jakiej są one odwiedzane w procesie przeszukiwania w głąb (utożsamiamy tu wierzchołki z liczbami 1, ..., 10 i przyjmujemy, że na liście $ZA[v]$ wierzchołki uszeregowane są rosnąco).



Rys. 2.5 Numeracja wierzchołków grafu z rys. 2.5 (w nawiasach) odpowiadająca kolejności, w jakiej są one odwiedzane w procesie przeszukiwania w głąb

Technika przeszukiwania w głąb przenosi się w oczywisty sposób na grafy zorientowane. Nietrudno sprawdzić, że efektem wywołania procedury $WG(v)$ jak również $WG1(v)$, jest w przypadku zorientowanym odwiedzenie w $O(n+m)$ krokach wszystkich wierzchołków u takich, że istnieje droga z v do u (por. zad. 2.11).

Przeszukiwanie grafu wszerz 2.3

Podamy teraz nieco inną metodę przeszukiwania grafu, zwaną *przeszukiwaniem wszerz* (ang. breadth first search). Zanim ją opiszemy zauważmy, że przy przeszukiwaniu w głąb, im wierzchołek zostaje później odwiedzony, tym zostaje wcześniej wykorzystany – ściślej, jest tak przy założeniu, że ten drugi wierzchołek jest odwiedzony przed wykorzystaniem pierwszego. Jest to natychmiastowy wniosek z faktu, iż wierzchołki odwiedzone, lecz jeszcze nie wykorzystane, są gromadzone na stosie. Przeszukiwanie wszerz polega, z grubsza mówiąc, na zastąpieniu stosu przez kolejkę. Po takiej modyfikacji, im wcześniej wierzchołek jest odwiedzony (umieszczony w kolejce), tym wcześniej jest wykorzystany (usunięty z kolejki). Wykorzystanie wierzchołka odbywa się przez odwiedzenie od razu wszystkich jeszcze nieodwiedzonych sąsiadów tego wierzchołka. Całą procedurę przedstawiono poniżej.

```
1  procedure WS(v);
   (* przeszukiwanie grafu wszerz z wierzchołka v;
      zmienne NOWY, ZA są globalne *)
2  begin
3    KOLEJKA := Ø; KOLEJKA ⇐ v; NOWY[v] := false;
4    while KOLEJKA ≠ Ø do
5      begin p ⇐ KOLEJKA; odwiedź p;
6        for u ∈ VA[p] do
7          if NOWY[u] then
8            begin KOLEJKA ⇐ u; NOWY[u] := false
9            end
10     end
11 end
```

W analogiczny sposób jak dla przeszukiwania w głąb można łatwo pokazać, że wywołanie $WS(v)$ powoduje odwiedzenie wszystkich wierzchołków składowej spójnej grafu zawierającej wierzchołek v, przy czym każdy wierzchołek jest odwiedzany dokładnie raz. Złożoność obliczeniowa przeszukiwania wszerz jest również rzędu $m+n$, gdyż każdy wierzchołek jest umieszczany w kolejce i usuwany z kolejki dokładnie raz, a liczba wszystkich iteracji pętli 6 jest oczywiście rzędu liczby krawędzi grafu.

Oba typy przeszukiwania grafu – w głąb i wszerz – mogą być użyte do znajdowania drogi między ustalonymi wierzchołkami v i u. Wystarczy przeszukać graf z wierzchołka v aż do momentu odwiedzenia wierzchołka u. Zaletą przeszukiwania w głąb jest fakt, że w momencie odwiedzania wierzchołka u stos zawiera ciąg wierzchołków określający drogę z v do u. Staje się to oczywiste jeśli zauważyć, że każdy wierzchołek kładziony na stosie jest sąsiadem szczytowego elementu stosu. Wadą przeszukiwania w głąb jest jednak to, że otrzymana droga nie jest na ogół najkrótszą drogą z v do u.

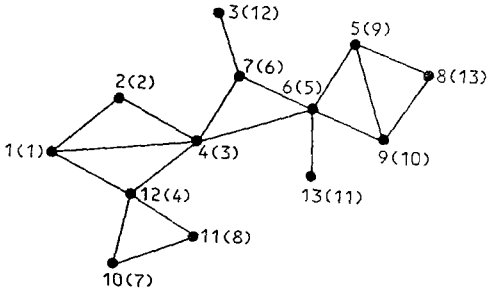
Wady tej nie ma metoda znajdowania drogi oparta na przeszukiwaniu wszerz. Zmodyfikujmy procedurę *WS* przez zmianę wierszy 7 ÷ 9

if *NOWY*[*u*] **then**
begin *KOLEJKA* ⇐ *u*; *NOWY*[*u*] := **false**; *POPRZ*[*u*] := *p*
end

Po zakończeniu działania tak zmodyfikowanej procedury tablica *POPRZ* zawiera, dla każdego odwiedzonego wierzchołka *u*, wierzchołek *POPRZ* [*u*], z którego doszliśmy do *u*. Zauważmy, że najkrótsza droga z *u* do *v* jest wyznaczana przez ciąg wierzchołków $u = u_1, u_2, \ldots, u_k = v$, gdzie $u_{i+1} = POPRZ[u_i]$ dla $1 \leqslant i < k$, oraz *k* jest pierwszym wskaźnikiem *i*, dla którego $u_i = v$. Istotnie, w kolejce umieszczane są najpierw wierzchołki znajdujące się w odległości 0 od *v* (tzn. sam wierzchołek *v*), następnie kolejno wszystkie nowe wierzchołki osiągalne z *v*, tzn. wierzchołki znajdujące się w odległości 1 od *v* itd. Przez odległość rozumiemy tu oczywiście długość najkrótszej drogi. Ogólnie, przyjmijmy założenie indukcyjne, że odwiedziliśmy już wszystkie wierzchołki znajdujące się w odległości od *v* nie przekraczającej *r*, że kolejka zawiera wszystkie wierzchołki znajdujące się w odległości *r* od *v*, i tylko te wierzchołki, oraz że tablica *POPRZ* określa poprawnie najkrótszą drogę z każdego już odwiedzonego wierzchołka do *v*, w sposób opisany poprzednio. Wykorzystując każdy wierzchołek *p* znajdujący się w kolejce, nasza procedura odwiedza pewne nowe wierzchołki, i każdy taki nowy wierzchołek *u* znajduje się oczywiście w odległości $r+1$ od *v*, przy czym określając *POPRZ* [*u*] := *p* przedłużamy najkrótszą drogę z *p* do *v* do najkrótszej drogi z *u* do *v*. Po wykorzystaniu wszystkich wierzchołków z kolejki, znajdujących się w odległości *r* od *v*, zawiera ona oczywiście zbiór wierzchołków znajdujących się w odległości $r+1$ od *v*, i łatwo zauważyć, że spełniony jest warunek indukcyjny dla odległości $r+1$.

Na rysunku 2.6 przedstawiono graf, którego wierzchołki ponumerowano zgodnie z kolejnością, w jakiej są one odwiedzane w procesie przeszukiwania wszerz.

Podobnie jak w przypadku przeszukiwania w głąb, procedura *WS* może być użyta bez żadnych modyfikacji, gdy listy incydencji *ZA* [*v*], $v \in V$ reprezentują pewien graf zorientowany. Oczywiście odwiedzone zostają jedynie te wierzchołki, do których istnieje droga z wierzchołka, z którego rozpoczynamy przeszukiwanie (por. zad. 2.12).



Rys. 2.6 Numeracja wierzchołków grafu z rys. 2.5 (w nawiasach) odpowiadająca kolejności, w jakiej są one odwiedzane w procesie przeszukiwania wszerz

Drzewa rozpinające 2.4

Drzewem nazywamy dowolny niezorientowany graf spójny bez cykli. Dla dowolnego spójnego grafu niezorientowanego $G = \langle V, E \rangle$ każde drzewo (V, T), gdzie $T \subseteq E$, nazywamy *drzewem rozpinającym* grafu G. Krawędzie takiego drzewa nazywamy *gałęziami*, a wszystkie pozostałe krawędzie grafu G nazywamy *cięciwami* (oczywiście o gałęziach i cięciwach w grafie możemy mówić tylko w kontekście ustalonego drzewa rozpinającego).

Zauważmy, że każde drzewo o n wierzchołkach zawiera $n-1$ krawędzi. Fakt ten można w prosty sposób udowodnić przez indukcję względem n. Jest on oczywiście prawdziwy dla $n = 1$. Jeśli $n > 1$, to zauważmy najpierw, że w każdym drzewie o n wierzchołkach istnieje wierzchołek „wiszący", tzn. wierzchołek stopnia 1. Istotnie, rozważmy w takim drzewie dowolną drogę $v_1 - v_2 - \ldots - v_k$ o maksymalnej długości ($v_i \neq v_j$ dla $i \neq j$). Wierzchołki v_1, v_k są „wiszące", gdyż nie może od żadnego z nich prowadzić druga krawędź do żadnego spośród wierzchołków $v_1, \ldots, v_k$ (drzewo nie zawiera cykli) ani do żadnego innego wierzchołka (droga jest maksymalna). Usuwając z naszego drzewa wierzchołek v_k i krawędź $\{v_{k-1}, v_k\}$ (albo też wierzchołek v_1 i krawędź $\{v_1, v_2\}$) otrzymujemy oczywiście drzewo o $n-1$ wierzchołkach, które — na mocy założenia indukcyjnego — ma $n-2$ krawędzie. Zatem nasze pierwotne drzewo miało $n-2+1 = n-1$ krawędzi.

Procedury przeszukiwania grafu w głąb i wszerz mogą być w prosty sposób zastosowane do znajdowania drzew rozpinających. W obu przypadkach osiągnięcie nowego wierzchołka u z wierzchołka v powoduje włączenie krawędzi $\{v, u\}$ do drzewa

ALGORYTM 2.3 (Znajdowanie drzewa rozpinającego grafu spójnego metodą przeszukiwania w głąb).

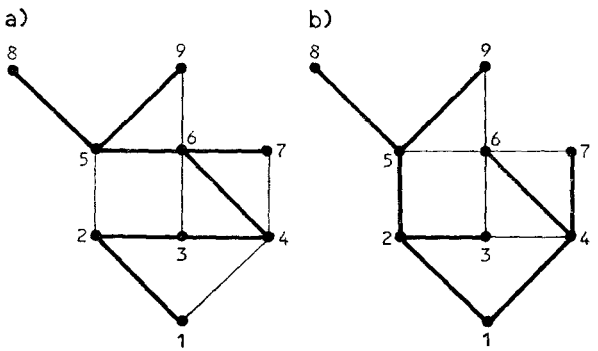
Dane: Spójny graf $G = \langle V, E \rangle$ reprezentowany przez listy incydencji $ZA[v]$, $v \in V$.

Wyniki: Drzewo rozpinające $\langle V, T \rangle$ grafu G.

```
1  procedure WGD(v);
   (* przeszukiwanie w głąb połączone ze znajdowaniem krawędzi drzewa;
      zmienne NOWY, ZA, T są globalne *)
2  begin NOWY[v] := false;
3    for u ∈ ZA[v] do
4      if NOWY[u] then (* {v, u} jest nową gałęzią *)
5        begin T := T ∪ {v, u}; WGD(u)
6        end
7  end; (* WGD *)
```

```
 8  begin (* program główny *)
 9    for u ∈ V do NOWY[u] := true; (* inicjalizacja *)
10    T := Ø; (* T = zbiór dotychczas znalezionych gałęzi *)
11    WGD(r) (* r jest dowolnym wierzchołkiem grafu *)
12  end
```

Aby udowodnić, że algorytm 2.3 poprawnie konstruuje drzewo rozpinające dowolnego grafu spójnego wystarczy zauważyć następujące trzy fakty. Po pierwsze, w momencie dodawania do zbioru T nowej gałęzi $\{v, u\}$ (wiersz 5) istnieje w $\langle V, T\rangle$ droga z r do v (fakt ten dowodzimy przez indukcję). Tak więc algorytm konstruuje graf spójny. Po drugie, każda nowa gałąź $\{v, u\}$ dodawana do zbioru T łączy odwiedzony już wierzchołek v (tzn. $NOWY[v]$ = **false**) z nowym wierzchołkiem u. Wynika stąd, że skonstruowany graf $\langle V, T\rangle$ nie zawiera cykli. Istotnie, ostatnia krawędź „zamykająca" cykl musiałaby łączyć dwa odwiedzone już wierzchołki. Wreszcie po trzecie, z własności przeszukiwania w głąb wynika, że procedura *WGD* odwiedza wszystkie wierzchołki grafu spójnego. Graf $\langle V, T\rangle$ skonstruowany przez nasz algorytm jest zatem drzewem rozpinającym grafu G. Złożoność obliczeniowa algorytmu jest oczywiście $O(n+m)$, tzn. jest tego samego rzędu co przeszukiwania w głąb. Przykład drzewa rozpinającego skonstruowanego przez algorytm 2.3 przedstawiono na rysunku 2.7a. Każde drzewo rozpinające wyznaczone przez ten algorytm ma pewną ciekawą własność, którą teraz opiszemy. Nazwijmy wierzchołek r, od którego rozpoczynamy przeszukiwanie grafu *korzeniem* drzewa rozpinającego $\langle V, T\rangle$. Oczywiście w drzewie $\langle V, T\rangle$ istnieje dokładnie jedna droga od dowolnego wierzchołka do korzenia (por. zad. 10).



Rys. 2.7 Drzewo rozpinające skonstruowane przez
(a) algorytm 2.3 oraz
(b) algorytm 2.5

Dla dwóch różnych wierzchołków v, u drzewa $\langle V, T\rangle$ powiemy, że u jest *potomkiem* wierzchołka v, jeśli v leży na drodze (w drzewie $\langle V, T\rangle$) z u do korzenia. Jeśli przy tym $v-u$, to mówimy, że u jest *synem* wierzchołka v, a v jest *ojcem* wierzchołka u.

TWIERDZENIE 2.4

Niech $\langle V, T\rangle$ będzie drzewem rozpinającym grafu spójnego $G = \langle V, E\rangle$ skonstruowanym przez algorytm 2.3 i niech $\{u, v\} \in E$. Wtedy albo u jest potomkiem v, albo v jest potomkiem u.

DOWÓD

Załóżmy, bez zmniejszenia ogólności, że wierzchołek v zostaje odwiedzony wcześniej niż u. Rozważmy proces przeszukiwania w głąb z wierzchołka v. Oczywiście po jego zakończeniu musi być $NOWY[u]$ = **false**, jako że $v-u$. Lecz to oznacza, że krawędzie dodane do zbioru T w czasie tego procesu zawierają drogę z v do u, a co za tym idzie v leży na drodze z u do korzenia. ■

W podobny sposób można skonstruować drzewo rozpinające używając techniki przeszukiwania wszerz.

ALGORYTM 2.5 (Znajdowanie drzewa rozpinającego grafu spójnego metodą przeszukiwania wszerz).

Dane: Spójny graf $G = \langle V, E\rangle$ reprezentowany przez listy indencji $ZA[v]$, $v \in V$.

Wyniki: Drzewo rozpinające $\langle V, T\rangle$ grafu G.

```
1  begin
2    for u ∈ V do NOWY[u] := true; (* inicjalizacja *)
3    T := Ø; (* T = zbiór dotychczas znalezionych gałęzi *)
4    KOLEJKA := Ø; KOLEJKA ⇐ r;
5    NOWY[r] := false; (* r = korzeń drzewa rozpinającego *)
6    while KOLEJKA ≠ Ø do
7      begin v ⇐ KOLEJKA;
8        for u ∈ ZA[v] do
9          if NOWY[u] then (* {v, u} jest nową gałęzią *)
10           begin KOLEJKA ⇐ u; NOWY[u] := false; T := T ∪ {v, u}
11           end
12     end
13 end
```

W podobny sposób jak w przypadku algorytmu 2.3 dowodzimy, że powyższy algorytm poprawnie konstruuje drzewo rozpinające dowolnego grafu spójnego, w $O(n+m)$ krokach.

Przykład drzewa rozpinającego skonstruowanego przez ten algorytm przedstawiono na rys. 2.7b.

Rozważania z punktu 2.3 prowadzą bezpośrednio do następującego twierdzenia:

TWIERDZENIE 2.6

Niech $\langle V, T\rangle$ będzie drzewem rozpinającym grafu spójnego $G = \langle V, E\rangle$ skonstruowanym przez algorytm 2.5. Wtedy droga w $\langle V, T\rangle$ z dowolnego wierzchołka v do korzenia r jest najkrótszą drogą z v do r w grafie G. ■

Nieco ogólniejsze zagadnienie znajdowania najkrótszych dróg w grafie, którego krawędziom przypisano „długości", niekoniecznie równe jedności, omówiono szczegółowo w rozdziale 3.

Rozważania tego punktu przenoszą się łatwo na dowolne grafy, niekoniecznie spójne. Maksymalny podgraf bez cykli dowolnego grafu G nazywamy *lasem rozpinającym* grafu G. Jest oczywiste, że las rozpinający grafu o k składowych spójnych jest określony przez drzewa rozpinające tych składowych, a zatem zawiera $n-k$ krawędzi.

Znajdowanie fundamentalnego zbioru cykli w grafie

Ściśle związane z problemem znajdowania drzewa rozpinającego jest zagadnienie konstrukcji fundamentalnego zbioru cykli. Jeśli do drzewa rozpinającego $\langle V, T\rangle$ grafu $G = \langle V, E\rangle$ dodamy dowolną cięciwę $e \in E \setminus T$, to nietrudno zauważyć, że powstały podgraf $\langle V, T \cup \{e\}\rangle$ zawiera dokładnie jeden cykl*, oznaczmy go przez C_e. Oczywiście C_e zawiera krawędź e. Zbiór $\mathscr{C} = \{C_e : e \in E \setminus T\}$ nazywamy *fundamentalnym zbiorem cykli* grafu G (względem drzewa rozpinającego $\langle V, T\rangle$). Poniżej udowodnimy, że każdy cykl grafu G można w pewien naturalny sposób otrzymać z cykli zbioru $\mathscr{C}$.

Oznaczmy, dla dowolnych zbiorów A, B

$$A \oplus B = (A \cup B) \setminus (A \cap B)$$

Zbiór $A \oplus B$ nazywamy *różnicą symetryczną* zbiorów A, B.

Zauważmy, że różnica symetryczna zbiorów $A_1, \ldots, A_k$ zawiera — niezależnie od rozmieszczenia nawiasów — dokładnie te elementy, które występują w nieparzystej liczbie zbiorów A_i. Istotnie, nietrudno to wykazać przez indukcję względem k. Dla $k = 1$ i $k = 2$ jest to oczywiście prawdą. Dla $k > 2$ nasza różnica symetryczna ma postać $A \oplus B$, gdzie A jest różnicą symetryczną zbiorów $A_1, \ldots, A_p$, B natomiast jest różnicą symetryczną zbiorów $A_{p+1}, \ldots, A_{p+q}$, przy czym $p, q < k$ i $p+q = k$. Zbiór $A \oplus B$ zawiera dokładnie te elementy, które występują tylko w jednym ze zbiorów A, B. Korzystając z założenia indukcyjnego wnioskujemy, że $A \oplus B$ jest zbiorem tych elementów, które występują w nieparzystej liczbie zbiorów spośród $A_1, \ldots, A_p$ i parzystej liczbie spośród $A_{p+1}, \ldots, A_{p+q}$, lub też nieparzystej liczbie zbiorów spośród $A_{p+1}, \ldots, A_{p+q}$ i parzystej spośród $A_1, \ldots, A_p$. Jest to dokładnie zbiór tych elementów, które występują w nieparzystej liczbie zbiorów spośród $A_1, \ldots, A_k$.

Z udowodnionej powyżej własności wynika, że możemy pomijać nawiasy w różnicy symetrycznej dowolnej liczby zbiorów i pisać $A_1 \oplus A_2 \oplus \ldots \oplus A_k$.

Zbiór C krawędzi grafu nazywamy *pseudocyklem*, jeśli każdy wierzchołek grafu $\langle V, C\rangle$ ma stopień parzysty. Przykładem pseudocyklu jest zbiór pusty i dowolny cykl grafu.

* W punkcie tym przez cykl będziemy zawsze rozumieć cykl elementarny.

LEMAT 2.7

Różnica symetryczna dowolnej liczby pseudocykli jest pseudocyklem.

DOWÓD

Wystarczy oczywiście rozważyć przypadek dwóch pseudocykli, C_1 i C_2. Dla dowolnego wierzchołka v oznaczmy przez $S_1(v)$ i $S_2(v)$ zbiór krawędzi odpowiednio z C_1 i C_2 incydentnych z v. Zbiory $S_1(v)$ i $S_2(v)$ są parzystej liczności, parzysta jest więc również liczność zbioru krawędzi z $C_1 \oplus C_2$ incydentnych z v, jako że $|S_1(v) \oplus S_2(v)| = |S_1(v)| + |S_2(v)| - 2|S_1(v) \cap S_2(v)|$. □

Możemy teraz udowodnić zapowiedziane już twierdzenie dotyczące fundamentalnego zbioru cykli.

2.5

TWIERDZENIE 2.8

Niech $G = \langle V, E \rangle$ będzie spójnym grafem niezorientowanym, a $\langle V, T \rangle$ jego drzewem rozpinającym. Dowolny cykl grafu G można jednoznacznie przedstawić jako różnicę symetryczną pewnej liczby cykli fundamentalnych. Ogólniej, dowolny pseudocykl C grafu G wyraża się jednoznacznie jako

$$C = \bigoplus_{e \in C \setminus T} C_e \tag{2.1}$$

DOWÓD

Różnica symetryczna $\bigoplus_{e=C\setminus T} C_e$, będąca pseudocyklem na mocy poprzedniego lematu, składa się ze zbioru cięciw $C \setminus T$ i pewnych gałęzi. Wynika to z faktu, iż każda cięciwa $e \in C \setminus T$ należy do dokładnie jednego cyklu fundamentalnego, mianowicie do C_e. Zbiór

$$C \oplus \bigoplus_{e \in C \setminus T} C_e \tag{2.2}$$

jest – znów na mocy poprzedniego lematu – pseudocyklem, przy czym może on zawierać tylko gałęzie. Żaden niepusty pseudocykl nie może być jednak zawarty w T, jako że każdy niepusty podgraf bez cykli zawiera wierzchołek stopnia 1 („wiszący"). Stąd wniosek, że zbiór (2.2) jest pusty, co jest równoważne równości (2.1). Jednoznaczność przedstawienia (2.1) wynika łatwo z faktu, iż cykl C_e jest jedynym cyklem fundamentalnym zawierającym cięciwę e ($e \in E \setminus T$). ■

Twierdzenie, które udowodniliśmy ma bardzo prostą interpretację w terminach przestrzeni liniowych. Niech $E = \{e_1, \ldots, e_m\}$ i przyporządkujmy każdemu pseudocyklowi C wektor $\langle a_1, \ldots, a_m \rangle$, gdzie

$$a_i = \begin{cases} 1 & \text{jeśli } e_i \in C \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Sumie takich wektorów, przy założeniu, że współrzędne obliczane są modulo 2, odpowiada różnica symetryczna odpowiednich pseudocykli. Lemat 2.7 orzeka, iż wektory odpowiadające pseudocyklom tworzą podprzestrzeń m-wymiarowej przestrzeni liniowej nad ciałem dwuelementowym (złożonym z 0,1, z operacją dodawania modulo 2 i mnożenia). Zauważmy, że dowolna kombinacja liniowa

w przestrzeni nad ciałem dwuelementowym odpowiada różnicy symetrycznej – jako że jedynym niezerowym współczynnikiem może być 1. Z kolei twierdzenie 2.8 mówi, że cykle fundamentalne określają bazę naszej podprzestrzeni.

Znalezienie fundamentalnego zbioru cykli ma istotne znaczenie przy analizie obwodów elektrycznych. Mówiąc dokładniej, każdemu cyklowi fundamentalnemu w grafie odpowiadającym danemu układowi elektrycznemu możemy przyporządkować równanie wyrażające *prawo Kirchhoffa dla napięć* (por. np. poz. [61]): suma spadków napięć wzdłuż cyklu jest równa zeru. Wtedy żadne z tych równań nie jest zależne od pozostałych, natomiast jest od nich zależne dowolne równanie wyrażające prawo Kirchhoffa dla dowolnego cyklu grafu.

Opiszmy teraz prosty algorytm znajdowania zbioru cykli fundamentalnych. Algorytm ten oparty jest na przeszukiwaniu w głąb i ma strukturę podobną do rekurencyjnego algorytmu znajdowania drzewa rozpinającego (algorytm 2.3). Każdy nowy wierzchołek napotykany w procesie przeszukiwania umieszczany jest na stosie – reprezentowanym przez tablicę *STOS* – i zdejmowany ze stosu po wykorzystaniu. Oczywiście stos zawiera zawsze ciąg wierzchołków z aktualnie odwiedzanego wierzchołka v do korzenia. Dlatego też, jeśli analizowana przez nas krawędź $\{v, u\}$ zamyka cykl (tzn. $WGN[v] > WGN[u] > 0$ i u nie znajduje się bezpośrednio pod elementem szczytowym stosu), to wierzchołek u – na mocy twierdzenia 2.4 – znajduje się na stosie i cykl zamknięty przez krawędź $\{v, u\}$ reprezentowany jest przez szczytową partię stosu, aż do wierzchołka u.

ALGORYTM 2.9 (Znajdowanie zbioru cykli fundamentalnych grafu)
Dane: Graf $G = \langle V, E \rangle$ reprezentowany przez listy incydencji $ZA[v]$, $v \in V$.
Wyniki: Zbiór cykli fundamentalnych grafu G.

```
1   procedure CYKLE(v);
    (* znajdowanie fundamentalnego zbioru cykli dla składowej zawierającej
       wierzchołek v; zmienne d, num, STOS, ZA, WGN są globalne *)
2   begin d := d+1; STOS[d] := v; num := num+1; WGN[v] = num;
3     for u ∈ ZA[v] do
4       if WGN[u] = 0 then CYKLE(u)
5       else if (u ≠ STOS[d-1]) and (WGN[v] > WGN[u]) then
                (* {v, u} zamyka nowy cykl *)
6               wypisz cykl o wierzchołkach
7               STOS[d], STOS[d-1], ..., STOS[c], gdzie STOS[c] = u;
8     d := d-1 (* wykorzystany wierzchołek v jest zdejmowany ze stosu *)
9   end; (* CYKLE *)
10  begin (* program główny *)
11    for v ∈ V do WGN[v] := 0; num := 0; (* inicjalizacja *)
12    d := 0; STOS[0] := 0; (* d = liczba elementów na stosie *)
13    for v ∈ V do
14      if WGN[v] = 0 then CYKLE(v)
15  end
```

Oszacujemy teraz złożoność obliczeniową tego algorytmu. Zauważmy najpierw, że całkowita liczba kroków, nie licząc wypisywania cykli (wiersze 6, 7) jest — tak jak we wszystkich algorytmach opartych na przeszukiwaniu w głąb — $O(n+m)$. Do tego należy dodać sumaryczną długość wszystkich cykli. Długość ta nie przekracza $(m-n+1)n$, co daje całkowitą złożoność algorytmu $O(nm+n)$ (lub $O(nm)$, jeśli odrzucimy zdegenerowany przypadek $m = 0$).

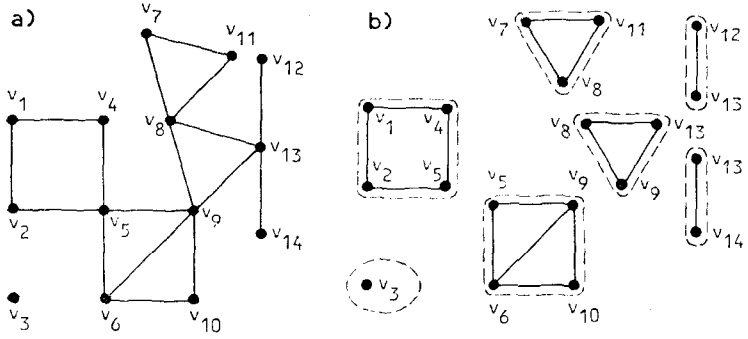
Znajdowanie składowych dwuspójnych 2.6

Wierzchołek a grafu niezorientowanego $G = \langle V, E\rangle$ nazywamy *punktem artykulacji*, jeśli usunięcie tego wierzchołka i wszystkich incydentnych z nim krawędzi powoduje zwiększenie liczby składowych spójnych grafu. Równoważnie możemy powiedzieć, że a jest punktem artykulacji, jeśli istnieją wierzchołki u, v różne od a takie, że każda droga z u do v — a zakładamy, że istnieje co najmniej jedna taka droga — przechodzi przez wierzchołek a. Graf niezorientowany nazywamy *dwuspójnym*, jeśli jest on spójny i nie zawiera punktów artykulacji. Dowolny maksymalny podgraf dwuspójny grafu G nazywamy *składową dwuspójną*, albo *blokiem* tego grafu.

W niektórych zastosowaniach dwuspójność grafu jest cechą bardzo pożądaną. Wyobraźmy sobie na przykład, że wierzchołki grafu reprezentują węzły pewnej sieci informacyjnej, a krawędzie odpowiadają liniom przesyłowym. Jeśli nasz graf jest dwuspójny, to awaria pojedynczego węzła w nie powoduje nigdy utraty połączenia między żadnymi dwoma węzłami różnymi od w. Znajomość bloków grafu jest też ważna ze względu na fakt, że wiele zagadnień grafowych — takich jak znalezienie wszystkich cykli elementarnych lub też stwierdzenie czy graf jest płaski (tzn. daje się narysować na płaszczyźnie tak, by żadne dwie krawędzie się nie przecinały) — sprowadza się w naturalny sposób do analogicznych problemów dla bloków danego grafu.

Zauważmy, że jeśli $\langle V_1, B_1\rangle$, $\langle V_2, B_2\rangle$ są dwoma różnymi blokami grafu G, to $V_1 \cap V_2 = \emptyset$ albo $V_1 \cap V_2 = \{a\}$, gdzie a jest punktem artykulacji grafu G. Istotnie, rozważmy podgraf $\langle V_1 \cup V_2, B_1 \cup B_2\rangle$. Gdyby $|V \cap V_2| \geqslant 2$, to podgraf ten byłby dwuspójny, wbrew założeniu o maksymalności $\langle V_1, B_1\rangle$ i $\langle V_2, B_2\rangle$. Również przypadek $V_1 \cap V_2 = \{a\}$, gdzie a nie jest punktem artykulacji grafu G, jest niemożliwy. Wtedy bowiem istnieje w G droga między wierzchołkami $u \in V_1$, $u \neq a$ oraz $v \in V_2$, $v \neq a$, nie zawierająca wierzchołka a. Nietrudno zauważyć, że podgraf powstały z $\langle V_1 \cup V_2, B_1 \cup B_2\rangle$ przez dodanie wierzchołków i krawędzi tej drogi jest dwuspójny, wbrew założeniu o maksymalności $\langle V_1, B_1\rangle$ i $\langle V_2, B_2\rangle$. Przykład grafu, jego punktów artykulacji i bloków przedstawiono na rys. 2.8.

Znajdowanie punktów artykulacji i bloków grafu jest klasycznym problemem, który można efektywnie rozwiązać przez zanurzenie w procedurze przeszukiwania w głąb (p. Tarjan [66]). Zanim podamy szczegółowy algorytm, poczynimy najpierw parę wstępnych obserwacji.



Rys. 2.8 (a) Graf w punktach artykulacji v_5, v_8, v_9, v_{13}
(b) Bloki tego grafu

TWIERDZENIE 2.10

Niech $D = \langle V, T \rangle$ będzie drzewem rozpinającym o korzeniu r, spójnego grafu $G = \langle V, E \rangle$ skonstruowanym przez przeszukiwanie w głąb (algorytm 2.3). Wierzchołek $v \in V$ jest punktem artykulacji grafu G wtedy i tylko wtedy, gdy albo $v = r$ i v ma co najmniej dwóch synów w D, albo $v \neq r$ i istnieje syn w wierzchołka v taki, że ani w, ani żaden jego potomek nie jest połączony krawędzią z żadnym przodkiem wierzchołka v.

DOWÓD

Rozważmy najpierw przypadek $v = r$. Jeśli wierzchołek v ma tylko jednego syna (albo jest wierzchołkiem izolowanym), to jego usunięcie nie zwiększa liczby składowych spójnych, jako że nie narusza to spójności drzewa. Jeśli natomiast ma on co najmniej dwóch synów, to po usunięciu wierzchołka v synowie ci leżą w różnych składowych spójnych. Istotnie, z twierdzenia 2.4 wynika, że każda droga pomiędzy dwoma różnymi synami musi przechodzić przez korzeń – w przeciwnym przypadku zawierałaby cięciwę $\{u, t\}$, gdzie ani u nie jest potomkiem t, ani też t nie jest potomkiem u. Podobnie jest w przypadku $v \neq r$. Jeśli po usunięciu wierzchołka v istnieje droga od jego syna w do korzenia, to tak jak poprzednio z twierdzenia 2.4 wnioskujemy, że droga ta musi zawierać krawędź łączącą w lub pewnego potomka wierzchołka w z pewnym przodkiem wierzchołka v. ■

Idea algorytmu jest następująca: Przeszukujemy graf w głąb z pewnego wierzchołka r, obliczając dla każdego wierzchołka v dwa parametry: $WGN[v]$ i $L[v]$. Pierwszy z nich, to po prostu numer wierzchołka v w kolejności, w jakiej wierzchołki odwiedzane są przy przeszukiwaniu w głąb poczynając od wierzchołka r. Po oznaczeniu przez $D = \langle V, T \rangle$ drzewa odpowiadającego naszemu procesowi przeszukiwania w głąb, drugi parametr określa minimalną wartość $WGN[u]$, gdzie $u = v$ lub wierzchołek u jest połączony cięciwą z wierzchołkiem v lub dowolnym jego potomkiem w D. Parametr $L[v]$ łatwo obliczyć

indukcyjnie względem drzewa D – jeśli znamy $L[w]$ dla wszystkich synów w wierzchołka v, to oznaczając

$$A = min\{L[w]: w \text{ jest synem wierzchołka } v\}$$
$$B = min\{WGN[u]: \{u, v\} \in E \setminus T\}$$

mamy

$$L[v] = min\{WGN[v], A, B\}$$

Łatwym wnioskiem z twierdzenia 2.10 jest fakt, że v jest punktem artykulacji lub korzeniem wtedy i tylko wtedy, gdy $L[w] \geqslant WGN[v]$ dla pewnego syna w wierzchołka v (zakładamy $n > 1$).

ALGORYTM 2.11 (Znajdowanie składowych dwuspójnych grafu)

Dane: Graf $G = \langle V, E \rangle$ bez wierzchołków izolowanych, reprezentowany przez listy incydencji $ZA[v]$, $v \in V$.

Wyniki: Zbiory krawędzi wszystkich składowych dwuspójnych.

```
1  procedure DWUSP(v, p);
   (* przeszukuj w głąb z wierzchołka v zakładając, że p jest ojcem wierz-
      chołka v w tym procesie, oraz wypisz krawędzie znalezionych składowych
      dwuspójnych; zmienne num, L, WGN, STOS są globalne *)
2  begin num := num+1; WGN[v] := num;
3    L[v] := WGN[v];
4    for u ∈ ZA[v] do
5      if WGN[u] = 0 then (* wierzchołek u jest nowy, {v, u} jest gałęzią *)
6        begin STOS ⇐ {v, u}) DWUSP(u, v);
7          L[v] := min(L[v], L[u]);
8          if L[u] ⩾ WGN[v] then
             (* v jest korzeniem lub punktem artykulacji, a szczytowa partia
                stosu, aż do {v, u} włącznie, zawiera składową dwuspójną *)
9            begin (* wypisz krawędzie składowej dwuspójnej *)
10             repeat e ⇐ STOS; write(e)
11             until e = {v, u};
12             write (';') (* znak końca składowej dwuspójnej *)
13           end
14       end
15       else (* WGN[u] > 0, tzn. u był już odwiedzony *)
16       if (u ≠ p) and (WGN[u] < WGN[v]) then
           (* krawędź {v, u} jest cięciwą i nie występuje na stosie *)
17       begin STOS ⇐ {v, u},
18         L[v] := min(L[v], WGN[u])
19       end
```

```
20  end; (* DWUSP *)
21  begin (* program główny *)
22    for u ∈ V do WGN[u] := 0; (* inicjalizacja *)
23    STOS := Ø; num := 0;
24    for u ∈ V do
25      if WGN[u] = 0 then DWUSP(u, 0)
26  end
```

Pokażemy teraz, że wywołanie procedury $DWUSP(v, 0)$ (wiersz 25) dla jeszcze nie odwiedzonego wierzchołka v powoduje wyodrębnienie i wypisanie wszystkich bloków składowej spójnej grafu zawierającej wierzchołek v. Dowód przebiega przez indukcję względem liczby bloków tej składowej. Jeśli składowa ta nie zawiera punktów artykulacji, to nietrudno zauważyć, że wywołanie $DWUSP(v, 0)$ powoduje umieszczenie wszystkich krawędzi tej składowej na stosie, a następnie (por. pętla 10) wypisanie tych krawędzi. Załóżmy teraz, że nasza składowa zawiera $k > 1$ bloków, i że algorytm działa poprawnie dla dowolnej składowej zawierającej mniej niż k bloków. Rozpatrzmy pierwszą napotkaną krawędź $\{v, u\}$ taką, że $L[u] \geqslant WGN[v]$ w wierszu 8. Zgodnie z naszymi poprzednimi rozważaniami nierówność ta oznacza, że v jest korzeniem lub punktem artykulacji. Żaden z potomków wierzchołka v (w drzewie odpowiadającym przeszukiwaniu w głąb realizowanemu przez algorytm) nie jest punktem artykulacji, i w konsekwencji krawędzie szczytowej części stosu, aż do $\{v, u\}$ włącznie – będące krawędziami grafu łączącymi potomków wierzchołka v, łącznie z samym v – tworzą blok, który jest wypisywany w pętli 10. Zmodyfikujmy teraz naszą składową przez usunięcie powyższego bloku (bez usuwania wierzchołka v). Tak zmodyfikowana składowa ma $k-1$ bloków, i wykonanie dla niej procedury $DWUSP(v, 0)$ powoduje – na mocy założenia indukcyjnego – poprawne wypisanie tych bloków. Działanie algorytmu dla zmodyfikowanej składowej różni się od przypadku oryginalnej składowej jedynie tym, że dla pierwszego napotkanego punktu artykulacji (oryginalnej składowej) v pętla 4 nie jest wykonywana dla wierzchołków u należących do usuniętego bloku. Stąd łatwy wniosek, że $DWUSP(v, 0)$ poprawnie wyznacza wszystkie k bloków naszej składowej, a tym samym cały algorytm poprawnie wyznacza wszystkie bloki grafu.

Oszacujmy teraz złożoność obliczeniową algorytmu. Pętle 22 i 25 wykonują $O(n)$ kroków, przy czym w przypadku drugiej pętli nie liczymy kroków wykonanych przez wywołanie $DWUSP(u, 0)$ dla każdego nieodwiedzonego jeszcze wierzchołka. Wywołanie takie, dla składowej spójnej o n_i wierzchołkach i m_i krawędziach, wykonuje $O(n_i+m_i)$ kroków – nie licząc kroków w pętli 10 – jako że procedura taka przeszukuje składową w głąb wykonując liczbę kroków ograniczoną przez stałą dla każdej analizowanej krawędzi. Każda krawędź jest zdejmowana ze stosu i wypisywana dokładnie raz, co daje w sumie $O(m)$ kroków

konywanych przez pętlę 10 w czasie wykonywania całego algorytmu. Sumując wszystkie powyższe składniki otrzymujemy ostatecznie całkowitą złożoność algorytmu równą $O(n+m)$.

Drogi Eulera 2.7

Drogą Eulera w grafie $G = \langle V, E\rangle$ nazywamy dowolną drogę przechodzącą przez każdą krawędź grafu dokładnie raz, tzn. drogę $v_1, \ldots, v_{m+1}$ taką, że każda krawędź $e \in E$ występuje w ciągu $v_1, \ldots, v_{m+1}$ dokładnie raz jako $e = \{v_i, v_{i+1}\}$ dla pewnego i. Jeśli $v_1 = v_{m+1}$, to drogę taką nazywamy *cyklem Eulera*. Zagadnienie istnienia drogi Eulera w danym grafie było po raz pierwszy rozważane przez Eulera w roku 1736, i podany przez niego warunek konieczny i dostateczny istnienia takiej drogi (por. twierdzenie 2.12) uważany jest za historycznie pierwsze twierdzenie teorii grafów.

TWIERDZENIE 2.12

Droga Eulera w grafie istnieje wtedy i tylko wtedy gdy graf jest spójny i zawiera co najwyżej dwa wierzchołki stopnia nieparzystego.

DOWÓD

Dowód dostateczności warunku podanego w twierdzeniu otrzymamy jako wniosek z analizy algorytmu znajdowania drogi Eulera, który opiszemy w tym punkcie. Konieczność warunku jest oczywista, gdyż jeśli wierzchołek v różny od v_1 i v_{m+1} występuje w drodze Eulera $v_1, \ldots, v_{m+1}$ k razy, to oznacza to, iż stopień tego wierzchołka w grafie wynosi $2k$. Wynika stąd, że wierzchołki stopnia nieparzystego, jeśli istnieją, są końcami drogi Eulera. Warto tu zauważyć, że nie istnieje graf o tylko jednym wierzchołku stopnia nieparzystego. Istotnie, oznaczając stopień wierzchołka v przez $d(v)$ mamy

$$\sum_{v \in V} d(v) = 2m$$

jako że w powyższej sumie każda krawędź $\{u, v\}$ liczona jest dwukrotnie: raz w $d(u)$ i raz w $d(v)$. Wynika stąd, że liczba wierzchołków stopnia nieparzystego jest zawsze parzysta. ■

Jeśli w grafie spójnym nie ma wierzchołków stopnia nieparzystego, to każda droga Eulera jest cyklem, gdyż końce drogi Eulera nie będącej cyklem są zawsze wierzchołkami stopnia nieparzystego. Załóżmy, że u, v są jedynymi wierzchołkami stopnia nieparzystego grafu spójnego $G = \langle V, E\rangle$, i utwórzmy graf G^* przez dodanie dodatkowego wierzchołka t i krawędzi $\{u, t\}$, $\{v, t\}$ (lub po prostu $\{u, v\}$ jeśli $\{u, v\} \notin E$). Wtedy G^* jest grafem spójnym bez wierzchołków stopnia nieparzystego, a drogi Eulera w G są w oczywistej wzajemnie jednoznacznej odpowiedniości z cyklami Eulera w G^*. Z tego względu w pozostałej części tego punktu zajmować się będziemy wyłącznie cyklami Eulera.

