Grafika Komputerowa i Multimedia

Wykład 9

Algorytmy kompresji bezstratnej

Algorytm Huffmana i LZ77

Damian Grela
e-mail: **dgrela@pk.edu.pl**
http://www.dgrela.pl

Algorytm Huffmana

- Algorytm **Huffmana**, nazwany tak od nazwiska jego twórcy, został ogłoszony drukiem w 1952 roku, a więc ponad pół wieku temu.
- Mimo to pozostaje nadal jednym z najbardziej rozpowszechnionych algorytmów kompresji bezstratnej, dzięki efektywności, względnej prostocie i brakowi zastrzeżeń patentowych.

Algorytm Huffmana

- W słowniku kompresora Huffmana znajdują się wyłącznie elementy alfabetu wiadomości, tak więc rozmiar słownika jest stały i z góry znany (słownik nie jest rozbudowywany w czasie kompresji).
- Kody przydzielone poszczególnym znakom alfabetu są natomiast różne. Kody dobiera się w taki sposób, żeby łączna długość skompresowanej wiadomości była jak najkrótsza.

- Osiąga się to wtedy, gdy długość kodu dla danego elementu alfabetu jest odwrotnie proporcjonalna do jego częstości występowania w wiadomości (najczęściej występujące elementy alfabetu mają najkrótsze kody).
- Cechą charakterystyczną zestawu kodów Huffmana jest jego prefiksowość, oznacza to, że żaden kod Huffmana nie jest początkiem innego kodu. Znacznie upraszcza to budowę dekodera.

- W idealnym przypadku częstości występowania poszczególnych znaków alfabetu wyznaczamy dokładnie, analizując całą wiadomość. W przypadku wiadomości bardzo długich, bądź też generowanych na bieżąco, jest to niemożliwe.

Wtedy mamy do wyboru kilka rozwiązań:

1. Estymacja statyczna na podstawie początku wiadomości, lub wiedzy o procesie generującym wiadomość.
2. Estymacja blokowa, dzielimy wiadomość na duże bloki i dla każdego budujemy oddzielny słownik.
3. Estymacja adaptacyjna, statystyki są korygowane na bieżąco na bazie określonej ilości ostatnich znaków (tzw. okno kompresora).

- Nie należy uważać, że algorytm Huffmana jest zawsze lepszy od LZW, choć istotnie z reguły algorytm Huffmana daje lepszą kompresję (ale nie zawsze!).

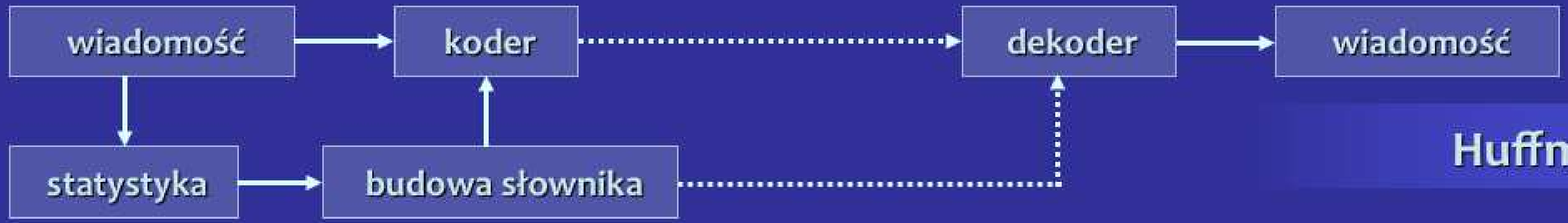
- Należy pamiętać o dwóch potencjalnych wadach algorytmu Huffmana:
  - Algorytm Huffmana nie posiada w słowniku sekwencji dłuższych niż 1 znak alfabetu.
  - Algorytm Huffmana wymaga by słownik został przesłany do dekodera.

Huffman vs. LZW

- Pierwszą wadę łatwo sobie uzmysłowić, niech sekwencją do kompresji będzie ABCABCABCABC...
- Tu koder LZW szybko sobie w słownik wstawi symbol ABC (czy też BCA, CAB, obojętnie) i będzie emitował jeden symbol, co więcej wkrótce zbuduje sobie również dłuższe sekwencje typu ABCABC.

- Huffman będzie zaś bezradnie powtarzał trzy symbole na każde ABC...
- Druga wada jest oczywista – konieczność przesłania słownika zwiększa rozmiar kompresowanego pliku, co pogarsza ostateczną efektywność kompresji.

Huffman vs. LZW



Huffman

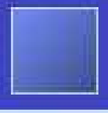
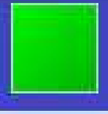
LZW

Sposób budowy słownika

- Pierwszą czynnością przy kompresji wiadomości jest wyznaczenie częstości występowania znaków alfabetu w wiadomości, metodą dokładną, lub za pomocą estymacji.
- Wróćmy do naszej przykładowej linii obrazu:

i wyznaczmy częstość występowania czterech znaków alfabetu.

