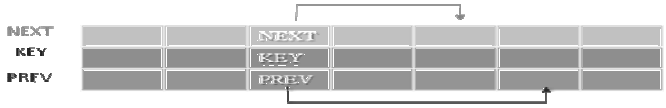
ALGORYTMY I STRUKTURY DANYCH – LISTA

- **Lista** – kolekcja elementów zapewniająca dostęp do dowolnego elementu

- **Reprezentacja list**
  - dowiązaniowa – typy list
    - lista pojedynczo wiązana (jednokierunkowa)

    - lista podwójnie wiązana (dwukierunkowa)

    - jednokierunkowa lub dwukierunkowa z wartownikiem
  - tablicowa
    - wielotablicowa (tablice równoległe)



    - jednotablicowa

- **Operacje na listach**

  ★ dla listy jednokierunkowej bez wartownika

  - wyszukanie elementu w liście (o zadanej wartości klucza)
    - **dane**: lista i key[x] (x – szukany element)
    - **wynik**: wskaźnik do pierwszego wystąpienia x lub NIL

```
p ← head
while(p ≠ NILL and key[p] ≠ k)
      p ← next[p]
ret (p)
```

  - wstawienie elementu do listy (z zachowaniem porządku)
    - **dane**: lista uporządkowana niemalejąco i element do dodania (x)
    - **wynik**: lista z elementem x

```
if(head = NILL)
   head ← x                          } lista pusta
else
   if(key[head] > key[x])
      next[x] ← head                 } wstawienie na początek
      head ← x
   else
      p← head, n ← next[head]
      while(n ≠ NILL and key[n] ≤ key[x])
         p←n, n ← next[n]            } wstawienie w środku
      nex[x] ← n                     lub na końcu listy
      next[p] ← x
```

- inne operacje: usunięcie elementu z listy, wyznaczenie pierwszego, następnego, poprzedniego elementu, usunięcie listy, łączenie list, kopiowanie listy (fragmentu listy), …

ALGORYTMY I STRUKTURY DANYCH – STOS

- **Stos –** kolekcja elementów zapewniająca dostęp do jednego końca

- **Reprezentacja stosu**
  - listowa
  - tablicowa

- **Operacje na stosie**
    - dla reprezentacji tablicowej

  - odłożenie elementu na stos
    - **dane**: stos S o rozmiarze N i element x
    - **wynik**: element x w S lub błąd – „stos przepełniony”
    - **założenia**: wskaźnik stosu (sp) wskazuje pierwszą wolną pozycję
  -

    push(x)
    if(sp ≤ N)
    S[sp] ← x
    sp ← sp +1
    else
    „stos przepełniony”

  - zdjęcie elementu ze stosu
    - zdjąć można ***tylko*** element ostatnio odłożony
    - **dane**: stos S o rozmiarze N i element x
    - **wynik**: element x w S lub błąd – „stos pusty”
    - **założenia**: wskaźnik stosu (sp) wskazuje pierwszą wolną pozycję

    pop()
    if(sp > 1)
    sp ← sp -1
    ret (x ← S[sp])
    else
    „stos pusty”

- inne operacje:
  - sprawdzenie czy stos jest pusty
  - usunięcie wszystkich elementów ze stosu

ALGORYTMY I STRUKTURY DANYCH – KOLEJKA

- **Kolejka** – kolekcja elementów zapewniająca dostęp do elementów na początku i na końcu – FIFO

- **Reprezentacja stosu**
  - listowa
  - tablicowa

- **Operacje na kolejce FIFO**
    - dla reprezentacji listowej – lista dwukierunkowa z wartownikiem

  - umieszczenie elementu w kolejce
    - **dane**: kolejka Q (head wskazuje na wartownika Q) i element x
    - **wynik**: element x w Q

      enqueue(x)
      next[x]← head
      prev[x]←prev[head]
      next[prev[head]]← x
      prev[head]← x

  - pobranie elementu z kolejki
    - pobrać można ***tylko*** element najdawniej umieszczony
    - **dane**: kolejka Q (head wskazuje na wartownika Q)
    - **wynik**: element x

      dequeue()
      x← next[head]
      next[head]←next[x]
      prev[next[x]]← prev[x]
      ret(x)

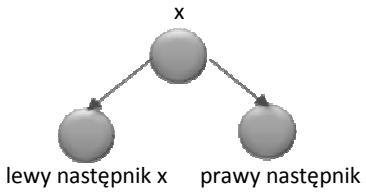
- inne operacje:
  - sprawdzenie czy kolejka jest pusta
  - usunięcie wszystkich elementów z kolejki

- **Typy kolejek**
  - FIFO – z kolejki pobierany element najdawniej w niej umieszczony
  - LIFO – z kolejki pobierany element ostatnio w niej umieszczony (stos)
  - blokująca – ograniczony rozmiar
  - priorytetowa – z kolejki pobierany element o największym priorytecie (wartości pewnego klucza)
    - najefektywniejsza realizacja za pomocą kopca

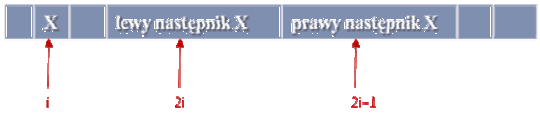
ALGORYTMY I STRUKTURY DANYCH – DRZEWO BINARNE

- **Drzewo binarne**
  - ukorzenione i uporządkowane
  - nie zawiera żadnych węzłów lub
  - składa się z trzech rozłącznych zbiorów węzłów: korzenia, lewego poddrzewa i prawego poddrzewa

- **Reprezentacja stosu**
  - dowiązaniowa



  - tablicowa

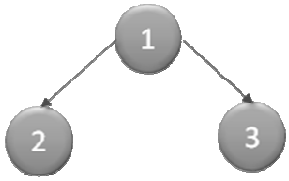


- **Podstawowa operacje na drzewie binarnym**
  - dla reprezentacji dowiązaniowej

  - przechodzenie drzewa
    - **dane**: drzewo wskazywane przez root
    - **wynik**: wypisane klucze węzłów

  - w porządku ***preorder***

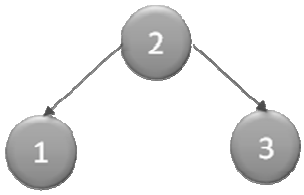
t←root
preorder(t)
if(t ≠ NILL)
visit(t)
preorder(left[t])
preorder(right[t])



- kolejność wypisywania: 1,2,3

  - w porządku ***inorder***

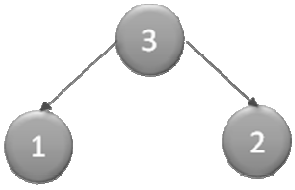
t←root
inorder(t)
if(t ≠ NILL)
inorder(left[t])
visit(t)
inorder(right[t])



- kolejność wypisywania: 1,2,3

  - w porządku ***postorder***

t←root
postorder(t)
if(t ≠ NILL)
postorder(left[t])
postorder(right[t])
visit(t)



- kolejność wypisywania: 1,2,3

  - **złożoność**: O(n) (n – liczba węzłów)