ALGORYTM 2.13 (Znajdowanie cyklu Eulera)

Dane: Graf spójny $G = \langle V, E \rangle$ bez wierzchołków stopnia nieparzystego, reprezentowany przez listy incydencji $ZA[v]$, $v \in V$.

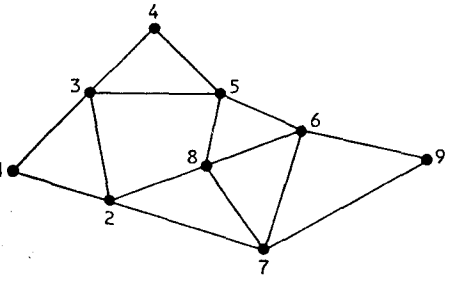
Wyniki: Cykl Eulera reprezentowany przez ciąg wierzchołków na stosie CE.

```
1  begin
2    STOS := Ø; CE := Ø;
3    v := dowolny wierzchołek grafu;
4    STOS ⇐ v;
5    while STOS ≠ Ø do
6      begin v := top(STOS); (* v = szczytowy element stosu *)
7        if ZA[v] ≠ Ø then
8          begin u := pierwszy wierzchołek listy ZA[v];
9            STOS ⇐ u;
             (* usuń krawędź {v, u} z grafu *)
10           ZA[v] := VA[v]\{u}; ZA[u] := ZA[u]\{v};
11           v := u
12         end
13       else (* ZA[v] = Ø *)
14         begin v ⇐ STOS; CE ⇐ v
15         end
16     end
17 end
```

Zasadę działania algorytmu można wyjaśnić w następujący sposób: Niech v_0 będzie wierzchołkiem wybranym w wierszu 3. Pętla 5 zaczyna konstruować drogę o początku w v_0, przy czym wierzchołki tej drogi kładzione są na *STOS*, a jej krawędzie usuwane są z grafu. Postępowanie to jest kontynuowane aż do momentu gdy nie jest możliwe rozszerzenie drogi o nowy wierzchołek, tzn. gdy $ZA[v] = \emptyset$ w wierszu 7. Zauważmy, że musi być wtedy $v = v_0$, gdyż w każdym innym przypadku oznaczałoby to, że stopień wierzchołka v jest nieparzysty. Tak więc z naszego grafu został usunięty cykl, a wierzchołki tego cyklu zostały umieszczone na stosie *STOS*. Zauważmy, że stopień dowolnego wierzchołka w tak zmodyfikowanym grafie pozostaje parzysty. Wierzchołek $v = v_0$ jest teraz przenoszony ze stosu *STOS* do stosu *CE*, a „bieżącym" wierzchołkiem v staje się szczytowy element stosu *STOS*. Z wierzchołka tego powtarzany jest analogiczny proces (jeśli $ZA[v] \neq \emptyset$), który wobec parzystości stopni wszystkich wierzchołków znajduje – i kładzie na *STOS* – pewien cykl przechodzący przez wierzchołek v. Wierzchołek ten jest następnie przenoszony do stosu *CE*. Postępowanie to jest kontynuowane aż do momentu, gdy *STOS* staje się pusty. Jest oczywiste, że wierzchołki umieszczane na stosie *CE* tworzą pewną drogę, przy czym wierzchołek v przenoszony jest na stos *CE* dopiero wtedy, gdy $ZA[v] = \emptyset$, tzn. gdy wszystkie krawędzie incydentne z tym wierzchołkiem reprezentowane są (przez

ry sąsiednich wierzchołków) na jednym ze stosów. Stąd już łatwo wynika, że zakończeniu działania algorytmu stos *CE* zawiera cykl Eulera.

Oszacujemy teraz złożoność obliczeniową naszego algorytmu. W tym celu uważmy, że każda iteracja głównej pętli (wiersz 5) albo umieszcza wierzchołek stosie *STOS* i usuwa krawędź z grafu, albo przenosi wierzchołek ze stosu *STOS* stosu *CE*. Tak więc liczba iteracji tej pętli jest O (*m*). Z kolei liczba kroków każdej iteracji jest ograniczona przez stałą. Zakładamy tu, że każda z list ycydencji $ZA[v]$, $v \in V$ jest zrealizowana w ten sposób, że każdy wierzchołek tej liście zawiera wskaźnik do poprzedniego i następnego wierzchołka oraz erzchołek u na liście $ZA[v]$ zawiera wskaźnik do wierzchołka v na liście



1, 2, 3, 4, 5, 6, 7, 2, 8, 6, 9, 7, 8, 5, 3, 1

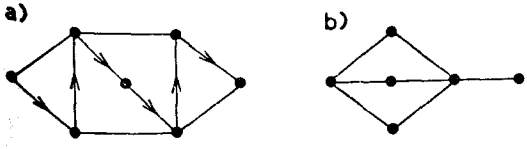
Rys. 2.9 Graf i cykl Eulera w tym grafie znaleziony przez algorytm 2.13

$ZA[u]$. Umożliwia to usunięcie krawędzi $\{v, u\}$ (wiersz 10) w czasie ograniczonym przez stałą (por. p. 2.1). Z powyższych rozważań wnioskujemy, że całkowita złożoność algorytmu jest O (*m*). Oczywiście algorytm ten jest optymalny, jako że samo wypisanie cyklu Eulera wymaga $\Omega(m)$ kroków.

Na rysunku 2.9 przedstawiono graf i pewien cykl Eulera znaleziony przez algorytm 2.13.

Algorytmy z powracaniem 2.8

Rozważmy teraz problem podobny do rozwiązanego w poprzednim paragrafie, z tą jednak różnicą, że tym razem interesować nas będą drogi przechodzące dokładnie raz przez każdy wierzchołek — a nie każdą krawędź — danego grafu. Ta niewielka, zdawać by się mogło, modyfikacja powoduje, jak się przekonamy, znaczną zmianę charakteru problemu. Przypomnijmy, że drogę o powyższej własności nazywamy drogą Hamiltona (cykl Hamiltona definiujemy w oczywisty sposób). Przykład drogi Hamiltona w grafie oraz grafu, w którym taka droga nie istnieje pokazano na rys. 2.10.



Rys. 2.10 (a) Droga Hamiltona w grafie (b) Graf, w którym nie istnieje droga Hamiltona

W przeciwieństwie do dróg Eulera, nie jest znany żaden prosty warunek konieczny i dostateczny dla istnienia drogi Hamiltona – mimo iż problem ten jest jednym z centralnych zagadnień teorii grafów. Nie jest również znany żaden algorytm, który sprawdzałby istnienie drogi Hamiltona w dowolnym grafie używając liczby kroków ograniczonej przez wielomian zmiennej n (liczby wierzchołków grafu). Są pewne podstawy – lecz brak dowodu matematycznego – na to, by przypuszczać, że ten stan rzeczy jest spowodowany nie tyle naszą niewiedzą, co po prostu faktem, że taki algorytm nie istnieje. Problem istnienia drogi Hamiltona należy bowiem do klasy tzw. *problemów NP-zupełnych* [27]. Szczegółowe omówienie tych problemów wykracza poza ramy tej książki. Wspomnimy tu tylko, że jest to szeroka klasa problemów – często fundamentalnych dla teorii grafów, logiki, teorii liczb, optymalizacji dyskretnej i innych dziedzin – dla żadnego z których nie jest znany algorytm wielomianowy (tzn. o liczbie kroków ograniczonej przez pewien wielomian od wymiaru problemu), przy czym istnienie algorytmu wielomianowego dla chociaż jednego z nich pociągałoby automatycznie istnienie algorytmów wielomianowych dla wszystkich tych problemów. Właśnie fakt fundamentalności wielu problemów *NP*-zupełnych w różnych dziedzinach i to, że mimo niezależnych wysiłków specjalistów w tych dziedzinach nie udało się znaleźć algorytmu wielomianowego dla żadnego z tych problemów skłania do przypuszczenia, że taki algorytm nie istnieje.

Powróćmy do konkretnego problemu istnienia drogi Hamiltona. Oczywisty algorytm, który możemy zastosować, to „pełny przegląd wszystkich możliwości": generujemy wszystkie $n!$ różnowartościowych ciągów wierzchołków, i dla każdego z nich sprawdzamy, czy określa drogę Hamiltona. Postępowanie takie wymaga co najmniej $n!\,n$ kroków, która to funkcja rośnie szybciej niż dowolny wielomian, a nawet szybciej niż dowolna funkcja eksponencjalna postaci a^n, $a > 1$, jako że

$$n!\,n > \lfloor n/2 \rfloor^{\lfloor n/2 \rfloor} = a^{\lfloor n/2 \rfloor \log_a \lfloor n/2 \rfloor}$$

Opiszemy teraz ogólną technikę pozwalającą na znaczną redukcję liczby kroków w algorytmach typu pełnego przeglądu wszystkich możliwości. Aby tę technikę zastosować szukane rozwiązanie musi mieć postać ciągu $\langle x_1, \ldots, x_n \rangle$. Główna idea metody polega na tym, że nasze rozwiązanie konstruujemy stopniowo, rozpoczynając od ciągu pustego ε (długości 0). Ogólnie, mając dane rozwiązanie częściowe $\langle x_1, \ldots, x_i \rangle$ staramy się znaleźć dopuszczalną wartość x_{i+1}, co do której nie możemy od razu wykluczyć, że $\langle x_1, \ldots, x_i, x_{i+1} \rangle$ daje się rozszerzyć do pewnego rozwiązania (lub już nim jest). Jeśli taka dopuszczalna, jeszcze nie wykorzystana wartość x_{i+1} istnieje, to dodajemy ją do naszego rozwiązania częściowego i kontynuujemy nasz proces dla ciągu $\langle x_1, \ldots, x_i, x_{i+1} \rangle$. Jeśli nie, to powracamy do rozwiązania częściowego $\langle x_1, \ldots, x_{i-1} \rangle$ i kontynuujemy nasz proces poszukując nowej jeszcze nie wykorzystanej dopuszczalnej wartości x_i' – stąd nazwa „algorytm z powracaniem" (ang. backtracking).

Mówiąc dokładniej zakładamy, że dla każdego $k > 0$ jest dany pewien ustalony zbiór A_k, z którego wybierać będziemy kandydatów na k-tą współrzędną

związania częściowego. Oczywiście zbiory A_k muszą być tak określone, by dla każdego rozwiązania całkowitego $\langle x_1, \ldots, x_n \rangle$ naszego problemu oraz dla każdego $k \leqslant n$ zbiór A_k zawierał element x_k (w praktyce nie możemy na ogół wyuczyć sytuacji, w której zbiór A_k zawiera pewne „zbędne" elementy, nie występujące na k-tej współrzędnej żadnego rozwiązania całkowitego). Zakładamy również istnienie pewnej prostej funkcji, która dowolnemu rozwiązaniu częściowemu $\langle x_1, \ldots, x_i \rangle$ przyporządkowuje wartość $P(x_1, \ldots, x_i)$ (**true** lub **false**) tak, że jeśli $P(x_1, \ldots, x_i) =$ **false**, to ciągu $\langle x_1, \ldots, x_i \rangle$ na pewno nie da się rozszerzyć do rozwiązania. Jeśli $P(x_1, \ldots, x_i) =$ **true**, to mówimy, że wartość x_i jest *dopuszczalna* (dla rozwiązania częściowego $\langle x_1, \ldots, x_{i-1} \rangle$) — nie oznacza to bynajmniej, że $\langle x_1, \ldots, x_{i-1} \rangle$ na pewno rozszerza się do pełnego rozwiązania. Proces ten możemy zapisać w postaci następującego schematu:

```
1   begin
2     k := 1;
3     while k > 0 do
4       if istnieje jeszcze niewykorzystany element y ∈ A_k
5           taki, że P(X[1], ..., X[k−1], y) then
6         begin X[k] := y; (* element y został wykorzystany *)
7           if ⟨X[1], ..., X[k]⟩ jest rozwiązaniem całkowitym then
8             write (X[1], ..., X[k]);
9           k := k+1
10        end
11      else (* powrót do krótszego rozwiązania częściowego; wszystkie elementy zbioru A_k stają się znowu niewykorzystane *)
12        k := k−1
13  end
```

Algorytm ten znajduje wszystkie rozwiązania, jeśli tylko założyć, że zbiory A_k są skończone i istnieje n takie, że $P(x_1, \ldots, x_n) =$ **false** dla wszystkich $x_1 \in A_1, \ldots, x_n \in A_n$ (ten ostatni warunek oznacza, że wszystkie rozwiązania są długości mniejszej od n). Pokażemy nieco ogólniejszą własność:

Niech $s > 0$ i niech $\langle x_1, \ldots, x_{s-1} \rangle$ będzie pewnym rozwiązaniem częściowym skonstruowanym przez algorytm. Rozważmy pierwszą iterację pętli 3, dla której $k = s$, $X[i] = x_i$, $1 \leqslant i \leqslant s$. Począwszy od tej iteracji algorytm generuje wszystkie rozwiązania całkowite będące rozszerzeniem ciągu $\langle x_1, \ldots, x_{s-1} \rangle$ i dochodzi do stanu, w którym $k = s-1$.

Oczywiście dla $s = 1$ własność powyższa wyraża po prostu poprawność algorytmu. Ogólnej postaci tej własności dowodzimy przez indukcję („wstecz") względem s. Jest ona prawdziwa dla $s = n$, gdyż nie istnieją żadne elementy dopuszczalne dla $\langle x_1, \ldots, x_{s-1} \rangle$, i wykonuje się od razu drugi człon instrukcji warunkowej w wierszu 11, tzn. zmienna k przybiera wartość $s-1$. Załóżmy teraz prawdziwość naszej własności dla pewnego $s > 1$. Pokażemy jej prawdziwość dla $s-1$. Niech $\langle x_1, \ldots, x_{s-2} \rangle$ będzie dowolnym rozwiązaniem częściowym skon-

struowanym przez algorytm i rozważmy pierwszą iterację pętli 3, dla której $k = s-1$, $X[i] = x_i$, $1 \leqslant i \leqslant s-1$. Jeśli istnieje element y dopuszczalny dla $\langle x_1, \ldots, x_{s-2} \rangle$, to konstruowane jest rozwiązanie częściowe $\langle x_1, \ldots, x_{s-2}, y \rangle$ (wiersz 6) i zmienna k przybiera wartość s (wiersz 9). Następnie, zgodnie z naszym założeniem indukcyjnym, następuje wygenerowanie wszystkich rozwiązań będących rozszerzeniem ciągu $\langle x_1, \ldots, x_{s-2}, y \rangle$ i dochodzimy do stanu, w którym $k = s-1$, po czym proces powtarzany jest dla następnego niewykorzystanego elementu y dopuszczalnego dla $\langle x_1, \ldots, x_{s-2} \rangle$, aż do momentu wyczerpania wszystkich takich elementów (sytuacja taka może wystąpić od razu na samym początku). Zmienna k zmniejszana jest wtedy o 1 i osiąga wartość $s-2$ (wiersz 12). Z powyższych rozważań wynika, że algorytm poprawnie generuje wszystkie rozwiązania będące rozszerzeniem ciągu $\langle x_1, \ldots, x_{s-2} \rangle$, co kończy dowód kroku indukcyjnego.

Udowodniona przez nas własność bezpośrednio prowadzi do następującej prostej i przejrzystej wersji rekurencyjnej schematu algorytmu z powracaniem:

```
1  procedure AP(k);
   (* generowanie wszystkich rozwiązań będących rozszerzeniem ciągu
      X[1], ..., X[k-1]; tablica X jest globalna *)
2  begin
3    for y ∈ A_k takich, że P(X[1], ..., X[k-1], y) do
4      begin X[k] := y;
5        if X[1], ..., X[k] jest rozwiązaniem całkowitym then
6          write (X[1], ..., X[k]);
7          AP(k+1)
8      end
9  end
```

Wygenerowanie wszystkich rozwiązań całkowitych możemy spowodować przez wywołanie $AP(1)$. Prezentację algorytmu z powracaniem rozpoczęliśmy od nieco bardziej skomplikowanej nierekurencyjnej wersji jedynie dlatego, że w wersji rekurencyjnej „powracanie" nie występuje w jawnej postaci, będąc częścią implementacji mechanizmu rekursji.

Zastosujemy teraz algorytm z powracaniem do znajdowania cykli Hamiltona w grafie $G = \langle V, E \rangle$. Każdy taki cykl możemy reprezentować przez ciąg $\langle x_1, x_2, \ldots, x_{n+1} \rangle$, przy czym $x_1 = x_{n+1} = v_0$, gdzie v_0 jest pewnym ustalonym wierzchołkiem grafu, $x_i - x_{i+1}$ dla $1 \leqslant i \leqslant n$, oraz $x_i \neq x_j$ dla $1 \leqslant i < j \leqslant n$. Zgodnie z tymi warunkami możemy przyjąć:

$$A_k = V$$

$$P(x_1, \ldots, x_{k-1}, y) \Leftrightarrow y \in ZA[x_{k-1}] \wedge y \notin \{x_1, \ldots, x_{k-1}\}$$

ALGORYTM 2.14 (Znajdowanie wszystkich cykli Hamiltona w grafie)

Dane: Graf $G = \langle V, E \rangle$ reprezentowany przez listy incydencji $ZA[v]$, $v \in V$.

Wyniki: Lista wszystkich cykli Hamiltona grafu G.

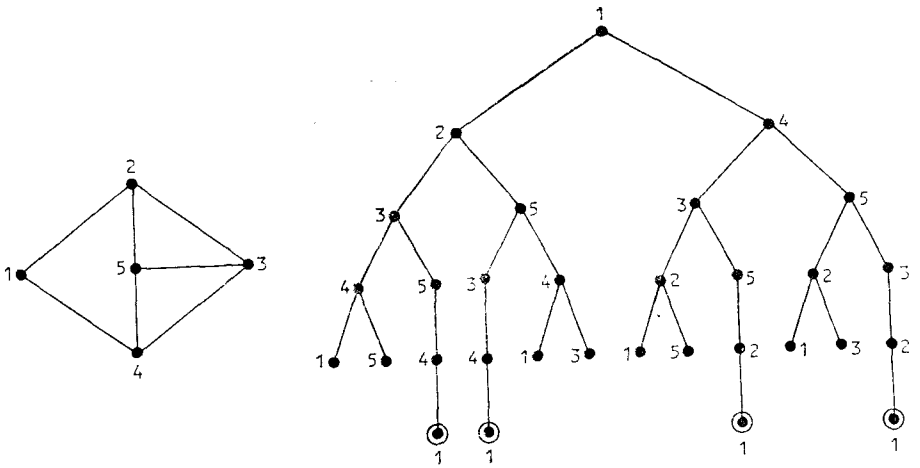
```
1  procedure HAMILT(k);
   (* generowanie wszystkich cykli Hamiltona będących rozszerzeniem ciągu
      ⟨X[1], ..., X[k−1]⟩; tablica X jest globalna *)
2  begin
3    for y ∈ ZA[X[k−1]] do
4      if (k = n+1) and (y = v0) then write (X[1], ..., X[n], v0)
5      else if DOP[y] then
6        begin X[k] := y; DOP[y] := false;
7          HAMILT(k+1);
8          DOP[y] := true
9        end
10 end; (* HAMILT *)
11 begin (* program główny *)
12   for v ∈ V do DOP[v] := true; (* inicjalizacja *)
13   X[1] := v0; (* v0 = dowolny ustalony wierzchołek grafu *)
14   DOP[v0] := false;
15   HAMILT(2)
16 end
```

Działanie tego, jak również dowolnego algorytmu z powracaniem, można zilustrować jako proces przeszukiwania pewnego drzewa. Każdy wierzchołek drzewa odpowiada w naturalny sposób pewnemu ciągowi $\langle x_1, \ldots, x_k \rangle$, przy czym wierzchołki odpowiadające ciągom postaci $\langle x_1, \ldots, x_k, y \rangle$ są synami tego wierzchołka (korzeń odpowiada ciągowi pustemu ε). Rozważmy „pełne" drzewo D złożone ze wszystkich możliwych ciągów postaci $\langle x_1, \ldots, x_k \rangle$, gdzie $0 \leqslant k \leqslant n$ i $x_i \in A_i$ dla $1 \leqslant i \leqslant k$, oraz załóżmy chwilowo, że każdy element $y \in A_k$ jest dopuszczalny dla $\langle x_1, \ldots, x_{k-1} \rangle$, jeśli $k \leqslant n$, oraz żaden element nie jest dopuszczalny dla $\langle x_1, \ldots, x_{k-1} \rangle$, jeśli $k > n$; innymi słowy

$$P(x_1, \ldots, x_{k-1}, x_k) = \begin{cases} \textbf{true} & \text{jeśli } k \leqslant n \\ \textbf{false} & \text{jeśli } k > n \end{cases}$$

($x_i \in A_i$ dla $1 \leqslant i \leqslant k$). Nietrudno wtedy zauważyć, że wywołanie $AP(1)$ spowoduje przeszukanie w głąb całego drzewa D (poczynając od korzenia ε). W przypadku mniej trywialnej funkcji P określającej dopuszczalność wierzchołków proces przeszukiwania „opuszcza" rozpatrywanie wierzchołków poddrzewa o korzeniu w każdym napotkanym wierzchołku „niedopuszczalnym" (tzn. odpowiadającym ciągowi $\langle x_1, \ldots, x_k \rangle$, dla którego $P(x_1, \ldots, x_k) = \textbf{false}$). Na tym właśnie polega efektywność algorytmu z powracaniem w stosunku do pełnego przeglądu wszystkich możliwości. Zilustrowano to, dla algorytmu 2.14, na rysunku 2.11.

Należy zaznaczyć, że w większości zastosowań liczba kroków algorytmu z powracaniem, choć mniejsza na ogół niż dla pełnego przeglądu, rośnie jednak w najgorszym przypadku wykładniczo ze wzrostem wymiaru problemu. Tak jest nawet w przypadku, gdy poszukujemy tylko jednego, a nie wszystkich rozwiązań



Rys. 2.11 Graf i drzewo ilustrujące algorytm z powracaniem znajdowania wszystkich cykli Hamiltona w tym grafie

(wtedy po prostu przerywamy wykonywanie algorytmu po osiągnięciu pierwszego rozwiązania; zauważmy, że gdy problem nie ma żadnych rozwiązań, modyfikacja ta nie ma wpływu na przebieg wykonywania algorytmu).

Zadania

2

2.1 Udowodnij, że dla dowolnego grafu niezorientowanego macierz sąsiedztwa wierzchołków B wyraża się przez macierz incydencji A w następujący sposób:
$B = AA^T - diag\,[d_1, \ldots, d_n]$
gdzie A^T jest macierzą transponowaną względem A, d_i jest stopniem i-tego wierzchołka, $diag\,[d_1, \ldots, d_n]$ zaś jest macierzą wymiaru $n \times n$ o elementach $d_1, \ldots, d_n$ na głównej przekątnej i zerach poza główną przekątną.

2.2 Określmy dla dowolnego grafu niezorientowanego *macierz sąsiedztwa krawędzi* $C = [c_{ij}]$, gdzie $c_{ij} = 1$, jeśli i-ta i j-ta krawędź są incydentne ze wspólnym wierzchołkiem, lub $c_{ij} = 0$ w przeciwnym przypadku (przyjmujemy $c_{ii} = 0$). Jak można wyrazić macierz C przez macierz incydencji? Czy macierz sąsiedztwa krawędzi wyznacza graf z dokładnością do izomorfizmu?

2.3 Podaj szczegółową realizację list incydencji i procedur usuwania i dodawania krawędzi w czasie ograniczonym przez stałą (należy pamiętać, że przy usuwaniu krawędzi $\{u, v\}$ trzeba usunąć odpowiednie elementy z obu list $ZA\,[u]$ i $ZA\,[v]$ – zakładamy, że wskaźnik do jednego z tych elementów jest dany jako parametr procedury).

2.4 Napisz procedury przejścia pomiędzy każdą parą sposobów reprezentacji grafu opisanych w punkcie 2.1 oraz oszacuj złożoność obliczeniową zaproponowanych procedur.

2.5 Niektóre algorytmy grafowe używają jedynie $O(n)$, lub przynajmniej

$O(n+m)$, spośród n^2 elementów macierzy sąsiedztwa wierzchołków B. Jednakże na początku algorytmu zwykle następuje inicjalizacja wszystkich elementów macierzy B, przez co złożoność algorytmu jest automatycznie $\Omega(n^2)$. Podaj sposób uniknięcia tej kosztownej inicjalizacji. (Wskazówka: Dla każdego elementu b_{ij} użytego po raz pierwszy przez algorytm umieszczamy na stosie wskaźnik do tego elementu, oraz umieszczamy przy b_{ij} wskaźnik do w ten sposób nowo utworzonego elementu stosu. Aby sprawdzić, czy dany element b_{ij} był już inicjalizowany wystarczy zbadać, czy wskaźnik przy b_{ij} prowadzi do wskaźnika na stosie określającego powrót do b_{ij}) [1].

2.6 Podaj algorytm o złożoności $O(n)$ badania czy dany graf zorientowany reprezentowany przez macierz sąsiedztwa wierzchołków zawiera wierzchołek, do którego dochodzą krawędzie ze wszystkich pozostałych $n-1$ wierzchołków, i od którego nie odchodzi żadna krawędź (algorytm taki pokazuje, że warunek (c) jest istotny w twierdzeniu 2.1).

2.7 Zbadaj technikę różniącą się od przeszukiwania wszerz jedynie tym, że nowo osiągnięte wierzchołki umieszczane są na stosie (algorytm taki opisany jest dokładnie przez procedurę *WS*, w której *KOLEJKA* została zastąpiona przez *STOS*) [35].

2.8 Zbadaj technikę przeszukiwania grafu, w której odwiedzone, lecz jeszcze niewykorzystane wierzchołki gromadzone są w kolejce, i którą dokładniej opisać można w następujący sposób: Badamy ostatni (najpóźniej umieszczony) wierzchołek v kolejki (początkowo jest nim dowolny wierzchołek grafu). Jeśli istnieje nowy wierzchołek $u \in ZA[v]$, to umieszczamy go na końcu kolejki i powtarzamy powyższy proces aż do momentu gdy umieszczony w kolejce wierzchołek nie ma nowych sąsiadów (fragment ten przypomina przeszukiwanie w głąb). Następnie badamy pierwszy (najwcześniej umieszczony) wierzchołek kolejki. Jeśli istnieje nowy sąsiad tego wierzchołka, to umieszczamy go na końcu kolejki i powtarzamy dla niego opisany powyżej proces typu przeszukiwania w głąb. W przeciwnym przypadku pierwszy wierzchołek kolejki jest wykorzystany i usuwamy go z kolejki, a następnie rozpatrujemy następny znajdujący się za nim wierzchołek. Cały proces kontynuowany jest aż do momentu, gdy kolejka staje się pusta. Podaj szczegółową implementację tej techniki.

2.9 Udowodnij, że poniższa procedura odwiedza każdy wierzchołek grafu spójnego dokładnie raz.

```
procedure PRZESZUKIWANIE(r);
(* początkowo STATUS[v] = nowy dla każdego v ∈ V *)
begin
  odwiedź r; STATUS[r] := odwiedzony;
  while istnieje w V wierzchołek odwiedzony do
    begin v := dowolny wierzchołek odwiedzony;
      if istnieje nowy u ∈ ZA[v] then
```

```
            begin u := pierwszy nowy wierzchołek listy ZA[v];
              odwiedź u; STATUS[u] := odwiedzony
            end
          else STATUS[v] := wykorzystany
        end
    end
```

Pokaż, że przeszukiwanie w głąb i wszerz jak również techniki opisane w zadaniach 2.7 i 2.8 można opisać jako szczególne przypadki tej procedury.

2.10 Niech $D = \langle V, T\rangle$ będzie podgrafem grafu spójnego $G = \langle V, E\rangle$. Udowodnij, że następujące warunki są równoważne:
(a) D jest drzewem rozpinającym grafu G.
(b) $|T| = |V|-1$ i D nie zawiera cykli.
(c) Dla każdej pary wierzchołków $u, v \in V$ istnieje w D dokładnie jedna droga z u do v.
(d) $|T| = |V|-1$ i dla każdej pary wierzchołków $u, v \in V$ istnieje w D co najmniej jedna droga z u do v.
(e) $|T| = |V|-1$ i dla każdej pary wierzchołków $u, v \in V$ istnieje w D co najwyżej jedna droga z u do v.

2.11 Zbadaj dokładniej technikę przeszukiwania w głąb dla grafów zorientowanych wspomnianą na końcu punktu 2.2. Pokaż, że jeśli w grafie zorientowanym G istnieje droga z wierzchołka r do każdego innego wierzchołka, to algorytm 2.3 (zmodyfikowany tak, że w wierszu 5 dodawana jest krawędź zorientowana $\langle v, u\rangle$) konstruuje podgraf $\langle V, T\rangle$, który – po zignorowaniu orientacji krawędzi – jest drzewem rozpinającym grafu G. Pokaż, że w $\langle V, T\rangle$ istnieje dokładnie jedna droga z r do każdego wierzchołka $v \in V$. Czy twierdzenie 2.4 przenosi się na przypadek zorientowany? Jak należałoby je zmodyfikować?

2.12 Zbadaj dokładniej technikę przeszukiwania wszerz grafu zorientowanego. Pokaż, że jeśli w grafie zorientowanym G istnieje droga z wierzchołka r do każdego innego wierzchołka, to algorytm 2.5 znajduje podgraf $\langle V, T\rangle$ zawierający dokładnie jedną drogę z r do każdego innego wierzchołka, przy czym jest to najkrótsza droga w G.

2.13 Zastosuj przeszukiwanie w głąb do znajdowania w spójnym grafie niezorientowanym cyklu (o powtarzających się krawędziach i wierzchołkach), przechodzącego każdą krawędź w każdym kierunku dokładnie raz.

2.14 *Mostem* grafu G nazywamy każdą krawędź, której usunięcie powoduje zwiększenie liczby składowych spójnych grafu. Podaj algorytm znajdowania wszystkich mostów grafu w czasie $O(m+n)$. (Wskazówka: Zastosuj przeszukiwanie w głąb, w podobny sposób jak do znajdowania punktów artykulacji).

2.15 Graf $G = \langle V, E\rangle$ jest *dwudzielny*, jeśli istnieje podział $V = A \cup B$, $A \cap B = \emptyset$ taki, że każda krawędź $e \in E$ jest postaci $e = \{a, b\}$, $a \in A$, $b \in B$. Udowodnij, że graf jest dwudzielny wtedy i tylko wtedy, gdy nie zawiera

cykli elementarnych nieparzystej długości. Podaj dwa algorytmy testujące czy dany graf jest dwudzielny w czasie O $(m+n)$ – jeden oparty na przeszukiwaniu w głąb, a drugi na przeszukiwaniu wszerz.

2.16 Podaj algorytm testujący w czasie O $(m+n)$ acykliczność danego grafu zorientowanego.

2.17 Graf zorientowany nazywamy *silnie spójnym*, jeśli dla każdej pary wierzchołków u, v istnieje droga z u do v. Przez *składową silnie spójną* rozumiemy dowolny maksymalny podgraf silnie spójny. Pokaż, że składowe silnie spójne określają podział zbioru wierzchołków na rozłączne niepuste podzbiory. Podaj algorytm znajdowania składowych silnie spójnych w czasie O $(m+n)$. (Wskazówka: Zastosować przeszukiwanie w głąb).

2.18 Podaj algorytm znajdowania w czasie O $(c(m+n))$ wszystkich cykli elementarnych grafu zorientowanego, gdzie c jest liczbą cykli. (Wskazówka: Zastosuj przeszukiwanie w głąb).

2.19 Podaj warunek konieczny i dostateczny na istnienie cyklu Eulera w grafie zorientowanym (definiujemy go w identyczny sposób jak dla grafów niezorientowanych). Pokaż, że algorytm 2.13 (po usunięciu wiersza 9) poprawnie konstruuje cykl Eulera w przypadku zorientowanym.

2.20 *Cyklem de Bruijna rzędu n* nazywamy cykliczny ciąg binarny długości 2^n, w którym każdy z 2^n ciągów binarnych długości n występuje jako podciąg n kolejnych wyrazów. Udowodnij istnienie cyklu de Bruijna rzędu n dla każdego $n \geqslant 1$. (Wskazówka: Rozważ graf, którego wierzchołki odpowiadają ciągom binarnym długości $n-1$, krawędź natomiast każda odpowiada ciągowi binarnemu długości n, przy czym krawędź $b_1 \ldots b_n$ prowadzi z $b_1 \ldots b_{n-1}$ do $b_2 \ldots b_n$. Udowodnij istnienie drogi Eulera w takim grafie) [8].

2.21 Zmodyfikuj podany w punkcie 2.8 schemat algorytmu z powracaniem tak, by wyznaczał on jedno rozwiązanie zamiast wszystkich.

2.22 Udowodnij, poprzez wskazanie odpowiednich grafów, że liczba kroków wykonywanych (w najgorszym przypadku) przez algorytm 2.14 rośnie wykładniczo ze wzrostem n.

2.23 Zastosuj algorytm z powracaniem do rozwiązania następujących problemów:

(a) Znalezienie rozmieszczenia 8 wzajemnie nieatakujących się hetmanów na szachownicy.

(b) Znalezienie kliki o maksymalnej liczności w grafie niezorientowanym. (*Kliką* nazywamy dowolny podzbiór wierzchołków, w którym każda para różnych wierzchołków połączona jest krawędzią grafu).

(c) Znalezienie pokolorowania wierzchołków grafu minimalną liczbą kolorów tak, by żadna krawędź nie łączyła dwóch wierzchołków tego samego koloru.

(d) Dla danych liczb całkowitych $a_1, \ldots, a_n, b$ znalezienie zbioru indeksów $J \subseteq \{1, \ldots, n\}$ takiego, że $\sum_{j \in J} a_j = b$, jeśli taki zbiór J istnieje.

(e) Stwierdzenie czy dane dwa grafy są izomorficzne.

Znajdowanie najkrótszych dróg w grafie

Pojęcia wstępne

W rozdziale tym będziemy rozważać grafy zorientowane $G = \langle V, E \rangle$ o krawędziach z wagami. Oznacza to, że każdej krawędzi $\langle u, v \rangle \in E$ jest przyporządkowana pewna liczba rzeczywista $a(u, v)$ zwana *wagą* tej krawędzi. Przyjmujemy dodatkowo $a(u, v) = \infty$, jeśli $u \not\mapsto v$. Jeśli ciąg wierzchołków

$$v_0, v_1, \ldots, v_p \tag{3.1}$$

określa drogę w G, to jej *długość* definiujemy jako

$$\sum_{i=1}^{p} a(v_{i-1}, v_i)$$

(Zauważmy, że przyjmując w dowolnym grafie wagę każdej krawędzi równą jedności otrzymujemy zwykłą definicję długości drogi w sensie liczby krawędzi; podobnie jak poprzednio przyjmujemy, że długość drogi wynosi 0 dla $p = 0$). Interesować nas będzie znalezienie najkrótszej drogi między ustalonymi wierzchołkami $s, t \in V$. Długość takiej najkrótszej drogi będziemy oznaczać przez $d(s, t)$ i będziemy nazywać *odległością* od s do t (tak określona odległość może być ujemna). Jeśli nie istnieje żadna droga od s do t, to przyjmujemy $d(s, t) = \infty$. Zauważmy, że jeśli każdy cykl naszego grafu ma dodatnią długość, to najkrótsza droga jest zawsze drogą elementarną, tzn. w ciągu (3.1) nie ma powtórzeń. Z drugiej strony, jeśli istnieje w grafie cykl ujemnej długości, to odległość pomiędzy niektórymi parami wierzchołków jest nieokreślona, gdyż obchodząc ten cykl dostateczną liczbę razy możemy wskazać drogę pomiędzy tymi wierzchołkami o długości mniejszej od dowolnej liczby rzeczywistej. W takim przypadku można by pytać o długość najkrótszej drogi elementarnej, jednak tak postawiony problem jest prawdopodobnie znacznie trudniejszy, gdyż zawiera w sobie między innymi zagadnienie istnienia cyklu Hamiltona (por. zad. 3.1).

Można podać wiele interpretacji praktycznych problemu najkrótszych dróg. Wierzchołki mogą na przykład odpowiadać miastom, zaś każda krawędź pewnej drodze, której długość dana jest przez wagę tej krawędzi. Szukamy wtedy naj-

ótszego połączenia między miastami. Waga krawędzi może też odpowiadać sztowi (lub czasowi) przesłania informacji między wierzchołkami. W takim ypadku szukamy najtańszej (lub najszybszej) drogi przesłania informacji. szcze inną sytuację otrzymujemy, gdy waga krawędzi $\langle u, v\rangle$ jest równa prawdopodobieństwu $p(u, v)$ bezawaryjnej pracy kanału przesyłania informacji. Jeśli założymy, że awarie kanałów są od siebie niezależne, to prawdopodobieństwo sprawności drogi przesyłania informacji jest równe iloczynowi prawdopodobieństw odpowiadających jej krawędziom. Problem znajdowania najbardziej niezawodnej drogi można łatwo sprowadzić do zagadnienia najkrótszych dróg, przez zamianę wag $p(u, v)$ na

$$a(u, v) = -\log p(u, v)$$

szcze jedno zastosowanie znajdowania najkrótszych dróg poznamy w punkcie 3.4.

W rozdziale tym podawać będziemy algorytmy znajdowania odległości między wierzchołkami, a nie samych dróg. Mając jednak odległość możemy, przy założeniu dodatniej długości wszystkich cykli, łatwo wyznaczyć najkrótsze drogi. Wystarczy w tym celu zauważyć, że dla dowolnych $s, t \in V (s \neq t)$ istnieje wierzchołek v taki, że

$$d(s, t) = d(s, v) + a(v, t)$$

Istotnie, własność taką ma przedostatni wierzchołek dowolnej najkrótszej drogi z s do t. Z kolei możemy znaleźć wierzchołek u taki, że $d(s, v) = d(s, u) + a(u, v)$ itd. Z dodatniości długości wszystkich cykli łatwo wynika, że tak utworzony ciąg $t, v, u, \ldots$ nie zawiera powtórzeń i kończy się na wierzchołku s. Oczywiście określa on (po odwróceniu kolejności) najkrótszą drogę z s do t.

Otrzymujemy więc następujący algorytm:

ALGORYTM 3.1 (Znajdowanie najkrótszej drogi)

Dane: Odległości $D[v]$ od ustalonego wierzchołka s do wszystkich pozostałych wierzchołków $v \in V$,
ustalony wierzchołek t,
macierz wag krawędzi $A[u, v]$, $u, v \in V$.

Wyniki: $STOS$ zawiera ciąg wierzchołków określający najkrótszą drogę z s do t.

```
1  begin
2    STOS := Ø; STOS ⇐ t; v := t;
3    while v ≠ s do
4      begin
5        u := wierzchołek, dla którego D[v] = D[u] + A[u, v];
6        STOS ⇐ u;
7        v := u
8      end
9  end
```

Niech $\langle V, E\rangle$ będzie grafem zorientowanym, $|V| = n$, $|E| = m$. Jeśli wybranie wierzchołka u w wierszu 5 następuje przez przegląd wszystkich wierzchołków to złożoność naszego algorytmu jest $O(n^2)$. Jeśli przeszukujemy tylko listę $PRZED[v]$ zawierającą wszystkie wierzchołki u takie, że $u \to v$, to złożoność ta jest $O(m)$.

Zauważmy, że w przypadku dodatnich wag krawędzi problem najkrótszych dróg w grafie niezorientowanym łatwo sprowadzić do analogicznego problemu dla pewnego grafu zorientowanego. Wystarczy w tym celu każdą krawędź $\{u, v\}$ zastąpić przez dwie krawędzie $\langle u, v\rangle$, $\langle v, u\rangle$, każda o takiej samej wadze jak $\{u, v\}$. Jednakże w przypadku wag niedodatnich prowadzi to do powstania cykli o niedodatniej długości.

W rozdziale tym będziemy zawsze zakładać, że $G = \langle V, E\rangle$ jest grafem zorientowanym, $n = |V|$, $m = |E|$. Dla uproszczenia i uniknięcia zdegenerowanych przypadków przy szacowaniu złożoności algorytmów przyjmować będziemy, że $m = \Omega(n)$ (tzn. $n = O(m)$). Wyklucza to sytuacje, w których „większość" wierzchołków jest izolowanych. Będziemy również zakładać, że wagi krawędzi są pamiętane w tablicy $A[u, v]$, $u, v \in V$ ($A[u, v]$ zawiera wagę $a(u, v)$).

Najkrótsze drogi z ustalonego wierzchołka 3

Większość znanych algorytmów znajdowania odległości między dwoma ustalonymi wierzchołkami s, t jest oparta na postępowaniu, które w ogólnym zarysie można przedstawić następująco: Mając daną macierz wag krawędzi $A[u, v]$, $u, v \in V$ obliczamy pewne ograniczenia górne $D[v]$ na odległość od s do wszystkich wierzchołków $v \in V$. Za każdym razem, gdy stwierdzamy, że

$$D[u] + A[u, v] < D[v] \tag{3.2}$$

oszacowanie $D[v]$ polepszamy: $D[v] := D[u] + A[u, v]$.