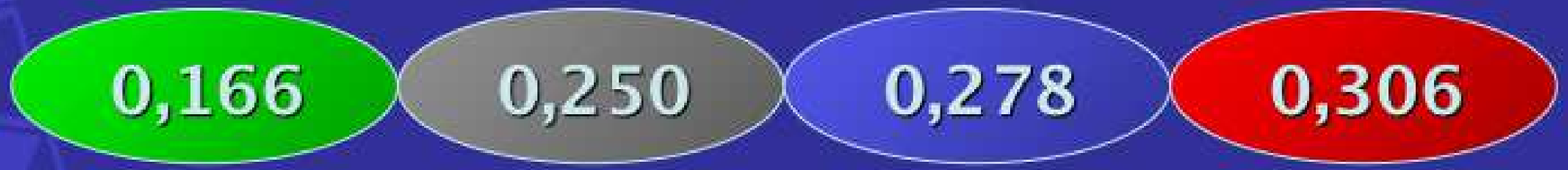
Sposób budowy słownika

	10/36 = 0,278
	11/36 = 0,306
	6/36 = 0,166
	9/36 = 0,250

- Można przewidywać, że najdłuższy kod przypadnie w udziale kolorowi zielonemu (jest najmniej pikseli w tym kolorze), najkrótszy zaś czerwonemu (tych jest najwięcej). Kod drugi z kolei będzie niebieski, trzeci szary.

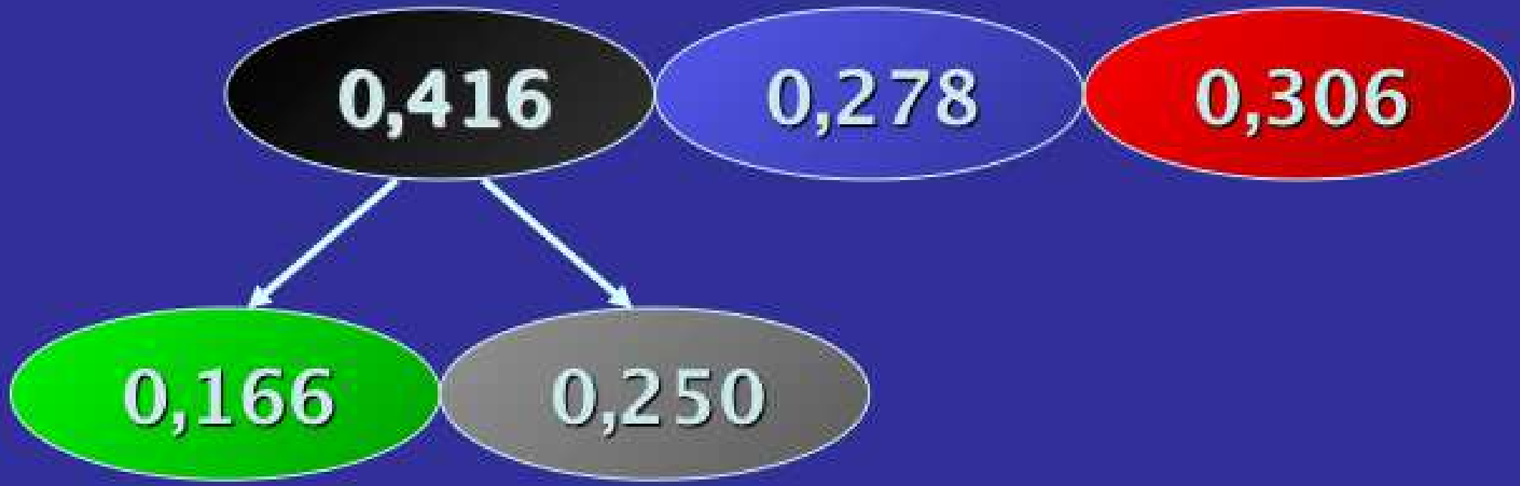
Sposób budowy słownika

- Czas zbudować słownik, robimy to za pomocą drzewa. Zaczynamy od czterech liści drzewa, są to nasze elementy alfabetu, ustawione w kolejności od najrzadszego, do najczęstszego:



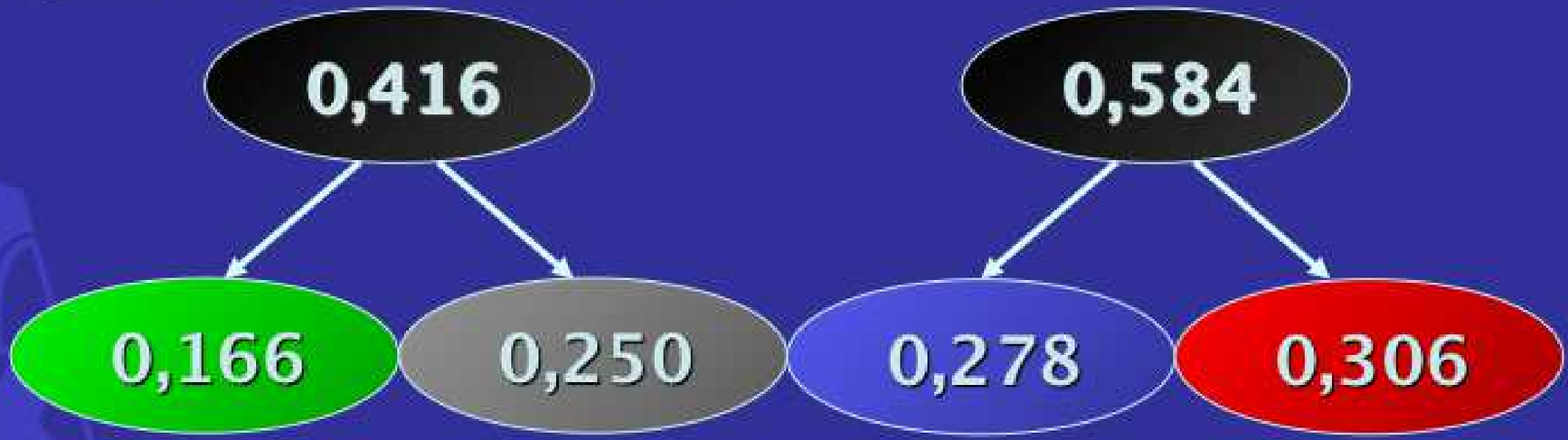
Sposób budowy słownika

- Następnie bierzemy dwa najrzadziej spotykane elementy i tworzymy dla nich wspólny korzeń drzewa, który zajmuje ich miejsce.



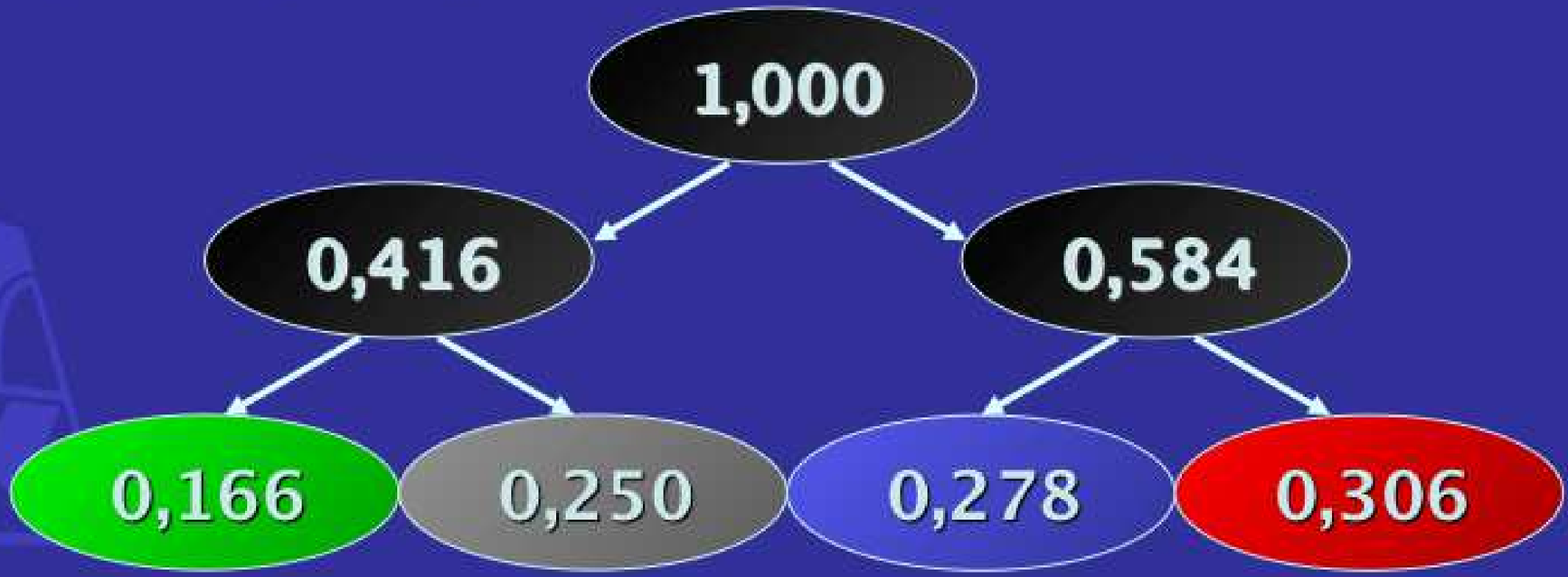
Sposób budowy słownika

- Czynność tę powtarzamy, aż do połączenia w drzewo o jednym korzeniu wszystkich początkowych „liści”.



