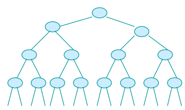
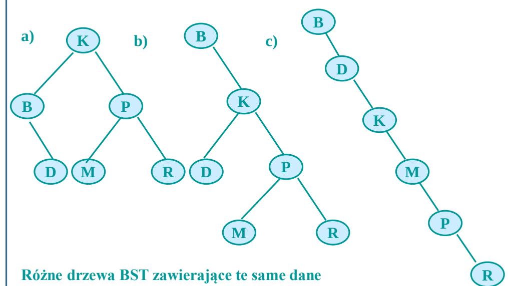
Algorytmy i struktury danych

Wykład 11:

- Metody równoważenia drzew BST
- Drzewa samoorganizujące się (drzewa *splay)*
- Binarne drzewa wyrażeń
- Tablice rzadkie
- Tablice rozproszone



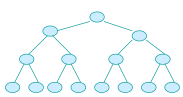
Techniki równoważenia drzew BST



Różne drzewa BST zawierające te same dane

Techniki równoważenia drzew BST

Ile maksymalnie węzłów może mieć drzewo binarne w zależności od wysokości h?



Wysokość	Węzłów na poziomie h	Węzłów w drzewie
1	$2^0=1$	$1=2^1-1$
2	$2^1=2$	$3=2^2-1$
3	$2^2=4$	$7=2^3-1$
4	$2^3=8$	$15=2^4-1$
...	...	...
11	$2^{10}=1024$	$2047=2^{11}-1$
...		
14	$2^{13}=8192$	$16383=2^{14}-1$
...	...	...
h	2^{h-1}	$n=2^h-1$

dla np. n=10 000

$$h=\lceil log_2(n+1)\rceil=$$

$$=\lceil log_2 10001\rceil=\lceil 13{,}3\rceil=14$$

Techniki równoważenia drzew BST

Sposoby równoważenia drzew BST:

1. **Okresowe równoważenie drzewa BST**
2. **Stałe poprawianie drzewa w miarę wstawiania węzłów (np. z wykorzystaniem rotacji; przykład: drzewa AVL, drzewa czerwono-czarne)**

Okresowe równoważenie drzewa BST

1. Linearyzacja (przejście) drzewa i uporządkowanie danych przed ponownym utworzeniem drzewa:

- zdegenerowane drzewo jest poddawane linearyzacji
- przed ponownym utworzeniem drzewa dane porządkowane są w tablicy;
- na korzeń drzewa wybierany jest element bliski wartości środkowej; element ten wyznacza podział tablicy na lewą i prawą podtablicę;
- lewy następnik korzenia: środek lewej podtablicy; prawy następnik korzenia: środek prawej podtablicy;
- otrzymane drzewo jest dobrze zrównoważone (liczba elementów na drodze od korzenia do dowolnego liścia jest rzędu *lg n*);

Techniki równoważenia drzew BST

Tworzenie drzewa po uprzednim posortowaniu jego elementów (idea algorytmu):

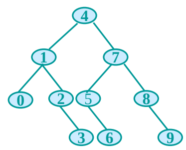
```
void balance(T data[ ], int first, int last) {
        if (first<=last) {
                int mid=(first+last)/2;
                insert(data[mid]);
                balance(data, first, mid-1);
                balance(data, mid+1, last);
        }
}
```

Techniki równoważenia drzew BST

Przykład użycia funkcji *balance*

Strumień danych	5	1	9	8	7	0	2	3	4	6
Strumień uporządkowany	0	1	2	3	4	5	6	7	8	9

0 1 2 3 4 5 6 7 8 9



Techniki równoważenia drzew BST

2. Algorytm DSW

- Idea: rozłożyć niezrównoważone drzewo (sprowadzić do listy) i złożyć je ponownie;
- Kroki algorytmu DSW (C.**D**ay, Q.**S**tout, B.**W**arren):
  1) przekształcenie drzewa w winorośl;
  2) przekształcenie winorośli w drzewo zrównoważone (z wykorzystaniem rotacji).