Postępowanie przerywamy, gdy nie jest możliwa dalsza poprawa żadnego z ograniczeń $D[v]$. Można łatwo pokazać, że wartość każdej ze zmiennych $D[v]$ jest wtedy równa $d(s, v)$, odległości od s do v. Zauważmy, że aby wyznaczyć odległość od s do t obliczamy tu odległości od s do wszystkich wierzchołków grafu. Nie jest znany żaden algorytm znajdowania odległości między dwoma ustalonymi wierzchołkami, który byłby w istotny sposób efektywniejszy od znanych algorytmów wyznaczania odległości od ustalonego wierzchołka do wszystkich pozostałych.

Opisany przez nas ogólny schemat postępowania jest niepełny, gdyż nie określa kolejności w jakiej wybierane są wierzchołki u, v do sprawdzania warunku (3.2). Kolejność ta ma – jak się okaże – bardzo istotny wpływ na efektywność algorytmu. Opiszemy teraz bardziej szczegółowo metody znajdowania odległości z ustalonego wierzchołka zwanego *źródłem* – oznaczać go będziemy zawsze przez s – do wszystkich pozostałych wierzchołków grafu.

Podamy najpierw algorytm dla ogólnego przypadku, w którym zakładamy tylnie brak cykli o ujemnej długości. Z algorytmem tym wiąże się zwykle nazwiska L. R. Forda [22] i R. E. Bellmana [3].

GORYTM 3.2 (Znajdowanie odległości od źródła do wszystkich pozostałych rzchołków – metoda Forda-Bellmana)

ne: Graf zorientowany $\langle V, E\rangle$ o n wierzchołkach z wyróżnionym źródłem $s \in V$,
macierz wag krawędzi $A[u, v]$, $u, v \in V$
(graf nie zawiera cykli ujemnych długości).

yniki: Odległości od źródła do wszystkich wierzchołków grafu: $D[v] = d(s, v)$, $v \in V$.

```
begin
  for v ∈ V do D[v] := A[s, v]; D[s] := 0;
  for k := 1 to n−2 do
    for v ∈ V \ {s} do
      for u ∈ V do D[v] := min(D[v], D[u]+A[u, v])
end
```

.2 owodnimy poprawność tego algorytmu. W tym celu oznaczmy przez $d^{(m)}(v)$ gość najkrótszej spośród dróg z s do v zawierających co najwyżej m krawę- ($d^{(m)}(v) = \infty$, jeśli nie istnieje żadna taka droga). Mamy wtedy

$$d^{(m+1)}(v) = min\{d^{(m)}(u) + a(u, v): u \in V\}, v \in V \tag{3.3}$$

otnie, dla każdego $u \in V$ mamy oczywiście $d^{(m+1)}(v) \leqslant d^{(m)}(u) + a(u, v)$, przy m równość występuje, gdy u jest przedostatnim wierzchołkiem dowolnej jkrótszej drogi z s do v.

Pokażemy teraz, że jeśli przy wejściu do kolejnej iteracji pętli 3

$$d(s, v) \leqslant D[v] \leqslant d^{(m)}(v), \quad \text{dla wszystkich } v \in V, \tag{3.4}$$

po zakończeniu tej iteracji

$$d(s, v) \leqslant D[v] \leqslant d^{(m+1)}(v), \quad \text{dla wszystkich } v \in V \tag{3.5}$$

otnie, zakładając spełnienie warunku (3.4) i analizując działanie instrukcji wierszu 5 widzimy, że po zakończeniu naszej iteracji pętli 3 mamy

$$d(s, v) \leqslant D[v] \leqslant min\{d^{(m)}(u) + a(u, v): u \in V\}$$

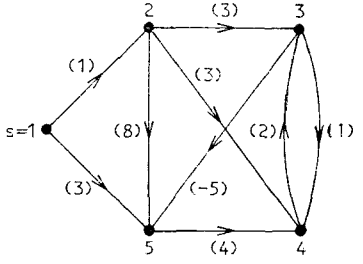
o po uwzględnieniu równości (3.3) daje warunek (3.5).

Zauważmy, że przy wejściu do pętli 3 mamy $D[v] = d^{(1)}(v)$, $v \in V$, zatem po wykonaniu $n-2$ iteracji tej pętli $d(s, v) \leqslant D[v] \leqslant d^{(n-1)}(v)$, $v \in V$. Wystarczy teraz pokazać, że $d^{(n-1)}(v) = d(s, v)$. Tak jest w istocie, gdyż każda droga o więcej niż $n-1$ krawędziach zawiera cykl, którego usunięcie nie zwiększa długości drogi (założyliśmy bowiem brak cykli ujemnej długości). Dowód poprawności algorytmu jest tym samym zakończony.

Oczywiste jest, że złożoność czasowa algorytmu jest O (n^3). Obliczenia możemy oczywiście zakończyć, gdy wykonanie pętli 4 nie powoduje zmiany żadnej spośród zmiennych $D[v]$, $v \in V$. Może to nastąpić dla $k < n-2$, jednak taka modyfikacja algorytmu nie zmienia w istotny sposób jego złożoności. W przypadku grafów rzadkich ($m \ll n^2$) wygodnie jest reprezentować graf przez listy incydencji $PRZED[v]$, $v \in V$. Po zamianie wiersza 5 na

for $u \in PRZED[v]$ **do** $D[v] := min(D[v], D[u] + A[u, v])$

otrzymujemy algorytm o złożoności O (nm).



$$A = \begin{bmatrix} 1 & \infty & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{bmatrix}$$

k	D[1]	D[2]	D[3]	D[4]	D[5]
	0	1	∞	∞	3
1	0	1	4	4	−1
2	0	1	4	3	−1
3	0	1	4	3	−1

Rys. 3.1 Działanie algorytmu Forda-Bellmana

Działanie algorytmu Forda-Bellmana zilustrowano na rys. 3.1 ($V = \{1, ..., 5\}$, wagi krawędzi są podane w nawiasach, pętle 4 oraz 5 są wykonywane w kolejności wzrastających numerów wierzchołków).

Przypadek nieujemnych wag — algorytm Dijkstry

W dwóch ważnych przypadkach, a mianowicie gdy wagi wszystkich krawędzi są nieujemne lub graf jest acykliczny, znane są efektywniejsze algorytmy. Opiszemy najpierw algorytm dla pierwszego przypadku pochodzący od Dijkstry [11]. Drugim przypadkiem zajmiemy się w następnym punkcie.

ALGORYTM 3.3 (Znajdowanie odległości od źródła do wszystkich pozostałych wierzchołków w grafie o nieujemnych wagach krawędzi – metoda Dijkstry)

Dane: Graf zorientowany $\langle V, E \rangle$ o wyróżnionym źródle $s \in V$, macierz wag krawędzi $A[u, v]$, $u, v \in V$
(wszystkie wagi są nieujemne)

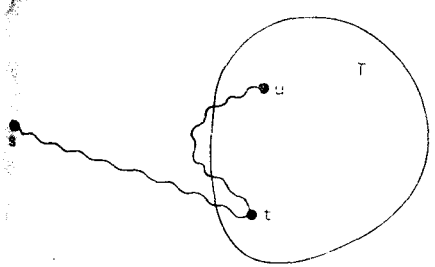
Wyniki: Odległość od źródła do wszystkich wierzchołków grafu: $D[v] = d(s, v)$, $v \in V$.

```
begin
  for v ∈ V do D[v] := A[s, v]; D[s] := 0;
  T := V \ {s};
  while T ≠ Ø do
    begin
      u := dowolny wierzchołek r ∈ T taki, że
                                  D[r] = min{D[p]: p ∈ T};
      T := T \ {u};
      for v ∈ T do D[v] := min(D[v], D[u] + A[u, v])
    end
end
```

Aby zrozumieć działanie algorytmu pokażemy, że następujący warunek jest niezmiennikiem pętli 4:

$$\begin{aligned} &\text{dla każdego } v \in V \setminus T \quad D[v] = d(s, v), \\ &\text{dla każdego } v \in T \quad D[v] = \text{długość najkrótszej spośród tych dróg z } s \text{ do } v, \text{ dla których przedostatni wierzchołek należy do zbioru } V \setminus T. \end{aligned} \tag{3.6}$$

Istotnie, w wierszu 6 znajdujemy wierzchołek $u \in T$ taki, że wartość $D[u]$ jest minimalną (spośród wszystkich) wartością $D[t]$, dla $t \in T$. Pokażemy, że $D[u] = d(s, u)$. Tak jest w istocie, gdyż jeśli najkrótsza droga z s do u ma długość mniejszą od $D[u]$, to na mocy drugiej części warunku (3.6) jej przedostatni



Rys. 3.2 Uzasadnienie algorytmu Dijkstry

wierzchołek należy do zbioru T. Niech t będzie pierwszym wierzchołkiem drogi należącym do zbioru T. Odcinek początkowy z s do t naszej drogi stanowi najkrótszą drogę z s do t, przy czym jej przedostatni wierzchołek nie należy do zbioru T. Wobec drugiej części warunku (3.6) mamy $D[t] = d(s, t)$. Korzystając z założenia o nieujemności wag otrzymujemy

$$D[t] = d(s, t) \leqslant d(s, u) < D[u]$$

wbrew zasadzie, zgodnie z którą został wybrany wierzchołek u.

Tak więc $D[u] = d(s, u)$ i możemy, w wierszu 7, usunąć u ze zbioru T bez naruszenia pierwszej części warunku (3.6). Aby zapewnić spełnienie również drugiej części tego warunku, należy jeszcze sprawdzić drogi z s do $v \in T$, których przedostatnim wierzchołkiem jest u, i dokonać odpowiedniej aktualizacji zmiennych $D[v]$, $v \in T$. Tego właśnie dokonuje pętla 8.

Warunek (3.6) jest oczywiście spełniony przy wejściu do pętli 4. Po zakończeniu działania algorytmu mamy $T = \emptyset$, a więc zgodnie z warunkiem (3.6), $D[v] = d(s, v)$, $v \in V$.

Przechodzimy teraz do oszacowania złożoności algorytmu Dijkstry. Pętla 4 jest wykonywana $n-1$ razy, przy czym każde jej wykonanie wymaga $O(n)$ kroków: $O(n)$ kroków dla znalezienia wierzchołka u w wierszu 6 (zakładamy że zbiór T jest reprezentowany jako lista) oraz $O(n)$ kroków dla wykonania pętli 8. Złożoność algorytmu jest więc $O(n^2)$.

Przez staranniejszy dobór struktur danych można otrzymać wersję o złożoności $O(m \log n)$. Należy w tym celu zbiór T reprezentować przez drzewo binarne o wysokości $O(\log n)$ i tej własności że dla dowolnych jego wierzchołków u, v

$$\text{jeśli } u \text{ jest synem } v, \text{ to } D[u] \leqslant D[v] \tag{3.7}$$

(podobną strukturę danych stosuje się w algorytmie sortowania Heapsort, por. poz. [21], [1]). Wierzchołek u, dla którego wartość $D[u]$ jest minimalna, jest wtedy korzeniem drzewa. Korzeń ten możemy usunąć w $O(\log n)$ krokach zachowując własność zmniejszania się wartości $D[j]$ na każdej drodze do korzenia. Wystarczy przesunąć na miejsce korzenia jego syna s o większej (lub równej) wartości $D[j]$, następnie na powstałe miejsce po s przesunąć syna o większej wartości $D[j]$ itd. Jeśli graf reprezentować przez listy $ZA[u]$, $u \in V$, to wiersz 8 możemy zamienić na

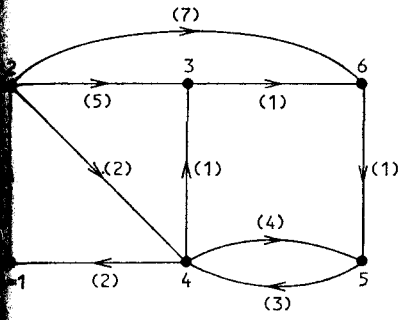
```
for v ∈ ZA[u] do
  if D[u]+A[u, v] < D[v] then
    begin
      D[v] := D[u]+A[u, v];
      przesuń wierzchołek v drzewie w kierunku korzenia tak, by zachować
      warunek (3.7)
    end
```

Jeśli założymy istnienie tablicy wskaźników do wierzchołków naszego drzewa, to przesunięcie wierzchołka v, o którym mowa w tym fragmencie może być dokonane w $O(\log n)$ krokach. Wystarczy zamienić kolejno v z wierzchołkami znajdującymi się bezpośrednio nad nim.

Przy takim zmodyfikowanym algorytmie każda krawędź grafu jest analizowana dokładnie raz, przy czym związane jest z tym $O(\log n)$ kroków na przesunięcie odpowiedniego wierzchołka w drzewie reprezentującym zbiór T. Daje

w sumie $O(m \log n)$ kroków. Do tego dochodzi $O(n \log n)$ kroków potrzebnych dla zbudowania naszego drzewa i dla usunięcia z niego $n-1$ razy korzenia. Ostatkowita złożoność algorytmu $O(m \log n)$ (por. zad. 3.6).

Nie wiadomo, czy istnieje algorytm, o złożoności $O(m)$ znajdowania odległości od ustalonego wierzchołka do wszystkich pozostałych wierzchołków grafu o nieujemnych wagach wszystkich krawędzi. Można pokazać, że istnieje stała C taka, że zagadnienie to może dla dowolnego $k > 0$ być rozwiązane w czasie $Ck(m+n^{1+1/k})$ (por. poz. [37]).



D[1]	D[2]	D[3]	D[4]	D[5]	D[6]
0	①	∞	∞	∞	∞
0	1	⑥	③	∞	⑧
0	1	④	3	⑦	⑧
0	1	4	3	⑦	⑤
0	1	4	3	⑥	5

Rys. 3.3 Działanie algorytmu Dijkstry

Działanie algorytmu Dijkstry zilustrowano na rys. 3.3 ($V = \{1, \ldots, 6\}$, wagi krawędzi są podane w nawiasach, wartości $D[v]$, $v \in T$ są oznaczone kółkami, minimalna z nich kółkiem podwójnym).

Drogi w grafie acyklicznym 3.4

Zajmiemy się teraz drugim przypadkiem, w którym jest znany algorytm znajdowania odległości od ustalonego wierzchołka w czasie $O(n^2)$ – a mianowicie przypadkiem, gdy graf jest acykliczny (wagi krawędzi mogą być dowolne). Udowodnimy najpierw następujący lemat:

LEMAT 3.4

W dowolnym grafie acyklicznym możemy tak ponumerować wierzchołki, by każda krawędź była postaci $\langle v_i, v_j \rangle$, gdzie $i < j$. ■

Przykład takiej numeracji pokazano na rys. 3.4.

Dla dowodu pokażemy algorytm, który taką numerację konstruuje.

ALGORYTM 3.5 (Numerowanie wierzchołków grafu acyklicznego)

Dane: Graf zorientowany acykliczny $\langle V, E \rangle$ określony przez listy incydencji $ZA[v]$, $v \in V$.

Wyniki: Dla każdego wierzchołka $v \in V$ numer $NR[v]$ taki, że dla dowolnej krawędzi $\langle u, v \rangle \in E$ zachodzi $NR[u] < NR[v]$.

```
1  begin
2    for v ∈ V do LIDO[v] := 0;
3    for u ∈ V do
4      for v ∈ ZA[u] do LIDO[v] := LIDO[v]+1;
     (* LIDO[v] = liczba krawędzi dochodzących do v *)
5    STOS := Ø;
6    for v ∈ V do
7      if LIDO[v] = 0 then STOS ⇐ v;
8    num := 0;
9    while STOS ≠ Ø do
10     begin u ⇐ STOS;
11       num := num+1; NR[u] := num;
12       for v ∈ ZA[u] do
13         begin LIDO[v] := LIDO[v]−1;
14           if LIDO[v] = 0 then STOS ⇐ v
15         end
16     end
17 end
```

Algorytm jest oparty na następującym prostym fakcie: W dowolnym (niepustym) grafie acyklicznym istnieje wierzchołek, do którego nie dochodzi żadna krawędź. Aby się o tym przekonać wybierzmy dowolny wierzchołek w_1 grafu, następnie pewien wierzchołek w_2 taki, że $w_2 \to w_1$, następnie wierzchołek w_3 taki, że $w_3 \to w_2$ itd. Po skończonej liczbie kroków musimy w ten sposób dojść do pewnego wierzchołka w_i, do którego nie dochodzi żadna krawędź, jako że na mocy acykliczności żaden wierzchołek nie może się powtarzać w ciągu $w_1, w_2, w_3, \ldots$.

W naszym algorytmie wierzchołki, do których nie dochodzi żadna krawędź, są gromadzone na stosie. Wybierany jest, w wierszu 10, szczytowy element u stosu (mógłby to być dowolny spośród elementów znajdujących się na stosie) i wierzchołkowi temu nadawany jest najmniejszy spośród niewykorzystanych jeszcze numerów. W ten sposób gwarantujemy, że wszystkie krawędzie odchodzące od tego wierzchołka będą prowadziły do wierzchołków o większych numerach. Następnie wierzchołek u, wraz z odchodzącymi od niego krawędziami jest usuwany z grafu. Powoduje to zmniejszenie się o jeden liczby krawędzi dochodzących do każdego wierzchołka v takiego, że $u \to v$ – liczba ta pamiętana jest w $LIDO[v]$. Jeśli dla któregoś z wierzchołków v liczba ta zostaje zredukowana do zera, to v jest kładziony na stos. Na mocy acykliczności grafu i naszych poprzednich uwag całkowite opróżnienie stosu, powodujące zakończenie wykonywania algorytmu (por. pętla 9) następuje nie wcześniej niż po nadaniu numerów wszystkim wierzchołkom grafu.

Łatwo zauważyć, że każda krawędź jest analizowana przez algorytm raz w wierszu 4 i raz w wierszu 12. Złożoność algorytmu jest więc O (m) (przypom-

my, że w mocy pozostaje założenie $m = \Omega(n)$ — w przeciwnym razie należałoby napisać $O(m+n)$).

Wobec algorytmu 3.5 przy opisie algorytmu znajdowania dróg w grafie acyklicznym możemy zakładać, że każda krawędź prowadzi od wierzchołka o mniejszym numerze do wierzchołka o większym numerze.

ALGORYTM 3.6 (Znajdowanie odległości od źródła do wszystkich wierzchołków w grafie acyklicznym)

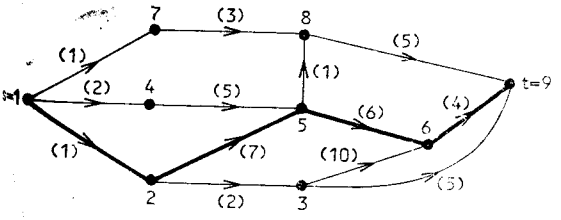
Dane: Graf zorientowany $\langle V, E\rangle$, gdzie $V = \{v_1, \ldots, v_n\}$ i dla dowolnej krawędzi $\langle v_i, v_j\rangle \in E$ mamy $i < j$. Graf ten określony jest przez listy incydencji $PRZED[v]$, $v \in V$.

Wyniki: Odległości od v_1 do wszystkich wierzchołków grafu: $D[v_i] = d(v_1, v_i)$, $i = 1, \ldots, n$.

```
1  begin
2    D[v1] := 0,
3    for j := 2 to n do D[vj] := ∞;
4    for j := 2 to n do
5      for vi ∈ PRZED[vj] do D[vj] := min(D[vj], D[vi]+A[vi, vj])
6  end
```

Nietrudno pokazać, przez indukcję względem j, że po wykonaniu pętli 4 dla pewnej wartości j mamy $D[v_j] = d(v_1, v_j)$, $i = 1, 2, \ldots, j$. Wystarczy skorzystać z faktu, iż wszystkie wierzchołki pośrednie najkrótszej drogi z v_1 do v_j mają numery mniejsze od j.

Złożoność algorytmu jest oczywiście $O(m)$, gdyż każda krawędź $\langle v_i, v_j\rangle$ jest analizowana w wierszu 5 dokładnie raz.



Rys. 3.4 Najdłuższa droga w grafie acyklicznym

Algorytmy 3.5 i 3.6 mogą mieć zastosowanie przy metodzie planowania przedsięwzięć zwanej *PERT* (Project Evaluation and Review Technique) lub *CPM* (Critical Path Method). Metoda ta polega na skonstruowaniu grafu (tzw. *sieci PERT* lub *sieci CPM*), którego krawędzie odpowiadają pewnym elementarnym zadaniom, z których składa się przedsięwzięcie, a ich wagi wskazują na czas potrzebny na wykonanie poszczególnych zadań. Zakładamy ponadto, że dla dowolnych krawędzi $\langle u, v\rangle$, $\langle v, t\rangle$ tego grafu, zadanie reprezentowane przez $\langle u, v\rangle$ musi być zakończone przed rozpoczęciem zadania reprezentowanego przez

$\langle v, t\rangle$. Łatwo zauważyć, że taki graf musi być acykliczny. Naszym zadaniem jest znalezienie najdłuższej drogi z wierzchołka s odpowiadającego rozpoczęciu przedsięwzięcia do wierzchołka t odpowiadającego jego zakończeniu (por. rys. 3.4). Drogę taką nazywamy *ścieżką krytyczną*. Jej długość określa czas potrzebny na wykonanie całego przedsięwzięcia. Oczywiście zagadnienie to sprowadza się do problemu najkrótszej drogi przez zamianę znaku każdej wagi $a(u, v)$, gdzie $u \to v$.

Najkrótsze drogi między wszystkimi parami wierzchołków, domknięcie przechodnie relacji 3.5

Zagadnienie wyznaczania odległości między wszystkimi parami wierzchołków można oczywiście rozwiązać stosując n razy – kolejno dla każdego z wierzchołków – jedną z opisanych poprzednio metod znajdowania odległości od ustalonego wierzchołka. Otrzymujemy w ten sposób algorytm o złożoności $O(n^4)$ (przy wykorzystaniu metody Forda-Bellmana) lub też $O(n^3)$ dla grafów acyklicznych lub nieujemnych wag. Okazuje się jednak, że w ogólnym przypadku n-krotne zastosowanie metody Forda-Bellmana nie jest najlepszą metodą. Pokażemy teraz dwie efektywniejsze metody.

Rozpatrzmy w tym celu graf zorientowany $G = \langle V, E\rangle$, gdzie $V = \{v_1, \ldots, v_n\}$ i załóżmy, że $A = [a_{ij}]$ jest macierzą wag ($a_{ij} = a(v_i, v_j)$). Oznaczając przez $d_{ij}^{(m)}$ długość najkrótszej drogi z v_i do v_j o co najwyżej m krawędziach otrzymujemy następujące oczywiste równania (por. (3.3)):

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{jeśli } i = j \\ \infty & \text{jeśli } i \neq j \end{cases} \tag{3.8}$$

$$d_{ij}^{(m+1)} = min\,\{d_{ik}^{(m)} + a_{kj} : 1 \leqslant k \leqslant n\} \tag{3.9}$$

Zauważmy, że równanie (3.9) wykazuje pewne podobieństwo do definicji iloczynu dwóch macierzy kwadratowych – jeśli operację *min* traktować jako „sumę", operację „+" zaś jako „iloczyn", to macierz $[d_{ij}^{(m+1)}]$ jest „iloczynem" macierzy $[d_{ij}^{(m)}]$ i $A = [a_{ij}]$. Oznaczmy taki „iloczyn" dwóch macierzy A i B przez $A*B$, oraz zauważmy, że elementem jednostkowym przy tej operacji jest macierz

$$U = \begin{bmatrix} 0 & \infty & \infty & \ldots & \infty \\ \infty & 0 & \infty & \ldots & \infty \\ \infty & \infty & 0 & \ldots & \infty \\ \vdots & \vdots & \vdots & & \vdots \\ \infty & \infty & \infty & \ldots & 0 \end{bmatrix}$$

równań (3.8) i (3.9) wynika teraz łatwo, że $[d_{ij}^{(0)}] = U$ oraz

$$d_{ij}^{(m)} = \underbrace{((\ldots((A*A)*A)\ldots)*A)}_{m \text{ razy}} \quad (m \geqslant 1) \tag{3.10}$$

ożе zajść jeden z dwu następujących przypadków:

(1) $d_{ij}^{(n-1)} = d_{ij}^{(n)}$ i w konsekwencji $d_{ij}^{(m)} = d_{ij}^{(n-1)}$ dla każdego $m \geqslant n$. Wtedy oczywiście $d_{ij}^{(n-1)} = d(v_i, v_j)$;

(2) $d_{ij}^{(n-1)} \neq d_{ij}^{(n)}$. Oznacza to, iż istnieje cykl ujemnej długości.

oczyn $A*B$ dwóch macierzy wymiaru $n \times n$ możemy wyznaczyć w czasie $O(n^3)$ dodawań i $n-1$ porównań na każdy spośród n^2 elementów iloczynu $A*B$). acierz $[d_{ij}^{(n-1)}]$, a tym samym odległości między wszystkimi parami wierzchoł- w możemy zatem obliczyć w czasie $O(n^4)$.

Złożoność takiego algorytmu jest na razie taka sama, jak w przypadku rotnego użycia algorytmu Forda-Bellmana. Możemy ją jednak obniżyć, jeśli ważymy, że działanie $*$ jest łączne (tzn. $(A*B)*C = A*(B*C)$, por. zad. 3.8). kt ten pozwala na obliczenie iloczynu (3.10) poprzez kolejne podnoszenie ma- rzy A do kwadratu i tym samym zastąpienie $n-1$ mnożeń macierzy przez $\lceil \log n \rceil$ ożeń. Otrzymujemy w ten sposób algorytm o złożoności $O(n^3 \log n)$ znajdu- y odległości między wszystkimi parami wierzchołków w grafie bez cykli ujem- długości.

Jeszcze bardziej efektywny algorytm pochodzi od Warshalla [70] i Floy- [20]. Idea tego algorytmu jest następująca: Oznaczmy przez $d_{ij}^{(m)}$ długości krótszej spośród dróg z v_i do v_j o wierzchołkach pośrednich w zbiorze $\{v_1, \ldots, v_m\}$.

Mamy wtedy następujące równania:

$$d_{ij}^{(0)} = a_{ij} \tag{3.11}$$

$$d_{ij}^{(m+1)} = \min d_{ij}^{(m)}, \ldots, d_{i,m+1}^{(m)} + d_{m+2,j}^{(m)} \text{ dla } m = 0,1, \ldots, n-1 \tag{3.12}$$

Uzasadnienie drugiego równania jest proste. Rozważmy najkrótszą spośród dróg z v_i do v_j o wierzchołkach pośrednich w zbiorze $\{v_1, \ldots, v_m, v_{m+1}\}$. Jeśli droga ta nie zawiera v_{m+1}, to $d_{ij}^{(m+1)} = d_{ij}^{m}$. Jeśli zawiera, to dzieląc ją na odcinki od v_i do v_{m+1} oraz od v_{m+1} do v_j otrzymujemy równość $d_{ij}^{(m+1)} = d_{i,m+1}^{(m)} + d_{m+1,j}^{(m)}$.

Równania (3.11), (3.12) dają możliwość łatwego obliczenia odległości $d(v_i, v_j) = d_{ij}^{(n)}$, $1 \leqslant i, j \leqslant n$.

ALGORYTM 3.7 (Wyznaczanie odległości między wszystkimi parami wierzchołków – metoda Floyda)

Dane: Macierz wag krawędzi $A[i,j]$, $1 \leqslant i,j \leqslant n$ grafu zorientowanego bez cykli o ujemnej długości.

Wyniki: Odległości między wszystkimi parami wierzchołków: $D[i,j] = d(v_i, v_j)$.

```
1  begin
2    for i := 1 to n do
3      for j := 1 to n do D[i,j] := A[i,j];
4    for i := 1 to n do D[i,i] := 0;
5    for m := 1 to n do
6      for i := 1 to n do
7        for j := 1 to n do
8          D[i,j] := min(D[i,j], D[i,m]+D[m,j])
9  end
```

Uzasadnienie poprawności algorytmu pozostawiamy Czytelnikowi.

Złożoność tego algorytmu jest oczywiście $O(n^3)$. Warto zauważyć, że taką samą złożoność miał algorytm Forda-Bellmana znajdowania odległości od ustalonego wierzchołka do wszystkich pozostałych. Ciekawe, że w ogólnym przypadku (tzn. bez założenia o nieujemności wag lub o acykliczności grafu) nie jest znany żaden algorytm znajdowania odległości między jedną ustaloną parą wierzchołków, który byłby istotnie efektywniejszy od algorytmu znajdowania odległości między wszystkimi parami.

Ściśle związane z problemem obliczania najkrótszych dróg w grafie jest zagadnienie domknięcia przechodniego relacji binarnej. Przypomnijmy, że przez relację binarną na zbiorze V rozumiemy dowolny podzbiór $E \subseteq V \times V$. Relacja taka jest przechodnia jeśli spełnia warunek

$$\text{jeśli } \langle x, y\rangle \in E \text{ i } \langle y, z\rangle \in E, \text{ to } \langle x, z\rangle \in E$$

dla dowolnych $x, y, z \in E$. Zauważmy, że relację binarną $E \subseteq V \times V$ można reprezentować jednoznacznie przez graf zorientowany $G = \langle V, E\rangle$. Dla dowolnej takiej relacji określamy

$E^* = \{\langle x, y\rangle$: istnieje w $\langle V, E\rangle$ droga o niezerowej długości od x do $y\}$. Nietrudno zauważyć, że E jest relacją przechodnią na zbiorze V, oraz $E \subseteq E^*$. Co więcej, E jest najmniejszą relacją przechodnią zawierającą E, tzn. dla dowolnej relacji przechodniej $F \supseteq E$ mamy $E^* \subseteq F$. Relację E^* nazywamy *domknięciem przechodnim* relacji E.

Jeśli relację E reprezentować przez graf $\langle V, E\rangle$, w którym każda krawędź ma wagę 1, to domknięcie przechodnie E^* możemy wyznaczyć w czasie $O(n^3)$ przez algorytm 3.7; po wykonaniu tego algorytmu mamy

$$\langle v_i, v_j\rangle \in E^* \Leftrightarrow D[i,j] < \infty$$

W przypadku obliczania domknięcia przechodniego wygodnie jest przyjąć

$$A[i,j] = \begin{cases} 0 & \text{gdy } \langle v_i, v_j\rangle \notin E \text{ (zamiast } \infty) \\ 1 & \text{gdy } \langle v_i, v_j\rangle \in E \end{cases} \tag{3.13}$$

Wiersz 7 algorytmu 3.7 możemy wówczas zmienić na

$$D[i,j] := D[i,j] \vee (D[i,m] \wedge D[m,j])$$

gdzie $\vee$ i $\wedge$ są zwykłymi działaniami boolowskimi:

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

Po wykonaniu tak zmodyfikowanego algorytmu (pochodzi on od Warshalla [70]) mamy oczywiście

$$D[i, j] = \begin{cases} 1 & \text{jeśli } \langle v_i, v_j \rangle \in E^* \\ 0 & \text{jeśli } \langle v_i, v_j \rangle \notin E^* \end{cases}$$

Warto tu zaznaczyć, że jest znany algorytm znajdowania domknięcia przechodniego efektywniejszy od algorytmu Warshalla (por. [53]). Wykorzystuje on związek tego problemu z mnożeniem macierzy omówiony na początku tego punktu, który dla macierzy A określonej przez (3.13) prowadzi do zwykłego mnożenia macierzy boolowskich zgodnie ze wzorem

$$c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$$

Okazuje się (por. poz. [64]), że takiego mnożenia można dokonać w czasie $O(n^{\log 7})$*, co daje algorytm znajdowania domknięcia przechodniego o złożoności $O(n^{\log 7} \log n)$. Algorytm taki ma znaczenie raczej teoretyczne, gdyż metoda mnożenia macierzy w czasie $O(n^{\log 7})$ jest dość skomplikowana, i w konsekwencji ujawnia swą wyższość nad zwykłą „szkolną" metodą dopiero przy bardzo dużych wartościach n. W praktyce najefektywniejszy okazuje się zwykle odpowiednio zaprogramowany algorytm Warshalla (por. poz. [65] oraz zad. 3.9).

Innym sposobem znajdowania domknięcia przechodniego relacji E jest zastosowanie przeszukiwania w głąb (lub wszerz) grafu $\langle V, E \rangle$. Sposób ten jest szczególnie korzystny, gdy relacja E jest symetryczna — domknięcie przechodnie jest wtedy oczywiście wyznaczone przez podział na składowe spójne, w grafie niezorientowanym wyznaczonym przez E, i może być otrzymane w czasie $O(m+n)$. Szczegóły pozostawiamy Czytelnikowi (por. zad. 3.12).

Zadania 3.6

3.1 Niech $G = \langle V, E \rangle$ będzie dowolnym grafem zorientowanym. Ustalmy pewien wierzchołek $s \in V$, dodajmy nowy wierzchołek t, zamieńmy każdą krawędź $\langle v, s \rangle \in E$ na krawędź $\langle v, t \rangle$ oraz przyporządkujmy każdej krawędzi otrzymanego grafu wagę -1. Udowodnij, że najkrótsza droga elementarna z s do t ma długość $-n$ wtedy i tylko wtedy, gdy w G istnieje cykl Hamiltona.

* Wykładnik log 7 może być jeszcze trochę zmniejszony (patrz poz. [55]).

3.2 Zmodyfikuj algorytm 3.1 tak, by wyznaczał on
(a) najkrótszą drogę elementarną z s do t;
(b) wszystkie najkrótsze drogi elementarne z s do t,
przy założeniu, iż graf nie zawiera cykli o długości ujemnej (może zawierać cykle zerowej długości). Przeprowadź analizę zaproponowanych algorytmów.

3.3 Zmodyfikuj algorytmy 3.2, 3.3 i 3.6 w taki sposób, by wyznaczały one najkrótsze drogi z s do wszystkich pozostałych wierzchołków, bez wykorzystania algorytmu 3.1.

Wskazówka: Należy dla każdego wierzchołka $v \in V$ pamiętać i uaktualniać w czasie działania algorytmu wierzchołek $PRE[v]$ taki, że

$$D[v] = D[PRE[v]] + A[PRE[v], v]$$

3.4 Czy można warunki (3.4) i (3.5) zastąpić równościami

$$D[v] = d^{(m)}(v),\ D[v] = d^{(m+1)}(v)?$$

3.5 Udowodnij, że w grafie bez cykli o niedodatniej długości, równania

$$d(s, s) = 0$$

$$d(s, v) = min\{d(s, u) + a(u, v) : u \in V \setminus \{v\}\}$$

wyznaczają jednoznacznie odległości $d(s, v)$, $v \in V$. Pokaż przykład grafu o cyklu długości 0, dla którego ten układ równań ma nieskończenie wiele rozwiązań.

3.6 Uzupełnij szczegóły opisu i analizy wersji algorytmu Dijkstry o złożoności $O(m \log n)$.

3.7 Wyznacz najkrótszą drogę z s do t dla grafu z rys. 3.4.

3.8 Udowodnij, że działanie $*$ określone na macierzach kwadratowych $A = [a_{ij}]$, $B = [b_{ij}]$ wymiaru $n \times n$ wzorem $A * B = [c_{ij}]$, gdzie $c_{ij} = min\{a_{ik} + b_{kj} : 1 \leqslant k \leqslant n\}$ jest łączne.

3.9 Udowodnij, że następująca modyfikacja algorytmu Warshalla wyznacza domknięcie przechodnie relacji [65]:

```
1  begin (* relacja E określona przez macierz A, por. (3.13) *)
2    for m := 1 to n do
3      for i := 1 to n do
4        if A[i, m] = 1 then
5          for j := 1 to n do A[i, j] := A[i, j] ∨ A[m, j]
6  end (* A[i, j] = 1 ⇔ ⟨v_i, v_j⟩ ∈ E* *)
```

Załóżmy, że wiersz macierzy A mieści się w słowie maszynowym, i że pętlę 5 traktujemy jako jedną operację elementarną (dodanie logiczne m-tego wiersza do i-tego).
Jaka jest wtedy złożoność algorytmu?

3.10 Zmodyfikuj algorytm Floyda tak, by oprócz odległości $D[i, j]$ wyznaczał macierz $M[i, j]$, gdzie $M[i, j]$ jest maksymalnym numerem wierzchołka

pewnej najkrótszej drogi z v_i do v_j ($M[i,j] = 0$, jeśli ta droga nie zawiera wierzchołków pośrednich). Udowodnij, że procedura rekurencyjna

```
procedure PISZDROGE(i, j);
  begin
    if M[i,j] = 0 then if i = j then write(i)
                                 else write(i, j)
                  else begin
                         PISZDROGE(i, M[i,j]);
                         PISZDROGE(M[i,j], j)
                       end
  end
```

wypisuje najkrótszą drogę z v_i do v_j w czasie $O(n)$. Podaj wersję nierekurencyjną tej procedury.

11 Przez przepustowość drogi rozumiemy minimalną wagę krawędzi tej drogi. Podaj algorytm wyznaczający maksymalne przepustowości dróg między: (a) ustaloną parą wierzchołków; (b) wszystkimi parami wierzchołków.

12 Na podstawie techniki przeszukiwania w głąb podaj algorytm znajdujący domknięcie przechodnie dowolnej relacji $E \subseteq V \times V$ w czasie $O(n^2+nm)$ oraz algorytm o złożoności $O(n^2)$ dla relacji symetrycznych.

Przepływy w sieciach i zagadnienia pokrewne

4

Maksymalny przepływ w sieci

4.1

Przez *sieć* rozumieć będziemy parę $S = \langle G, c\rangle$, gdzie $G = \langle V, E\rangle$ jest dowolnym grafem zorientowanym, a $c\colon E \to R$ jest funkcją, która każdej krawędzi $\langle u, v\rangle$ przyporządkowuje nieujemną liczbę rzeczywistą $c(u, v)$ zwaną *przepustowością* tej krawędzi. Zbiory V, E nazywamy odpowiednio zbiorem *wierzchołków* i zbiorem *krawędzi* sieci S.

Dla dowolnej funkcji

$$f\colon E \to R \tag{4.1}$$

i dowolnego wierzchołka v sieci S rozważmy wielkość

$$Div_f(v) = \sum_{u:v\to u} f(v,u) - \sum_{u:u\to v} f(u,v)$$

Jeśli $f(u, v)$ interpretujemy jako przepływ z u do v, to wielkość $Div_f(v)$ określa „ilość przepływu netto" wypływającego z wierzchołka v. Wielkość ta może być dodatnia (jeśli z wierzchołka v więcej wypływa niż do niego wpływa), ujemna (jeśli więcej do wierzchołka wpływa niż z niego wypływa, tzn. następuje „akumulacja" przpływu w wierzchołku v) lub równa zeru. Ten ostatni przypadek będzie nas najbardziej interesować. Wyróżnijmy w naszej sieci dwa wierzchołki, *źródło* s i *ujście* t ($s \neq t$). Przez *przepływ* z s do t w sieci S będziemy rozumieć dowolną funkcję postaci (4.1) spełniającą warunki

$$0 \leqslant f(u, v) \leqslant c(u, v) \qquad \text{dla każdego } \langle u, v\rangle \in E \tag{4.2}$$

oraz

$$Div_f(v) = 0 \qquad \text{dla każdego } v \in V \setminus \{s, t\} \tag{4.3}$$

Wielkość

$$W(f) = Div_f(s)$$

nazywamy *wartością przepływu* f.

Zgodnie z poprzednio opisanymi intuicjami mamy tu do czynienia z przepływem, który nie powstaje ani nie jest akumulowany w żadnym z wierzchołków różnym od s, t, oraz spełnia warunek mówiący, iż przez krawędź $\langle u, v\rangle$ możemy przesłać co najwyżej $c(u, v)$ jednostek przepływu. Przepływ taki może opisywać zachowanie się gazu lub cieczy w rurociągu, strumieni samochodów w sieci autostrad, przesyłanie towarów drogą kolejową (bez magazynowania ich na stacjach pośrednich), przesyłanie informacji w sieci informacyjnej itp.

Interesować nas będzie głównie znalezienie maksymalnego przepływu (tzn. przepływu o maksymalnej wartości) w danej sieci. Udowodnimy najpierw parę prostych i intuicyjnie oczywistych faktów. Przez *przekrój* $P(A)$ sieci S odpowiadający podzbiorowi $A \subseteq V$ $(A \neq \emptyset,\ A \neq V)$ rozumiemy zbiór krawędzi $\langle u, v\rangle \in E$ takich, że $u \in A$ i $v \in V \setminus A$, tzn.

$$P(A) = E \cap (A \times (V \setminus A))$$

Dla dowolnego przepływu f w sieci S, przepływ przez przekrój $P(A)$ definiujemy w naturalny sposób:

$$f(A, V \setminus A) = \sum_{e \in P(A)} f(e)$$

LEMAT 4.1

Jeśli $s \in A$, $t \in V \setminus A$, to dla dowolnego przepływu f z s do t

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A) \tag{4.4}$$

DOWÓD

Sumujmy równania $Div_f(v) = 0$ (por. (4.3)) dla wszystkich $v \in A$. Suma ta składa się z pewnej liczby składników $f(u, v)$, opatrzonych znakiem plus lub minus, gdzie co najmniej jeden z wierzchołków u, v należy do zbioru A. Jeśli oba wierzchołki należą do A, to $f(u, v)$ występuje ze znakiem plus w $Div_f(u)$ i ze znakiem minus w $Div_f(v)$ (i nie występuje w wyrażeniu $Div_f(r)$ dla żadnego r różnego od u i v), co w sumie daje 0. Każdy ze składników $f(u, v)$, $u \in A$, $v \in V \setminus A$ występuje dokładnie raz – ze znakiem plus – w $Div_f(u)$, co daje w sumie $f(A, V \setminus A)$. Podobnie składniki $f(u, v)$, $u \in V \setminus A$, $v \in A$ odpowiedzialne są za składnik $-f(V \setminus A, A)$ w (4.4). Z drugiej strony nasza suma jest równa $Div_f(s) = W(f)$, jako że $Div_f(v) = 0$ dla każdego $v \in A \setminus \{s\}$. ■

Przyjmując $A = V \setminus \{t\}$, z lematu 4.1 otrzymujemy w szczególności

$$\begin{aligned} Div_f(s) = W(f) &= f(V \setminus \{t\}, \{t\}) - f(\{t\}, V \setminus \{t\}) = \\ &= -(f(\{t\}, V \setminus \{t\}) - f(V \setminus \{t\}, \{t\})) = -Div_f(t) \end{aligned}$$

co wyraża intuicyjny fakt, iż do ujścia wpływa dokładnie tyle przepływu, ile wypływa ze źródła. Ogólnie, lemat 4.1 mówi, że globalną ilość przepływu możemy mierzyć w dowolnym przekroju oddzielającym s od t.