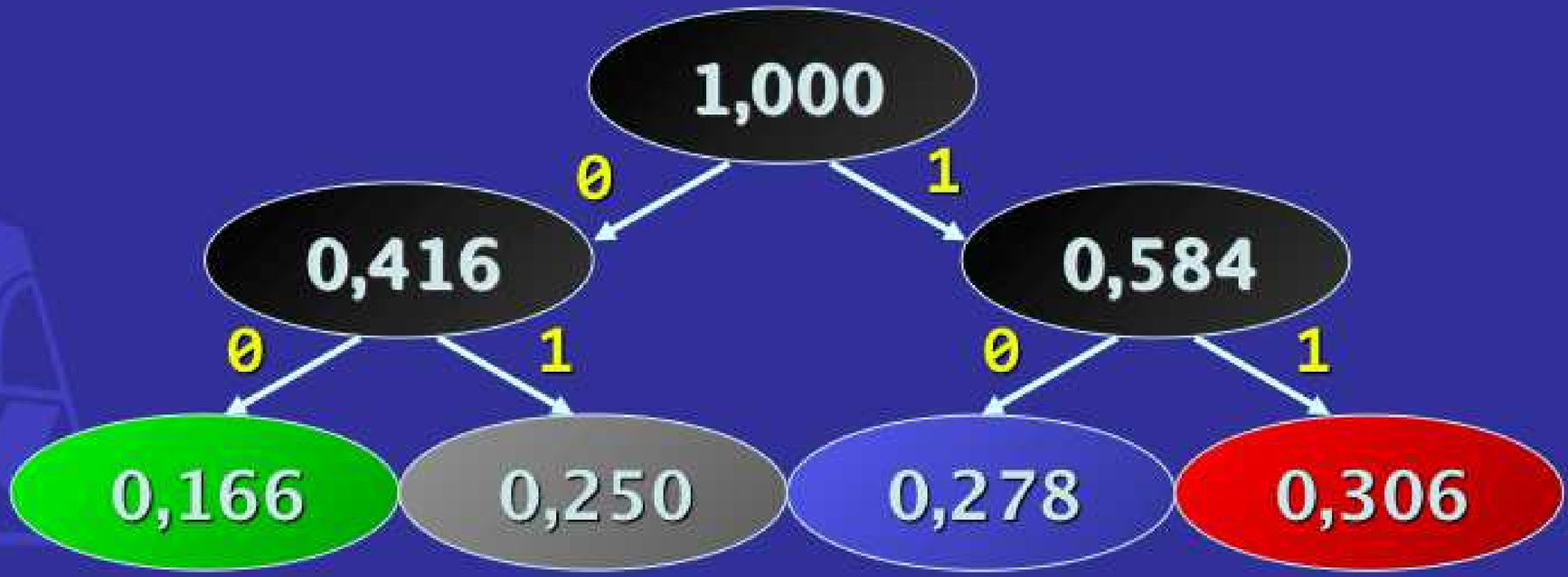
Sposób budowy słownika

- Czynność tę powtarzamy, aż do połączenia w drzewo o jednym korzeniu wszystkich początkowych „liści”.



Sposób budowy słownika

- Następnie wszystkie gałęzie odchodzące od korzenia w lewo oznaczamy jako „0”, a odchodzące w prawo jako „1”.



Sposób budowy słownika

- Te cyfry binarne tworzą nam kod danego elementu alfabetu, który czytamy idąc od korzenia do liścia.

	10
	11
	00
	01

- Sprawdźmy teraz jak bliski idealnemu jest ten kod.
- Górną granicę możliwej kompresji wiadomości wyznacza Twierdzenie Shannona o kodowaniu źródła, mówi ono, że minimalna ilość bitów jaka jest konieczna do przesłania symbolu wiadomości jest równa entropii informacyjnej tej wiadomości.

Trochę teorii informacji

- Entropia wyraża się wzorem:

$$E(X) = -\sum_{i=1}^{n} p(x_i) \log_2(p(x_i))$$

gdzie X to nasza informacja, zaś $p(x_i)$, to prawdopodobieństwo wystąpienia symbolu x_i w wiadomości.

- Biorąc pod uwagę naszą linię pikseli, możemy wyliczyć, że entropia wynosi dla niej 1,96625537.
- Średnia zaś długość kodu Huffmana, jaki uzyskaliśmy, wynosi 2 (bo wszystkie kody mają długość 2).

- Widać więc, że nie jest źle, można udowodnić, że żadne kombinowanie nie da lepszych rezultatów.
- Przykładowo, gdybyśmy kolorowi czerwonemu (jako najczęstszemu) dali kod 1-bitowy, to zielony i szary musiałyby dostać 3 bity, wtedy średnia długość kodu wyniosłaby $1 \cdot 0{,}306 + 2 \cdot 0{,}278 + 3 \cdot 0{,}250 + 3 \cdot 0{,}166 = 2{,}11$, a więc popsuliśmy.

- Skoro entropia wiadomości wynosi 1,96625537, to minimalna ilość bitów potrzebna do przesłania całości wynosi 36 razy tyle, a więc (zaokrąglając w górę do całkowitego bitu) 71 bitów.

- Huffman daje nam 72 bity, kodowanie LZW (5 bitów na symbol, 22 symbole) jest gorsze, bo 110 bitów, ale nie zapominajmy, że przy Huffmanie trzeba jeszcze przesłać słownik, albo tablicę prawdopodobieństw (wtedy słownik można zbudować w dekoderze).
- Tablica prawdopodobieństw to mogą być po prostu ilości poszczególnych pikseli, co wymaga minimum 24 bitów (po 6 na kolor) i tu zbliżamy się powoli do LZW.

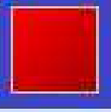
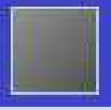
- Tutaj warto zauważyć ogólną małą celowość stosowania kompresji do naszej nieszczęsnej linii pikseli.
- Dlaczego? Otóż zauważmy, że algorytm Huffmana przydzielił po prostu dwubitowy kod każdemu z 4 kolorów, równie dobrze sami moglibyśmy wpaść na to, że skoro kolory są 4, to trzeba 2 bitów do ich zapisania... Bez żadnej zabawy w kompresje...