Zdefiniujmy *przepustowość przekroju* $P(A)$ następująco:

$$c(A, V \setminus A) = \sum_{e \in P(A)} c(e)$$

Przez *minimalny przekrój* między s i t będziemy rozumieć dowolny przekrój $P(A)$, $s \in A$, $t \in V \setminus A$ o minimalnej przepustowości.

Jednym z podstawowych faktów teorii przepływu w sieciach jest klasyczne twierdzenie o maksymalnym przepływie i minimalnym przekroju.

TWIERDZENIE 4.2 (Ford i Fulkerson[23]).
Wartość każdego przepływu z s do t nie przekracza przepustowości minimalnego przekroju między s i t, przy czym istnieje przepływ osiągający tę wartość.

DOWÓD

Niech $P(A)$ będzie minimalnym przekrojem. Na mocy lematu 4.1 dla dowolnego przepływu f mamy

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A) \leqslant f(A, V \setminus A) = \\ = \sum_{e \in P(A)} f(e) \leqslant \sum_{e \in P(A)} c(e) = c(A, V \setminus A) \tag{4.5}$$

Istnienie przepływu, dla którego powyższa nierówność przechodzi w równość (przepływ taki jest oczywiście maksymalny) jest faktem nieco głębszym. Otrzymamy go jako wniosek z analizy algorytmu przedstawionego w następnym punkcie. ■

Wszystkie znane algorytmy znajdowania maksymalnego przepływu polegają na kolejnym zwiększaniu przepływu, przy czym modyfikacja przepływu zwiększająca jego wartość opiera się najczęściej na tzw. *technice ścieżek rozszerzających.*

Powiemy, że krawędź e sieci S jest *użyteczna* z u do v względem przepływu f, jeśli

$$e = \langle u, v \rangle \quad \text{i} \quad f(e) < c(e) \tag{4.6}$$

lub

$$e = \langle v, u \rangle \quad \text{i} \quad f(e) > 0 \tag{4.7}$$

W zależności od tego, czy spełniony jest pierwszy czy drugi z powyższych warunków, mówimy odpowiednio o *zgodnej* lub *przeciwnej* krawędzi użytecznej z u do v. *Ścieżką rozszerzającą* (*długości* l) dla danego przepływu f z s do t nazywamy dowolny naprzemienny ciąg (parami różnych) wierzchołków i krawędzi

$$v_0, e_1, v_1, e_2, v_2, \ldots, v_{l-1}, e_l, v_l \tag{4.8}$$

taki, że $v_0 = s$, $v_l = t$ oraz dla każdego $i \leqslant l$ krawędź e_i jest użyteczna z v_{i-1} do v_i względem przepływu f.

Znajomość ścieżki rozszerzającej postaci (4.8) pozwala na łatwe zwiększenie wartości przepływu f o wielkość

$$\delta = min\{\Delta(e_i): 1 \leqslant i \leqslant l\}$$

gdzie

$$\Delta(e_i) = \begin{cases} c(e_i) - f(e_i) & \text{jeśli krawędź } e_i \text{ jest zgodna} \\ f(e_i) & \text{jeśli krawędź } e_i \text{ jest przeciwna} \end{cases}$$

Wystarczy w tym celu zwiększyć o δ przepływ w każdej zgodnej krawędzi e_i:

$$f'(e_i) = f(e_i) + \delta$$

az zmniejszyć przepływ o δ w każdej przeciwnej krawędzi e_i:

$$f'(e_i) = f(e_i) - \delta$$

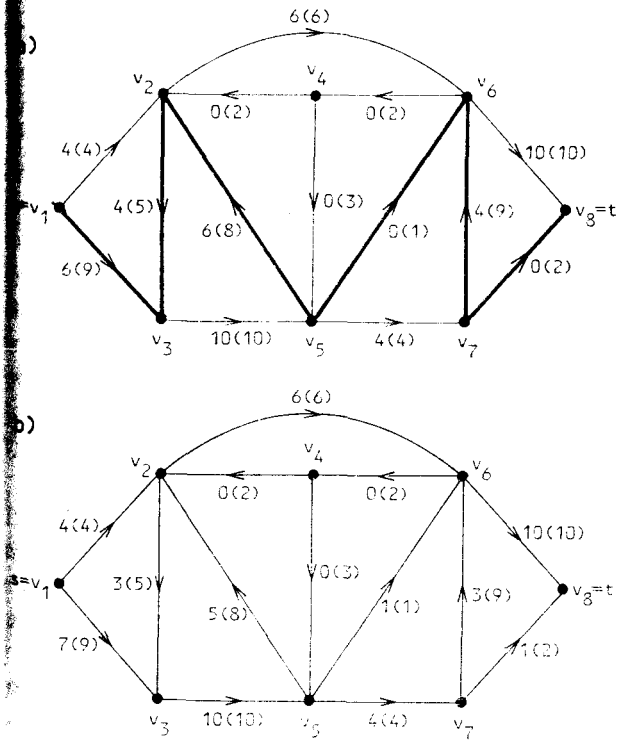
k określona funkcja f' (przyjmijmy $f'(e) = f(e)$ dla krawędzi e nie należących ścieżki (4.8)) jest przepływem. Istotnie, modyfikacja wartości $Div_f(v_i)$, $1 \leqslant i \leqslant l-1$ polega na zwiększeniu jej o wartość δ związaną ze zmianą przepływu w e_{i+1} oraz zmniejszeniu o wartość δ związaną ze zmianą przepływu w e_i – t tak niezależnie od typu krawędzi e_i, e_{i+1}. W sumie te zmiany kompensują , i w rezultacie $Div_{f'}(v_i) = Div_f(v_i) = 0$ $(1 \leqslant i \leqslant l-1)$. Spełniony jest również arunek $0 \leqslant f'(e) \leqslant c(e)$ dla każdej krawędzi e. Wartość przepływu zwiększa natomiast o δ:

$$W(f') = Div_{f'}(s) = Div_f(s) + \delta = W(f) + \delta$$

oces zwiększania przepływu wzdłuż ścieżki rozszerzającej pokazano na rys. 4.1. zy każdej krawędzi e podano przepływ w tej krawędzi $f(e)$ oraz – w nawiasie – zepustowość $c(e)$. Nasza ścieżka rozszerzająca ma postać

$$v_1, \langle v_1, v_3\rangle, v_3, \langle v_2, v_3\rangle, v_2, \langle v_5, v_2\rangle, v_5, \langle v_5, v_6\rangle, v_6, \langle v_7, v_6\rangle, v_7, \langle v_7, v_8\rangle, v_8$$

daje zwiększenie przepływu o $\delta = \Delta(v_5, v_6) = 1$, z 10 do 11.



Rys. 4.1 Zwiększanie przepływu wzdłuż ścieżki naprzemiennej:
(a) przed modyfikacją
(b) po modyfikacji

TWIERDZENIE 4.3

Następujące trzy warunki są równoważne:

(a) Przepływ z s do t jest maksymalny.

(b) Nie istnieje ścieżka rozszerzająca dla f.

(c) $W(f) = c(A, V \setminus A)$ dla pewnego $A \subseteq V$ takiego, że $s \in A$, $t \notin A$.

DOWÓD

(a)⇒ (b) Jeśli przepływ jest maksymalny, to oczywiście nie istnieje dla niego ścieżka rozszerzająca, jako że istnienie takiej ścieżki umożliwiałoby zwiększenie przepływu.

(b) ⇒ (c) Przypuśćmy, że dla pewnego przepływu f nie istnieje ścieżka rozszerzająca. Określmy zbiór $A \subseteq V$ jako zbiór tych wierzchołków v, dla których istnieje ścieżka

$$v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k$$

taka, że $k \geqslant 0$, $v_0 = s$, $v_k = v$, oraz e_i jest krawędzią użyteczną z v_{i-1} do v_i, $1 \leqslant i \leqslant k$. Oczywiście $s \in A$, natomiast $t \notin A$, gdyż oznaczałoby to istnienie ścieżki rozszerzającej dla f. Rozważmy przekrój $P(A)$. Dla każdej krawędzi $e \in P(A)$ musi być $f(e) = c(e)$, zgodnie z definicją zbioru A (por. warunek (4.6)), tak więc $f(A, V \setminus A) = c(A, V \setminus A)$. Korzystając znów z definicji zbioru A, otrzymujemy w podobny sposób $f(V \setminus A, A) = 0$ (por. warunek (4.7)). Na mocy lematu 4.1 wnioskujemy stąd, że

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A) = c(A, V \setminus A)$$

(c) ⇒ (a) wynika z udowodnionego już faktu, iż wartość dowolnego przepływu nie przekracza $c(A, V \setminus A)$, por. (4.5). ■

Nietrudno sprawdzić, że przepływ z rysunku 4.1b jest maksymalny. Przekrój „nasycony" $P(A)$ występujący w dowodzie twierdzenia 4.3 jest wyznaczony przez $A = \{v_1, v_2, v_3, v_5\}$.

Do wywnioskowania twierdzenia o maksymalnym przepływie i minimalnym przekroju z twierdzenia 4.3 brakuje nam jeszcze pewnego, dość subtelnego szczegółu. (Zachęcamy Czytelnika do zastanowienia się jakiego, przed przystąpieniem do dalszego czytania). Otóż brakującym faktem jest po prostu istnienie przepływu maksymalnego w dowolnej sieci. Innymi słowy należy pokazać, że nie jest możliwa sytuacja, w której na przykład istnieje dla każdego $\varepsilon > 0$ przepływ o wartości przekraczającej $7 - \varepsilon$, lecz nie istnieje przepływ o wartości równej 7 (warto zauważyć, że podobny problem dla minimalnego przekroju nie istnieje – liczba różnych przekrojów, w odróżnieniu od liczby różnych przepływów, jest skończona). Ten prosty fakt – będący w gruncie rzeczy wnioskiem z nieostrości nierówności w warunku (4.2) – można udowodnić na wiele sposobów. My udowodnimy go w następnym punkcie przez rozważenie pewnego algorytmu znajdowania maksymalnego przepływu.

lgorytm znajdowania maksymalnego rzepływu 4.2

ozważania z poprzedniego paragrafu sugerują ideę następującego prostego gorytmu znajdowania maksymalnego przepływu: Startując od dowolnego zepływu (np. $f(e) = 0$ dla każdej krawędzi $e \in E$) poszukujemy ścieżek rozerzających i zwiększamy wzdłuż nich aktualny przepływ.

Powstają tu jednak dwa pytania. Po pierwsze, czy taki algorytm zakończy oje działanie po skończonej liczbie kroków, oraz po drugie, czy otrzymany zepływ będzie maksymalny. Odpowiedź na drugie pytanie — przy założeniu, że gorytm po skończonej liczbie kroków zatrzyma się z braku dalszych ścieżek zszerzających — jest twierdząca, co wynika bezpośrednio z twierdzenia 4.3. atomiast Ford i Fulkerson w pracy [23] dali, w pewnym sensie, negatywną odpoiedź na pierwsze pytanie. Pokazali oni mianowicie przykład sieci, w której ożna tak „złośliwie" dobierać kolejne rozważane ścieżki rozszerzające, że oces nigdy się nie kończy, co więcej, wartość przepływu jest przez cały czas niejsza od jednej czwartej wartości przepływu maksymalnego.

Edmonds i Karp (por. poz. [17]) pokazali, że jeśli aktualny przepływ zwiększamy zawsze wzdłuż najkrótszej ścieżki rozszerzającej, to maksymalny przepływ iągany jest po użyciu co najwyżej $mn/2$ ścieżek (jak zwykle $n = |V|$, $m = |E|$). alezienie najkrótszej ścieżki łatwo zrealizować przez algorytm analogiczny do zeszukiwania wszerz (por. p. 2.3). Mówiąc dokładniej, startując ze źródła s zeszukujemy wszerz graf G_f złożony z krawędzi $\langle u, v\rangle$ takich, że istnieje w naej sieci krawędź użyteczna z u do v względem aktualnego przepływu f, aż do omentu osiągnięcia ujścia t. Znaleziona przez ten proces najkrótsza droga z s o t odpowiada oczywiście najkrótszej ścieżce rozszerzającej z s do t w naszej sieci. iorąc pod uwagę fakt, że procesu przeszukiwania wszerz możemy dokonać czasie $O(m+n)$, złożoność całego algorytmu znajdowania maksymalnego przeływu możemy oszacować jako $O(mn(m+n))$. Nie będziemy się tu bliżej zajmować tym algorytmem, jako że w dalszej części tego paragrafu pokażemy efekwniejszą metodę, o złożoności $O(n^3)$.

Ciekawą technikę zastosował do znajdowania maksymalnego przepływu inic w pracy [13]. Polega ona na konstruowaniu pewnej pomocniczej sieci cyklicznej (tzn. o grafie nie zawierającym cykli), której struktura dokładnie odzwierciedla wszystkie najkrótsze ścieżki rozszerzające z s do t względem aktualnego przepływu f. Sieć taką — oznaczmy ją przez S_f — konstruujemy przez przeszukiwanie wszerz wspomnianego już grafu G_f o krawędziach określonych przez istnienie w naszej sieci użytecznych krawędzi względem przepływu f. Sieć S_f zawiera źródło s, ujście t, oraz krawędzie grafu G_f postaci $\langle u, v\rangle$, gdzie u znajduje się w odległości d, a v w odległości $d+1$ od s, przy czym $0 \leqslant d < l$, a l jest *długością* sieci S_f, tzn. odległością od s do t w grafie G_f. Przepustowość $c_f(u, v)$ określamy jako $c(u, v) - f(u, v)$ lub $f(v, u)$, w zależności od tego, czy krawędź

$\langle u, v\rangle$ reprezentuje krawędź zgodną czy przeciwną (lub sumę tych wartości, jeśli istnieje jednocześnie zgodna i przeciwna krawędź użyteczna z u do v).

Poniżej przedstawiono procedurę *PSA* konstruującą sieć S_f. Oryginalna sieć S jest dana przez listy incydencji $ZA[v]$, $PRZED[v]$, $v \in V$, oraz przez tablicę przepustowości krawędzi $C[u, v]$, $u, v \in V$, natomiast aktualny przepływ f określony jest przez tablicę $F[u, v]$, $u, v \in V$. Zmienne odnoszące się do konstruowanej sieci pomocniczej S_f odróżnione są od zmiennych odnoszących się do oryginalnej sieci S przez pierwszą literę X: mamy zbiór wierzchołków XV, listy incydencji $XZA[v]$, $XPRZED[v]$, $v \in XV$, oraz tablice przepustowości i przepływów XC i XF. Odległość wierzchołka od źródła w sieci S_f pamiętana jest w tablicy ODL. Struktura procedury *PSA* jest następująca: Po inicjalizacji w wierszach $3 \div 7$ następuje przeszukiwanie wszerz grafu G_f, poczynając od źródła s (p. główna pętla w wierszu 9). Kolejno odwiedzane w procesie przeszukiwania wierzchołki v są umieszczane w kolejce (p. wiersze 13 i 21), każdy wierzchołek u pobierany z kolejki (wiersz 10) wykorzystywany jest natomiast w dwóch pętlach (p. wiersze 11 i 19). Pierwsza z nich przeszukuje zgodne krawędzie użyteczne $\langle u, v\rangle$, druga zaś przeciwne krawędzie użyteczne $\langle v, u\rangle$. Zauważmy, że sieć może zawierać jednocześnie zgodną krawędź użyteczną $\langle u, v\rangle$ oraz przeciwną krawędź użyteczną $\langle v, u\rangle$. Krawędź $\langle u, v\rangle$ włączana do S_f ma wtedy przepustowość równą sumie przepustowości wnoszonych przez te krawędzie (por. wiersze 23 i 27).

```
1   procedure PSA; (* konstruowanie pomocniczej sieci acyklicznej S_f;
    zmienne V, XV, PRZED, ZA, XPRZED, XZA, ODL, C, F, XC, XF, s, t są
    globalne *)
2   begin
3     for u ∈ V do (* inicjalizacja *)
4       begin ODL[u] := ∞; XPRZED[u] := Ø; XZA[u] := Ø;
5         for v ∈ V do XC[u, v] := 0;
6         for v ∈ V do XF[u, v] := 0 (* inicjalizacja przepływu zerowego *)
7       end;
8     KOLEJKA := Ø; XV := Ø; ODL[s] := 0;
      (* przeszukiwanie wszerz rozpoczynając od źródła s *)
9     KOLEJKA ⇐ s;
10    while KOLEJKA ≠ Ø do
11      begin u ⇐ KOLEJKA; XV := XV ∪ {u};
12        for v ∈ ZA[u] do (* przeszukiwanie krawędzi zgodnych *)
13          if (ODL[u] < ODL[v] ⩽ ODL[t]) and (F[u, v] < C[u, v]) then
14            begin if ODL[v] = ∞ then KOLEJKA ⇐ v; (* v nowy *)
15              ODL[v] := ODL[u] + 1;
                (* dodaj ⟨u, v⟩ do sieci S_f *)
16              XZA[u] := XZA[u] ∪ {v};
17              XPRZED[v] := XPRZED[v] ∪ {u};
18              XC[u, v] := C[u, v] − F[u, v]
```

```
19            end;
20          for v ∈ PRZED[u] do (* przeszukiwanie krawędzi przeciwnych *)
21            if (ODL[u] < ODL[v] ⩽ ODL[t]) and (F[v, u] > 0) then
22              begin if ODL[v] = ∞ then KOLEJKA ⇐ v; (* nowy *)
23                ODL[v] := ODL[u]+1;
24                if XC[u, v] = 0 then (* dodaj ⟨u, v⟩ do sieci S_f *)
25                  begin XZA[u] := XZA[u] ∪ {v};
26                    XPRZED[v] := XPRZED[v] ∪ {u}
27                  end;
28                XC[u, v] := XC[u, v]+F[v, u]
29              end
30        end
31  end
```

Niech po zakończeniu działania procedury, $ODL[t] = l$. Jeśli $l = \infty$, to oznacza to, iż w naszym procesie nie osiągnęliśmy ujścia t, tzn. nie istnieje w naszej sieci ścieżka rozszerzająca z s do t. Na mocy twierdzenia 4.3 aktualny przepływ w sieci S jest maksymalny. Jeśli $l < \infty$, to najkrótsza ścieżka rozszerzająca z s do t ma długość l, co więcej, łatwo zauważyć, że drogi z s do t w sieci pomocniczej odpowiadają wzajemnie jednoznacznie ścieżkom rozszerzającym długości l w oryginalnej sieci (ściśle rzecz biorąc, niektóre z tych „ścieżek rozszerzających" mogą używać jednocześnie zgodnej i przeciwnej krawędzi użytecznej z u do v). Warto zauważyć, że po osiągnięciu ujścia, gdy $ODL[t] < \infty$, nowo napotkane wierzchołki nie są już rozważane w procedurze *PSA* (p. warunki w wierszach 13 i 21) – żaden taki wierzchołek, znajdując się w odległości l od s, nie może być wierzchołkiem drogi długości l z s do t.

Nietrudno oszacować złożoność procedury *PSA*. Każdy wierzchołek v jest umieszczany w kolejce i zdejmowany z niej co najwyżej raz. Każda krawędź incydentna z v jest analizowana dokładnie raz (w pętli 12 lub 20), przy czym liczba kroków dla każdej takiej krawędzi jest ograniczona przez stałą. Całkowita liczba kroków jest więc rzędu $n+m$, co jest zdominowane przez n^2 kroków inicjalizacji tablic XC i XF (wiersze 5, 6). Pomocniczą sieć acykliczną można więc skonstruować w czasie $O(n^2)$ (kosztownej inicjalizacji tablic XC i XF można by uniknąć, lecz zastosowanie takiej zmodyfikowanej procedury o złożoności $O(n+m)$ nie zmieniłoby oszacowania $O(n^3)$ dla całego algorytmu znajdowania maksymalnego przepływu).

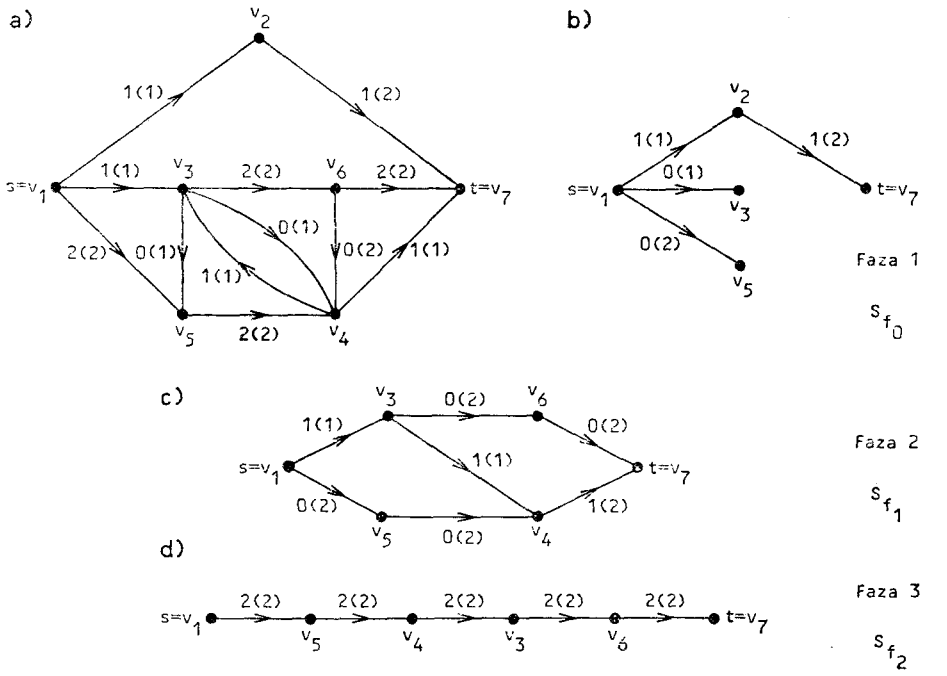
Istota pomysłu Dinica polega na rozbiciu procesu zwiększania przepływu wzdłuż ścieżek rozszerzających na *fazy*, odpowiadające wykorzystaniu najkrótszych ścieżek określonej długości. Faza rozpoczyna się skonstruowaniem pomocniczej sieci acyklicznej, następnie w sieci pomocniczej jest znajdowany tzw. przepływ pseudomaksymalny. Przez *przepływ pseudomaksymalny* w sieci S_f długości l, nazywamy dowolny przepływ f^* w S_f taki, że nie istnieje w S_f ścieżka

rozszerzająca długości l względem przepływu f^*; innymi słowy, dla dowolnej drogi z s do t w S_f, określonej przez ciąg wierzchołków

$$v_0, v_1, \ldots, v_l \qquad (v_0 = s, v' = t)$$

istnieje krawędź $\langle v_i, v_{i+1}\rangle$, $0 \leqslant i < l$ taka, że $f^*(v_i, v_{i+1}) = c_f(v_i, v_{i+1})$. Przepływ pseudomaksymalny jest następnie „przenoszony" z sieci pomocniczej do sieci oryginalnej: przepływ $f^*(u, v)$ dodawany jest do $f(u, v)$, a jeśli powoduje to przekroczenie przepustowości $c(u, v)$ — ta „nadwyżka" jest likwidowana oraz uwzględniana przez odpowiednie zmniejszenie przepływu $f(v, u)$ (w taki sposób efektem naszej modyfikacji jest zawsze zwiększenie $Div_f(u)$ o $f^*(u, v)$ i zmniejszenie $Div_f(v)$ o $f^*(u, v)$). Nietrudno sprawdzić, że taka modyfikacja przepływu f, dokonana dla wszystkich krawędzi $\langle u, v\rangle$ sieci pomocniczej, określa pewien nowy przepływ f' taki, że $W(f') = W(f) + W(f^*)$. Fazę uważamy za zakończoną. Metoda Dinica polega na wykonywaniu kolejnych faz, rozpoczynając od przepływu zerowego, aż do momentu, gdy przepływ w naszej sieci jest maksymalny.

Pokazano to na rysunku 4.2 (jak zwykle, przy każdej krawędzi pokazano przepływ w tej krawędzi oraz, w nawiasie, jej przepustowość). Startując od przepływu zerowego f_0 w sieci S, otrzymujemy pomocniczą sieć acykliczną S_{f_0} (rys. 4.2b). Znajdujemy w niej przepływ pseudomaksymalny (metoda efektywnego



Rys. 4.2 (a) Sieć S i przepływ maksymalny w tej sieci.
(b), (c) Pomocnicze sieci acykliczne odpowiadające kolejnym fazom algorytmu Dinica, wraz z zaznaczonymi przepływami pseudomaksymalnymi w każdej z nich.

jdowania takiego przepływu będzie omówiona w dalszym ciągu), i przenosimy do sieci S, otrzymując przepływ f_1. Z kolei konstruujemy sieć S_{f_1} (rys. 4.2c) aleziony w niej przepływ pseudomaksymalny przenosimy do sieci S, co daje pływ f_2. W ostatniej fazie konstruujemy sieć S_{f_2} (rys. 4.2d). Po przeniesieniu pływu pseudomaksymalnego w S_{f_2} do S otrzymujemy w S przepływ f (po- ny na rys. 4.2a), który jest maksymalny, jako że $W(f) = 4 = c(\{s\}, V \setminus \{s\})$. twierdzenie 4.3). Rysunek 4.2c dobrze ilustruje fakt, że przepływ pseudo- symalny nie musi być maksymalny. Istotnie, pokazany na tym rysunku pływ pseudomaksymalny możemy powiększyć wzdłuż następującej ścieżki zerzającej długości 5:

$$v_1, \langle v_1, v_5\rangle, v_5, \langle v_5, v_4\rangle, v_4, \langle v_3, v_4\rangle, v_3, \langle v_3, v_6\rangle, v_6, \langle v_6, v_7\rangle, v_7$$

ważmy jeszcze, że krawędź $\langle v_4, v_3\rangle$ sieci S_{f_2} (por. rys. 4.2d) powstała z dwóch ciwnie skierowanych krawędzi sieci S.

Podstawowym faktem odpowiedzialnym za efektywność metody Dinica jest że niezależnie od przepustowości krawędzi liczba faz nie przekracza n. Dla odu tego faktu oznaczmy przez S_k pomocniczą sieć acykliczną zbudowaną -tej fazie, przez l_k zaś długość tej sieci. (Uwaga: Ostatnie wywołanie proce- *PSA*, w którym nie jest osiągane ujście, nie jest liczone jako faza).

AT 4.4

liczba faz przekracza k, to $l_k < l_{k+1}$.

ÓD

aczmy przez $d_k(v)$ odległość w S_k od s do v. Mamy udowodnić, że $d_{k+1}(t) > d_k(t)$. Rozważmy w S_{k+1} dowolną drogę $v_0, v_1, \ldots, v_{l_{k+1}}$ z $s = v_0$ do $t = v_{l_{k+1}}$.

Załóżmy, że dla pewnego $p \leqslant l_{k+1}$ wszystkie wierzchołki v_i, $0 \leqslant i \leqslant p$ eżą do sieci S_k. Pokażemy najpierw, że wtedy

$$d_k(v_{i+1}) \leqslant d_k(v_i)+1, \qquad 0 \leqslant i < p \tag{4.9}$$

puśćmy, że tak nie jest, tzn. że $d_k(v_{i+1}) > d_k(v_i)+1$ dla pewnego i. Zgodnie onstrukcją sieci S_k oznacza to, że S_k nie zawiera krawędzi $\langle v_i, v_{i+1}\rangle$ (ani też wiście krawędzi $\langle v_{i+1}, v_i\rangle$). W czasie k-tej fazy nie jest więc modyfikowany ani przepływ z v_i do v_{i+1}, ani też z v_{i+1} do v_i. Przeczy to jednak istnieniu początku $(k+1)$-szej fazy krawędzi w S użytecznej z v_i do v_{i+1}, której nie na początku k-tej fazy. Sprzeczność ta dowodzi nierówności (4.9). Z nierów- ci tych, wobec $d_k(v_0) = 0$, wnioskujemy przez indukcję względem i, że

$$d_k(v_i) \leqslant i = d_{k+1}(v_i), \qquad 0 \leqslant i \leqslant p \tag{4.10}$$

li wszystkie wierzchołki drogi $v_0, v_1, \ldots, v_{l_{k+1}}$ należą do sieci S_k, to otrzymu- y stąd w szczególności $d_k(t) \leqslant d_{k+1}(t)$. Równość $d_k(t) = d_{k+1}(t)$ byłaby mo- a jedynie gdy $d_k(v_i) = d_{k+1}(v_i)$, $0 \leqslant i \leqslant l_{k+1}$, lecz łatwo zauważyć, że pow- zenie się drogi $v_0, v_1, \ldots, v_{l_{k+1}}$ w S_k i S_{k+1} przeczyłoby pseudomaksymalności pływu znalezionego w S_k. Musi więc być $d_{k+1}(t) > d_k(t)$.

Rozważmy teraz przypadek, w którym istnieje wierzchołek drogi $v_0, v_1, \ldots, v_{l_{k+1}}$ nie należący do S_k. Niech v_{p+1} będzie pierwszym takim wierzchołkiem (oczywiście $1 \leqslant p+1 \leqslant l_{k+1} - 1$). Stosując podobne rozumowanie jak poprzednio wnioskujemy, że pojawienie się w S_{k+1} krawędzi $\langle v_p, v_{p+1}\rangle$, której nie było w S_k możemy przypisać tylko temu, iż $d_k(v_p) = d_k(t) - 1$, i do wierzchołka v_{p+1} dochodzimy już po osiągnięciu ujścia t (tak, że nie jest on włączany do S_k).
Wobec (4.10) i faktu, że wierzchołki $v_0, v_1, \ldots, v_{p-1}$ należą do sieci S_k otrzymujemy

$$l_k = d_k(t) = d_k(v_p)+1 \leqslant d_{k+1}(v_p)+1 = p+1 < l_{k+1} \qquad \blacksquare$$

Uwzględniając fakt, że $l_1 \geqslant 1$ oraz $l_k \leqslant n-1$ dla dowolnego k, otrzymujemy

WNIOSEK 4.5

Liczba faz w metodzie Dinica nie przekracza $n-1$.

Pozostało nam teraz podać efektywny algorytm znajdowania przepływu pseudomaksymalnego w pomocniczej sieci acyklicznej. Metoda, którą podamy pochodzi od Malhotry, Kumara i Maheshwari [48] i jest zrealizowana przez procedurę *MAXPSA* poniżej. Aby ją opisać, wprowadzimy najpierw pojęcie *potencjału wierzchołka sieci*, tzn. maksymalnej ilości przepływu, który można „przepuścić" przez ten wierzchołek. Definiujemy go jako minimum z dwóch wielkości: maksymalnego możliwego przepływu do wierzchołka v (*potencjał wejściowy*) oraz maksymalnego możliwego przepływu z wierzchołka v (*potencjał wyjściowy*). Dokładniej, *potencjał* wierzchołka v dowolnej sieci $S = \langle G, c\rangle$ o źródle s i ujściu t równy jest

$$Pot\ (v) = min\ \{Potwe\ (v),\ Potwy\ (v)\}$$

gdzie

$$Potwe\,(v) = \begin{cases} \sum\limits_{u:u\to v} c\,(u, v) & \text{jeśli } v \neq s \\ \infty & \text{jeśli } v = s \end{cases}$$

oraz

$$Potwy\,(v) = \begin{cases} \sum\limits_{u:v\to u} c(v, u) & \text{jeśli } v \neq t \\ \infty & \text{jeśli } v = s \end{cases}$$

Procedura *MAXPSA* oblicza najpierw potencjały wszystkich wierzchołków w pomocniczej sieci acyklicznej (wiersze 3 ÷ 12), umieszczając wierzchołki o zerowym potencjale na stosie *STOS*. Jest oczywiste, że wierzchołek o zerowym potencjale może być usunięty z sieci, wraz ze wszystkimi incydentnymi z nim krawędziami, bez żadnego wpływu na jakikolwiek przepływ w tej sieci. Usunięcie takiego wierzchołka, a ściślej incydentnych z nim krawędzi, może spowodować wyzerowanie potencjałów pewnych sąsiednich wierzchołków, których usunięcie zeruje po-

cjały dalszych wierzchołków itd. Proces ten jest realizowany przez pętlę 17. wyjściu z tej pętli (gdy $STOS = \emptyset$), zbiór XN zawiera wierzchołki pomocni- sieci acyklicznej o niezerowym potencjale (obliczanym w sieci zmodyfiko- nej przez kolejne usuwanie wierzchołków o zerowym potencjale). Jeśli $XN \neq \emptyset$, następuje ważny moment: znajdowany jest w XN wierzchołek r o minimalnym encjale, $p = POT[r]$ (wiersze 38 ÷ 41). Następnie znajdujemy w naszej sieci klicznej przepływ z r do t o wartości p (wiersze 42 ÷ 65) oraz przepływ z s r o wartości p (wiersze 66 ÷ 91) – w sumie przepływ z s do t o wartości p. iszemy pierwszą część (przepływ z r do t) – druga jest w pełni analogiczna. epływ nasz będzie używać jedynie zgodnych (w pomocniczej sieci acyklicznej) wędzi użytecznych, a metodę jego wyznaczania intuicyjnie najlepiej wyjaśnić następujący sposób. Wyobraźmy sobie, że w wierzchołku r umieściliśmy dunek" o wielkości $Q[r] = p$, i chcemy ten ładunek „przepchnąć" do ujścia t by ilość ładunku przeniesiona przez żadną krawędź nie przekraczała jej prze- towości. W tym celu przeszukujemy naszą sieć wszerz, startując z wierzchoł- r. Jak zwykle przy przeszukiwaniu wszerz osiągnięte już, lecz jeszcze niewy- zystane wierzchołki pamiętane są w kolejce. Każdy pobierany z kolejki wierz- łek v (wiersz 43) nie będący ujściem jest „rozładowywany" (wiersze 49÷64). ga to na przeniesieniu ładunku $Q[v]$ do wierzchołków u, do których prowa- krawędź z v. Ładunek ten jest przenoszony do kilku pierwszych wierzchołków liście $XZA[v]$, przy czym ładunek przeniesiony do każdego takiego wierzchoł- u, z wyjątkiem być może ostatniego, jest równy przepustowości $XC[v, u]$. ten sposób we wszystkich użytych krawędziach $\langle v, u\rangle$, z wyjątkiem być może tniej, jest ustalany przepływ $XF[v, u] = XC[v, u]$, i te „nasycone" kra- dzie możemy z sieci usunąć (wiersze 57÷60; przypomnijmy, że poszuku- y przepływu pseudomaksymalnego, a więc możemy się ograniczyć do wyko- stywania jedynie zgodnych krawędzi użytecznych pomocniczej sieci acyklicz-). Usuwaniu krawędzi towarzyszy odpowiednie zmniejszenie potencjałów ersze 44 ÷ 46). Wierzchołki sieci są odwiedzane zgodnie ze schematem prze- kiwania wszerz, a zatem, oznaczając $d = ODL[r]$, cały ładunek z wierzchoł- r jest przenoszony do wierzchołków znajdujących się w odległości $d+1$ od s, tępnie do wierzchołków znajdujących się w odległości $d+2$ od s, i tak dalej, cały ten ładunek osiąga ujście t. Zauważmy, że ładunek dochodzący do każ- o wierzchołka v nie przekracza p, i wierzchołek ten możemy rozładować li $v \neq t$), jako że p będąc równe minimalnemu potencjałowi nie przekracza pewno potencjału wyjściowego wierzchołka v.

Po znalezieniu, w analogiczny sposób, przepływu o wartości p z s do r rwsza iteracja głównej pętli (wiersz 15) – złożonej z eliminacji wierzchołków erowym potencjale i zwiększeniu przepływu o wartość minimalnego aktualnie tencjału – jest zakończona. Następują dalsze iteracje, aż do momentu, gdy ma już wierzchołków o niezerowym potencjale (tzn $XN = \emptyset$). Przepływ leziony w naszej sieci jest wtedy oczywiście pseudomaksymalny.

```
1   procedure MAXPSA; (* wyznaczanie przepływu pseudomaksymalnego
    w pomocniczej sieci acyklicznej metodą Malhotry, Kumara i Maheshwari;
    zmienne XV, XC, XF, XPRZED, XZA, s, t są globalne *)
2   begin
3     STOS := Ø; (* STOS zawierać będzie wierzchołki o zerowym potencjale *)
4     for v ∈ XV do (* wyznaczanie potencjału wierzchołka v *)
5       begin POTWE[v] := 0; POTWY[v] := 0;
6         if v = s then POTWE[v] := ∞
7         else for u ∈ XPRZED[v] do POTWE[v] := POTWE[v]+XC[u, v];
8         if v = t then POTWY[v] := ∞
9         else for u ∈ XZA[v] do POTWY[v] := POTWY[v]+XC[v, u];
10        POT[v] := min(POTWE[v], POTWY[v]);
11        if POT[v] = 0 then STOS⇒ v
12      end;
13    for v ∈ XV do Q [v] := 0 (* inicjalizacja ładunków wierzchołków *)
14    XN := XV; (*XN zawierać będzie wierzchołki o niezerowym potencjale*)
15    while XN ≠ Ø do (* główna pętla *)
16      begin
          (* wyeliminowanie wierzchołków o zerowym potencjale *)
17        while STOS ≠ Ø do
18          begin v ⇐ STOS; XN := XN\{v};
              (* usuwanie krawędzi dochodzących do v *)
19            for u ∈ XPRZED[v] do (* usuń krawędź ⟨u, v⟩ *)
20              begin POTWY[u] := POTWY[u]−(XC[u, v]−XF[u, v]);
21                XZA[u] := XZA[u]\{v};
                  XPRZED[v] := XPRZED [v]\{u};
22                if POT[u] ≠ 0 then (* zmodyfikuj POT[u] *)
23                  begin POT[u] := min(POTWE[u], POTWY[u]);
24                    if POT[u] = 0 then STOS ⇐ u
25                  end
26              end
              (* usuwanie krawędzi wychodzących z v *)
27            for u ∈ XZA[v] do (* usuń krawędź ⟨v, u⟩ *)
28              begin POTWE[u] := POTWE[u]−(XC[v, u]−XF[v, u]);
29                XPRZED[u] := XPRZED[u]\{v};
                  XZA[v] := XZA[v]\{u};
30                if POT[u] ≠ 0 then (* zmodyfikuj POT[u] *)
31                  begin POT[u] := min(POTWE[u], POTWY[u]);
32                    if POT[u] = 0 then STOS ⇐ u
33                  end
34              end
35        end;
```

```
(* XN jest zbiorem wierzchołków o niezerowym potencjale *)
if XN ≠ Ø then (* przepływ nie jest jeszcze maksymalny *)
  begin
    (* znalezienie wierzchołka r o minimalnym potencjale *)
    p := ∞;
    for v ∈ VN do if POT[v] < p then
                    begin r := v; p := POT[r]
                    end;
    (* znalezienie przepływu o wartości p z r do t *)
  KOLEJKA := Ø; KOLEJKA ⇐ r; Q[r] := p;
  repeat v ⇐ KOLEJKA;
    POTWE[v] := POTWE[v] − Q[v];
    POTWY[v] := POTWY[v] − Q[v];
    POT[v] := POT[v] − Q[v];
    if POT[v] = 0 then STOS ⇐ v;
    if v = t then
    else
      begin (* rozładowanie wierzchołka v *)
        u := pierwszy wierzchołek na liście XZA[v];
        while Q[v] > 0 do
          begin if Q[u] = 0 then KOLEJKA ⇐ u;
            delta := min(Q[v], XC[v, u] − XF[v, u]);
            XF[v, u] := XF[v, u] + delta;
            Q[v] := Q[v] − delta;
            Q[u] := Q[u] + delta;
            if XF[v, u] = XC[v, u] then (* usuń ⟨v, u⟩ *)
              begin XZA[v] := XZA[v] \ {u};
                XPRZED[u] := XPRZED[u] \ {v}
              end;
            if Q[v] > 0 then
              u := następny po u wierzchołek listy XZA[v]
          end
      end (* koniec rozładowania wierzchołka v *)
  until v = t; (* przepływ z r do t znaleziony *)
  (* znalezienie przepływu o wartości p z s do r *)
  KOLEJKA := Ø; KOLEJKA ⇐ r; Q[r] := p;
  repeat v ⇐ KOLEJKA;
    if v ≠ r then (* POT[v] jeszcze nie zmniejszony *)
      begin POTWE[v] := POTWE[v] − Q[v];
        POTWY[v] := POTWY[v] − Q[v];
        POT[v] := POT[v] − Q[v];
        if POT[v] = 0 then STOS ⇐ v
      end;
```

```
75              if v = s then Q[v] := 0
76              else
77                begin (* rozładowanie wierzchołka v *)
78                  u := pierwszy wierzchołek na liście PRZED[v];
79                  while Q[v] > 0 do
80                    begin if Q[u] = 0 then KOLEJKA ⇐ u;
81                      delta := min(Q[v], XC[u, v] − XF[u, v]);
82                      XF[u, v] := XF[u, v] + delta;
83                      Q[v] := Q[v] − delta;
84                      Q[u] := Q[u] + delta;
85                      if XF[u, v] = XC[u, v] then (* usuń ⟨u, v⟩ *)
86                        begin XPRZED[v] := XPRZED[v] \ {u};
87                              XZA[u] := XZA[u] \ {v}
88                        end;
89                      if Q[v] > 0 then
90                        u := następny po u wierzchołek listy XPRZED[v]
91                    end
92                end (* koniec rozładowania wierzchołka v *)
93            until v = s (* przepływ z s do r znaleziony *)
94          end (* koniec zwiększania przepływu wzdłuż pojedynczej
                wielościeżki *)
95      end (* koniec głównej pętli *)
96  end (* koniec procedury MAXPSA *)
```

Oszacujemy teraz liczbę kroków wykonywanych przez procedurę *MAXPSA*. Wyznaczenie potencjałów (wiersze 3 ÷ 12) wymaga $O(n+m)$ kroków, jako że każda krawędź $\langle u, v\rangle$ jest analizowana co najwyżej dwa razy: przy obliczaniu $POTWY[u]$ i $POTWE[v]$. Całkowita liczba kroków we wszystkich iteracjach głównej pętli, wykonana przez fragment procedury eliminujący wierzchołki o zerowym potencjale (wiersze 17 ÷ 35) jest również $O(n+m)$, gdyż każdy wierzchołek kładziony jest na *STOS* dokładnie raz, a przy zdejmowaniu tego wierzchołka ze stosu są usuwane wszystkie incydentne z nim krawędzie (pętle 19 i 27) tak, że każda krawędź jest usuwana co najwyżej raz. Każda iteracja głównej pętli powoduje wyzerowanie potencjału co najmniej jednego wierzchołka, mianowicie wierzchołka r. Liczba iteracji tej pętli nie przekracza zatem n. Każda iteracja, oprócz zanalizowanego już procesu eleminacji wierzchołków o zerowych potencjałach, zawiera dwie części będące zasadniczo procesami przeszukiwania wszerz – z r do t, oraz z s do r. Złożoność tych części jest rzędu liczby analizowanych krawędzi. Analizowanie krawędzi może być „niszczące" – gdy analizowana krawędź jest nasycana przepływem i usuwana z sieci, lub „nieniszczące", gdy krawędź pozostaje w sieci i może być analizowana w dalszych iteracjach głównej pętli. Oczywiście w czasie wszystkich iteracji głównej pętli

lizujemy w sposób niszczący $O(m)$ krawędzi, natomiast w czasie każdej acji analizujemy w sposób nieniszczący co najwyżej n krawędzi, po jednej każdy odwiedzony wierzchołek. W sumie daje to liczbę kroków we wszystkich acjach głównej pętli $m+n^2$, tzn. $O(n^2)$. Wnioskujemy stąd, że całkowita oność obliczeniowa procedury *MAXPSA* jest $O(n^2)$.

Zakładaliśmy w czasie naszej analizy, że usunięcia krawędzi z naszej sieci rsze 19, 27, 58 i 85) można dokonać w czasie ograniczonym przez stałą. Jest możliwe, jeśli wystąpienie wierzchołka v na liście $XZA[u]$ i wierzchołka u liście $XPRZED[v]$ są wzajemnie powiązane wskaźnikami, oraz każdy ele- t listy zawiera wskaźnik zarówno do poprzedniego, jak do następnego ele- ntu listy.

Możemy teraz zebrać opisane już procedury *PSA* i *MAXPSA* w kompletny rytm znajdowania maksymalnego przepływu.

GORYTM 4.6 (Znajdowanie maksymalnego przepływu w sieci).

ne: Sieć dana przez listy incydencji $PRZED[v]$, $ZA[v]$, $v \in V$, przepustowości $C[u, v]$, $u, v \in V$, źródło $s \in V$ oraz ujście $t \in V$.

niki: Maksymalny przepływ $F[u, v]$, $u, v \in V$.

```
begin
  for u ∈ V do
    for v ∈ V do F[u,v] := 0; (* zerowy przepływ początkowy *)
    repeat PSA; (* skonstruuj pomocniczą sieć acykliczną *)
      if ODL[t] ≠ ∞ then (* przepływ F nie jest maksymalny *)
        begin MAXPSA; (* znajdź maksymalny przepływ w sieci
                         pomocniczej *)
          (* przenieś przepływ z sieci pomocniczej do głównej *)
          for u ∈ XV do (* XV = zbiór wierzchołków sieci pomocniczej *)
            for v ∈ XV do
              begin F[u,v] := F[u,v] + XF[u,v];
                if F[u,v] > C[u,v] then
                  begin F[v,u] := F[v,u] - (F[u,v] - C[u,v]);
                    F[u,v] := C[u,v]
                  end
              end
        end (* koniec fazy *)
    until ODL[t] = ∞ (* przepływ F jest maksymalny *)
end
```

Z przeprowadzonej już analizy procedur *PSA* i *MAXPSA*, oraz z faktu, iż zba faz nie przekracza n (por. wniosek 4.5) otrzymujemy

NIOSEK 4.7

gorytm 4.6 poprawnie konstruuje maksymalny przepływ w czasie $O(n^3)$. ■

Zauważmy, że przez analizę tego algorytmu wykazaliśmy istnienie przepływu maksymalnego w dowolnej sieci, a tym samym dostarczyliśmy brakującego fragmentu do dowodu fundamentalnego twierdzenia o maksymalnym przepływie i minimalnym przekroju (twierdzenie 4.2).

Warto zauważyć, że choć algorytm 4.6 nie podpada bezpośrednio pod schemat kolejnego zwiększania przepływu wzdłuż ścieżek rozszerzających, to jednak metoda użyta w tym algorytmie może być uważana za uogólnienie techniki ścieżek rozszerzających. Zwiększenie przepływu (o wartość równą minimalnemu potencjałowi) odbywa się wzdłuż nieco ogólniejszej struktury niż pojedyncza ścieżka rozszerzająca z s do t. Strukturę tę można nazwać „wielościeżką rozszerzającą" z s do t, przechodzącą przez wierzchołek r (o minimalnym potencjale). Wykorzystanie takiej wielościeżki powoduje wyeliminowanie co najmniej jednego wierzchołka – a nie, tak jak w przypadku pojedynczej ścieżki rozszerzającej, jednej krawędzi – stąd mniejsza liczba iteracji i tym samym lepsze oszacowanie złożoności algorytmu.

Odnotujmy na zakończenie następującą oczywistą lecz ważną własność algorytmu 4.6:

TWIERDZENIE 4.7

Jeśli wszystkie przepustowości krawędzi są liczbami całkowitymi, to maksymalny przepływ f znaleziony przez algorytm 4.6 jest całkowitoliczbowy, tzn. $f(e)$ jest liczbą całkowitą dla każdej krawędzi e. ■

4. Skojarzenia o maksymalnej liczności w grafach dwudzielnych

Skojarzeniem w grafie niezorientowanym $G = \langle V, E \rangle$ nazywamy dowolny zbiór krawędzi $M \subseteq E$ taki, że żadne dwie krawędzie ze zbioru M nie są incydentne ze wspólnym wierzchołkiem. Innymi słowy, M jest skojarzeniem, jeśli dla każdych dwóch krawędzi $e_1, e_2 \in M$ mamy $e_1 = e_2$ lub $e_1 \cap e_2 = \emptyset$. Dla każdej krawędzi $\{u, v\} \in M$ mówimy, że M *kojarzy* u z v, natomiast każdy wierzchołek, który nie jest incydentny z żadną krawędzią skojarzenia będziemy nazywać *wolnym*. Szczególnie interesujące są problemy związane ze skojarzeniami w grafie *dwudzielnym*, tzn. grafie niezorientowanym $G = \langle V, E \rangle$ takim, że zbiór wierzchołków można rozbić na dwa rozłączne zbiory, $V = X \cup Y$, $X \cap Y = \emptyset$ takie, że każda krawędź $e \in E$ ma postać $e = \{x, y\}$, gdzie $x \in X$, $y \in Y$ (por. zad. 4.15). Graf taki oznaczać będziemy zwykle przez $\langle X, Y, E \rangle$ (oznaczenie takie zakłada ustalenie pewnego rozbicia zbioru wierzchołków na zbiory X, Y – nie jest ono na ogół jednoznacznie wyznaczone przez zbiory V i E).

W punkcie tym zajmiemy się problemem znalezienia w danym grafie dwudzielnym skojarzenia o maksymalnej liczności. To klasyczne zagadnienie kombinatoryczne znane jest również pod nazwą „problemu małżeństw". Nazwa ta

pochodzi od następującej interpretacji: Niech X i Y będą odpowiednio zbiorem chłopców i dziewcząt, i niech $\{x, y\} \in E$ oznacza, że chłopiec x zna dziewczynę y. Wtedy każde skojarzenie M odpowiada możliwemu zbiorowi par małżeńskich, w którym każda para utworzona została z chłopca i dziewczyny, którzy się znają, przy czym każda z osób uczestniczy w co najwyżej jednej parze.

Okazuje się, że problem znajdowania skojarzenia o maksymalnej liczności w grafie dwudzielnym daje się w prosty sposób sprowadzić do wyznaczania maksymalnego przepływu w pewnej sieci. Niech $H = \langle X, Y, E\rangle$ będzie dowolnym grafem dwudzielnym. Utwórzmy sieć $S(H)$ o źródle s, ujściu t ($s \neq t$ i $s, t \notin X \cup Y$), zbiorze wierzchołków

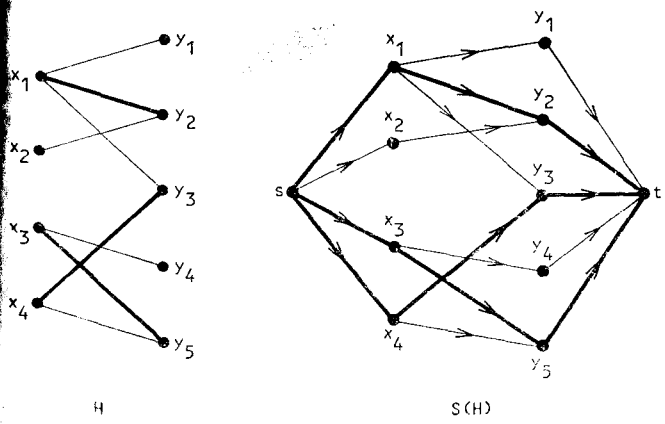
$$V^* = \{s, t\} \cup X \cup Y$$

zbiorze krawędzi

$$E^* = \{\langle s, x\rangle : x \in X\} \cup \{\langle y, t\rangle : y \in Y\} \cup \{\langle x, y\rangle : x \in X \wedge y \in Y \wedge \{x, y\} \in E\}$$

przepustowości $c(u, v) = 1$ dla każdej krawędzi $\langle u, v\rangle \in E^*$.

Na rysunku 4.3 pokazano konstrukcję sieci $S(H)$ dla pewnego grafu dwudzielnego.



Rys. 4.3 Konstrukcja sieci $S(H)$ dla grafu dwudzielnego H oraz przepływ zerojedynkowy odpowiadający skojarzeniu $M = \{\{x_1, y_2\}, \{x_3, y_5\}, \{x_4, y_3\}\}$

Zauważmy, że wobec twierdzenia 4.7 istnieje w $S(H)$ maksymalny przepływ zerojedynkowy, tzn. przepływ maksymalny f taki, że $f(e) = 0$ lub $f(e) = 1$ dla każdej krawędzi $e \in E$.

TWIERDZENIE 4.8

Istnieje wzajemnie jednoznaczna odpowiedniość pomiędzy skojarzeniami w H i przepływami zerojedynkowymi w $S(H)$, przy której skojarzeniu $M =$

$= \{\{x_1, y_1\}, \ldots, \{x_k, y_k\}\}$ o liczności k ($x_i \in X$, $y_i \in Y$ dla $1 \leqslant i \leqslant k$) odpowiada przepływ f_M o wartości k określony następująco:

$$f_M(s, x_i) = f_M(x_i, y_i) = f_M(y_i, t) = 1, \qquad 1 \leqslant i \leqslant k \tag{4.11}$$

oraz $f_M(e) = 0$ dla pozostałych krawędzi e sieci $S(H)$; przepływowi f o wartości k odpowiada natomiast skojarzenie M_f, $|M_f| = k$ określone następująco:

$$M_f = \{\{x, y\} : x \in X \wedge y \in Y \wedge f(x, y) = 1\} \tag{4.12}$$

Dowód ■

Jeśli $M = \{\{x_1, y_1\}, \ldots, \{x_k, y_k\}\}$ jest skojarzeniem o liczności k, to wierzchołki $x_1, \ldots, x_k$, jak również $y_1, \ldots, y_k$, są parami różne. Wynika stąd natychmiast, że $Div_{f_M}(x_i) = Div_{f_M}(y_i) = 0$, $1 \leqslant i \leqslant k$, tzn. funkcja f_M określona przez (4.11) jest przepływem z s do t w $S(H)$. Łatwo również zauważyć, że różnym skojarzeniom M odpowiadają różne przepływy f_M. Z drugiej strony, jeśli f jest przepływem zerojedynkowym w $S(H)$, to ilość przepływu wpływającego do (a więc i wypływającego z) każdego wierzchołka $x \in X$ nie przekracza jedności (jedyną krawędzią dochodzącą do x jest krawędź $\langle s, x\rangle$ o przepustowości równej 1). Wynika stąd, że w zbiorze M_f określonym przez (4.12) nie ma krawędzi postaci $\{x, y_1\}$, $\{x, y_2\}$, $y_1 \neq y_2$. Dla podobnych przyczyn nie ma w M_f krawędzi postaci $\{x_1, y\}$, $\{x_2, y\}$, $x_1 \neq x_2$. Tak więc M_f jest skojarzeniem w H. Nietrudno też zauważyć, że odwzorowanie $f \to M_f$, jest odwrotnością odwzorowania $M \to f_M$, tzn. $f_{M_f} = f$ i $M_{f_M} = M$. ■

Twierdzenie 4.8 pozwala użyć do wyznaczenia skojarzenia o maksymalnej liczności dowolnego algorytmu znajdowania maksymalnego przepływu (całkowitoliczbowego). Jeśli skorzystamy z algorytmu 4.6, to skojarzenie o maksymalnej liczności znajdziemy w $O(n^3)$ krokach. Okazuje się jednak, że szczególna postać sieci $S(H)$ pozwala na bardziej efektywny algorytm. Algorytm ten, pochodzący od Hopcrofta i Karpa [33] stosuje ogólny schemat Dinica, tzn. składa się z faz odpowiadających zwiększaniu przepływu wzdłuż najkrótszych ścieżek rozszerzających określonej długości. Jednakże szczególna postać sieci pozwala zarówno na lepsze ograniczenie na liczbę faz, jak i efektywniejszy algorytm znajdowania przepływu pseudomaksymalnego w każdej fazie.

Zauważmy na początek, że każda ścieżka rozszerzająca w $S(H)$ jest nieparzystej długości i składa się, po odrzuceniu pierwszej i ostatniej krawędzi, z ciągu na przemian występujących krawędzi zgodnych i przeciwnych (zaczynającego i kończącego się krawędzią zgodną). Dla danego skojarzenia M, nazwijmy ścieżką naprzemienną (długości $l = 2k+1$, z X do Y) względem M dowolny zbiór krawędzi $P \subseteq E$ postaci

$$P = \{\{x_0, y_1\}, \{y_1, x_1\}, \{x_1, y_2\}, \ldots, \{y_k, x_k\}, \{x_k, y_{k+1}\}\} \tag{4.13}$$

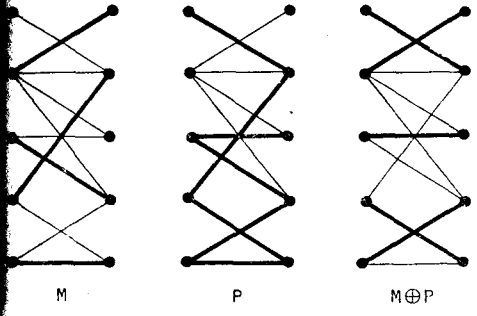
gdzie $k \geqslant 0$, wszystkie wierzchołki $x_0, \ldots, x_k, y_1, \ldots, y_{k+1}$ są różne, x_0 jest wolnym wierzchołkiem w X, y_{k+1} jest wolnym wierzchołkiem w Y, oraz co druga krawędź należy do M, tzn.

$$P \cap M = \{\{y_1, x_1\}, \{y_2, x_2\}, \ldots, \{y_k, x_k\}\}$$

ścieżkę naprzemienną można oczywiście jednoznacznie określić podając ciąg x_0, $y_1, x_1, y_2, \ldots, y_k, x_k, y_{k+1}$. Jest jasne, że przy opisanej w twierdzeniu 4.8 odpowiedniości między skojarzeniami w H a przepływami zerojedynkowymi w sieci $S(H)$ ścieżka naprzemienna (4.13) w naturalny sposób odpowiada pewnej ścieżce rozszerzającej względem przepływu f_M w $S(H)$, mianowicie ścieżce

$$s, \langle s, x_0\rangle, x_0, \langle x_0, y_1\rangle, y_1, \langle x_1, y_1\rangle, x_1, \langle x_1, y_2\rangle, y_2, \ldots$$
$$\ldots, y_k, \langle x_k, y_k\rangle, x_k, \langle x_k, y_{k+1}\rangle, y_{k+1}, \langle y_{k+1}, t\rangle, t$$

przy czym zwiększenie przepływu f_M wzdłuż tej ścieżki daje przepływ odpowiadający skojarzeniu określonemu przez różnicę symetryczną $M \oplus P$. Zilustrowano to na rysunku 4.4.



Rys. 4.4 Powiększanie skojarzenia M wzdłuż ścieżki naprzemiennej P

Z powyższych rozważań bezpośrednio wynika następujące twierdzenie:

TWIERDZENIE 4.9 (Berge)

Skojarzenie M w grafie dwudzielnym H ma maksymalną liczność wtedy i tylko wtedy, gdy nie istnieje w H ścieżka naprzemienna względem M ■

Przyjrzyjmy się teraz jak wygląda przepływ pseudomaksymalny w pomocniczej sieci acyklicznej skonstruowanej dla sieci $S(H)$ i pewnego przepływu całkowitoliczbowego – a więc zerojedynkowego – w tej sieci. Potencjał każdego wierzchołka różnego od s i t wynosi 0 lub 1, gdyż potencjał wejściowy każdego wierzchołka $x \in X$ i potencjał wyjściowy każdego wierzchołka $y \in Y$ w sieci pomocniczej jest równy jedności. Nietrudno zauważyć, że „wielościeżka rozszerzająca" znaleziona przez każdą iterację głównej pętli procedury *MAXPSA* (por. poprzedni punkt) ma postać pojedynczej ścieżki rozszerzającej. Co więcej, wszystkie wierzchołki pośrednie (tzn. różne od s i t) takiej ścieżki mają potencjał równy jedności, a więc po zwiększeniu przepływu wzdłuż tej ścieżki ich potencjał spada do zera i są one usuwane z sieci. Jasne jest teraz, że przepływ w pomocniczej sieci acyklicznej długości $l+2$ odbywa się wzdłuż maksymalnego zbioru dróg długości $l+2$ z s do t o parami rozłącznych zbiorach wierzchołków pośrednich. Odpowiada mu w naturalny sposób maksymalny zbiór ścieżek naprzemiennych długości l o parami rozłącznych zbiorach wierzchołków (przez „maksymalny" rozumiemy taki, który nie daje się powiększyć o dodatkową ścieżkę naprzemienną długości l o zbiorze wierzchołków nie zawierających żadnego wierzchołka dotychczasowych ścieżek).

Algorytm Hopcrofta–Karpa jest opisany szczegółowo poniżej jako algorytm 4.10. W algorytmie tym pomocnicza sieć acykliczna – a właściwie pomocniczy graf acykliczny, jako że przepustowości wszystkich krawędzi są równe jedności – jest konstruowany przez procedurę *PGA*. Aby objaśnić jej działanie dla danego grafu dwudzielnego $H = \langle X, Y, E\rangle$ i skojarzenia M w H, wygodnie jest wprowadzić graf zorientowany H_M przez zorientowanie wszystkich krawędzi $e \in M$ z Y do X, oraz wszystkich krawędzi $e \in E \setminus M$ z X do Y (drogi w H_M z wolnego wierzchołka w X do wolnego wierzchołka w Y odpowiadają dokładnie ścieżkom naprzemiennym względem M). Procedura *PGA* konstruuje najpierw krawędzie ze źródła s do wszystkich wolnych wierzchołków $x \in X$ (wiersze 6 ÷ 10). Następnie jest przeszukiwany wszerz graf H_M, poczynając od wolnych wierzchołków w X. Po znalezieniu każdego wolnego wierzchołka $y \in Y$ dodajemy do pomocniczego grafu acyklicznego krawędź $\langle y, t\rangle$ (wiersze 14 ÷ 17), przy czym od momentu dodania pierwszej takiej krawędzi nie rozpatrujemy już w procesie przeszukiwania wierzchołków znajdujących się w większej lub równej odległości od s niż t (p. warunek w wierszu 20). Powiększanie aktualnego skojarzenia wzdłuż maksymalnego zbioru najkrótszych ścieżek rozszerzających o parami rozłącznych zbiorach wierzchołków jest realizowane przez procedurę *FAZA*. Przeszukuje ona pomocniczy graf acykliczny w głąb, poczynając od s. Główna pętla w wierszu 40 przypomina nierekurencyjną procedurę przeszukiwania w głąb *WG1* (por. p. 2.2). Za każdym razem, gdy w naszym procesie osiągamy ujście t (p. wiersz 50), zawartością stosu *STOS* jest ciąg wierzchołków reprezentujący pewną ścieżkę naprzemienną (zauważmy, że ujście t nie jest kładzione na *STOS*, a tym samym przez cały czas $NOWY[t]$ = **true** i może ono być odwiedzane wielokrotnie). Pętla 52 dokonuje modyfikacji tablicy *SKOJ* odpowiadającej zwiększeniu skojarzenia wzdłuż takiej ścieżki. Zauważmy, że jeśli w trakcie wykonywania naszej procedury pewnemu wierzchołkowi $v \in V$ przypisywana jest wartość $NOWY[v] :=$ **false**, to procedura ta albo znajduje potem ścieżkę naprzemienną przechodzącą przez ten wierzchołek, albo też stwierdza, że nie istnieje żadna ścieżka przechodząca przez ten wierzchołek i nie przecinająca żadnej z uprzednio znalezionych ścieżek. Stąd łatwy wniosek, że procedura *FAZA* znajduje maksymalny zbiór nieprzecinających się najkrótszych ścieżek naprzemiennych. Poprawność całego algorytmu wynika więc z analizy metody Dinica przeprowadzonej w poprzednim punkcie.

ALGORYTM 4.10 (Znajdowanie skojarzenia o maksymalnej liczności metodą Hopcrofta–Karpa)

Dane: Graf dwudzielny $H = \langle X, Y, E\rangle$ reprezentowany przez listy incydencji $ZA[x]$, $x \in X$.

Wyniki: Skojarzenie o maksymalnej liczności reprezentowane przez tablicę *SKOJ* (*SKOJ* $[v]$ jest wierzchołkiem skojarzonym z v lub zerem jeśli v jest wolny).

```
1   procedure PGA;
    (* konstruowanie pomocniczego grafu acyklicznego;
       zmienne V, XV, X, Y, SKOJ, ZA, XZA, s, t są globalne *)
2   begin
3     for u ∈ V ∪ {s, t} do (* inicjalizacja *)
4       begin ODL[u] := ∞; XZA[u] := Ø
5       end;
      (* umieść wolne wierzchołki x ∈ X w kolejce i na liście XZA[s] *)
6     KOLEJKA := Ø; XV := {s}; ODL[s] := 0;
7     for x ∈ X do
8       if SKOJ[x] = 0 then (* x wolny *)
9         begin KOLEJKA ⇐ x; XZA[s] := XZA[s] ∪ {x}; ODL[x] := 1
10        end;
      (* przeszukiwanie wszerz poczynając od wolnych wierzchołków w X *)
11    while KOLEJKA ≠ Ø do
12      begin u ⇐ KOLEJKA; XV := XV ∪ {u};
13        if u ∈ Y then
14          if SKOJ[u] = 0 then (* u wolny, dodaj krawędź ⟨u, t⟩ *)
15            begin XZA[u] := XZA[u] ∪ {t}; XV := XV ∪ {t};
16              ODL[t] := ODL[u]+1
17            end
18          else (* SKOJ[u] ≠ 0 *)
19            begin x := SKOJ[u];
20              if ODL[t] = ∞ then (* dodaj krawędź ⟨u, x⟩ *)
21                begin KOLEJKA ⇐ x; XZA[u] := XZA[u] ∪ {x};
22                    ODL[x] := ODL[u]+1
23                end
24            end
25        else (* u ∈ X *)
26          for y ∈ ZA[u] do
27            if ODL[u] < ODL[y] then (* ODL[y] = ODL[u]+1 lub ∞ *)
28              begin if ODL[y] = ∞ then KOLEJKA ⇐ y; (* y nowy *)
29                XZA[u] := XZA[u] ∪ {y}; ODL[y] := ODL[u]+1
30              end
31      end
32  end; (* PGA *)
33  procedure FAZA;
    (* zwiększenie aktualnego skojarzenia wzdłuż maksymalnego zbioru dróg z s
       do t o parami rozłącznych zbiorach wierzchołków pośrednich w pomo-
       cniczym grafie acyklicznym; zmienne XV, POCZ XZA, SKOJ, s, t są
       globalne *)
34  begin
35    for u ∈ XV do (* inicjalizacja *)
```

```
36      begin NOWY[u] := true;
37        P[u] := POCZ[u] (* POCZ[u] = wskaźnik do początku listy
                    XZA[u]; P[u] = wskaźnik do aktualnie analizowanego ele-
                    mentu listy XZA[u] *)
38      end;
      (* przeszukiwanie w głąb pomocniczego grafu acyklicznego poczynając
        od źródła s *)
39    STOS := Ø; STOS ⇐ s; NOWY[s] := false;
40    while STOS ≠ Ø do (* główna pętla *)
41      begin u := top(STOS); (* u = szczytowy element stosu *)
42        (* znajdź pierwszy nowy wierzchołek na liście XZA[u] *)
43        if P[u] = nil then b := false else b := not NOWY[P[u] ↑.wierzch];
44        while b do
45          begin P[u] := P[u] ↑.nast;
46            if P[u] = nil then b := false
47            else b := not NOWY[P[u] ↑.wierzch]
48          end;
49        if P[u] ≠ nil then (* nowy wierzchołek znaleziony *)
50          if P[u] ↑.wierzch = t then (* znaleziona ścieżka naprzemienna *)
51            (* zwiększenie aktualnego skojarzenia wzdłuż ścieżki naprzemien-
                nej reprezentowanej na stosie *)
52              while top(STOS) ≠ s do
53                begin y ⇐ STOS; x ⇐ STOS;
54                  SKOJ[x] := y; SKOJ[y] := x
55                end
56              (* na stosie pozostaje jedynie źródło s *)
57          else (* P[u] ↑.wierzch ≠ t *)
58            begin v := P[u] ↑.wierzch; STOS ⇐ v; NOWY[v] := false
59            end
          else (* P[u] = nil, tzn. nie ma już nowych wierzchołków na liście
                XZA[u] *)
60        u ⇐ STOS (* zdejmij szczytowy element stosu *)
61      end
62  end; (* FAZA *)
63  begin (* program główny *)
64    for v ∈ V do SKOJ[v] := 0; (* inicjalizacja: skojarzenie puste *)
65    PGA; (* skonstruuj pomocniczy graf acykliczny *)
66    repeat FAZA; PGA
67    until not XV[t]
68  end
```

Podstawowym faktem odpowiedzialnym za dużą efektywność algorytmu Hopcrofta–Karpa jest to, że liczba faz jest rzędu $\sqrt{n}$. Do dowodu ograniczenia na liczbę faz potrzebny nam będzie następujący lemat:

LEMAT 4.11

Niech M i N będą dwoma skojarzeniami w grafie dwudzielnym $H = \langle X, Y, E\rangle$ niech $|M| = r < s = |N|$. Różnica symetryczna $M \oplus N$ zawiera wtedy co najmniej $s - r$ ścieżek naprzemiennych względem M, o parami rozłącznych zbiorach wierzchołków.

DOWÓD

Rozważmy graf $H^* = \langle X, Y, M \oplus N\rangle$, oraz oznaczmy przez $C_1, \ldots, C_p$ składowe spójne tego grafu. Każdy wierzchołek grafu H^* jest incydentny z co najwyżej jedną krawędzią z $M \setminus N$ i z co najwyżej jedną krawędzią z $N \setminus M$ (gdyż M i N są skojarzeniami). Wynika stąd łatwo, że każda składowa C_i ma jedną z następujących trzech postaci:

(1) izolowany wierzchołek;

(2) cykl parzystej długości o krawędziach na przemian w $M \setminus N$ i $N \setminus M$;

(3) droga o krawędziach na przemian w $M \setminus N$ i $N \setminus M$ (nie musi to być ścieżka naprzemienna: oba jej końce mogą należeć do X, lub oba do Y).

Oznaczmy przez E_i zbiór krawędzi składowej C_i i rozważmy wielkość $\delta_i = |E_i \cap N| - |E_i \cap M|$. Mamy $\delta_i \in \{-1, 0, 1\}$, przy czym $\delta_i = 1$ wtedy i tylko wtedy, gdy C_i jest ścieżką naprzemienną względem M. Co więcej $\delta_i = 1$ dla co najmniej $s - r$ wskaźników, jako że

$$\sum_{i=1}^{p} \delta_i = \sum_{i=1}^{p} (|E_i \cap N| - |E_i \cap M|) = \sum_{i=1}^{p} |E_i \cap N| - \sum_{i=1}^{p} |E_i \cap M| =$$
$$= |N \setminus M| - |M \setminus N| = |N| - |M| = s - r$$

Dowód lematu jest tym samym zakończony. ■

Możemy teraz przystąpić do dowodu zapowiedzianego już ograniczenia na liczbę faz algorytmu Hopcrofta–Karpa.

TWIERDZENIE 4.12

Liczba faz algorytmu Hopcrofta–Karpa nie przekracza $2\lfloor\sqrt{s}\rfloor + 1$, gdzie s jest maksymalną licznością skojarzenia w danym grafie.

DOWÓD

Oznaczmy przez $P_0, P_1, \ldots, P_{s-1}$ kolejne ścieżki naprzemienne konstruowane przez algorytm, oraz zdefiniujmy $M_0 = \emptyset$, $M_i = M_{i-1} \oplus P_{i-1}$ dla $1 \leqslant i \leqslant s$. Zgodnie z algorytmem, P_i jest ścieżką naprzemienną względem M_i, i wiemy już że $|P_0| \leqslant |P_1| \leqslant \ldots \leqslant |P_{s-1}|$ (por. lemat 4.4). Liczba faz to nic innego jak ilość różnych liczb w ciągu $|P_0|, \ldots, |P_{s-1}|$. Oznaczmy $r = \lfloor s - \sqrt{s} \rfloor$ i niech r^* będzie najmniejszym indeksem takim, że $|P_{r^*}| = |P_r|$. Zgodnie z konstrukcją pomocniczego grafu acyklicznego, P_{r^*} jest najkrótszą ścieżką naprzemienną względem M_{r^*} (w istocie P_i jest najkrótszą ścieżką naprzemienną względem M_i dla każdego i, lecz fakt ten wymagałby dowodu, por. zad. 4.11). Wobec lematu 4.11 istnieje co

najmniej $|M_s| - |M_{r^*}| = s - r^*$ rozłącznych ścieżek naprzemiennych względem M_{r^*}, zatem najkrótsza ścieżka naprzemienna względem M_{r^*} zawiera co najwyżej $r^*/(s-r^*)$ krawędzi skojarzenia M_{r^*}.
Stąd

$$|P_r| = |P_{r^*}| \leqslant \frac{2r^*}{s-r^*} + 1 \leqslant \frac{2r}{s-r} + 1 = \frac{2\lfloor s - \sqrt{s} \rfloor}{\lceil \sqrt{s} \rceil} + 1 = \frac{2s}{\lceil \sqrt{s} \rceil} - 1 \leqslant$$

$$\leqslant 2\sqrt{s} - 1 \leqslant 2\lfloor \sqrt{s} \rfloor + 1$$

Długość każdej ścieżki jest nieparzysta, a więc ciąg $|P_0|, \ldots, |P_r|$ zawiera co najwyżej $\lfloor \sqrt{s} \rfloor + 1$ różnych liczb. Ciąg $|P_{r+1}|, \ldots, |P_{s-1}|$ może zawierać co najwyżej $(s-1)-(r+1)+1 = s-r-1 = \lceil \sqrt{s} \rceil - 1 \leqslant \lfloor \sqrt{s} \rfloor$ dalszych liczb, co w sumie daje żądane ograniczenie $2\lfloor \sqrt{s} \rfloor + 1$. ■

Łatwo już teraz oszacować złożoność całego algorytmu 4.10. Zarówno przeszukiwanie wszerz w procedurze *PGA*, jak i przeszukiwanie w głąb w procedurze *FAZA* wymaga $O(m+n)$ kroków, co przy liczbie faz rzędu $\sqrt{n}$ daje całkowitą złożoność $O((m+n)\sqrt{n})$ lub, używając jedynie liczby wierzchołków jako wymiaru problemu, $O(n^{5/2})$.

Warto wspomnieć, że znany jest algorytm o złożoności $O(n^{5/2})$ znajdowania skojarzenia o maksymalnej liczności w dowolnym, niekoniecznie dwudzielnym grafie (por. poz. [19]). Algorytm ten jest jednak bez porównania bardziej skomplikowany. Inny od przytoczonego tu, algorytm dla przypadku grafu dwudzielnego podał Galil w pracy [26].

4. Systemy różnych reprezentantów

Niech $\langle A_1, \ldots, A_n \rangle$ będzie dowolnym ciągiem zbiorów (niekoniecznie rozłącznych i niekoniecznie różnych). *Systemem różnych reprezentantów* dla $\langle A_1, \ldots, A_n \rangle$ nazywamy dowolny ciąg $\langle a_1, \ldots, a_n \rangle$ taki, że $a_i \in A_i$, $1 \leqslant i \leqslant n$ oraz $a_i \neq a_j$ dla $i \neq j$. Mówimy, że w takim systemie różnych reprezentantów element a_i *reprezentuje* zbiór A_i. Problem istnienia i konstrukcji systemu różnych reprezentantów znany jest w wielu nieformalnych sformułowaniach. Jednym z nich jest tzw. „problem komisji": Mamy n komisji, przy czym A_i jest zbiorem osób należących do i-tej komisji. Należy wybrać w każdej komisji przewodniczącego tak, by żadna osoba nie przewodniczyła więcej niż jednej komisji.

Nietrudno zauważyć, że problem komisji sprowadza się do szczególnego przypadku problemu małżeństw. Istotnie, utwórzmy zbiory

$$X = A_1 \cup \ldots \cup A_n = \{x_1, \ldots, x_m\} \tag{4.14}$$

$$Y = \{y_1, y_n\} \tag{4.15}$$

(elementy $x_1, \ldots, x_m, y_1, \ldots, y_n$ są parami różne), oraz

$$E = \{\{x_i, y_j\} : 1 \leqslant i \leqslant m \wedge 1 \leqslant j \leqslant n \wedge x_i \in A_j\} \tag{4.16}$$

Oczywiście każdy system różnych reprezentantów $\langle a_1, \ldots, a_n \rangle$ odpowiada jednoznacznie skojarzeniu o liczności n w grafie dwudzielnym $H = \langle X, Y, E \rangle$, mianowicie skojarzeniu $\{\{a_1, y_1\}, \ldots, \{a_n, y_n\}\}$.

Biorąc pod uwagę fakt, że graf H ma $m+n$ wierzchołków i $\sum_{j=1}^{n} |A_j|$ krawędzi, otrzymujemy

WNIOSEK 4.13

Dla danego ciągu zbiorów $\langle A_1, \ldots, A_n \rangle$ możemy znaleźć system różnych reprezentantów lub też stwierdzić, że żaden taki system nie istnieje, w czasie $O(\sqrt{n} \sum_{j=1}^{n} |A_j|)$. ∎

DOWÓD

Liczba faz w algorytmie Hopcrofta–Karpa zastosowanym do grafu H nie przekracza $2\lfloor\sqrt{n}\rfloor+1$ (por. twierdzenie 4.12). Stąd liczba kroków algorytmu jest $O(\sqrt{n}(n+\sum_{j=1}^{n} |A_j|))$. Możemy zakładać, że wszystkie zbiory A_j są niepuste, w przeciwnym razie system różnych reprezentantów oczywiście nie istnieje. Przy takim założeniu $n+\sum_{j=1}^{n} |A_j| \leqslant 2\sum_{j=1}^{n} |A_j|$, co daje żądane oszacowanie złożoności algorytmu. ∎

Udowodnimy teraz klasyczne twierdzenie Ph. Halla [30] dające warunek konieczny i dostateczny istnienia systemu różnych reprezentantów dla danego ciągu zbiorów.

TWIERDZENIE 4.14

System różnych reprezentantów dla ciągu $\langle A_1, \ldots, A_n \rangle$ istnieje wtedy i tylko wtedy, gdy

$$|\bigcup_{j \in J} A_j| \geqslant |J|, \qquad \text{dla każdego } J \subseteq \{1, \ldots, n\} \tag{4.17}$$

DOWÓD

Jeśli istnieje system różnych reprezentantów $\langle a_1, \ldots, a_n \rangle$, to mamy dla każdego J

$$|\bigcup_{j \in J} A_j| \geqslant |\bigcup_{j \in J} \{a_j\}| = |J|$$

Załóżmy teraz, że nie istnieje system różnych reprezentantów dla $\langle A_1, \ldots, A_n \rangle$, i rozważmy skojarzenie M o maksymalnej liczności w grafie $H = \langle X, Y, E \rangle$, odpowiadającym ciągowi $\langle A_1, \ldots, A_n \rangle$ (por. ze wzorami (4.14), (4.15), (4.16)). Oczywiście $|M| \leqslant n$, a więc istnieje w Y wierzchołek wolny, powiedzmy b_0. Oznaczmy przez X_0 zbiór tych wierzchołków $x \in X$, do których istnieje „częściowa ścieżka naprzemienna" z b_0 postaci

$$b_0, a_1, b_1, \ldots, b_{k-1}, a_k$$

gdzie $k \geqslant 1$, $a_k = x$, $\{a_i, b_i\} \in M$ dla $1 \leqslant i \leqslant k-1$, oraz $\{b_{i-1}, a_i\} \in E \setminus M$ dla $1 \leqslant i \leqslant k$. X_0 nie zawiera oczywiście wierzchołków wolnych, jako że taki

wierzchołek odpowiadałby ścieżce naprzemiennej, wbrew założeniu, że M ma maksymalną liczność. Niech Y_0 będzie zbiorem wierzchołków skojarzonych przez M z wierzchołkami z X_0. Oznaczmy $J = \{j: 1 \leqslant j \leqslant n \wedge y_j \in Y_0 \cup \{b_0\}\}$. Nietrudno zauważyć, że

$$\bigcup_{j \in J} A_j = X_0$$

a więc

$$|\bigcup_{j \in J} A_j| = |X_0| = |Y_0| = |J| - 1 < |J|$$

tzn. warunek (4.17) nie jest spełniony. ∎

Oczywiście twierdzenie Halla nie ma większego znaczenia z algorytmicznego punktu widzenia, gdyż sprawdzenie warunku (4.17) wymaga rozważenia wszystkich 2^n możliwych zbiorów $J \subseteq \{1, \ldots, n\}$.

Udowodnimy jeszcze jedno twierdzenie związane ze skojarzeniami w grafie dwudzielnym. Nazwijmy *pokryciem wierzchołkowym* w grafie $G = \langle V, E \rangle$ dowolny zbiór $P \subseteq V$ taki, że każda krawędź $e \in E$ jest incydentna z pewnym wierzchołkiem ze zbioru P. Zauważmy, że P jest pokryciem wierzchołkowym wtedy i tylko wtedy, gdy $V \setminus P$ jest niezależnym zbiorem wierzchołków, tzn. zbiorem wierzchołków, z których żadne dwa nie są połączone krawędzią. W dalszym ciągu będą nas interesować pokrycia wierzchołkowe o minimalnej liczności w grafach dwudzielnych.

Niech M będzie skojarzeniem o maksymalnej liczności w grafie dwudzielnym $H = \langle X, Y, E \rangle$, i niech A będzie zbiorem wierzchołków $a \in X \cup Y$ osiągalnych z wolnych wierzchołków w X przez „częściowe ścieżki naprzemienne", analogicznie do określonych w dowodzie twierdzenia Halla.

LEMAT 4.15

Jeśli graf H nie zawiera izolowanych wierzchołków, to $P = (X \setminus A) \cup (Y \cap A)$ jest pokryciem wierzchołkowym o minimalnej liczności, natomiast $N = (X \cap A) \cup (Y \setminus A)$ jest niezależnym zbiorem wierzchołków o maksymalnej liczności.

DOWÓD

Pokażemy najpierw, że P jest pokryciem wierzchołkowym. Przypuśćmy, że pewna krawędź $\{x, y\} \in E$, $x \in X$, $y \in Y$ nie jest incydentna z żadnym z wierzchołków w P. Oznacza to, iż $x \in A$ i $y \notin A$. Nie może być $\{x, y\} \in M$, gdyż w takim przypadku wierzchołek x może być osiągnięty jedynie przez ścieżkę przechodzącą przez y, co pociągałoby $y \in A$. Lecz sytuacja, w której $\{x, y\} \in E \setminus M$ jest również niemożliwa, gdyż ścieżkę prowadzącą do x można by powiększyć o krawędź $\{x, y\}$, co znów pociągałoby $y \in A$. Otrzymana sprzeczność dowodzi, iż P jest pokryciem wierzchołkowym. Zauważmy, że żaden wierzchołek w P nie jest wolny. Dla wierzchołków z $X \setminus A$ wynika to bezpośrednio z definicji zbioru A, natomiast dla wierzchołków z $Y \cap A$ stąd, iż wolny wierzchołek w $Y \cap A$ odpowiadałby

cieżce naprzemiennej względem M, wbrew naszemu założeniu o maksymalnoci $|M|$. W podobny sposób dowodzimy, że co najwyżej jeden wierzchołek dolnej krawędzi $\{x, y\} \in M$ należy do zbioru P: Jeśli $y \in Y \cap A$, to $x \in A$, i tym mymy $x \notin P$. Stąd wniosek, że $|P| \leqslant |M|$. Lecz każde pokrycie wierzchołkowe usi z definicji zawierać co najmniej jeden wierzchołek każdej krawędzi, a dla rawędzi skojarzenia wierzchołki te są oczywiście parami różne. Musi więc być wsze $|P| \geqslant |M|$, co dowodzi, że nasze pokrycie wierzchołkowe ma minimalną zność, i że $|P| = |M|$. Druga część lematu wynika z pierwszej na mocy naszych oprzednich uwag i tego, iż $N = (X \cup Y) \setminus P$. ∎

Odnotujmy wniosek z powyższego dowodu (zauważmy, że wierzchołki zolowane nie mają żadnego wpływu ani na skojarzenia, ani na pokrycia wierzchołkowe).