Trochę teorii informacji

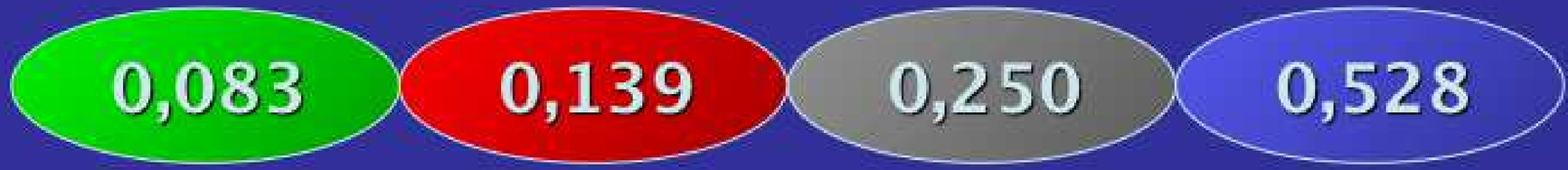
- Żadna kompresja bezstratna nie ma szans.
- W naszej informacji prawdopodobieństwa wystąpienia kolorów są dość wyrównane, stąd entropia jest nieco mniejsza od 2 bitów na symbol.

Nowy przykład

- Dla porównania wyobraźmy sobie linię złożoną z 3 pikseli zielonych, 5 czerwonych, 9 szarych i 19 niebieskich (warto zauważyć, że kolejność, wbrew pozorom jest nieistotna).
- Tym razem entropia wiadomości jest znacznie mniejsza i wynosi 1,68. Daje to teoretyczną nadzieję na spakowanie wiadomości do 61 bitów.

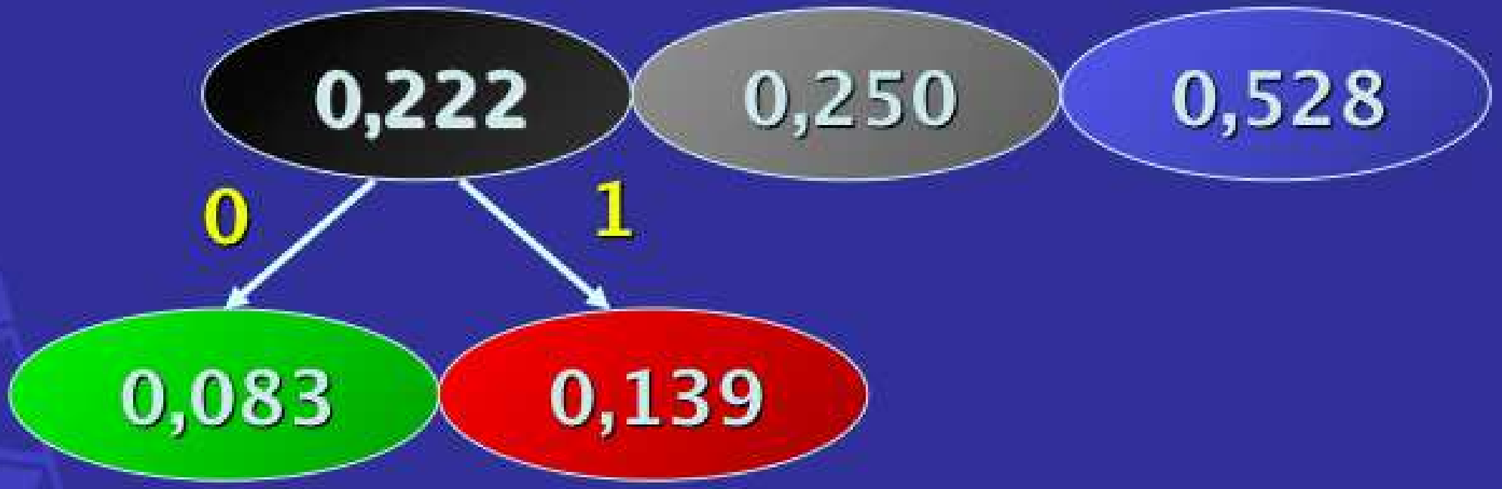
	19/36 = 0,528
	5/36 = 0,139
	3/36 = 0,083
	9/36 = 0,250

- Proces tworzenia drzewa Huffmana dla nowego przypadku:

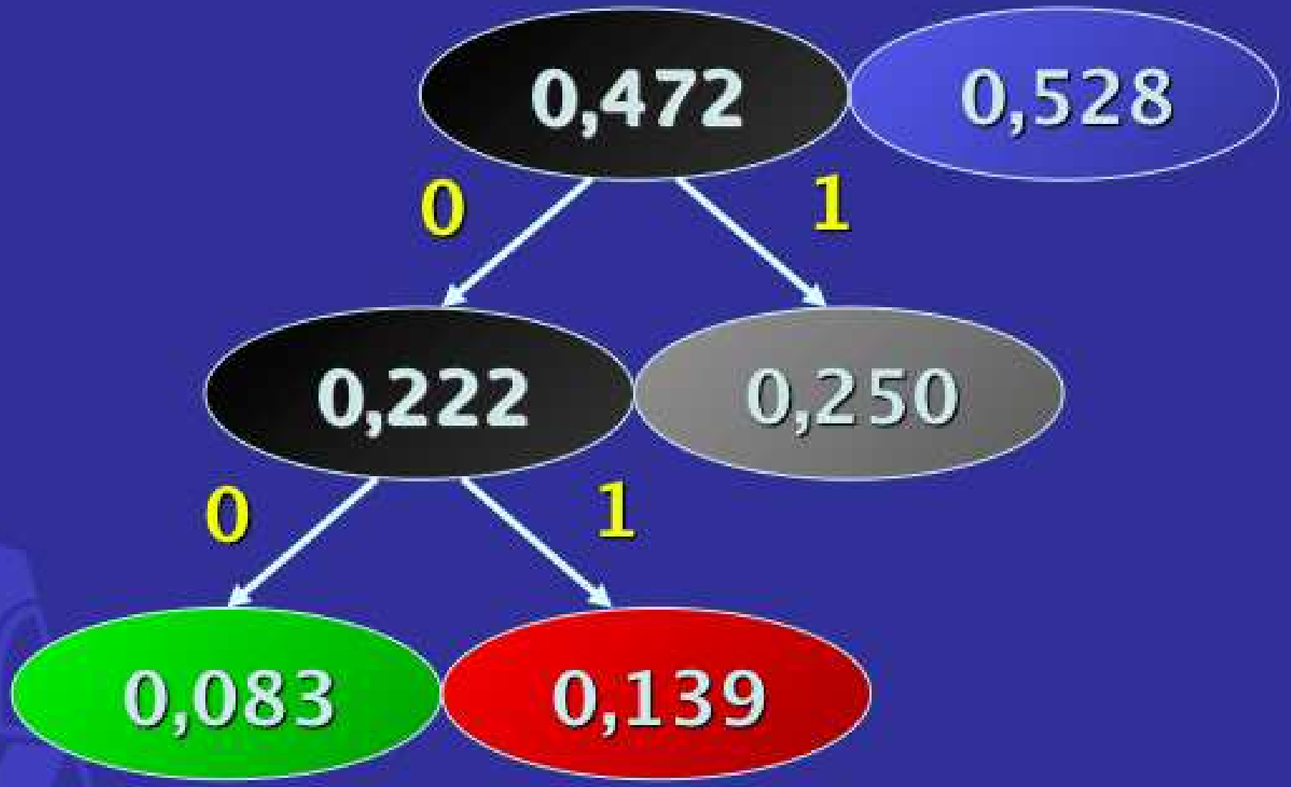


Nowy przykład

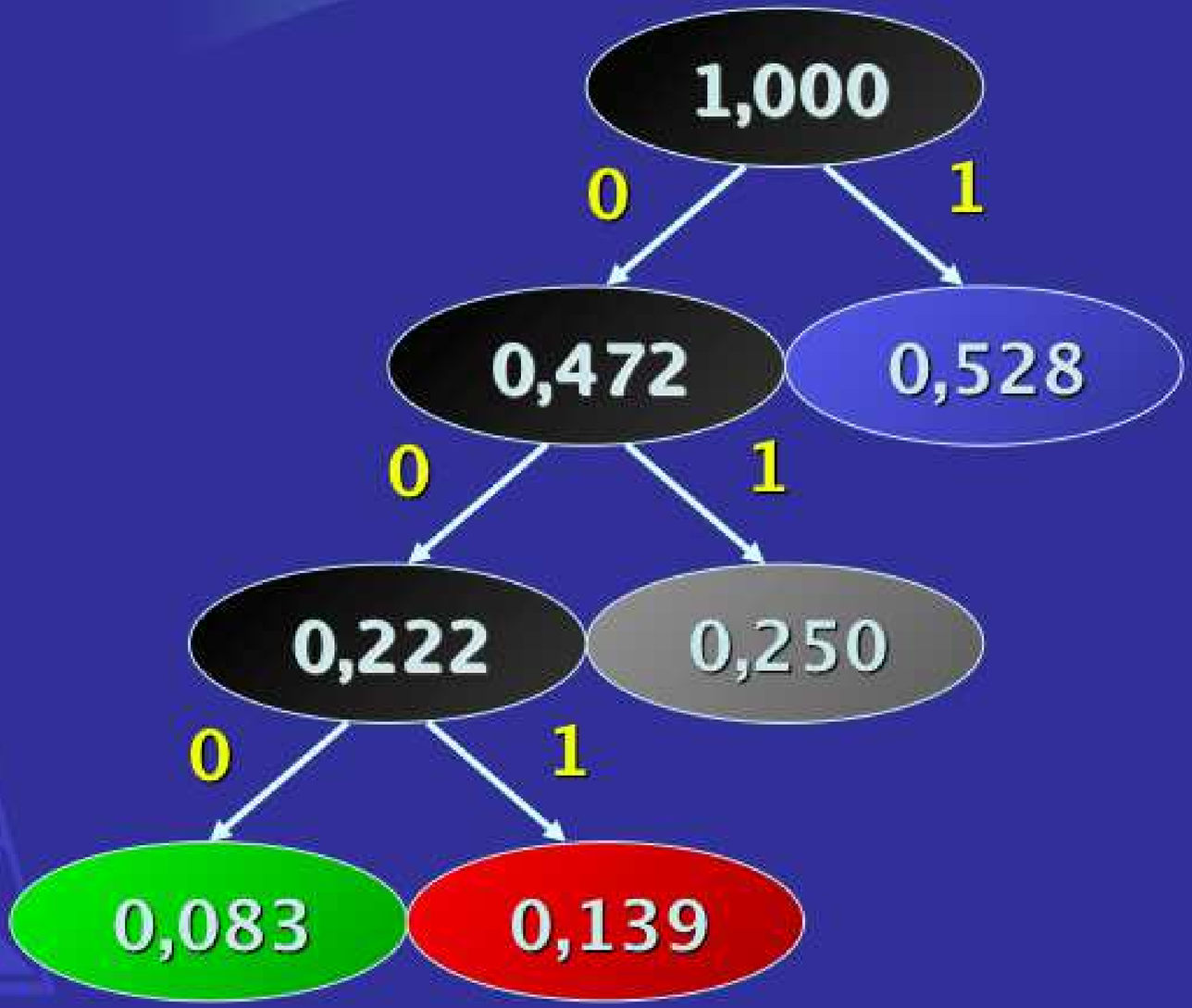
- Proces tworzenia drzewa Huffmana dla nowego przypadku:



Nowy przykład



Nowy przykład



Nowy przykład

- Gotowy słownik:

	1
	001
	000
	01

Nowy przykład

- Obliczmy najpierw średnią oczekiwaną ilość bitów na piksel:
  $0{,}528 \cdot 1 + 0{,}250 \cdot 2 + 0{,}139 \cdot 3 + 0{,}083 \cdot 3 = 1{,}694$
  a więc niewiele więcej niż entropia.

- Długość skompresowanej wiadomości wyniesie natomiast $19 + 9 \cdot 2 + 5 \cdot 3 + 3 \cdot 3 = 61$ bitów! Wycisnęliśmy więc „maksa”, ale nie zapominajmy, że jakoś trzeba przesłać słownik…

Kompresja LZ77

- Algorytm LZ77, opracowany przez Ziva i Lempela jest zbliżony w swojej zasadzie działania do LZW, jednak jako słownika używa określonej ilości ostatnio skompresowanych danych źródłowych.
- Kompresor odnajduje ciągi danych, które już wcześniej wystąpiły w danych, następnie zamiast tychże danych zapisuje długość powtórzonego ciągu, oraz jego przesunięcie do tyłu.

Kompresja LZ77

- Tak więc słownikiem kompresji jest po prostu N bajtów ostatnio przetwarzanych danych źródłowych.
- W praktycznych implementacjach N jest dość duże i wynosi od 2 kB do 64 kB.
- Dekoder oczywiście używa jako słownika zdekompresowanych danych, zatem podobnie jak LZW, LZ77 nie wymaga przesyłania słownika wraz z zakodowanym strumieniem danych.

Kompresja LZ77

- Ostatnie N bajtów danych można traktować jako okno przesuwające się nad danymi, stąd LZ77 jest nazywany algorytmem z przesuwającym się oknem.
- Poniższy przykład pokazuje ideę kompresji LZ77, w roli głównej nasza nieśmiertelna linia pikseli ☺

Kompresja LZ77

- Zakładamy, że rozmiar okna obejmie całą linię (jest przecież bardzo krótka).
- Na początku okno jest puste, zatem jedyne co możemy zrobić, to wyemitować symbol 0.

Kompresja LZ77

- Kolejny niebieski symbol można już zapisać jako parę „długość-przesunięcie” [1, -1], ale LZ77 pozwala na więcej, mianowicie na długość większą niż przesunięcie, co teoretycznie oznacza wyjście poza bufor, a w praktyce oznacza powtarzanie zawartości bufora (jest to więc zaawansowana forma kompresji ByteRun), a więc można zapisać dwa kolejne niebieskie piksele jako [2, -1].

Kompresja LZ77

- Piksel czerwony to 1, kolejne trzy zapiszemy w stylu ByteRun jako [3, -1].
- Zauważmy, że przesunięcie liczę tu od końca okna, można też liczyć je od początku, to kwestia umowna.
- Zielony piksel da nam 2, dwa niebieskie zaś mamy już w naszym oknie i to na dwa sposoby: [2, -7] albo [2, -8], w praktyce używa się zawsze mniejszego przesunięcia.

Kompresja LZ77

- Szary piksel jest pierwszy raz, więc 3, czerwony można albo zapisać jawnie jako 1, albo jako przesunięcie [1, -5], zależnie od tego, co zajmuje mniej miejsca.
- Dla przykładowej linii pikseli kompresja LZ77 nie będzie tak efektywna jak Huffman czy LZW, dlatego, że działa ona efektywnie gdy większość okna jest wypełniona danymi, a rozmiar okna jest spory.

Kompresja LZ77

- Przykład kodowania ciągu znaków:

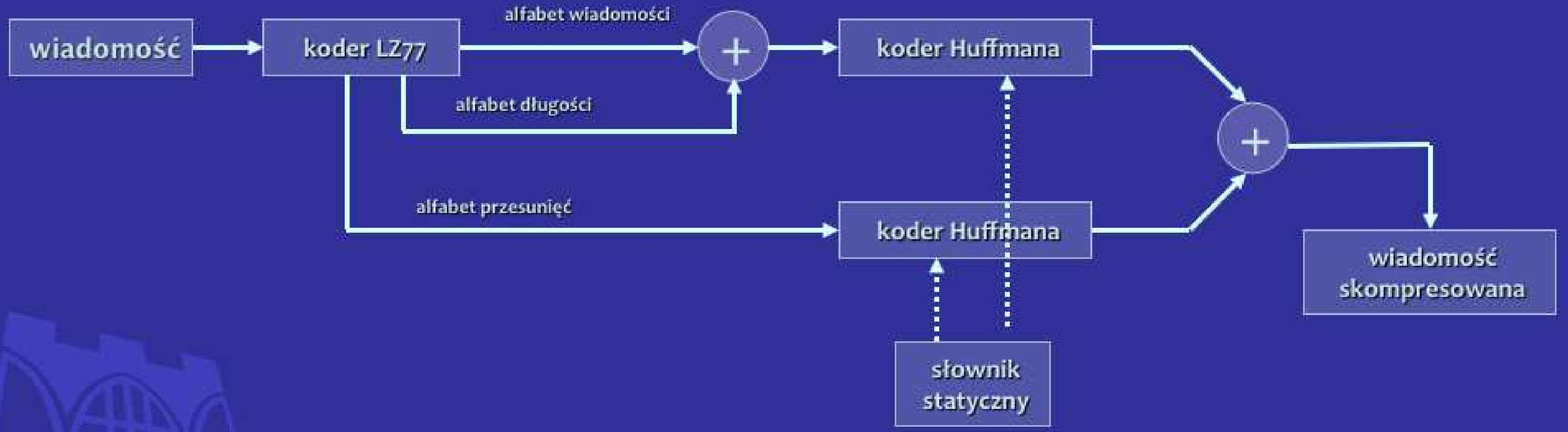
Pozycja	1	2	3	4	5	6	7	8	9
Znak	A	A	B	C	B	B	A	B	C

Krok	Pozycja	Dopasowanie	Znak	Wyjście
1.	1	--	A	[0,0] A
2.	2	A	B	[1,-1] B
3.	4	--	C	[0,0] C
4.	5	B	B	[1,-2] B
5.	7	A B	C	[2,-5] C

Standard zlib

- **Zlib** to biblioteka do bezstratnej kompresji danych, oraz nazwa formatu danych produkowanego między innymi przez tę bibliotekę. Format danych jest zdefiniowany w dokumencie **RFC 1950**, natomiast algorytmów kompresji może być użytych kilka, jednak najczęściej używany jest algorytm **deflate**, który jest kombinacją kodowania LZ77 i Huffmana. Algorytm deflate jest zdefiniowany w dokumencie **RFC 1951**. Bloki skompresowanych danych zabezpieczone są sumą kontrolną liczoną algorytmem ADLER-32.

Standard zlib



- W praktyce blok danych zlib spakowany algorytmem deflate może używać jednej z trzech metod kompresji:

1. Bez kompresji.

2. Kompresja ze statycznym słownikiem kodów Huffmana. W praktyce stosowana bardzo rzadko. Słownik statyczny jest oparty na założeniu, że najczęściej będą się trafiały znaki z alfabetu długości LZ77 od 3 do 114 bajtów (7-bitowe kody Huffmana), później kody alfabetu wiadomości od 0 do 143, kody długości LZ77 od 115 do 258 (8-bitowe kody Huffmana), najrzadziej kody z alfabetu wiadomości od 144 do 255 (9-bitowe kody Huffmana). Ta statystyka nie zawsze odpowiada rzeczywistej.

3. Kompresja z dynamicznym słownikiem kodów Huffmana (drzewo jest budowane dla każdego bloku), przed danymi przesyłane są długości kodów dla wszystkich znaków obu alfabetów (wiadomość + długości, oraz przesunięcia). Przy założeniu alfabetyczności kodu Huffmana (dla dowolnych dwóch kodów Huffmana o tej samej długości, kolejność kodów jest taka sama jak kodowanych symboli) długości kodów wystarczają do zbudowania kompletnego drzewa.

Kod alfabetyczny

A – 011

B – 101

C - 111

Kod niealfabetyczny

A – 100

B – 001

C - 110

Techniki zmniejszania drzewa Huffmana

- W standardzie zlib tylko drzewo Huffmana dla alfabetu wiadomości (obejmujące bajty od 0 do 255) jest pełne (w sumie 511 węzłów).
- W celu zmniejszenia drzewa długości LZ77 i przesunięć LZ77 (zwłaszcza to ostatnie byłoby bardzo duże, 65535 węzłów!) stosuje się technikę dzielenia elementu alfabetu na kod i przyrostek.

Techniki zmniejszania drzewa Huffmana

- Kod określa pewien zakres znaków alfabetu, natomiast przyrostek konkretny element z tego zakresu.
- Dzięki temu drzewo wspólnego alfabetu wiadomości i długości LZ77 ma 575 węzłów zamiast 1023, drzewo alfabetu przesunięć LZ77 zredukowano do 63 węzłów.
- Zakresy znaków w obu przypadkach zwiększają się w przybliżeniu wykładniczo.

- Oto przykładowy sposób zakodowania przesunięcia 5500 bajtów:

Kod Huffmana dla zakresu 24	10101111100

zakres 24: 4097 – 6144 11 bitów (1404) 4096 + 1404 = 5500

Zastosowania zlib

- Zlib jest wolny od patentów i ma licencję umożliwiającą stosowanie go w dowolnych projektach, również komercyjnych. Dzięki temu jest szeroko stosowanym standardem.
- Oto niektóre zastosowania:
  - format zapisu obrazów PNG,
  - protokół internetowy HTTP/1.1,
  - pakiet OpenSSH,
  - kompresory ZIP i GZIP.

Dziękuję za uwagę...

Damian Grela
e-mail: **dgrela@pk.edu.pl**
http://www.dgrela.pl