NIOSEK 4.16

dowolnym grafie dwudzielnym minimalna liczność pokrycia wierzchołkowego st równa maksymalnej liczności skojarzenia. ∎

Warto podać tu inne równoważne sformułowanie powyższego faktu w terinach macierzy zerojedynkowych. Przez *linię* takiej macierzy będziemy rozumieć wiersz lub kolumnę. Poniższe twierdzenie zwane jest często „twierdzeniem ęgierskim".

WIERDZENIE 4.17 (König i Egerváry).

la dowolnej macierzy zerojedynkowej maksymalna liczność zbioru (wystąpień) dynek, z których żadne dwie nie leżą na jednej linii jest równa minimalnej liczie linii, którymi można pokryć wszystkie jedynki.

OWÓD

la danej macierzy zerojedynkowej $A = [a_{ij}]$ o n wierszach i m kolumnach twórzmy graf dwudzielny $H = \langle X, Y, E \rangle$, gdzie $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$ oraz

$$E = \{\{x_i, y_j\} : 1 \leqslant i \leqslant n \wedge 1 \leqslant j \leqslant m \wedge a_{ij} = 1\}$$

atwo zauważyć, że zbiory jedynek w A, z których żadne dwie nie leżą na tej mej linii odpowiadają dokładnie skojarzeniom w H, natomiast każdy zbiór linii w A pokrywających wszystkie jedynki reprezentuje pokrycie wierzchołkowe H. Nasze twierdzenie jest więc po prostu innym sformułowaniem wniosku .16. ∎

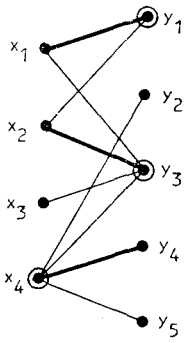
Na rysunku 4.5 przedstawiono graf dwudzielny wraz ze skojarzeniem o maksymalnej liczności i pokryciem wierzchołkowym o minimalnej liczności, oraz odpowiadającą macierz zerojedynkową ilustrującą twierdzenie węgierskie.

Zauważmy, że zbiór A występujący w lemacie 4.15 można znaleźć w czasie $O(|X|+|Y|+|E|)$ stosując przeszukiwanie w głąb (lub wszerz). Jeśli skorzystamy z algorytmu Hopcrofta–Karpa i zauważymy, że każdy wierzchołek izolo-

wany może być usunięty z dowolnego pokrycia wierzchołkowego i może być dodany do każdego niezależnego zbioru wierzchołków, to otrzymamy następujący wniosek:

WNIOSEK 4.18

Zarówno pokrycie wierzchołkowe o minimalnej liczności, jak i niezależny zbiór wierzchołków o maksymalnej liczności w dowolnym grafie dwudzielnym $H = \langle X, Y, E\rangle$ mogą być znalezione w czasie $O((m+n)\sqrt{n})$, gdzie $n = |X \cup Y|$, $m = |E|$. ∎



	y_1	y_2	y_3	y_4	y_5
x_1	1	0	1	0	0
x_2	1	0	1	0	0
x_3	0	0	1	0	0
x_4	0	1	1	1	1

Rys. 4.5 Ilustracja twierdzenia węgierskiego

Warto wspomnieć, że dla powyższych problemów nie jest znany żaden algorytm wielomianowy w przypadku dowolnych grafów, niekoniecznie dwudzielnych.

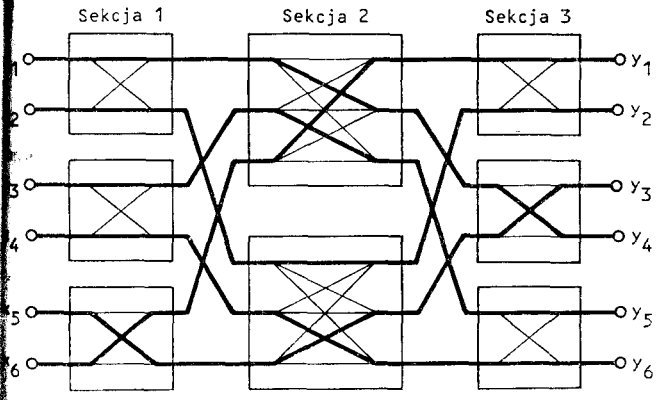
Na zakończenie tego punktu zajmiemy się jeszcze problemem istnienia wspólnego systemu różnych reprezentantów dla dwóch ciągów zbiorów, $\langle A_1, \ldots, A_n\rangle$ i $\langle B_1, \ldots, B_n\rangle$. Rozumienie takiego systemu jako po prostu systemu różnych reprezentantów zarówno dla pierwszego, jak i drugiego ciągu nie prowadzi do niczego ciekawego: sytuacja sprowadza się po prostu do rozpatrywania ciągu $\langle A_1 \cap B_1, \ldots, A_n \cap B_n\rangle$. Dlatego też zdefiniujemy *wspólny system różnych reprezentantów* dla naszych ciągów jako dowolny ciąg $\langle a_1, \ldots, a_n\rangle$ o tej własności, że istnieje permutacja σ zbioru $\{1, \ldots, n\}$ taka, że $\langle a_1, \ldots, a_n\rangle$ jest systemem różnych reprezentantów dla $\langle A_1, \ldots, A_n\rangle$ i dla $\langle B_{\sigma(1)}, \ldots, B_{\sigma(n)}\rangle$. Problem istnienia i konstrukcji takiego systemu łatwo sprowadzić (por. poz. [23]) do znajdowania.

Pokażemy teraz zastosowanie powyższego twierdzenia do teorii sieci komutacyjnych stosowanych w telefonii. Załóżmy, że dane są dwa zbiory „abonentów" X i Y, oraz niech $|X| = |Y| = N$. Naszym zadaniem jest zaprojektować urządzenie, składające się z pojedynczych zestyków, które byłoby w stanie zrealizować układ połączeń określony przez dowolne, wzajemnie jednoznaczne odwzorowanie $\varphi\colon X \to Y$. Urządzenie takie nazywać będziemy *siecią przestrajalną wymiaru* $N \times N$. Zakładamy, że każdy zestyk może się znajdować w jednym z dwóch stanów: zwarty i rozwarty. Najprostszym rozwiązaniem, lecz wymagającym aż N^2 zestyków, jest zastosowanie oddzielnego zestyku między każdą parą abonen-

ów $\langle x, y\rangle \in X \times Y$. Realizacja odwzorowania φ polega na zwarciu zestyków dla ar $\langle x, \varphi(x)\rangle$, $x \in X$ oraz rozwarciu wszystkich pozostałych. Urządzenie takie azywamy *komutatorem wymiaru* $N \times N$.

Zauważmy, że każda sieć przestrajalna będąc w stanie zrealizować wszykie $N!$ (wzajemnie jednoznacznych) odwzorowań z X na Y, musi mieć co najniej $N!$ możliwych stanów. Skoro stan jest określony przez kombinację stanów oszczególnych (dwustanowych) zestyków, to liczba zestyków musi być równa co ajmniej $\log N! = \Omega(N \log N)$. Poniżej pokażemy, że istnieje sieć przestrajalna awierająca $O(N \log N)$ zestyków.

Załóżmy, że $N = nk$, gdzie n i k są liczbami całkowitymi, większymi od dności. *Trójsekcyjną sieć Closa* (por. poz. [4], [7], [50]) typu $\langle n, k\rangle$ tworzymy trzech *sekcji*: pierwszej składającej się z n komutatorów wymiaru $k \times k$, drugiej kładającej się z k komutatorów wymiaru $n \times n$ i trzeciej składającej się – tak k pierwsza – z n komutatorów wymiaru $k \times k$. Sekcje te połączone są ze sobą następujący sposób: i-te wyjście j-tego komutatora sekcji pierwszej połączone st (na stałe) z j-tym wejściem i-tego komutatora sekcji drugiej, i-te wejście j-tego omutatora sekcji trzeciej połączone jest natomiast z j-tym wyjściem i-tego kotutatora sekcji drugiej, dla $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant k$. Trójsekcyjną sieć Closa pu $\langle 3, 2\rangle$ pokazano na rysunku 4.6.



ys. 4.6 Trójsekcyjna sieć Closa typu $\langle 3, 2\rangle$

WIERDZENIE 4.20. (Slepian [62], Diguid [10]).
Trójsekcyjna sieć Closa jest siecią przestrajalną.

DOWÓD
Niech φ będzie dowolnym wzajemnie jednoznacznym odwzorowaniem X na Y. Oznaczmy przez A_i zbiór wejść i-tego komutatora sekcji pierwszej, przez C_i zbiór wyjść i-tego komutatora sekcji trzeciej, oraz

$$B_i = \{x \in X : \varphi(x) \in C_i\}$$

Jest oczywiste, że zbiory $A_1, \ldots, A_n$ jak również $B_1, \ldots, B_n$ określają dwa podziały zbioru X na bloki liczności k każdy. Na mocy twierdzenia 4.19 zbiór X możemy

przedstawić w postaci rozłącznej sumy $X = X_1 \cup \ldots \cup X_k$, gdzie każdy ze zbiorów X_j jest zbiorem elementów pewnego wspólnego systemu różnych reprezentantów dla $\langle A_1, \ldots, A_n \rangle$ i $\langle B_1, \ldots, B_n \rangle$. Zauważmy, że dla $j = 1, \ldots, k$ możemy połączyć „abonentów" ze zbioru X_j z odpowiednimi „abonentami" w Y poprzez j-ty komutator sekcji środkowej. Istotnie, niech $X_j = \{a_1, \ldots, a_n\}$, gdzie $a_i \in A_i$ oraz $a_i \in B_{\sigma(i)}$, tzn. $\varphi(a_i) \in C_{\sigma(i)}$, dla $1 \leqslant i \leqslant n$ i pewnej permutacji σ zbioru $\{1, \ldots, n\}$. Komutatory sekcji pierwszej zawierające $a_1, \ldots, a_n$, jak również komutatory sekcji trzeciej zawierające $\varphi(a_1), \ldots, \varphi(a_n)$ są parami różne, a więc dla $i = 1, \ldots, n$ ścieżkę realizującą połączenie a_i z $\varphi(a_i)$ możemy utworzyć łącząc a_i z j-tym wyjściem i-tego komutatora sekcji pierwszej, następnie i-te wejście z $\sigma(i)$-tym wyjściem j-tego komutatora sekcji drugiej, i wreszcie j-te wejście $\sigma(i)$-tego komutatora sekcji trzeciej z $\varphi(a_i)$. ■

Na rysunku 4.6 grubszą linią pokazano połączenie realizujące następujące przyporządkowanie φ:

$$\varphi(x_1) = y_4 \qquad \varphi(x_2) = y_2 \qquad \varphi(x_3) = y_5$$
$$\varphi(x_4) = y_6 \qquad \varphi(x_5) = y_3 \qquad \varphi(x_6) = y_1$$

$(X_1 = \{x_1, x_3, x_6\}, X_2 = \{x_2, x_4, x_5\})$.

Załóżmy teraz, że N jest postaci 2^p (jeśli tak nie jest, zwiększamy N do najbliższej potęgi dwójki, tzn. do $2^{\lceil \log N \rceil}$). Utwórzmy trójsekcyjną sieć Closa typu $\langle N/2, 2 \rangle$. Jeśli każdy z dwu komutatorów sekcji środkowej zastąpimy z kolei przez trójsekcyjną sieć Closa typu $\langle N/4, 2 \rangle$, to otrzymana sieć (o pięciu sekcjach) pozostaje oczywiście przestrajalna. Postępowanie to możemy kontynuować aż do momentu, gdy wszystkie komutatory są wymiaru 2×2. Nietrudno zauważyć, że otrzymana sieć przestrajalna składa się z $2p+1$ sekcji, z których każda zawiera $N/2$ komutatorów wymiaru 2×2. Całkowita liczba zestyków równa jest więc

$$(2p+1) \cdot (N/2) \cdot 4 = 2N(2p+1) = 2N(2 \log N + 1) = O(N \log N)$$

Innym zastosowaniem twierdzenia 4.19 jest problem układania rozkładów zajęć. W najprostszym, najbardziej wyidealizowanym sformułowaniu problem ten dany jest przez pewien zbiór zajęć X o liczności nk, oraz przez dwa podziały $X = W_1 \cup \ldots \cup W_n$, $X = S_1 \cup \ldots \cup S_n$, gdzie W_i jest zbiorem zajęć prowadzonych przez i-tego wykładowcę, S_i zaś zbiorem zajęć, które mają być prowadzone w i-tej sali, przy czym zakładamy $|W_i| = |S_i| = k$, $1 \leqslant i \leqslant n$. Konstrukcja dokonana w dowodzie twierdzenia 4.19 dostarcza k wspólnych systemów różnych reprezentantów dla ciągów $\langle W_1, \ldots, W_n \rangle$ i $\langle S_1, \ldots, S_n \rangle$, przy czym systemy te w sumie wyczerpują cały zbiór zajęć X. Jeśli każde z zajęć trwa powiedzmy 1 godzinę, to przeprowadzając zajęcia określone przez j-ty system w ciągu j-tej godziny, $j = 1, \ldots, k$, otrzymujemy rozkład zajęć, w którym wszystkie zajęcia zostają przeprowadzone w ciągu k godzin, przy czym w ciągu tego czasu każda sala i każdy wykładowca jest zawsze wykorzystany. W praktyce układanie

rozkładu zajęć jest bardziej skomplikowane, gdyż występują dodatkowe ograniczenia (częstokroć sprzeczne!) mające swe źródło na przykład w tym, że pewne osoby chcą uczęszczać na wiele zajęć, które tym samym nie mogą się odbywać równocześnie.

4.5 Rozkład na łańcuchy

Rodzinę $\mathscr{C}$ podzbiorów zbioru skończonego X nazywamy *łańcuchem*, jeśli dla dowolnych $A, B \in \mathscr{C}$ mamy $A \subseteq B$ lub $B \subseteq A$. Innymi słowy, $\mathscr{C}$ jest łańcuchem, jeśli rodzinę tę możemy przedstawić jako $\mathscr{C} = \{C_1, \ldots, C_k\}$, gdzie $k = |\mathscr{C}|$ i $C_1 \subset C_2 \subset \ldots \subset C_k$. W punkcie tym zajmiemy się problemem rozkładu rodziny $\mathscr{P}(X)$ wszystkich podzbiorów zbioru X na minimalną liczbę łańcuchów.

Niech $n = |X|$. Łańcuch $\mathscr{C}$ będziemy nazywać *łańcuchem symetrycznym* w $\mathscr{P}(X)$, jeśli jest on postaci

$$C_{\lfloor n/2 \rfloor - j} \subset C_{\lfloor n/2 \rfloor - j + 1} \subset \ldots \subset C_{\lceil n/2 \rceil + j}$$

gdzie $0 \leqslant j \leqslant \lfloor n/2 \rfloor$ i $|C_i| = i$ dla $\lfloor n/2 \rfloor - j \leqslant i \leqslant \lceil n/2 \rceil + j$.

Pokażemy teraz konstrukcję, która z podziału rodziny $\mathscr{P}(X)$ na łańcuchy symetryczne konstruuje podział rodziny $\mathscr{P}(X \cup \{a\})$ $(a \notin X)$ na łańcuchy symetryczne. W tym celu każdy łańcuch symetryczny $A_1 \subset A_2 \subset \ldots \subset A_k$ w $\mathscr{P}(X)$ należący do naszego podziału zastępujemy najpierw następującymi dwoma łańcuchami w $\mathscr{P}(X \cup \{a\})$:

$$A_1 \subset A_2 \subset \ldots \subset A_k \tag{4.18}$$

$$A_1 \cup \{a\} \subset A_2 \cup \{a\} \subset \ldots \subset A_k \cup \{a\} \tag{4.19}$$

Otrzymujemy oczywiście w ten sposób pewien podział rodziny $\mathscr{P}(X \cup \{a\})$ na łańcuchy. Łańcuchy (4.18) i (4.19) nie są symetryczne w $\mathscr{P}(X \cup \{a\})$, lecz nietrudno zauważyć, że można je takimi uczynić przenosząc ostatni zbiór z drugiego łańcucha do pierwszego:

$$A_1 \subset A_2 \ldots \subset A_k \subset A_k \cup \{a\} \tag{4.20}$$

$$A_1 \cup \{a\} \subset A_2 \cup \{a\} \subset \ldots \subset A_{k-1} \cup \{a\} \tag{4.21}$$

Jeśli $k = 1$, to drugi z powyższych łańcuchów znika, tzn. wszystkie podzbiory występujące w (4.18) i (4.19) umieszczamy w jednym łańcuchu symetrycznym.

Startując od $\mathscr{P}(\emptyset) = \{\emptyset\}$, która to rodzina jest sama łańcuchem symetrycznym, i powtarzając opisaną powyżej konstrukcję n razy otrzymujemy podział rodziny $\mathscr{P}(X)$ na łańcuchy symetryczne.

Antyłańcuchem w $\mathscr{P}(X)$ nazywamy dowolną rodzinę $\mathscr{A} \subseteq \mathscr{P}(X)$ taką, że dla dowolnych $A, B \in \mathscr{A}$, $A \neq B$ mamy $A \not\subseteq B$ i $B \not\subseteq A$. Przykładem antyłańcucha jest $\mathscr{P}_k(X)$, rodzina wszystkich k-elementowych podzbiorów zbioru X $(0 \leqslant k \leqslant |X|)$. Niech $\mathscr{P}(X) = \mathscr{C}_1 \cup \ldots \cup \mathscr{C}_p$ będzie dowolnym podziałem rodziny $\mathscr{P}(X)$ na rozłączne łańcuchy i niech $\mathscr{A}$ będzie dowolnym antyłańcuchem

$w\ \mathscr{P}(X)$. Jest jasne, że co najwyżej jeden zbiór $A \in \mathscr{A}$ wpada do każdego z łańcuchów $\mathscr{C}_i$, a co za tym idzie $p \geqslant |\mathscr{A}|$. Mamy więc następujący wniosek:

WNIOSEK 4.21

Liczność każdego antyłańcucha w $\mathscr{P}(X)$ nie przekracza liczby bloków w dowolnym podziale rodziny $\mathscr{P}(X)$ na łańcuchy. ∎

Jeśli udałoby się nam wskazać taki antyłańcuch $\mathscr{A}$ w $\mathscr{P}(X)$ i taki podział na łańcuchy $\mathscr{P}(X) = \mathscr{C}_1 \cup \ldots \cup \mathscr{C}_p$, że $|\mathscr{A}| = p$, to oczywiście $\mathscr{A}$ byłby antyłańcuchem o maksymalnej liczności w $\mathscr{P}(X)$, nasz podział byłby natomiast podziałem rodziny $\mathscr{P}(X)$ na minimalną liczbę łańcuchów. W istocie łatwo się przekonać, że taki antyłańcuch i podział istnieją. Wystarczy przyjąć $\mathscr{A} = \mathscr{P}_{\lfloor n/2 \rfloor}(X)$ $(n = |X|)$, a jako podział rozważyć dowolny podział rodziny $\mathscr{P}(X)$ na łańcuchy symetryczne, na przykład podział otrzymany przez zastosowanie opisanej przez nas konstrukcji rekurencyjnej. Wynika to po prostu stąd, że każdy łańcuch symetryczny zawiera dokładnie jeden zbiór $\lfloor n/2 \rfloor$-elementowy. Skoro łańcuchy są rozłączne i pokrywają $\mathscr{P}(X)$, to musi ich być dokładnie tyle ile podzbiorów $\lfloor n/2 \rfloor$-elementowych. Przypomnijmy, że liczba podzbiorów k-elementowych zbioru n-elementowego jest dana przez współczynnik dwumienny $\binom{n}{k}$ (por. p. 1.6), i uporządkujmy otrzymane fakty:

TWIERDZENIE 4.22 (Sperner [63]).

Liczność dowolnego antyłańcucha w $\mathscr{P}(X)$, $|X| = n$ nie przekracza $\binom{n}{\lfloor n/2 \rfloor}$, tzn. $\mathscr{P}_{\lfloor n/2 \rfloor}(X)$ jest antyłańcuchem o maksymalnej liczności. ∎

TWIERDZENIE 4.23

Każdy podział rodziny $\mathscr{P}(X)$ na łańcuchy symetryczne składa się z $\binom{n}{\lfloor n/2 \rfloor}$ łańcuchów, gdzie $n = |X|$, i jest podziałem rodziny $\mathscr{P}(X)$ na minimalną możliwą liczbę łańcuchów. ∎

Opiszemy teraz nieco dokładniej realizację przedstawionej już rekurencyjnej metody konstrukcji rozkładu rodziny $\mathscr{P}(X)$ na łańcuchy. Przyjmujemy $X = \{1, \ldots, n\}$ i każdy łańcuch symetryczny $\mathscr{C}$ w $\mathscr{P}(X)$ reprezentujemy przez rekord zawierający tablicę $P[1..n]$, oraz zmienne całkowite *pocz*, *kon* takie, że

$$\mathscr{C} = \{\{P[1], P[2], \ldots, P[i]\}\colon pocz \leqslant i \leqslant kon\}$$

Poza tym rekord taki zawiera wskaźnik *nast* do rekordu reprezentującego następny łańcuch podziału.

ALGORYTM 4.24 (Znajdowanie podziału rodziny wszystkich podzbiorów zbioru $\{1, \ldots, n\}$ na łańcuchy symetryczne)

Dane: n

Wyniki: Lista rekordów, z których każdy reprezentuje blok podziału rodziny $\mathscr{P}(\{1, \ldots, n\})$ na łańcuchy symetryczne (*podz* jest wskaźnikiem od początku tej listy).

```
function PODZIAL(k);
(* wartością tej funkcji jest wskaźnik do początku listy rekordów, z których
   każdy reprezentuje łańcuch symetryczny w 𝒫(X), X = {1, ..., k}; wszy-
   stkie rekordy na liście określają podział rodziny 𝒫(X) na łańcuchy sy-
   metryczne *)
begin
  if k = 0 then (* konstrukcja podziału rodziny 𝒫(∅) = {∅} na łańcuchy
                   symetryczne – jedynym łańcuchem jest {∅} *)
    begin new(w); (* w = wskaźnik do rekordu reprezentującego łań-
      cuch *)
      with w ↑ do
        begin pocz := 0; kon := 0; nast := nil
        end
    end
  else (* k > 0*)
    begin w := PODZIAL(k-1);
      pp := w;
      while pp ≠ nil do
        with pp ↑ do
          begin (* łańcuch symetryczny A_1 ⊂ ... ⊂ A_r reprezentowany
            przez rekord pp ↑ jest zamieniany na dwa łańcuchy:
            A_1 ⊂ ... ⊂ A_r ⊂ A_r ∪ {k}, A_1 ∪ {k} ⊂ ... ⊂ A_{r-1} ∪ {k}, przy
            czym drugi jest pomijany jeśli r = 1,
            tzn. gdy pp ↑.pocz = pp ↑.kon *)
            pnast := nast;
            if pocz < kon then (* łańcuch długości r > 1 *)
              begin(* dodaj nowy łańcuch A_1 ∪ {k} ⊂ ... ⊂ A_{r-1} ∪ {k} *)
                new(pn); pn↑.nast := nast; nast := pn;
                pn↑.pocz := pocz+1; pn↑.kon := kon;
                pn ↑.P [1] := k;
                for i := 2 to kon do pn↑. P [i] := P [i-1]
              end;
            (* dodaj A_r ∪ {k} do łańcucha reprezentowanego przez pp↑ *)
            kon := kon+1;  P[kon] := k;
            pp := pnast
          end
    end;
  PODZIAL := w
end; (* PODZIAL *)
begin (* program główny *)
  podz := PODZIAL(n)
end
```

Pozostawiamy Czytelnikowi sprawdzenie, że liczba kroków wykonywanych przez algorytm jest rzędu sumy długości wszystkich otrzymanych łańcuchów podziału, tzn. $O(2^n)$. Na rysunku 4.7 pokazano listę łańcuchów symetrycznych otrzymaną przez algorytm dla $n = 5$.

```
  1   2   3   4   5 ]
[ 5   1   2   3 ]
[ 4   1   2   5 ]
  5 [ 4   1 ]
[ 3   1   4   5 ]
  5 [ 3   1 ]
  4 [ 3   5 ]
[ 2   3   4   5 ]
  5 [ 2   3 ]
  4 [ 2   5 ]
```

RYS. 4.7 Lista łańcuchów symetrycznych skonstruowana przez algorytm 4.24 dla $n = 5$ ("[" poprzedza $P[pocz]$, a "]" następuje za $P[kon]$ $pocz = 0$ dla pierwszego łańcucha

Zauważmy, że w każdym z rekordów listy utworzonej przez algorytm 4.24 elementy $P[i]$ dla $i > kon$ są nieokreślone. Jeśli określimy je w dowolny sposób tak, by ciąg $P[1], \ldots, P[n]$ był permutacją, to otrzymamy listę permutacji $\varphi_1, \varphi_2, \ldots, \varphi_m$, $m = \binom{n}{\lfloor n/2 \rfloor}$ o pewnej ciekawej własności. Otóż dla każdego podzbioru $A \subseteq \{1, \ldots, n\}$ istnieje permutacja φ_i taka, że zbiór pierwszych $|A|$ wyrazów tej permutacji jest równy A (permutację traktujemy tu jako ciąg długości n). Zauważmy, że nie może istnieć układ mniejszej liczby permutacji o powyższej własności, jako że każdy spośród $m = \binom{n}{\lfloor n/2 \rfloor}$ podzbiorów $\lfloor n/2 \rfloor$-elementowych musi występować jako odcinek początkowy w innej permutacji.

Pokażemy teraz zastosowanie powyższego układu permutacji do pewnej organizacji kartotek pochodzącej od Luma (por. poz. [47]). W tym celu rozważmy sytuację, w której rekordy w pewnej kartotece są opisane za pomocą n atrybutów. Załóżmy, że i-ty atrybut może, dla każdego konkretnego rekordu, przyjmować jedną z n_i wartości — dla uproszczenia oznaczmy te wartości przez $1, \ldots, n_i$. Niech R będzie zbiorem rekordów w naszej kartotece. Każdy rekord r jest wtedy opisany przez ciąg $\langle w_1(r), \ldots, w_n(r) \rangle$, gdzie $w_i(r)$ jest wartością i-tego atrybutu dla r ($1 \leqslant w_i(r) \leqslant n_i$). Naszą kartotekę możemy posortować „leksykograficznie" na wiele sposobów, jeśli weźmiemy pod uwagę fakt, iż pojęcie porządku leksykograficznego zakłada pewne uporządkowanie współrzędnych, od współrzędnej, której wartości „zmieniają się najwolniej", do współrzędnej, której wartości „zmieniają się najszybciej". Dokładniej dla dowolnej permutacji φ zbioru $\{1, \ldots, n\}$ określamy porządek $\leqslant_\varphi$ w następujący sposób:

$$\langle a_1, \ldots, a_n \rangle \leqslant_\varphi \langle b_1, \ldots, b_n \rangle \Leftrightarrow \langle a_1, \ldots, a_n \rangle = \langle b_1, \ldots, b_n \rangle \text{ lub}$$

istnieje k, $1 \leqslant k \leqslant n$ takie, że $a_{\varphi(i)} = b_{\varphi(i)}$ dla $1 \leqslant i \leqslant k$ i $a_{\varphi(k)} < b_{\varphi(k)}$. W metodzie Luma stosowanych jest $m = \binom{n}{\lfloor n/2 \rfloor}$ kopii kartotek, z których i-ta — oznaczmy ją przez K_i — posortowana jest zgodnie z porządkiem $\leqslant_{\varphi_i}$ na

ciągach $\langle w_1(r), \ldots, w_n(r)\rangle$. Zakładamy, że każde pytanie skierowane do kartoteki ma postać $Q = \langle x_1, \ldots, x_n\rangle$, gdzie każde x_i jest albo pewną wartością i-tego atrybutu, $1 \leqslant x_i \leqslant n_i$, albo też $x_i = *$, co oznacza, że pytanie nie nakłada żadnych warunków na wartość i-tego atrybutu. Odpowiedź na pytanie $\langle x_1, \ldots x_n\rangle$ oznaczamy przez $||\langle x_1, \ldots, x_n\rangle||$ i definiujemy następująco:

$$||\langle x_1, \ldots, x_n\rangle|| = \{r \in R: w_i(r) = x_i \text{ lub } w_i(r) = *, \text{ dla } 1 \leqslant i \leqslant n\}$$

Istotą metody Luma jest fakt, iż dla dowolnego pytania Q istnieje takie i, że odpowiedź składa się z odcinka kolejnych (względem porządku $\leqslant_{\varphi_i}$) rekordów w kartotece K_i. Aby tę własność udowodnić rozważmy dowolne pytanie $Q = \langle x_1, \ldots, x_n\rangle$. Zgodnie z podstawową własnością zbioru permutacji $\varphi_1, \ldots, \varphi_m$ istnieje wśród nich taka permutacja φ_i, która sprowadza w Q wszystkie współrzędne x_i różne od $*$ na początek, tzn. $x_{\varphi_i(j)} \neq *$ dla $1 \leqslant j \leqslant k$ i $x_{\varphi_i(j)} = *$ dla $k < j \leqslant n$. Lecz jasne jest, że w K_i odpowiedź $||Q||$ jest odcinkiem kolejnych rekordów rozpoczynającym się od rekordu r takiego, że

$$\langle w_{\varphi_i(1)}(r), \ldots, w_{\varphi_i(n)}(r)\rangle = \langle x_{\varphi_i(1)}, \ldots, x_{\varphi_i(k)}, 1, \ldots, 1\rangle$$

i kończącym się rekordem r' takim, że

$$\langle w_{\varphi_i(1)}(r'), \ldots, w_{\varphi_i(n)}(r')\rangle = \langle x_{\varphi_i(1)}, \ldots, x_{\varphi_i(k)}, n_{\varphi_i(k+1)}, \ldots, n_{\varphi_i(n)}\rangle$$

(sytuacja nie zmienia się istotnie gdy nie ma w naszej kartotece takiego rekordu r lub r').

Jeśli nasze kartoteki K_i są pamiętane na nośniku informacji o strukturze liniowej, to fakt, że wszystkie rekordy, które chcemy znaleźć, tworzą spójny odcinek, znacznie ułatwia proces wyszukiwania. Oczywiście główną niedogodnością, szczególnie dla dużych n, jest duża liczba kopii naszej kartoteki, które należy pamiętać.

Na zakończenie tego rozdziału warto zauważyć, że wiele rozważanych tu przez nas twierdzeń miało charakter *minimaksowy*: orzekało, iż minimum pewnej wielkości jest równe maksimum pewnej innej wielkości. Przykładem może być twierdzenie o maksymalnym przepływie i minimalnym przekroju, twierdzenie węgierskie oraz para twierdzeń 4.22 i 4.23. Okazuje się, że ten ostatni przykład jest szczególnym przypadkiem pewnego znacznie ogólniejszego twierdzenia pochodzącego od Dilwortha (por. poz. [12]) i dotyczącego skończonych zbiorów częściowo uporządkowanych. Tak jak w przypadku zbioru częściowo uporządkowanego $\langle \mathscr{P}(X), \subseteq\rangle$, dla dowolnego zbioru częściowo uporządkowanego $\langle P, \leqslant\rangle$ definiujemy *łańcuch* jako dowolny podzbiór $L \subseteq P$ taki, że $x \leqslant y$ lub $y \leqslant x$ dla dowolnych $x, y \in L$, natomiast *antyłańcuch* jako dowolny podzbiór $A \subseteq P$ taki, że dla dowolnych $x, y \in A$

$$x \leqslant y \Rightarrow x = y$$

A oto zapowiedziane twierdzenie Dilwortha:

TWIERDZENIE 4.25

Dla dowolnego skończonego zbioru częściowo uporządkowanego $\langle P, \leqslant \rangle$ minimalna liczba łańcuchów, które w sumie pokrywają P, jest równa maksymalnej liczności antyłańcucha.

DOWÓD

Zastosujemy indukcję względem liczności zbioru P. Jeśli $|P| \leqslant 1$, to twierdzenie jest oczywiście prawdziwe. Niech więc $|P| > 1$, niech L będzie dowolnym łańcuchem maksymalnym (tzn. nie będącym podzbiorem żadnego większego łańcucha) i niech m będzie maksymalną licznością antyłańcucha w $\langle P, \leqslant \rangle$. Pokażemy, że nasz zbiór P można pokryć m łańcuchami. Jeśli maksymalna liczność antyłańcucha w $\langle P \setminus L, \leqslant \rangle$ jest równa $m-1$ (nie może ona być oczywiście mniejsza od $m-1$), to na mocy założenia indukcyjnego istnieje rozkład na łańcuchy $P \setminus L = L_1 \cup \ldots \cup L_{m-1}$, a stąd otrzymujemy żądany rozkład $P = L_1 \cup \ldots \cup L_{m-1} \cup L$. Załóżmy więc, że w $\langle P \setminus L, \leqslant \rangle$ istnieje antyłańcuch m-elementowy $A = \{a_1, \ldots, a_m\}$. Utwórzmy zbiory elementów znajdujących się „nad" A:

$$G = \{x \in P: x > a \quad \text{dla pewnego } a \in A\}$$

oraz „pod" A:

$$D = \{x \in P: x < a \quad \text{dla pewnego } a \in A\}$$

Załóżmy, że L jest postaci $l_1 < \ldots < l_k$. Wtedy $l_1 \notin G$, jako że w przeciwnym przypadku łańcuch L można by powiększyć o pewien element $a < l_1$, $a \in A$, wbrew założeniu o maksymalności L. W podobny sposób dowodzimy, że $l_k \notin D$. A zatem $|G| < |P|$, $|D| < |P|$ i na mocy założenia indukcyjnego zastosowanego dla $\langle G, \leqslant \rangle$ i $\langle D, \leqslant \rangle$ istnieją rozkłady na łańcuchy

$$G = G_1 \cup \ldots \cup G_m \qquad \text{oraz} \qquad D = D_1 \cup \ldots \cup D_m$$

Po ewentualnym przenumerowaniu zbiorów łańcuchów $G_1, \ldots, G_m$ i $D_1, \ldots, D_m$ możemy zakładać, że $a_i \in G_i \cap D_i$, $1 \leqslant i \leqslant m$. Lecz $G \cup D = P$, jako że istnienie elementu $b \in P \setminus (G \cup D)$ przeczyłoby maksymalności antyłańcucha A. Tak więc otrzymujemy żądany rozkład na łańcuchy

$$P = L_1 \cup \ldots \cup L_m$$

przyjmując $L_i = G_i \cup D_i$, $i = 1, \ldots, m$. ∎

Zadania

4.

4.1 Udowodnij, że każdy przepływ f z s do t można przedstawić w postaci sumy $f = f_1 + \ldots + f_k$, $k \leqslant m$ (tzn. $f(e) = f_1(e) + \ldots + f_k(e)$ dla każdej krawędzi $e \in E$), gdzie f_i jest przepływem wzdłuż pojedynczej drogi z s do t, $i = 1, \ldots, k$. Wywnioskuj stąd, że możliwe jest uzyskanie przepływu maksymalnego przez kolejne zwiększanie przepływu wzdłuż odpowiednio dobra-

nych ścieżek rozszerzających o wszystkich krawędziach zgodnych, startując od przepływu zerowego. Pokaż sieć, w której $k = \Omega(m)$ dla każdego takiego rozkładu $f = f_1 + \ldots + f_k$.

4.2 Udowodnij, że jeśli każde zwiększenie przepływu odbywa się wzdłuż najkrótszej ścieżki rozszerzającej względem aktualnego przepływu, to długości kolejnych ścieżek tworzą ciąg niemalejący. (Warto zauważyć, że z własności tej łatwo wynika lemat 4.4, lecz wynikanie w przeciwną stronę nie jest tak oczywiste).

4.3 Udowodnij, że jeśli każde zwiększenie przepływu odbywa się wzdłuż najkrótszej ścieżki rozszerzającej, to startując od dowolnego przepływu uzyskujemy przepływ maksymalny po użyciu co najwyżej $mn/2$ ścieżek rozszerzających (por. poz. [17]).

4.4 Pokaż, że dla sieci o przepustowości każdej krawędzi równej zeru lub jedności, algorytm Dinica może być zrealizowany w czasie $O(n^{2/3}m)$ (por. poz. [18]).

4.5 Pokaż, że jeśli w sieci o przepustowościach całkowitych potencjał każdego wierzchołka różnego od s i t jest równy zeru lub jedności, to algorytm Dinica może być zrealizowany w czasie $O(n^{1/2}m)$ (por. poz. [18]).

4.6 Rozważmy sieć z wyróżnionymi wieloma źródłami $s_1, \ldots, s_p$ i ujściami $t_1, \ldots, t_q$, oraz niech dane będą *wydajności* źródeł $a_1, \ldots, a_p \geqslant 0$ i *zapotrzebowania* $b_1, \ldots, b_q \geqslant 0$ takie, że $a_1 + \ldots + a_p = b_1 + \ldots + b_q$. Podaj metodę znajdowania w takiej sieci przepływu f takiego, że $Div_f(s_i) = a_i$ dla $1 \leqslant i \leqslant p$, $Div_f(t_j) = -b_j$ dla $1 \leqslant j \leqslant q$ oraz $Div_f(v) = 0$ dla wszystkich pozostałych wierzchołków sieci – jeśli taki przepływ istnieje. (Wskazówka: Dodaj do sieci dodatkowe krawędzie $\langle s, s_i \rangle$ o przepustowościach a_i, dla $i = 1, \ldots, p$, oraz krawędzie $\langle t_j, t \rangle$ o przepustowościach b_j, dla $j = 1, \ldots, q$).

4.7 Dodajmy do problemu przepływu w sieci dodatkowe ograniczenie polegające na tym, że dla każdego wierzchołka v różnego od s i t ilość przepływu wpływającego do v i wypływającego z v nie może przekraczać *przepustowości* $g(v)$ tego wierzchołka. Jak taki zmodyfikowany problem sprowadzić do oryginalnego zagadnienia, bez przepustowości wierzchołków? (Wskazówka: Dla każdego wierzchołka v dodaj nowy wierzchołek v^*, krawędź $\langle v, v^* \rangle$ o przepustowości $g(v)$ i zamień każdą krawędź postaci $\langle v, u \rangle$ na $\langle v^*, u \rangle$).

4.8 Niech G będzie grafem zorientowanym nie zawierającym krawędzi $\langle s, t \rangle$. Udowodnij, że maksymalna liczba dróg w G z s do t o parami rozłącznych zbiorach wierzchołków pośrednich (tzn. różnych od s i t) jest równa minimalnej liczbie wierzchołków pośrednich, po usunięciu których nie istnieje w naszym grafie żadna droga z s do t (por. poz. [52]). (Wskazówka: Potraktuj G jako sieć o przepustowości każdego wierzchołka równej jedności – por. poprzednie zadanie – i skorzystaj z twierdzenia o maksymalnym przepływie i minimalnym przekroju).

4.9 Podaj algorytm znajdowania krawędzi o tej własności, że zwiększenie jej przepustowości powoduje zwiększenie wartości maksymalnego przepływu. Czy taka krawędź zawsze istnieje? (Wskazówka: Znajdź maksymalny przepływ f, a następnie zbiór A wierzchołków osiągalnych przez częściowe ścieżki rozszerzające z s, oraz zbiór B wierzchołków, z których w podobny sposób osiągnąć można t. Szukany zbiór to $(A \times B) \cap E$).

4.10 Rozważmy sieć, w której każdej krawędzi $e \in E$ przypisany jest, oprócz przepustowości, *koszt* $h(e)$ przesłania jednostki przepływu przez e. Zdefiniujmy koszt ścieżki rozszerzającej jako sumę kosztów krawędzi zgodnych minus sumę kosztów krawędzi przeciwnych tej ścieżki. Udowodnij, że przepływ f o wartości $w = W(f)$ ma minimalny koszt wśród wszystkich przepływów o wartości w wtedy i tylko wtedy, gdy nie istnieje cykl (ścieżka zamknięta) rozszerzający względem f o ujemnym koszcie. Pokazać, że zwiększenie o δ przepływu o minimalnym koszcie o wartości w wzdłuż ścieżki rozszerzającej o minimalnym koszcie prowadzi do przepływu o minimalnym koszcie o wartości $w+\delta$.

4.11 Udowodnij, że jeśli P jest najkrótszą ścieżką naprzemienną względem skojarzenia M w grafie dwudzielnym, P zaś najkrótszą ścieżką naprzemienną względem $M \oplus P$, to $|P'| \geqslant |P| + |P \cap P'|$ (por. poz. [33]).

4.12 Udowodnij, że jeśli M i N są skojarzeniami w grafie dwudzielnym H, to istnieje takie skojarzenie R, że wierzchołek grafu H jest wolny względem R wtedy i tylko wtedy, gdy jest wolny względem zarówno M, jak i N (patrz poz. [51], por. też lemat 4.11).

4.13 Graf dwudzielny $H = \langle X, Y, E \rangle$ nazywamy *wypukłym* na X, jeśli zbiór X można uporządkować w ciąg $x_1, \ldots, x_p$ tak, by dla każdego $y \in Y$ zbiór $A(y) = \{x \in X : x-y\}$ tworzył odcinek postaci $\{x_{P(y)}, x_{P(y)+1}, \ldots, \rangle_{K(y)}\}$. Podobnie definiujemy wypukłość na Y. Udowodnij, że jeśli graf H jest wypukły na X to skojarzenie o maksymalnej liczności można otrzymać przeglądając ciąg $x_1, \ldots, x_p$ i kojarząc wierzchołek x_i z tym spośród wolnych jeszcze wierzchołków $y \in Y$, $x-y$, dla którego wartość $K(y)$ jest minimalna (lub pozostawiając wierzchołek x_i wolny gdy żaden taki wierzchołek y nie istnieje) (por. poz. [28]).

4.14 Pokaż, że skojarzenie o maksymalnej liczności w grafie dwudzielnym $H = \langle X, Y, E \rangle$ wypukłym na X (por. poprzednie zadanie) można skonstruować w czasie $O(|X|+|Y| \log |Y|)$ i w czasie $O(|X|+|Y|)$, jeśli H jest wypukły zarówno na X jak i na Y. (Uwaga: Zakładamy, że graf dany jest poprzez tablice wartości $P(y)$ i $K(y)$, $y \in Y$; warto tu wspomnieć, że ograniczenie $O(|X|+|Y| \log |Y|)$ można znacznie polepszyć, jak to pokazano w pracy [46], korzystając z pewnej ogólnej techniki pochodzącej od Tarjana i podanej w pracy [67]).

4.15 Podaj przykład grafu (niedwudzielnego), dla którego nie zachodzi własność wyrażona we wniosku 4.16.

6 Udowodnij następujące twierdzenie, dualne względem twierdzenia węgierskiego: Dla dowolnej macierzy zerojedynkowej maksymalna liczba jedynek w linii jest równa minimalnej liczbie rozproszonych zbiorów jedynek pokrywających wszystkie jedynki w macierzy (zbiór jedynek nazywamy *rozproszonym*, jeśli żadne dwie z nich nie leżą na jednej linii). Podaj sformułowanie tego twierdzenia w terminach skojarzeń w grafie dwudzielnym.

7 *Macierzą bistochastyczną* nazywamy macierz wymiaru $n \times n$ o elementach rzeczywistych nieujemnych, w której suma elementów w każdym wierszu i w każdej kolumnie jest równa jedności. Przez *macierz permutacyjną* rozumiemy dowolną bistochastyczną macierz zerojedynkową. Udowodnij, że każdą macierz bistochastyczną B można przedstawić w postaci kombinacji $B = \mu_1 P_1 + \ldots + \mu_k P_k$, gdzie $P_1, \ldots, P_k$ są macierzami permutacyjnymi, $\mu_1, \ldots, \mu_k > 0$ i $\mu_1 + \ldots + \mu_k = 1$ (por. poz. [6]).

8 *Permanent* macierzy $A = [a_{ij}]$ wymiaru $n \times n$ definiujemy w następujący sposób:

$$per\, A = \sum a_{1,\sigma(1)}\, a_{2,\sigma(2)}, \ldots, a_{n,\sigma(n)}$$

gdzie sumowanie rozciąga się na wszystkie permutacje σ zbioru $\{1, \ldots, n\}$. Udowodnij, że permanent macierzy zerojedynkowej A wymiaru $n \times n$ jest równy zeru wtedy i tylko wtedy, gdy A zawiera podmacierz zerową wymiaru $p \times r$, gdzie $p + r = n + 1$ (por. poz. [24]).

9 Uzupełnij szczegóły naszkicowanego w punkcie 4.4 algorytmu o złożoności $O(\sqrt{n} \sum_{j=1}^{n} (|A_j| + |B_j|))$ znajdowania wspólnego systemu różnych reprezentantów dla ciągów $\langle A_1, \ldots, A_n \rangle$ i $\langle B_1, \ldots, B_n \rangle$ (por. zad. 4.5).

10 Podaj konstrukcję ciągu o elementach ze zbioru $\{1, \ldots, n\}$ o długości $l_n \approx \frac{2}{\pi} 2^n$ o tej własności, że każdy podzbiór $A \subseteq \{1, \ldots, n\}$ występuje w tym ciągu jako podciąg $|A|$ kolejnych wyrazów [43]. (Wskazówka: Skonstruuj $\binom{k}{\lfloor k/2 \rfloor}$ permutacji zbioru $\{1, \ldots, k\}$, $k = \lfloor n/2 \rfloor$, takich, że każdy podzbiór $K \subseteq \{1, \ldots, k\}$ występuje jako odcinek końcowy pewnej z tych permutacji, oraz $\binom{p}{\lfloor p/2 \rfloor}$, $p = \lceil n/2 \rceil$ permutacji zbioru $\{k+1, \ldots, n\}$ takich, że każdy podzbiór $L \subseteq \{k+1, \ldots, n\}$ występuje jako odcinek początkowy pewnej z tych permutacji. Następnie połącz te permutacje w odpowiedni sposób. Uwaga: Na mocy wzoru Stirlinga $\binom{k}{\lfloor k/2 \rfloor} \approx \sqrt{2/(\pi k)}\, 2^k$).

Matroidy

Algorytmy zachłanne rozwiązywania problemów optymalizacyjnych

5.

Rozważmy następującą macierz o współczynnikach rzeczywistych nieujemnych:

$$A = \begin{bmatrix} 7 & 5 & 1 \\ 3 & 4 & 3 \\ 2 & 3 & 1 \end{bmatrix}$$

Zajmiemy się rozwiązaniem następującego problemu optymalizacyjnego:

PROBLEM 1.

Znaleźć podzbiór elementów macierzy taki, że

(a) w każdej kolumnie znajduje się co najwyżej jeden wybrany element, oraz

(b) suma wybranych elementów jest największa z możliwych.

Spróbujmy rozwiązać ten problem w następujący sposób: wybieramy elementy kolejno, przy czym za każdym razem wybieramy największy spośród elementów, które możemy dodać nie naruszając warunku (a). Postępowanie to kontynuujemy aż do momentu, gdy dodanie dowolnego elementu narusza warunek (a). Algorytm tego typu będziemy nazywać *zachłannym*.

W przypadku problemu 1 i macierzy A algorytm zachłanny znajduje podzbiór

$$\begin{bmatrix} ⑦ & ⑤ & 1 \\ 3 & 4 & ③ \\ 2 & 3 & 1 \end{bmatrix}$$

który rzeczywiście daje największą możliwą sumę.

Nietrudno zauważyć, że algorytm zachłanny znajduje prawidłowo rozwiązanie problemu 1 dla dowolnej macierzy rzeczywistej o elementach nieujemnych.

Rozważmy nieco inny problem.

PROBLEM 2.

Znaleźć podzbiór elementów macierzy taki, że

(a) w każdej kolumnie i w każdym wierszu znajduje się co najwyżej jeden wybrany element, oraz

(b) suma wybranych elementów jest największa z możliwych.

Algorytm zachłanny zastosowany do problemu 2 i macierzy A daje tym azem podzbiór

$$\begin{bmatrix} ⑦ & 5 & 1 \\ 3 & ④ & 3 \\ 2 & 3 & ① \end{bmatrix}$$

sumie 12, co nie jest poprawnym rozwiązaniem, gdyż podzbiór

$$\begin{bmatrix} ⑦ & 5 & 1 \\ 3 & 4 & ③ \\ 2 & ③ & 1 \end{bmatrix}$$

na sumę 13. Tak więc w drugim kroku nie warto było być zachłannym – wybierając nieco mniejszy element (3 zamiast 4) ostatecznie lepiej na tym wychodzimy.

Powstaje pytanie: kiedy opłaca się być zachłannym? Sformułujmy je nieco ciślej.

Rozważać będziemy problemy optymalizacyjne następującego typu:

PROBLEM 3.

Dany jest zbiór skończony E, rodzina jego podzbiorów $\mathcal{I} \subseteq \mathcal{P}(E)$, oraz funkcja $w: E \to R^+$, gdzie R^+ oznacza zbiór liczb rzeczywistych nieujemnych. Znaleźć podzbiór $S \in \mathcal{I}$ o największej sumie $\sum_{e \in S} w(e)$.

Zarówno problem 1, jak i problem 2 są szczególnymi przypadkami problemu 3. W obu przypadkach E jest zbiorem pozycji macierzy, w zaś przyporządkowuje pozycji $\langle i, j \rangle$ macierzy $[a_{ij}]$ liczbę a_{ij}. Dla problemu 1

$S \in \mathcal{I} \Leftrightarrow$ każda kolumna zawiera co najwyżej jedną pozycję ze zbioru S

natomiast w przypadku problemu 2

$S \in \mathcal{I} \Leftrightarrow$ każda kolumna i każdy wiersz zawiera co najwyżej jedną pozycję ze zbioru S

Nasze pytanie możemy teraz sformułować następująco: przy jakich założeniach na rodzinę $\mathcal{I}$ algorytm zachłanny rozwiązuje poprawnie problem 3 dla dowolnej funkcji w?

Okazuje się, że można podać prostą odpowiedź na to pytanie. Wystarczy mianowicie, by para $\langle E, \mathcal{I} \rangle$ tworzyła tzw. matroid. Tym ważnym obiektom kombinatorycznym poświęcony jest następny punkt tego rozdziału (Czytelnika zainteresowanego w głębszym poznaniu teorii matroidów odsyłamy do pozycji [42], [72]).

Matroidy i ich podstawowe własności 5.2

Matroidy zostały wprowadzone przez H. Whitneya w pracy [73] w zupełnie innym kontekście niż algorytmy zachłanne, a mianowicie w badaniach nad abstrakcyjną teorią zależności nieliniowej. Istnieje wiele równoważnych definicji matroidu. Dla nas najbardziej wygodna będzie następująca:

Matroidem nazywamy dowolną parę $M = \langle E, \mathcal{I} \rangle$, gdzie E jest zbiorem skończonym a $\mathcal{I} \subseteq \mathcal{P}(E)$, spełniającą warunki

M1 = $\emptyset \in \mathcal{I}$, oraz jeśli $A \in \mathcal{I}$ i $B \subseteq A$, to $B \in \mathcal{I}$.

M2 = Dla dowolnych A, $B \in \mathcal{I}$ takich, że $|B| = |A| + 1$ istnieje element $e \in B \setminus A$ taki, że $A \cup \{e\} \in \mathcal{I}$.

(Warunek $\emptyset \in \mathcal{I}$ eliminuje zdegenerowany przypadek $\mathcal{I} = \emptyset$).

Zbiory z rodziny $\mathcal{I}$ nazywamy *zbiorami niezależnymi*, pozostałe zaś podzbiory z $\mathcal{P}(E) \setminus \mathcal{I}$ *zbiorami zależnymi* matroidu M. Wiąże się to ze wspomnianym już związkiem matroidów z teorią zależności liniowej – aksjomaty matroidu zostały wybrane tak, by odzwierciedlały najbardziej charakterystyczne własności niezależnych podzbiorów przestrzeni liniowej, ściślej mówiąc niezależnych podzbiorów zawartych w pewnym skończonym podzbiorze tej przestrzeni (przypomnijmy, że podzbiór $\{e_1, \ldots, e_n\}$ przestrzeni liniowej nazywamy niezależnym, jeśli nie istnieją skalary $\lambda_1, \ldots, \lambda_n$ nie wszystkie równe zeru takie, że $\lambda_1 e_1 + \ldots + \lambda_n e_n = 0$). Istotnie, dowolny podzbiór niezależnego podzbioru przestrzeni liniowej jest oczywiście niezależny. Jeśli $|B| = |A| + 1$ dla liniowo niezależnych podzbiorów A, B przestrzeni liniowej, to A rozpina przestrzeń wymiaru $|A|$, która może zawierać co najwyżej $|A|$ elementów zbioru B. Istnieje więc element $e \in B \setminus A$ nie należący do tej podprzestrzeni. Zbiór $A \cup \{e\}$ rozpina podprzestrzeń wymiaru $|A| + 1$, a więc jest liniowo niezależny.

Powiemy, że dwa matroidy $\langle E, \mathcal{I} \rangle$ i $\langle E', \mathcal{I}' \rangle$ są *izomorficzne*, jeśli istnieje wzajemnie jednoznaczne odwzorowanie f zbioru E na zbiór E' takie, że $A \in \mathcal{I}$ wtedy i tylko wtedy, gdy $f(A) \in \mathcal{I}'$. Często nie będziemy rozróżniać pomiędzy matroidami izomorficznymi.

Dla dowolnego podzbioru $C \subseteq E$ zajmiemy się teraz maksymalnymi podzbiorami niezależnymi zbioru C, tzn. podzbiorami niezależnymi $A \subseteq C$ o tej własności, że nie istnieje podzbiór niezależny B taki, że $A \subseteq B \subseteq C$.

TWIERDZENIE 5.1

Niech E będzie zbiorem skończonym, a $\mathcal{I}$ rodziną jego podzbiorów spełniającą warunek M1. Przy tych założeniach $M = \langle E, \mathcal{I} \rangle$ jest matroidem wtedy i tylko wtedy, gdy spełnia warunek

M3 Dla dowolnego podzbioru $C \subseteq E$ każde dwa maksymalne podzbiory niezależne zbioru C mają tę samą liczność.

DOWÓD

Załóżmy, że $M = \langle E, \mathcal{I} \rangle$ jest matroidem i dla pewnego $C \subseteq E$ istnieją dwa maksymalne podzbiory niezależne A, $B \subseteq C$ takie, że $|B| > |A|$. Wybierzmy dowolny podzbiór $B' \subseteq B$ (niezależny wobec M1!) taki, że $|B'| = |A| + 1$. Na mocy M2 istnieje element $e \in B' \setminus A \subseteq C$ taki, że $A \cup \{e\} \in \mathcal{I}$, wbrew maksymalności zbioru A.

Na odwrót, załóżmy, że spełnione są warunki M1 i M3. Wybierzmy dowolne podzbiory A, $B \in \mathcal{I}$ takie, że $|B| = |A| + 1$, oraz oznaczmy $C = A \cup B$. Przypuśćmy, że nie istnieje element $e \in B \setminus A$ taki, że $A \cup \{e\} \in \mathcal{I}$. Oznaczałoby to,

że A jest maksymalnym podzbiorem niezależnym zbioru C. Rozszerzając B do maksymalnego podzbioru niezależnego $B^* \subseteq C$ mielibyśmy wtedy $|A| < |B^*|$, wbrew warunkowi M3. Tak więc warunki M1 i M3 pociągają za sobą warunek M2. ∎

Z twierdzenia 5.1 wynika, że warunki M1 i M3 stanowią równoważny układ aksjomatów dla matroidów.

Liczność maksymalnego podzbioru zbioru $C \subseteq E$ nazywamy *rangą* tego zbioru i oznaczamy przez $r(C)$:

$$r(C) = max\{|A|: A \in \mathcal{I} \wedge A \subseteq C\}$$

Oczywiście podzbiór $C \subseteq E$ jest niezależny wtedy i tylko wtedy, gdy $r(C) = |C|$. Każdy maksymalny zbiór niezależny matroidu $M = \langle E, \mathcal{I} \rangle$ nazywamy — przez analogię do przestrzeni liniowych — *bazą* tego matroidu, a rangę $r(E)$ — będącą odpowiednikiem wymiaru przestrzeni liniowej — nazywamy *rangą matroidu*. Odnotujmy ważny wniosek z twierdzenia 5.1.

WNIOSEK 5.2
Każde dwie bazy matroidu mają tę samą liczność. ∎

Zauważmy jeszcze, że każdy zbiór niezależny $C \in \mathcal{I}$ możemy rozszerzyć do bazy $B \supseteq C$; wystarczy kolejno dodawać do C nowe elementy, których dołączenie nie narusza niezależności, aż do momentu, gdy takich elementów nie ma. Otrzymany zbiór jest maksymalnym zbiorem niezależnym, a więc bazą. Podobnie dowolny zbiór niezależny $A \subseteq C$ możemy rozszerzyć do maksymalnego niezależnego podzbioru zbioru C.

Odnotujmy niektóre własności rangi.

TWIERDZENIE 5.3
Dla dowolnych $A, B \subseteq E$ oraz $e, f \in E$

R1 $0 \leqslant r(A) \leqslant |A|$
R2 jeśli $A \subseteq B$, to $r(A) \leqslant r(B)$
R3 $r(A \cup B) + r(A \cap B) \leqslant r(A) + r(B)$
R4 $r(A) \leqslant r(A \cup \{e\}) \leqslant r(A) + 1$
R5 jeśli $r(A \cup \{e\}) = r(A \cup \{f\}) = r(A)$, to $r(A \cup \{e,f\}) = r(A)$

DOWÓD
Własności R1, R2 są oczywiste. Udowodnimy R3. Niech $\{e_1, \ldots, e_p\}$ będzie maksymalnym zbiorem niezależnym w $A \cap B$. Rozszerzmy go do maksymalnego zbioru niezależnego $\{e_1, \ldots, e_p, f_1, \ldots, f_q\} \subseteq A$, a następnie do maksymalnego zbioru niezależnego $\{e_1, \ldots, e_p, f_1, \ldots, f_q, g_1, \ldots, g_r\} \subseteq A \cup B$. Mamy $p = r(A \cap B)$ $p+q = r(A)$, $p+r \leqslant r(B)$, $p+q+r = r(A \cup B)$, a zatem

$$r(A \cup B) + r(A \cap B) = (p+q+r)+p = (p+q)+(p+r) \leqslant r(A)+r(B)$$

Warunek R4 jest oczywisty, R5 natomiast wynika łatwo z R3:

$$\begin{aligned} r(A) &\leqslant r(A \cup \{e,f\}) = r\big((A \cup \{e\} \cup (A \cup \{f\})\big) \\ &\leqslant r(A \cup \{e\}) + r(A \cup \{f\}) - r\big((A \cup \{e\}) \cap (A \cup \{f\})\big) = \\ &= r(A) + r(A) - r(A) = r(A) \end{aligned}$$ ∎

Kontynuując analogię z przestrzeniami liniowymi powiemy, że element e jest *zależny* od zbioru A, jeśli $r(A \cup \{e\}) = r(A)$, oraz oznaczymy przez $sp(A)$ zbiór wszystkich elementów zależnych od A:

$$sp(A) = \{e \in E : r(A \cup \{e\}) = r(A)\}$$

Zbiór $A \subseteq E$ nazywamy *podprzestrzenią* matroidu, jeśli $A = sp(A)$, tzn. jeśli $r(A \cup \{e\}) = r(A)+1$ dla dowolnego $e \in E \setminus A$.

TWIERDZENIE 5.4

Dla dowolnych $A, B \subseteq E$ oraz $e, f \in E$

S1 $A \subseteq sp(A)$

S2 jeśli $A \subseteq B$, to $sp(A) \subseteq sp(B)$

S3 $sp(sp(A)) = sp(A)$

S4 jeśli $f \notin sp(A)$ i $f \in sp(A \cup \{e\})$, to $e \in sp(A \cup (f))$ ■

Dowód

Warunek S1 jest oczywisty: jeśli $e \in A$, to $r(A \cup \{e\}) = r(A)$, czyli $e \in sp(A)$. Dla dowodu S2 skorzystamy z R3. Załóżmy, że $A \subseteq B$ oraz $e \in sp(A)$. Wtedy

$$\begin{aligned} r(B) &\leqslant r(B \cup \{e\}) = r((A \cup \{e\}) \cup B) \\ &\leqslant r(A \cup \{e\}) + r(B) - r(A \cup (B \cap \{e\})) = \\ &= r(A) + r(B) - r(A) = r(B) \end{aligned}$$

i w konsekwencji $e \in sp(B)$.

Z S1 i S2 wynika, że $sp(A) \subseteq sp(sp(A))$. Pokażemy teraz przeciwną inkluzję. Potrzebna będzie nam do tego równość

$$r(sp(A)) = r(A) \tag{5.1}$$

Aby ją udowodnić załóżmy, że $sp(A) \setminus A = \{e_1, \ldots, e_k\}$. Z definicji $sp(A)$ mamy $r(A \cup \{e_i\}) = r(A)$, $i = 1, \ldots, k$. Załóżmy, że dla pewnego $i < k$ mamy $r(A \cup \{e_1, \ldots, e_i\}) = r(A)$. Z R3 otrzymujemy wtedy

$$\begin{aligned} r(A \cup \{e_1, \ldots, e_i, e_{i+1}\}) &= r((A \cup \{e_1, \ldots, e_i\}) \cup (A \cup \{e_{i+1}\})) \\ &\leqslant r(A \cup \{e_1, \ldots, e_i\}) + r(A \cup \{e_{i+1}\}) - r(A) = \\ &= r(A) + r(A) - r(A) = r(A) \end{aligned}$$

Stąd, przez indukcję względem i otrzymujemy żądaną równość (5.1). Dowód inkluzji $sp(sp(A)) \subseteq sp(A)$ jest teraz prosty. Zakładając $e \in sp(sp(A))$, tzn. $r(sp(A) \cup \{e\}) = r(sp(A))$, mamy $r(A) \leqslant r(A \cup \{e\}) \leqslant r(sp(A) \cup \{e\}) = r(sp(A)) = r(A)$, czyli $e \in sp(A)$.

Wreszcie własność S4 wynika z definicji zależności elementu od zbioru: założywszy $f \notin sp(A)$ i $f \in sp(A \cup \{e\})$ mamy

$$r(A)+1 = r(A \cup \{f\}) \leqslant r(A \cup \{e, f\}) = r(A \cup \{e\}) \leqslant r(A)+1$$

a stąd $r(A \cup \{f\}) = r(A \cup \{e, f\})$, tzn. $e \in sp(A \cup \{f\})$. ■

Zauważmy, że z własności S3 wynika, że $sp(A)$ jest podprzestrzenią matroidu azywamy ją *podprzestrzenią rozpiętą* przez zbiór A.

Ostatnim ważnym pojęciem, które omówimy w tym punkcie jest pojęcie klu. *Cyklem* matroidu będziemy nazywać każdy minimalny zbiór zależny, tzn. ki zbiór zależny C, że $C\setminus\{e\}$ jest niezależny dla dowolnego $e \in C$. Pojęcie cyklu dzie miało szczególnie przejrzystą interpretację w przypadku matroidów gra- wych, które omawiamy w punkcie 5.5.

WIERDZENIE 5.5

la dowolnych cykli C, D są spełnione warunki

C1 Jeśli $C \subseteq D$, to $C = D$

C2 Jeśli $C \neq D$ oraz $e \in C \cup D$, to istnieje cykl $F \subseteq (C \cup D)\setminus\{e\}$

OWÓD

arunek C1 wyraża po prostu własność minimalności zawartą w definicji cyklu dowodnimy teraz C2. Zbiory $C\setminus\{e\}$ i $D\setminus\{e\}$ są niezależne, a więc $r(C\setminus\{e\}) = |C|-1$, $r(D\setminus\{e\}) = |D|-1$. Na mocy warunku R3

$$r(C \cup D)+r(C \cap D) \leqslant r(C)+r(D) = |C|+|D|-2$$
$$= |C \cup D|+|C \cap D|-2 \qquad (5.2)$$

biór $C \cap D$ jest niezależny, gdyż jest podzbiorem właściwym obu cykli. Stąd $(C \cap D) = |C \cap D|$ i nierówność (5.2) możemy przepisać jako

$$r(C \cup D) \leqslant |C \cup D|-2 \qquad (5.3)$$

łóżmy teraz, że cykl F, o którym mowa w warunku C2 nie istnieje. Wtedy biór $(C \cup D)\setminus\{e\}$ jest niezależny, $r((C \cup D\setminus\{e\}) = |C \cup D|-1$ i w kon- kwencji $r(C \cup D) \geqslant |C \cup D|-1$, wbrew nierówności (5.3). ■

Odnotujmy ważny wniosek z własności C2:

NIOSEK 5.6

eśli A jest zbiorem niezależnym, to dla dowolnego $e \in E$ zbiór $A \cup \{e\}$ zawiera o najwyżej jeden cykl.

OWÓD

eśliby istniały dwa różne cykle C, $D \subseteq A \cup \{e\}$, to mielibyśmy oczywiście $e \in C \cap D$ i wobec własności C2 istniałby cykl $F \subseteq (C \cup D)\setminus\{e\} \subseteq A$, wbrew za- łożeniu o niezależności zbioru A. ■

Nasze dotychczasowe intuicje dotyczące matroidów wiążą się z liniowo iezależnymi zbiorami wektorów przestrzeni liniowej. Podamy teraz dwa inne rzykłady matroidów.

Dla dowolnego zbioru skończonego E para $\langle E, \mathscr{P}(E)\rangle$ spełnia oczywiście arunki M1, M2. Nazywamy ją *matroidem wolnym* na zbiorze E. W takim ma- roidzie każdy zbiór $A \subseteq E$ jest niezależny i w konsekwencji $r(A) = |A|$.

Inny przykład otrzymujemy rozpatrując dowolny podział $\pi = \{D_1, \ldots, D_1\}$ zbioru E i definiując

$$\mathscr{I} = \{A \subseteq E: |A \cap D_i| \leqslant 1 \quad \text{dla } i = 1, \ldots, k\}$$

Zauważmy, że jeśli $A, B \in \mathcal{I}$ oraz $|B| = |A|+1$, to musi istnieć wskaźnik i taki, że $D_i \cap A = \emptyset$ lecz $D_i \cap B \neq \emptyset$. Element $e \in D_i \cap B$ możemy „przenieść" do A otrzymując zbiór $A \cup \{e\} \in \mathcal{I}$. Dowodzi to warunku M2. Tak określony matroid $M = \langle E, \mathcal{I} \rangle$ nazywamy *matroidem podziałowym*.

Znacznie ciekawsze przykłady matroidów poznamy w punktach 5.4, 5.5 i 5.6.

Twierdzenie Rado-Edmondsa

Powracamy do algorytmów zachłannych, o których była mowa w pierwszym punkcie. Sformułujemy najpierw ogólny schemat tego typu algorytmów. Rozważać będziemy zbiór skończony E, funkcję w: $E \to R^+$, oraz rodzinę $\mathcal{I} \subseteq \mathcal{P}(E)$. Wartość $w(e)$ nazywamy *wagą* elementu e.
Wagę podzbioru $A \subseteq E$ definiujemy następująco:

$$w(A) = \sum_{e \in A} w(e)$$

ALGORYTM 5.7 (Algorytm zachłanny)

```
1  begin
2    posortuj zbiór E według malejących wag tak, by
3      E = {e_1, ..., e_n}, gdzie w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_n);
4    S := ∅;
5      for i := 1 to n do
6      if S ∪ {e_i} ∈ I then S := S ∪ {e_i}
7  end
```

TWIERDZENIE 5.8. (Rado, Edmonds, por. poz. [15] i [56]).
Jeśli $M = \langle E, \mathcal{I} \rangle$ jest matroidem, to zbiór S znaleziony przez algorytm zachłanny jest zbiorem niezależnym o największej wadze. Na odwrót, jeśli $M = \langle E, \mathcal{I} \rangle$ nie jest matroidem, to istnieje funkcja w: $E \to R^+$ taka, że S nie jest zbiorem niezależnym o największej wadze.

DOWÓD

Załóżmy, że $M = \langle E, \mathcal{I} \rangle$ jest matroidem, i niech $S = \{s_1, \ldots, s_k\}$ będzie zbiorem wybranym przez algorytm zachłanny, przy czym $w(s_1) \geq w(s_2) \geq \ldots \geq w(s_k)$. Rozważmy dowolny zbiór niezależny $T = \{t_1, \ldots, t_m\}$, gdzie $w(t_1) \geq w(t_2) \geq \ldots \geq w(t_1)$. Zauważmy najpierw, że musi być $m \leq k$, gdyż zbiór S wybrany przez algorytm zachłanny jest bazą matroidu: każdy element $e_i \in S$ „odrzucony" w pewnym kroku algorytmu jest zależny od zbioru elementów poprzednio wybranych, a więc tym bardziej od całego zbioru S. Pokażemy teraz, że $w(T) \leq w(S)$, dokładniej, że dla dowolnego $i \leq m$ mamy $w(t_i) \leq w(s_i)$. Przypuśćmy bowiem, że $w(t_i) > w(s_i)$ i rozważmy zbiory niezależne

$$A = \{s_1, \ldots, s_{i-1}\}$$
$$B = \{t_1, \ldots, t_{i-1}, t_i\}$$

godnie z warunkiem M2, istnieje element $t_j, j \leqslant i$ taki, że zbiór $\{s_1, \ldots, s_{i-1}, t_j\}$ st niezależny. Mamy $w(t_j) \geqslant w(t_j) > w(s_i)$, a zatem istnieje wskaźnik $p \leqslant i$ ki, że $w(s_1) \geqslant \ldots \geqslant w(s_{p-1}) \geqslant w(t_j) > w(s_p)$, wbrew temu, że s_p jest elementem o największej wadze, którego dodanie do $\{s_1, \ldots, s_{p-1}\}$ nie narusza niezależności. Otrzymana sprzeczność dowodzi, że $w(t_i) \leqslant (s_i)$, $1 \leqslant i \leqslant m$.

Załóżmy teraz, że $M = \langle E, \mathscr{I} \rangle$ nie jest matroidem. Jeśli nie jest spełniony arunek M1, tzn. jeśli istnieją zbiory A, $B \subseteq E$ takie, że $A \subseteq B \in \mathscr{I}$, $A \notin \mathscr{I}$,
5.3 określmy

$$w(e) = \begin{cases} 1 & \text{jeśli } e \in A \\ 0 & \text{jeśli } e \in E \setminus A \end{cases}$$

atwo widzieć, że zbiór A nie jest wtedy zawarty w zbiorze S wybranym przez gorytm zachłanny. W konsekwencji $w(S) < w(B) = w(A)$. Jeśli natomiast arunek M1 jest spełniony, lecz nie zachodzi warunek M2, to istnieją zbiory niezależne A, B takie, że $|A| = k$, $|B| = k+1$ i dla każdego $e \in B \setminus A$ zbiór $A \cup \{e\}$ t zależny. Oznaczmy $p = |A \cap B|$ (oczywiście $p < k$) i niech $0 < \varepsilon < 1/(k-p)$. kreślmy

$$(e) = \begin{cases} 1+\varepsilon & \text{jeśli } e \in A \\ 1 & \text{jeśli } e \in B \setminus A \\ 0 & \text{w pozostałych przypadkach} \end{cases}$$

auważmy, że przy tak określonych wagach algorytm zachłanny najpierw wyerze wszystkie elementy zbioru A, a następnie odrzuci wszystkie elementy $\in B \setminus A$. W konsekwencji zostanie wybrany zbiór S o wadze mniejszej od wagi bioru B:

$$w(S) = w(A) = k(1+\varepsilon) = (k-p)(1+\varepsilon) + p(1+\varepsilon)$$
$$< (k-p)\frac{k+1-p}{k-p} + p(1+\varepsilon) = (k+1-p) + p(1+\varepsilon) = w(B) \qquad \blacksquare$$

Warto zwrócić uwagę na pewien dość zaskakujący fakt. Skoro wagę podzbioru kreśliliśmy jako sumę wag jego elementów, to wydawać by się mogło, że optymalny zbiór S będzie w istotny sposób zależał od wartości numerycznych wag oszczególnych elementów. Jednakże zależność ta jest bardzo słaba: zbiór S aleźy jedynie od uporządkowania wag poszczególnych elementów. W algorytmie achłannym po posortowaniu elementów wagi przestają nas zupełnie interesować.

Drugim faktem wartym odnotowania jest to, że w dowodzie twierdzenia 5.8 okazaliśmy, że zbiór S jest optymalny w bardzo silnym sensie. Nie tylko suma lementów jest maksymalna, lecz również w dowolnym zbiorze niezależnym T waga i-tego co do wielkości elementu jest nie większa od wagi i-tego co do wielkości elementu w S. Fakt ten nazywać będziemy *optymalnością w sensie Gale'a* por. poz. [25]). Tak więc w matroidzie nie można wybrać zbioru niezależnego łożonego z „mniejszej liczby elementów, ale za to większych (o większych wagach)".

Zauważmy jeszcze, że przy odwróceniu uporządkowania elementów algorytm zachłanny wybierze zbiór S, który nie tylko ma najmniejszą wagę, lecz również i-ty co do wielkości element (licząc od najmniejszego) jest niewiększy od i-tego co do wielkości elementu dowolnej bazy . Możemy teraz spojrzeć z ogólnego punktu widzenia na zastosowanie algorytmu zachłannego do problemu 1 z początku tego rozdziału: mieliśmy tam do czynienia z matroidem podziałowym określonym przez podział pozycji macierzy na kolumny.

W dalszym ciągu poznamy zastosowania algorytmu zachłannego dla innych, mniej banalnych matroidów. Pokażemy też jak w konkretnych przypadkach sprawdzać efektywnie warunek $S \cup \{e_i\} \in \mathcal{I}$ występujący w wierszu 6 algorytmu zachłannego. We wszystkich przypadkach otrzymamy oszacowanie złożoności algorytmu przez wielomian względem wymiaru problemu — co nie jest bynajmniej dla problemów optymalizacyjnych sytuacją typową. Cechą algorytmów zachłannych, w dużej mierze decydującą o ich dużej efektywności, jest fakt, że element raz włączony do rozwiązania, pozostaje w nim do końca. Nie występuje tu „sprawdzanie wszystkich możliwości" charakterystyczne dla algorytmów z powracaniem (por. p. 2.8), które zwykle prowadzi do wykładniczego wzrostu liczby kroków ze wzrostem wymiaru problemu.

Matroidy macierzowe 5.4

Matroidy macierzowe różnią się od matroidów określonych przez niezależne podzbiory przestrzeni liniowej jedynie postacią. Niech $\{v_1, \ldots, v_n\}$ będą elementami pewnej przestrzeni liniowej wymiaru m, i niech $\{b_1, \ldots, b_m\}$ będzie bazą tej przestrzeni. Wektory $v_1, \ldots, v_n$ mają jednoznaczne rozwinięcie względem bazy $b_1, \ldots, b_m$:

$$
\begin{aligned}
v_1 &= a_{11}\, b_1 + a_{21}\, b_2 + \ldots + a_{m1}\, b_m \\
v_2 &= a_{12}\, b_1 + a_{22}\, b_2 + \ldots + a_{m2}\, b_m \\
&\vdots \\
v_n &= a_{1n}\, b_1 + a_{2n}\, b_2 + \ldots + a_{mn}\, b_m
\end{aligned}
$$

Rozważmy macierz

$$
A = \begin{bmatrix}
a_{11} & a_{12} \ldots a_{1n} \\
a_{21} & a_{22} \ldots a_{2n} \\
\vdots & \vdots \quad \ldots \quad \vdots \\
a_{m1} & a_{m2} \ldots a_{mn}
\end{bmatrix} \tag{5.4}
$$

Jej kolumny — oznaczmy je przez $e_1, \ldots, e_n$ — odpowiadają wektorom $v_1, \ldots, v_n$, przy czym podzbiór wektorów jest liniowo niezależny wtedy i tylko wtedy, gdy odpowiadający mu podzbiór kolumn jest liniowo niezależny. Na odwrót, kolumny dowolnej macierzy A postaci (5.4) możemy traktować jako wektory pewnej m-wymiarowej przestrzeni liniowej. Każda taka macierz wyznacza ma-

oid $M(A) = \langle E, \mathscr{I} \rangle$, gdzie E jest zbiorem jej kolumn, natomiast $B \in \mathscr{I}$ wtedy ylko wtedy, gdy zbiór kolumn B jest liniowo niezależny. Tak określony matroid zywamy *matroidem macierzy* A. Matroid nazywamy *macierzowym*, jeśli jest omorficzny z) matroidem pewnej macierzy.

LGORYTM 5.9 (Algorytm zachłanny dla matroidu macierzowego)
ane: Macierz A o m wierszach i n kolumnach, w której kolumny ustawione są według nierosnących wag (wagi są nieujemne).
yniki: Niezależny zbiór kolumn o maksymalnej sumie wag (S zawiera numery tych kolumn).

```
    begin
2     S := Ø;
3     for j := 1 to n do
4       begin i := 0;
5         while (A[i, j] = 0) and (i < m) do i := i+1;
6         if A[i, j] ≠ 0 then (* j-ta kolumna niezerowa *)
7           begin S := S ∪ {j};
8             for k := j+1 to n do
9               for l := 1 to m do
10                A[l, k] := A[l, k] − A[l, j] * A[i, k]/A[i, j]
11          end (* A[i, k] = 0 dla j+1 ⩽ k ⩽ n *)
12      end
13  end
```

roces przekształcania macierzy A realizowany przez ten algorytm to nic innego ak znana z analizy numerycznej eliminacja Gaussa. W każdej iteracji pętli 3 lgorytm sprawdza (wiersz 5) czy j-ta kolumna składa się z samych zer. Jeśli ak ($A[i, j] = 0$ w wierszu 6), to oczywiście j-ta kolumna nie należy do żadnego iniowo niezależnego zbioru kolumn. Jeśli nie ($A[i, j] \neq 0$ w wierszu 6), to włączamy j-tą kolumnę do szukanego zbioru liniowo niezależnych kolumn i odejmujemy, dla wszystkich $k > j$, j-tą kolumnę pomnożoną przez $A[i, k]/A[i, j]$ od k-tej kolumny. Nie zmienia to zależności liniowej kolumn, a jednocześnie powoduje wyzerowanie wszystkich elementów w i-tym wierszu na prawo od $A[i, j]$ łatwo widzieć, że następne iteracje pętli 3 nie zmieniają tego stanu rzeczy). Po zakończeniu działania algorytmu, zbiór S zawiera numery niezerowych kolumn. Kolumny te są liniowo niezależne, gdyż po odpowiedniej permutacji wierszy zawierają one podmacierz wymiaru $|S| \times |S|$ o samych zerach powyżej głównej przekątnej i niezerowych elementach na tej przekątnej.

Złożoność algorytmu można łatwo oszacować, jeśli zauważymy, że dominującą częścią ($O(nm)$ kroków) bloku 4 jest pętla 8. Blok ten jest wykonywany n razy, co daje w sumie $O(n^2m)$ kroków.

Pokażemy teraz przykład zastosowania algorytmu 5.9 związany z planowaniem eksperymentów. Najogólniej rzecz biorąc rozważać będziemy eksperymenty, w których pewien obiekt poddawany jest jednocześnie działaniu wielu nieza-

leżnych czynników, przy czym sumaryczną zmianę obiektu możemy mierzyć ilościowo. Naszym zadaniem jest wyznaczenie wpływu poszczególnych czynników na zmianę obiektu. Przyjmujemy model liniowy, w którym zmiana obiektu wyraża się wzorem

$$b = c_1 x_1 + c_2 x_2 + \ldots + c_m x_m$$

gdzie x_i jest natężeniem i-tego czynnika, a $c_1, \ldots, c_m$ współczynnikami, które należy wyznaczyć. Najprościej byłoby wykonać m eksperymentów, poddając w i-tym eksperymencie obiekt działaniu wyłącznie i-tego czynnika o jednostkowym natężeniu – wtedy $x_i = 1$, $x_k = 0$ dla $k \neq i$, co umożliwia bezpośrednie wyznaczenie c_i. Jednakże takie rozwiązanie jest często niemożliwe ze względów technicznych. Przypuśćmy na przykład, że obserwujemy wpływ zawartości różnych minerałów w glebie na wzrost plonów pewnej uprawy, przy czym do dyspozycji mamy $n\,(n \geqslant m)$ nawozów będących mieszankami tych minerałów w ściśle określonych proporcjach. Równania odpowiadające możliwym eksperymentom możemy przedstawić następująco:

$$\begin{aligned} c_1 a_{11} + c_2 a_{21} + \ldots + c_m a_{m1} &= b_1 \\ c_1 a_{12} + c_2 a_{22} + \ldots + c_m a_{m2} &= b_2 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad &\vdots \\ c_1 a_{1n} + c_2 a_{2n} + \ldots + c_m a_{mn} &= b_n \end{aligned} \tag{5.5}$$

gdzie a_{ij} oznacza ilość i-tego minerału w j-tym eksperymencie (przyjmujemy pewne standardowe ilości nawozów i powierzchnie upraw). Rozważmy macierz $A = [a_{ij}]$. Jej kolumny odpowiadają eksperymentom (nawozom), a wiersze minerałom. Każdy liniowo niezależny zbiór m kolumn określa zbiór eksperymentów, które należy wykonać, by obliczyć współczynniki $c_1, \ldots, c_m$ z nakładu równań (5.5). Jeśli kolumny posortujemy według niemalejącego kosztu odpowiadających im eksperymentów, to algorytm zachłanny wyznaczy najtańszy układ eksperymentów.

Matroidy grafowe 5.

Niech $G = \langle V, E\rangle$ będzie grafem niezorientowanym.
Określmy $M(G) = \langle E, \mathscr{I}\rangle$, gdzie

$$\mathscr{I} = \{A \subseteq E\colon \text{graf } \langle V, A\rangle \text{ nie zawiera cykli}\}$$

TWIERDZENIE 5.10

Dla dowolnego grafu G $M(G)$ jest matroidem.

DOWÓD

Warunek M1 jest oczywiście spełniony. Wobec twierdzenia 5.1 wystarczy pokazać, że zachodzi również warunek M3. Tak jest w istocie, gdyż każdy maksymalny podzbiór $A \in \mathscr{I}$ zawarty w zbiorze $B \subseteq E$ to nic innego jak las rozpinający grafu $\langle V, B\rangle$ (por. p. 2.4). Liczność każdego takiego zbioru A wynosi więc

$$|V| - (\text{liczba składowych spójnych grafu } \langle V, B\rangle) \qquad \blacksquare$$

Matroid $M(G)$ nazywamy *matroidem grafu* G. Dowolny matroid nazywamy grafowym, jeśli jest on (izomorficzny z) matroidem pewnego grafu. Zauważmy, że cykle matroidu $M(G)$ to nic innego jak zbiory krawędzi cykli elementarnych – wyjaśnia to genezę nazwy „cykl" w przypadku dowolnych matroidów. Okazuje się, że każdy matroid grafowy $M(G)$ możemy traktować jako matroid macierzowy odpowiadający macierzy incydencji grafu G traktowanej jako macierz o elementach z ciała dwuelementowego $Z_2 = \{0, 1\}$. Dodawanie w takim ciele wykonujemy modulo 2, a mnożenie tak jak zwykłe mnożenie liczb całkowitych:

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Przypomnijmy, że kolumny macierzy incydencji odpowiadają krawędziom, wiersze wierzchołkom grafu, przy czym kolumna odpowiadająca krawędzi $\{u, v\}$ zawiera jedynki w wierszach odpowiadających wierzchołkom u, v i zera w pozostałych wierszach.

TWIERDZENIE 5.11

Niech $G = \langle V, E\rangle$ będzie dowolnym grafem, A zaś jego macierzą incydencji. Podzbiór kolumn macierzy A jest liniowo zależny nad ciałem Z_2 wtedy i tylko wtedy gdy odpowiadający mu podzbiór krawędzi zawiera cykl.

DOWÓD

Załóżmy, że zbiór $C \subseteq E$ jest cyklem i rozważmy odpowiadający mu zbiór kolumn macierzy A. Zbiór ten zawiera w każdym niezerowym wierszu dwie jedynki, jest więc liniowo zależny nad ciałem Z_2 – jego suma obliczona modulo 2 daje kolumnę zerową.

Na odwrót, załóżmy, że pewien niepusty zbiór kolumn macierzy A jest liniowo zależny nad Z_2. Oznacza to, że zawiera on niepusty podzbiór kolumn o sumie będącej kolumną zerową (zauważmy, że w przypadku ciała Z_2 kombinacja liniowa o niezerowych współczynnikach to po prostu suma). Niech $B \subseteq E$ będzie odpowiadającym mu zbiorem krawędzi. $\langle V, B\rangle$ jest wtedy grafem o parzystym stopniu każdego wierzchołka. Graf ten zawiera cykl, gdyż każdy niepusty graf bez cykli zawiera co najmniej jeden wierzchołek stopnia 1. ∎

Na mocy tego twierdzenia, do znajdowania bazy matroidu $M(G)$ o minimalnej wadze – innymi słowy minimalnego drzewa (ogólniej: lasu, tzn. grafu, którego wszystkie spójne składowe są drzewami) rozpinającego w G – możemy użyć algorytmu 5.9. Złożoność takiego rozwiązania wynosi $O(m^2 n)$, gdzie $n = |V|$ i $m = |E|$.

Istnieją jednak znacznie efektywniejsze algorytmy. Opiszemy jeden z nich, pochodzący od Kruskala [40]. Warto wspomnieć, że algorytm ten odegrał dużą

rolę w rozwoju teorii opisanej w tym rozdziale – algorytm 5.7 został sformułowany jako uogólnienie algorytmu Kruskala z matroidu grafowego na dowolne matroidy.

ALGORYTM 5.12 (por. poz. [40])

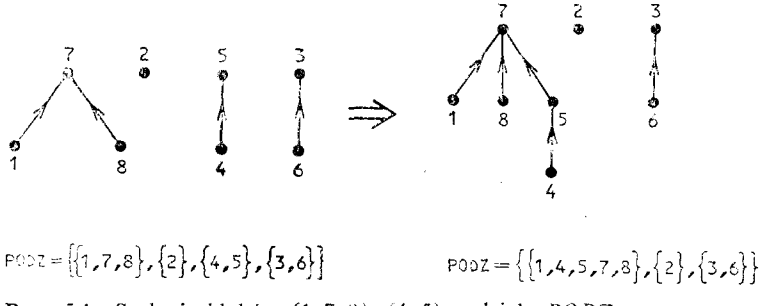
Dane: Graf $G = \langle V, E \rangle$ w postaci listy krawędzi $e_1, \ldots, e_m$ posortowanej według niemalejących wag.

Wyniki: Minimalny las rozpinający (S zawiera zbiór jego krawędzi).

```
1  begin
2    S := Ø;
3    PODZ := {{v}: v ∈ V};
4    for i := 1 to m do
5      if e_i = {u, v}, gdzie u ∈ A, v ∈ B, A, B ∈ PODZ, A ≠ B then
6        begin S := S ∪ {e_i};
7          PODZ := (PODZ \ {A, B}) ∪ {A ∪ B}
8        end
9  end
```

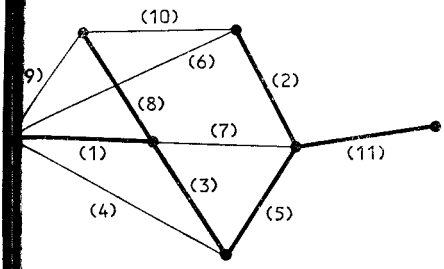
W algorytmie tym, będącym po prostu wersją algorytmu zachłannego dla matroidu $M(G)$, las rozpinający grafu G budujemy krok po kroku, kolejno dodając do S krawędzie nie powodujące zamknięcia cyklu. $PODZ$ zawiera podział zbioru V na zbiory wierzchołków składowych spójnych grafu określonego przez aktualną zawartość zbioru S. Krawędź e_i analizowaną w wierszu 5 dodajemy do S tylko wtedy gdy ma oba końce w różnych blokach podziału $PODZ$ – w przeciwnym razie jej dodanie utworzyłoby cykl. Po dodaniu e_i do S należy bloki $A, B \in PODZ$ zawierające końce krawędzi e_i zastąpić ich sumą $A \cup B$.

Złożoność algorytmu Kruskala zależy istotnie od reprezentacji podziału $PODZ$ i sposobu, w jaki dokonujemy scalania jego bloków. Istnieją bardzo efektywne metody rozwiązania tego problemu (por. poz. [67]). Dla naszych celów wystarczy następująca: Każdy blok podziału reprezentowany jest przez drzewo zorientowane, w którym z każdego wierzchołka istnieje droga do korzenia. Korzeń identyfikuje dany blok. Drzewo takie zawiera również informację



RYS. 5.1 Scalanie bloków $\{1, 7, 8\}$, $\{4, 5\}$ podziału $PODZ$

swojej wysokości. Identyfikacja bloku zawierającego dany wierzchołek następuje przez przejście drogi od tego wierzchołka do korzenia. Scalenie bloków A, B następuje poprzez dodanie krawędzi od korzenia drzewa reprezentującego jeden blok do korzenia drzewa reprezentującego drugi (rys. 5.1). Jeśli przy scalaniu



Rys. 5.2 Minimalne drzewo rozpinające skonstruowane przez algorytm Kruskala

dana krawędź prowadzi zawsze od drzewa o mniejszej (lub równej) wysokości, w trakcie algorytmu wysokość każdego drzewa jest nie większa od logarytmu liczby jego wierzchołków, a tym samym od log n (por. zad. 5.13).

Przykładowe minimalne drzewo rozpinające skonstruowane przez algorytm Kruskala przedstawiono na rys. 5.2. (w nawiasach podano wagi krawędzi).

Matroidy transwersalne 5.6

Niech $\mathscr{A} = (A_1, \ldots, A_n)$ będzie rodziną podzbiorów (niekoniecznie różnych) pewnego zbioru skończonego E. Powiemy, że zbiór $S \subseteq E$ jest *selektorem częściowym* rodziny $\mathscr{A}$, jeśli istnieje odwzorowanie różnowartościowe $\varphi\colon S \to \{1, \ldots, n\}$ takie, że $e \in A_{\varphi(e)}$ dla każdego $e \in S$. Równoważnie możemy powiedzieć, że zbiór $S = \{e_1, \ldots, e_k\}$ jest selektorem częściowym rodziny $\mathscr{A}$, jeśli dla pewnej permutacji f zbioru $\{1, \ldots, k\}$ i dla pewnego ciągu wskaźników $1 \leqslant i_1 \leqslant \ldots \leqslant i_k \leqslant n$ ciąg $\langle e_{f(1)}, \ldots, e_{f(k)}\rangle$ jest systemem różnych reprezentantów dla ciągu $\langle A_{i_1}, \ldots, A_{i_k}\rangle$ (por. p. 4.4). Zauważmy, że dla selektora częściowego S rodziny $(A_1, \ldots, A_n)$ istnieje na ogół wiele różnych funkcji różnowartościowych φ: $S \to \{1, \ldots, n\}$ spełniających warunek $e \in A_{\varphi e}$, $e \in S$.

Zasadniczym rezultatem tego punktu jest następujące twierdzenie pochodzące od Edmondsa i Fulkersona (por. poz. [16]):

TWIERDZENIE 5.13

Niech $\mathscr{A} = (A_1, \ldots, A_n)$ będzie rodziną podzbiorów zbioru skończonego E i niech $\mathscr{I}$ będzie rodziną selektorów częściowych rodziny A. Wtedy $M(\mathscr{A}) = \langle E, \mathscr{I}\rangle$ jest matroidem.

DOWÓD

Warunek M1 jest oczywiście spełniony. Aby pokazać M2 załóżmy, że $A, B \in \mathscr{I}$, $|A| = k$, $|B| = k+1$. Możemy zakładać, że $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_{k+1}\}$,

gdzie $a_i = b_i$ dla $1 \leqslant i \leqslant r = |A \cap B|$. Jeśli $r = k$ (tzn. $A \subseteq B$), to $A \cup \{b_{k+1}\} \in \mathscr{I}$ i warunek M2 jest spełniony. Niech więc $r < k$ i załóżmy, że $\langle a_1, \ldots, a_k \rangle$ jest systemem różnych reprezentantów dla $\langle A_{i_1}, \ldots, A_{i_k} \rangle$, a $\langle b_1, \ldots, b_{k+1} \rangle$ jest systemem różnych reprezentantów dla $\langle A_{j_1}, \ldots, A_{j_{k+1}} \rangle$ (wskaźniki $i_1, \ldots, i_k$ są parami różne, podobnie jak $j_1, \ldots, j_{k+1}$). Rozpatrzmy następującą konstrukcję, która albo znajduje element $b \in B \setminus A$ taki, że $A \cup \{b\} \in \mathscr{I}$ albo też zastępuje ciąg $\langle A_{i_1}, \ldots, A_{i_k} \rangle$ pewnym nowym ciągiem, dla którego $\langle a_1, \ldots, a_k \rangle$ jest również systemem różnych reprezentantów:

Konstrukcja

Wybierzmy wskaźnik $p \leqslant k+1$ taki, że $j_p \notin \{i_1, \ldots, i_k\}$. Jeśli można ten wskaźnik wybrać tak, by $r+1 \leqslant p \leqslant k+1$, to ciąg $\langle a_1, \ldots, a_k, b_p \rangle$ jest systemem różnych reprezentantów dla $\langle A_{i_1}, \ldots, A_{i_k}, A_{j_p} \rangle$, a co za tym idzie $A \cup \{b_p\} \in \mathscr{I}$. W przeciwnym przypadku, tzn. gdy $1 \leqslant p \leqslant r$, mamy $a_p = b_p$ i ciąg $\langle a_1, \ldots, a_k \rangle$ jest systemem różnych reprezentantów dla $\langle A_{i_1}, \ldots, A_{i_{p-1}}, A_{i_{p+1}}, \ldots, A_{i_k} \rangle$.

Przyjrzyjmy się bliżej temu ostatniemu ciągowi. Jeśli ciągi $\langle i_1, \ldots, i_r \rangle$ i $\langle j_1, \ldots, j_r \rangle$ pokrywały się na $q < r$ pozycjach, to ciągi $\langle i_1, \ldots, i_{p-1}, j_p, i_{p+1}, \ldots, i_r \rangle$, $\langle j_1, \ldots, j_r \rangle$ pokrywają się na $q+1$ pozycjach, jako że $j_p \neq i_p$. Powtarzając naszą konstrukcję albo znajdujemy żądany selektor częściowy $A \cup \{b\}$, $b \in B \setminus A$, albo też doprowadzamy do sytuacji, w której $\langle i_1, \ldots, i_r \rangle = \langle j_1, \ldots, j_r \rangle$. W tym ostatnim przypadku jednak $j_p \notin \{i_1, \ldots, i_k\}$ dla pewnego p, $r+1 \leqslant p \leqslant k+1$ (nie może bowiem być $\{j_{r+1}, \ldots, j_{k+1}\} \subseteq \{i_{r+1}, \ldots, i_k\}$). Wykonanie naszej konstrukcji jeszcze raz powoduje znalezienie elementu $b_p \in B \setminus A$ takiego, że $A \cup \{b_p\} \in \mathscr{I}$. ■

Matroid $M(\mathscr{A})$ nazywamy *matroidem transwersalnym* rodziny $\mathscr{A}$.

Opiszemy teraz algorytm zachłanny dla matroidów transwersalnych. Algorytm ten jest modyfikacją metody znajdowania systemów reprezentantów (dokładniej: skojarzenia o maksymalnej liczności w grafie dwudzielnym) opartej na technice ścieżek naprzemiennych opisanej w rozdziale 4. Przypomnijmy, że rodzinę $(A_1, \ldots, A_n)$ podzbiorów zbioru $E = \{e_1, \ldots, e_m\}$ reprezentujemy przez graf dwudzielny H o wierzchołkach $u_1, \ldots, u_m, v_1, \ldots, v_n$ i krawędziach postaci $\{u_i, v_j\}$, gdzie $e_i \in A_j$. Każdemu systemowi różnych reprezentantów $\langle e_{i_1}, \ldots, e_{i_k} \rangle$ ciągu $\langle A_{j_1}, \ldots, A_{j_k} \rangle$ (wskaźniki $j_1, \ldots, j_k$ parami różne) odpowiada niezależny zbiór krawędzi $M = \{\{u_{i_1}, v_{j_1}\}, \ldots, \{u_{i_k}, v_{j_k}\}\}$ w grafie H.

ALGORYTM 5.14 (Algorytm zachłanny dla matroidu transwersalnego)

Dane: Rodzina $\mathscr{A} = (A_1, \ldots, A_n)$ podzbiorów zbioru $E = \{e_1, \ldots, e_m\}$ reprezentowana za pomocą grafu dwudzielnego G. Elementy $e_1, \ldots, e_m$ poindeksowane są według nierosnących wag (wagi są nieujemne).

Wyniki: Selektor częściowy S o największej wadze dla rodziny A (oraz skojarzenie M grafu H określające funkcję $\varphi: S \to \{1, \ldots, n\}$, o której mowa w definicji selektora częściowego).

```
1  begin
2    S := Ø; M := Ø;
3    for i := 1 to m do
4      if istnieje w G ścieżka naprzemienna P względem M
5         o początku w e_i then
6         begin S := S ∪ {e_i}; M := P⊗M
7         end
8  end
```

łożoność tego algorytmu zależy oczywiście od metody poszukiwania ścieżki aprzemiennej P (wiersz 4 i 5). Jeśli zastosujemy metodę przeszukiwania w głąb ak jak w punkcie 2.2, to liczba kroków potrzebna do znalezienia ścieżki jest rzędu czby krawędzi grafu G, tzn. $O\left(\sum_{i=1}^{n} |A_i|\right)$. Całkowita złożoność algorytmu 5.14 est więc wtedy $O\left(m \sum_{i=1}^{n} |A_i|\right)$.

Pokażemy teraz dwa zagadnienia praktyczne, które mogą być rozwiązane rzy pomocy tego algorytmu.

Przypuśćmy, że pewien przedsiębiorca zatrudnia n osób, przy czym i-ta soba ma kwalifikacje do wykonywania dowolnej spośród prac z pewnego bioru A_i. Oznaczmy $\bigcup_{i=1}^{n} A_i = E = \{e_1, \ldots, e_m\}$ i załóżmy, że zysk przedsiębiorcy wykonywania pracy e_j wynosi $w(e_j)$, oraz że każda zatrudniona osoba może ykonać co najwyżej jedną pracę. Należy znaleźć takie przyporządkowanie prac ogącym je wykonać osobom, aby zysk był maksymalny. Tłumacząc to na język atroidów, problem polega tu na znalezieniu bazy o maksymalnej wadze matroidu ranswersalnego $M(\mathscr{A})$, gdzie $\mathscr{A} = (A_1, \ldots, A_n)$. Ściślej rzecz biorąc, należy znaleźć nie tylko selektor częściowy S o maksymalnej wadze, lecz również funkcję różnowartościową $\varphi: S \to \{1, \ldots, n\}$, dla której $e \in A_{\varphi_e}$, $e \in S$. Tego właśnie dokonuje algorytm 5.14.

Warto tu przypomnieć, że tak jak we wszystkich algorytmach zachłannych przedstawionych w tym rozdziale, baza o najmniejszej (największej) wadze zależy wyłącznie od uporządkowania elementów $e_1, \ldots, e_m$ według niemalejących wag. Oznaczmy przez $e_i \leqslant e_j$ fakt, iż $w(e_i) \leqslant w(e_j)$ („praca e_j jest nie mniej opłacalna niż e_i"). Wiemy, że zbiór prac $S = \{a_1, \ldots, a_k\}$ (gdzie $a_1 \geqslant \ldots \geqslant a_k$) jest optymalny w sensie Gale'a, tak więc dla każdego innego zbioru prac $T = \{b_1, \ldots, b_l\}$ (gdzie $b_1 \geqslant \ldots \geqslant b_l$), który może być wykonany przez zatrudnione osoby mamy $l \leqslant k$ oraz $b_i \leqslant a_i$ dla $1 \leqslant i \leqslant l$.

Drugi przykład, który pokażemy, dotyczy znalezienia optymalnej kolejności wykonywania zadań. Załóżmy, że jest dane m zadań $e_1, \ldots, e_m$, przy czym wykonanie każdego z nich wymaga tej samej ilości czasu, powiedzmy jednej godziny. Zadania wykonujemy kolejno, nie przerywając żadnego rozpoczętego zadania, przy czym w każdej chwili czasu może być wykonywane najwyżej jedno. Dla każdego i, $1 \leqslant i \leqslant m$ dany jest również *termin* d_i wykonania zadania e_i (liczba

godzin od chwili zerowej) oraz *kara w* (e_i), którą należy zapłacić za niewykonanie go w terminie (zakładamy, że kara ta nie zależy od tego, o ile godzin został przekroczony termin). Należy znaleźć kolejność wykonania zadań, która minimalizuje sumę kar. Równoważnie problem ten możemy sformułować jako znalezienie wykonywalnego w terminie zbioru zadań o maksymalnej sumie kar (maksymalizujemy tu sumę „unikniętych"kar). Należy przy tym znaleźć również kolejność wykonania zadań z tego optymalnego zbioru. Związek tego problemu z matroidami transwersalnymi jest następujący. Określmy $E = \{e_1, \ldots, e_m\}$, $n = max_{1 \leqslant i \leqslant m} d_i$ oraz $A_i = \{e_j \in E: d_j \geqslant i\}$ dla $1 \leqslant i \leqslant n$. Przy takich oznaczeniach dowolny selektor częściowy S rodziny $\mathscr{A} = (A_1, \ldots, A_n)$ określa wykonywalny w terminie zbiór zadań, przy czym funkcja różnowartościowa φ: $S \to \{1, \ldots, n\}$ taka, że $e \in A_{\varphi_e}$, $e \in S$ określa dla każdego zadania $e \in S$ godzinę $\varphi(e)$, w której to zadanie ma być wykonywane. Optymalne rozwiązanie S jest więc bazą o największej wadze matroidu transwersalnego $M(\mathscr{A})$ – możemy je znaleźć używając algorytmu 5.14.

Zadania

5.1 Udowodnij, że rodzina $\mathscr{B} \subseteq \mathscr{P}(E)$ jest bazą pewnego (jednoznacznie wyznaczonego przez $\mathscr{B}$) matroidu wtedy i tylko wtedy, gdy spełnia warunek

B1 Dla dowolnych $B_1, B_2 \in \mathscr{B}$ oraz $e \in B_1 \setminus B_2$ istnieje $f \in B_2 \setminus B_1$ takie, że $(B_1 \setminus \{e\}) \cup \{f\} \in \mathscr{B}$.

5.2 Dla następujących pojęć związanych z matroidem:
(a) rodzina $\mathscr{I}$ zbiorów niezależnych matroidu,
(b) rodzina $\mathscr{B}$ baz matroidu,
(c) rodzina $\mathscr{C}$ cykli matroidu,
(d) rodzina $\mathscr{F}$ podprzestrzeni matroidu,
(e) funkcja rangi r,
(f) operacja rozpinania podprzestrzeni sp,
wyraź każde z nich w terminach każdego z pozostałych.

5.3 Udowodnij, że rodzina $\mathscr{C} \subseteq \mathscr{P}(E)$ jest rodziną cykli pewnego matroidu wtedy i tylko wtedy, gdy spełnia warunki C1 i C2.

5.4 Udowodnij, że operacja sp: $\mathscr{P}(E) \to \mathscr{P}(E)$ jest operacją rozpinania podprzestrzeni w pewnym matroidzie wtedy i tylko wtedy gdy spełnia warunki S1, S2, S3, S4.

5.5 Udowodnij, że funkcja r jest funkcją rangi w pewnym matroidzie wtedy i tylko wtedy, gdy spełnia warunki R1, R2, R3 (lub też R1, R4, R5).

5.6 Udowodnij, że jeśli element należy do każdej bazy matroidu, to nie należy do żadnego cyklu tego matroidu.

5.7 Sprawdź, że dla dowolnego zbioru skończonego E i dowolnego $k \leqslant |E|$ para $M = \langle E, \mathscr{I} \rangle$, gdzie
$\mathscr{I} = \{A \subseteq E: |A| \leqslant k\}$
jest matroidem. Wyznaczyć pojęcia (a)÷(f) z zad. 5.2 dla tego matroidu.

5.8 Udowodnij, że w dowolnym matroidzie, $e \in sp(A)$ wtedy i tylko wtedy, gdy istnieje cykl C taki, że $C \setminus A = \{e\}$.

5.9 Udowodnij, że jeśli C_1, C_2 są dowolnymi dwoma różnymi cyklami matroidu oraz $f \in C_1 \setminus C_2$, to dla każdego $e \in C_1 \cap C_2$ istnieje cykl C_3 taki, że $f \in e \in C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.

5.10 Udowodnij, że jeśli $\mathscr{B}$ jest rodziną baz matroidu $M = \langle E, \mathscr{I} \rangle$, to $\mathscr{B}^* = \{E \setminus B: B \in \mathscr{B}\}$ jest rodziną baz pewnego matroidu M^* (M^* nazywamy matroidem *dualnym* do M). Udowodnij, że funkcja rangi matroidu dualnego wyraża się wzorem

$$r^*(A) = |A| + r(E \setminus A) - r(E)$$

5.11 Udowodnij, że dla dowolnego matroidu $M = \langle E, \mathscr{I} \rangle$ i dowolnego zbioru $S \subseteq E$ para $\langle S, \mathscr{I}' \rangle$, gdzie $\mathscr{I}' = \{A \in \mathscr{I}: A \subseteq S\}$ jest matroidem. Czy matroidem jest też zawsze $\langle S, \mathscr{I}'' \rangle$, gdzie $\mathscr{I}'' = \{S \cap A: A \in \mathscr{I}\}$?

5.12 Zastosuj algorytm 5.11 do macierzy

$$A = \left[\begin{array}{cccccc} 1 & 4 & 2 & 1 & 3 & 6 \\ 2 & 8 & 4 & 1 & 7 & 2 \\ -7 & 13 & -14 & 1 & 1 & 6 \\ 0 & 6 & 0 & 1 & 12 & 2 \end{array}\right]$$

(kolumny posortowane są według niemalejących wag).

5.13 Udowodnij, że przy opisanej w punkcie 4.5 realizacji algorytmu Kruskala wysokość drzewa reprezentującego dowolny blok B podziału $PODZ$ nie przekracza $\log |B|$.

5.14 Udowodnij, że każdy matroid podziałowy jest szczególnym przypadkiem matroidu transwersalnego.

5.15 Udowodnij, że jeśli istnieje system różnych reprezentantów dla $\langle A_1, \ldots, A_n \rangle$ oraz $\langle x_1, \ldots, x_k \rangle$ jest systemem różnych reprezentantów dla $\langle A_1, \ldots, A_k \rangle$ ($k < n$), to istnieje element x_{k+1} oraz permutacja f zbioru $\{1, \ldots, k+1\}$, taka że $\langle x_{f(1)}, \ldots, x_{f(k+1)} \rangle$ jest systemem różnych reprezentantów dla $\langle A_1, \ldots, A_{k+1} \rangle$.

Literatura

1. AHO A. V., HOPCROFT J. E., ULLMAN J. D.: *Projektowanie i analiza algorytmów komputerowych.* Warszawa, PWN 1983.
2. BANACHOWSKI L., KRECZMAR A.: *Elementy analizy algorytmów.* Wyd. 1 Warszawa, WNT 1981 (wznowienie w druku).
3. BELLMAN R. E.: On a routing problem. *Quart. Appl. Math.* 1958, **16**, s. 87–90.
4. BENEŠ V. E.: *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York, Academic Press 1965.
5. BEST M. R., VAN EMDE BOAS P., LENSTRA H. W.: A sharpened version of the Aandreaa-Rosenberg conjecture. *Techn. Rep.* ZW 30/74, Mathematisch Centrum, Amsterdam 1974
6. BIRKHOFF G.: Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman. Rev. Ser, A* 1946, **5**, s. 147–151.
7. CLOS C.: A study of non-blocking switching networks. *Bell System Tech. J.*, 1953, **32**, s. 406–424.
8. De BRUIJN N. G.: A combinatorial problem. *Indagationes Math.*, 1946, **8**, s. 461–467.
9. DEO N.: *Teoria grafów i jej zastosowania w technice i informatyce.* Warszawa, PWN 1980.
10. DIGUID A. M.: Structural properties of switching networks. *Techn. Rep. BTL-7*, Brown Univ., Providence, RI 1959.
11. DIJKSTRA E. W.: A note on two problems in connexion with graphs. *Numer. Math.*, 1959, **1**, s. 269–271.
12. DILWORTH R. P.: A decomposition theorem for partially ordered sets. *Ann. Math.*, 1950, **51**, s. 161–166.
13. Диниц Е. А.: Алгоритм решения задачи о максимальном потоке в сети со степенной оценкой. *Докл. Академии Наук СССР, Серия Мат., Физ.*, 1970, **194**, s. 754–757.
14. DUCKWORTH R., STEDMAN F.: *Tintinnalogia, or the art of ringing.* Kingsmead Reprints, Bath 1970 (wydana po raz pierwszy anonimowo w Londynie w roku 1668).
15. EDMONDS J.: Matroids and the greedy algorithms. *Math. Programming*, 1971, **1**, s. 127–136.
16. EDMONDS J., Fulkerson D. R.: Transversals and matroid patrition. *J. Res. NBS*, 1965, 69B, s. 147–153.
17. EDMONDS J., KARP R. M.: Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 1972, **19**, s. 248–264.
18. EVEN S.: *Graph algorithms.* Potomac, MD, Computer Science Press 1979.
19. EVEN S., KARIV O.: An $O(n^{2.5})$ algorithm for maximum matching in general graphs. *Proc.* 16th *Annual Symp. on Foundations of Computer Sience, IEEE*, 1975, s. 100–112.
20. FLOYD R. W.: Algorithm 97: Shortest path. *Comm. ACM*, 1962, **5**, s. 345.
21. FLOYD R. W.: Algorithm 245: treesort 3. *Comm. ACM*, 1964, 7, s. 701.
22. FORD L. R.: Network filow theory. *The Rand Corp.*, P–932, August 1956.
23. FORD L. R., Fulkerson D. R.: *Przepływy w sieciach.* Warszawa, PWN 1969.
24. FROBENIUS G., Über Matrizen aus nicht negativen Elementen. *Sitzb. Preuss. Acad. Wiss.*, 1917, s. 274–277.

25. GALE D.: Optimal assignments in an ordered set: an application of matroid theory. *J. Combinatorial Theory*, 1968, **4**, s. 176–180.
26. GALIL Z.: A new algorithm for the maximal flow problem. *Proc. 19th IEEE Symp. on Foundations of Computer Science*, Ann Arbor, MI, October 1978, s. 231–245.
27. GAREY M. R., JOHNSON D. S.: *Computers and Intractability. A Guide to the Theory of NP-Completeness*. San Francisco, CA, W. H. Freeman and Co., 1979.
28. GLOVER F.: Maximum matching in convex bipartite graph. *Nawal Res. Logist. Quart.*, 1967, **14**, s. 313–316.
29. *Guinnes book of world records*. New York, Sterling 1972.
30. HALL Ph.: On representatives of subsets. *J. London Math. Soc.*, 1935, **10**, s. 26–30.
31. HARARY F.: *Graph Theory*. Reading, MA, Addison-Wesley 1969.
32. HEAP B. R.: Permutations by interchanges. *Comput. J.*, 1963, **6**, s. 293–294.
33. HOPCROFT J., KARP R. M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 1973, **2**, s. 225–231.
34. HOROWITZ E., SAHNI S.: *Fundamentals of Data Structures*, Potomac, MD, Computer Science Press 1976.
35. HOROWITZ E., SAHNI S.: *Fundamentals of Computer Algorithms*. Potomac, MD, Computer Science Press 1978.
36. IGLEWSKI M., MADEY J., MATWIN S.: *Pascal. Język wzorcowy. Pascal* 360. Wyd. 4, Warszawa, WNT 1986.
37. JOHNSON D. B.: Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 1977, **24**, s. 1–13.
38. JOHNSON S. M.: Generation of permutations by adjacent transpositions. *Math. Comp.*, 1963, **17**, s. 282–285.
39. KAYE R.: A Grey code for set partitions. *Information Processing Lett.*, 1976, **5**, s. 171–173.
40. KRUSKAL J. B.: On the shortest spanning subgraph of a graph and the travelling salesman problem. *Proc. Amer. Math. Soc.*, 1956, **7**, s. 48–49.
41. KURATOWSKI K.: *Rachunek różniczkowy i całkowy. Funkcje jednej zmiennej*. Wyd. V, Warszawa, PWN 1971.
42. LAWLER E. L.: *Combinatorial Optimization: Networks and Matroids*. New York, Holt, Rinehart and Winston 1976.
43. LIPSKI W.: On strings containing all subsets as substrings. *Discrete Math.*, 1978, **21**, s. 253–259.
44. LIPSKI W.: More on permutation generation methods. *Computing*, 1979, **23**, s. 357–365.
45. LIPSKI W., PREPARATA F. P.: Yet another permutation sequence. *Techn. Rep. ACT*-10, Coordinated Science Lab., Univ. of Illinois, October 1978.
46. LIPSKI W., PREPARATA F. P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 1981, **15**, s. 329–346.
47. LUM V. Y.: Multiattribute retrieval with combined indexes. *Comm. ACM*, 1970, **13**, s. 660–665.
48. MALHOTRA V. M., KUMAR P. M., MAHESHWARI S. N.: An $O(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Lett.*, 1978, **7**, s. 277–278.
49. MAREK W., ONYSZKIEWICZ J.: *Elementy logiki i teorii mnogości w zadaniach*. Wyd. 2, Warszawa, PWN 1975.
50. MASSON G. M., GINGHER G. C., NAKAMURA S.: A sampler of circuit switching networks. *Computer*, June 1979, s. 32–48.
51. MENDELSOHN N. S., DULMAGE A. L.: Some generalizations of the problem of distinct representatives. *Canad. J. Math.*, 1958, **10**, s. 230–241.
52. MENGER K.: Zur allgemeinen Kurventheorie. *Fund. Math.*, 1927, **10**, s. 96–115.
53. MUNRO I.: Efficient determination of the transitive closure of a directed graph. *Information Processing Lett.*, 1971, **1**, s. 56–58.

54. NIJENHUIS A., WILF H. S.: *Combinatorial Algorithms*. New York, Academic Press 1975.
55. PAN V. JA : Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplications. *Proc. 20th Annual Symp. on Foundations of Computer Science*, October 29–31, 1979, San Juan, Puerto Rico, s. 28–38.
56. RADO R.: Note on independence function. *Proc. London Math. Soc.*, 1957, **7**, s. 337–343.
57. RASIOWA H.: *Wstęp do matematyki współczesnej*. Warszawa PWN, 1968.
58. REINGOLD E. N., NIEVERGELT J., DEO N.: *Combinatorial Algorithms. Theory and Practice*. Englewood Cliffs, N J, Prentice-Hall 1977.
59. RIVEST R. L., VUILLEMIN J.: A generalization and proof of the Aandreaa-Rosenberg conjecture. *Proc. 7th Annual ACM Symp. on Theory of Computing*, May 5–7, 1975, Albuquerque, NM, s. 6–11.
60. SEDGEWICK R.: Permutation generation methods. *Comput. Surveys*, 1977, **9**, s. 137–164 (p. też *Surveyors' Forum*, s. 315–317).
61. SESHU S., REED M. B.: *Linear graphs and electrical networks*. Reading, MA, Addison-Wesley 1961.
62. SLEPIAN D.: *Two theorems on a particular crossbar switching network*. Nie opublikowany artykuł, 1952.
63. SPERNER E.: Ein Satz über Untermengen einer endlichen Menge. *Math. Z.*, 1928, **27**, s. 544–548.
64. STRASSEN V.: Gaussian elimination is not optimal. *Numer. Math.*, 1969, **13**, s. 354–356.
65. SYSŁO M. M., DZIKIEWICZ J.: Computational experience with some transitive closure algorithms. *Computing*, 1975, **15**, s. 33–39.
66. TARJAN R. E.: Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1972, **1**, s. 146–160.
67. TARJAN R. E.: On the efficiency of a good but not linear set merging algorithm. *J. ACM*, 1975, **22**, s. 215–225.
68. TROTTER H. F.: Perm (Algorithm 115). *Comm. ACM*, 1962, **5**, s. 434–435.
69. TUTTE W. T.: *Introduction to the Theory of Matroids*. New York, American Elsevier 1970.
70. WARSHALL S.: A theorem on Boolean matrices. *J. ACM*, 1962, **9**, s. 11–12.
71. WELLS M. B.: Generation of permutations by transposition. *Math. Comp.*, 1961, **15**, s. 192–195.
72. WELSH D. J. A.: *Matroid Theory*. London, Academic Press 1976.
73. WHITNEY H.: On the abstract properties of linear independence. *Amer. J. Math.*, 1935, **57**, s. 509–533.
74. WILSON W. G.: *Change ringing*. London, Faber 1965.
75. WIRTH N.: *Wstęp do programowania systematycznego*. Wyd. 2, Warszawa, WNT 1987.
76. WIRTH N.: *Algorytmy + struktury danych = programy*. Wyd.1, Warszawa, WNT 1980 (wznowienie w druku).

Combinatorics for Programmers

Summary

This book is an exposition of selected topics in combinatorics, graph theory, and combinatorial computing. A special emphasis is put on the algorithmic approach to combinatorial problems: the problems discussed are complemented with detailed algorithms to solve them, and the computational complexity of these algorithms is analysed. The algorithms are slightly condensed informal versions of programs written in Pascal.

The first chapter contains an introduction to the most classical combinatorial objects and techniques (permutations, partitions of a set, partitions of an integer, binomial coefficients, generating functions, the inclusion-exclusion principle, etc.), together with many algorithms – not necessarily classical – for systematically generating the combinatorial objects discussed. Chapter 2 describes basic techniques used in the design of graph algorithms, including depth first search, breadth first search, and backtracking. Chapter 3 is on finding shortest paths in graphs with weighted edges. The next chapter describes methods of finding a maximum flow in a network. The augmenting path techniques are discussed, and the algorithm of Malhotra, Kumar and Maheshwari is presented in detail. Flow techniques are then applied to related problems of maximum matchings in bipartite graphs (the classical Hopcroft-Karp algorithm is included), and of systems of distinct representatives. The decomposition of a partially ordered set into chains is also treated. The last chapter explains the basic notions connected with matroids and describes the application of matroid greedy algorithms to combinatorial optimization.

There is a set of exercises at the end of each chapter.

The book is intended for computer programmers, computer scientists and students.

Комбинаторика для программистов

Резюме

В предлагаемой книге представлены избранные темы из области комбинаторики, теории графов и комбинаторных алгоритмов. Особое внимание уделено алгоритмической трактовке комбинаторных проблем: рядом с обсуждаемыми проблемами приводятся подробные алгоритмы их решения и проанализирована временная сложность этих алгоритмов. Эти алгоритмы — немного конденсированные неформальные варианты программ, написанных на языке Паскаль.

Первая глава содержит вводные сведения о самых классических комбинаторных объектах и техниках (перестановка, разбиение множества, разбиение числа, биномиальный коэффициент, производящая функция, принцип включения и исключения ит.д. (вместе со многими алгоритмами — необязательно классическими — систематического генерирования приведенных комбинаторных объектов. Вторая глава описывает основные техники, применяемые в проектировании алгоритмов обработки графов, включая поиск в глубину, поиск в ширину и перебор с возвратом. В третьей главе рассмотрены вопросы кратчайших путей в графах с ребрами с весами. В следующей главе описаны методы нахождения максимального потока в сети. Обсуждена техника увеличивающих путей и подробно представлены алгоритмы Малхотры, Кумара и Махешвари. Техника потоков применяется в дальнейшем к похожим проблемам, таким, как максимальное паросочетание в двудольном графе (классический алгоритм Хопкрофта-Карпа), система различных представителей. Обсуждается также разложение частично упорядоченного множества на цепи. В последней главе объясняются основные понятия, связанные с матроидами и описывается применение жадного алгоритма в комбинаторной оптимизации.

Все главы книги содержат соотствующий набор задач. Книга предназначена главным образом для программистов, заинтересованных нечисленным программированием и желающих пополнить практические знания теоретическими основами, а также для научных работников и студентов, занимающихся вычислительной техникой.

Biblioteka Inżynierii Oprogramowania

Wykaz wydanych książek

S. ALAGIĆ, M. A. ARBIB – Projektowanie programów poprawnych i dobrze zbudowanych
I. O. ANGELL – Wprowadzenie do grafiki komputerowej wyd. 1 i 2
L. BANACHOWSKI, A. KRECZMAR – Elementy analizy algorytmów
L. BANACHOWSKI, A. KRECZMAR, W. RYTTER – Analiza algorytmów i struktur danych
J. BIELECKI – System VSAM. Zasady stosowania w języku PL/I
L. BOLC, M. CICHY, Ł. RÓŻAŃSKA – Przetwarzanie języka naturalnego
S. BORAK, J. KLACZAK, S. KORCZAK, Z. PŁOSKI – System operacyjny George 3, wyd. 1 i 2 rozszerzone
J. M. BRADY – Informatyka teoretyczna w ujęciu programistycznym
K. L. CLARK, F. G. McCABE – Micro-Prolog
M. DĄBROWSKI, K. LAUS-MĄCZYŃSKA – Metody wyszukiwania i klasyfikacji informacji
P. DEMBIŃSKI, J. MAŁUSZYŃSKI – Matematyczne metody definiowania języków programowania
J. DEMINET - System operacyjny RSX–11
E. W. DIJKSTRA – Umiejętność programowania, wyd. 1 i 2
S. GASIK, P. KULCZYCKI, K. PIASECKI, J. WITASZEK – PL/I (F)
P. GIZBERT-STUDNICKI, J. KARCZMARCZUK – Snobol 4
M. GŁOWACKI – Systemy operacyjne DOS i OS
M. J. C. GORDON – Denotacyjny opis języków programowania
R. E. GRISWOLD, M. T. GRISWOLD – Icon
L. J. HOFFMAN – Poufność w systemach informatycznych
M. IGLEWSKI, J. MADEY, S. MATWIN – Pascal. Język wzorcowy. Pascal 6000, wyd. 2
M. IGLEWSKI, J. MADEY, S. MATWIN – Pascal. Język wzorcowy. Pascal 360, wyd. 3 zmienione i 4
W. ISZKOWSKI, M. MANIECKI – Programowanie współbieżne
R. JAGIELSKI – Tablice rozproszone
A. P. JERSZOW – Wprowadzenie do teorii programowania

C. B. JONES – Konstruowanie programowania metodą systematyczną
A. KASSUR, P. PERKOWSKI – Obliczeniowe aspekty projektowania układów elektronicznych
B. W. KERNIGHAN, D. M. RITCHIE – Język C, wyd. 1 i 2
F. KLUŹNIAK, S. SZPAKOWICZ – Prolog
H. KOPETZ – Niezawodność oprogramowania
W. LIPSKI – Kombinatoryka dla programistów, wyd. 1 i 2
J. MARTINEK – Lisp. Opis, realizacja i zastosowania
G. J. MYERS – Projektowanie niezawodnego oprogramowania
L. NIEMCZYCKI – Oprogramowanie teleprzetwarzania maszyn Jednolitego Systemu
H. OKTABA, W. RATAJCZAK – Simula 67
J. OLSZEWSKI – Projektowanie struktur systemów operacyjnych
W. PACHELSKI – Fortran dla maszyn Odra serii 1300
W. PACHELSKI – Fortran IV dla maszyn Jednolitego Systemu, wyd. 1 i 2 poprawione i rozszerzone
T. PAVLIDIS – Grafika i przetwarzanie obrazów. Algorytmy
P. PERKOWSKI – Technika symulacji cyfrowej
Przegląd metod i algorytmów numerycznych, cz. 1 – J. i M. Jankowscy wyd. 1 i 2
Przegląd metod i algorytmów numerycznych, cz. 2 – M. Dryja, J. i M. Jankowscy wyd. 1 i 2
I. C. PYLE – Ada
W. REISIG – Sieci Petriego
D. VAN TASSEL – Praktyka programowania, wyd. 1 i 2 rozszerzone
W. M. TURSKI – Metodologia programowania, wyd. 1 i 2 rozszerzone
E. Ch. TYUGU – Programowanie z bazą wiedzy
J. D. ULLMAN – Systemy baz danych
J. WALASEK – Konwersacyjne otoczenie programowe Pascala
N. WIRTH – Modula 2
N. WIRTH – Wstęp do programowania systematycznego, wyd. 1 i 2
R. WIT – Metody programowania nieliniowego. Minimalizacja funkcji gładkich
K. ZORYCHTA, W. OGRYCZAK – Programowanie liniowe – całkowitoliczbowe

WNT Warszawa 1989. Wydanie II. Nakład 3700+300 egz. Ark. wyd. 12,7. Ark. druk. 11,5. Format B5. Papier piśmienny kl. III. 71g. Pracę oddano do składania 4.V.1988 r. Podpisano do druku w marcu 1989 r. Druk ukończono w kwietniu 1989 r. Symbol IPT/82425/MEN. Poznańskie Zakłady Graficzne. Zam. 70482/88 — Z-8/940