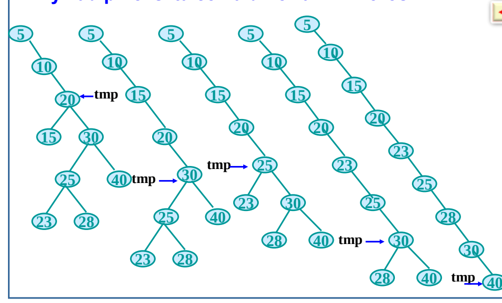
Techniki równoważenia drzew BST

Przekształcenie drzewa w winorośl:

```
UtwórzWinorośl (root, n ) {
    tmp=root;
    while ( tmp != 0 )
        if ( tmp ma lewy następnik) {
            obróć lewy następnik w prawo wokół tmp;
            ustaw tmp na następnik, który stał się poprzednikiem;
        }
        else
            ustaw tmp na jego prawy następnik;
}
```


Techniki równoważenia drzew BST

Przykład przekształcenia drzewa w winorośl



Techniki równoważenia drzew BST

Złożoność tworzenia winorośli

Pokaż algorytm

- **Przypadek optymistyczny (*drzewo jest już winoroślą*): pętla *while* wykonuje się *n* razy, nie są wykonywane żadne rotacje (tylko *n* instrukcji podstawienia); zatem *T(n)=O(n)***
- **Przypadek pesymistyczny (*korzeń nie ma prawego następnika*): pętla *while* wykonuje się *2n – 1* razy, w tym wykonywanych jest *n – 1* rotacji; zatem: *T(n)=O(n)***

Techniki równoważenia drzew BST

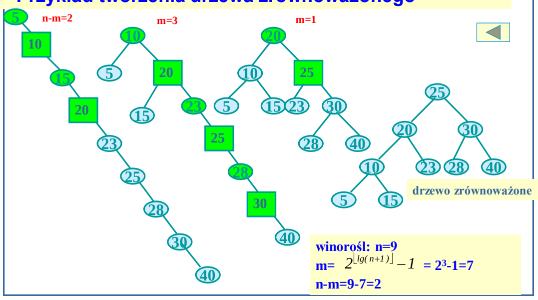
Tworzenie drzewa zrównoważonego

$m = 2^{\lfloor lg(n+1) \rfloor} - 1;$

UtwórzDrzewoZrównoważone(n) {
    wykonaj *n-m* rotacji w lewo, zaczynając od prawego następnika korzenia winorośli;
    while (*m>1*) {
        m = $\lfloor m/2 \rfloor$;
        wykonaj m rotacji w lewo, zaczynając od prawego następnika korzenia winorośli;
    }
}

Techniki równoważenia drzew BST

Przykład tworzenia drzewa zrównoważonego



winorośl: n=9
m= $2^{\lfloor lg(n+1) \rfloor} - 1$ **= 2^3-1=7**
n-m=9-7=2

Złożoność przekształcania winorośli w drzewo zrównoważone

- **Liczba rotacji wykonywanych w ramach pętli *while*:**

Dlaczego?

$$(2^{\lfloor lg(n+1) \rfloor - 1} - 1) + \ldots + 15 + 7 + 3 + 1 = \sum_{i=1}^{\lfloor lg(n+1) \rfloor - 1} (2^i - 1) \le n - \lfloor lg(n+1) \rfloor$$

Pokaż algorytm

$$S_n = a_1 \frac{q^n - 1}{q - 1}$$

- **Łączna liczba rotacji jest nie większa niż:**

$$\begin{aligned} n - m + (n - \lfloor lg(n+1) \rfloor) &= \\ &= 2n - m - \lfloor lg(n+1) \rfloor = \\ &= 2n - (2^{\lfloor lg(n+1) \rfloor} - 1) - \lfloor lg(n+1) \rfloor \le \\ &\le 2n - (n-1) - \lfloor lg(n+1) \rfloor \end{aligned}$$

$\Rightarrow$ *T(n)=O(n)*

Techniki równoważenia drzew BST

Złożoność równoważenia drzewa

1. **Przekształcenie drzewa niezrównoważonego w winorośl: *O(n)***
2. **Przekształcenie winorośli w drzewo zrównoważone: *O(n)***

T(n) = max { O(n), O(n) } = O(n)

Samoorganizujące się drzewa BST

Zmiana organizacji drzewa po dostępie do węzła

Cel: skrócenie czasu realizacji podstawowych operacji na drzewie

Idea: elementy (dane), które wykorzystywane są najczęściej przesuwane są w górę drzewa (bliżej korzenia)

Samoorganizujące się drzewa BST

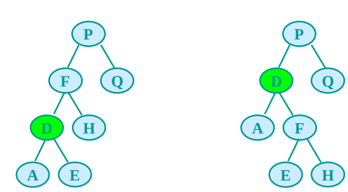
Idea:

Przy sięganiu do elementu (węzła) następuje korekta struktury drzewa poprzez:

a) pojedynczą rotację (wokół poprzednika)
b) przesunięcie elementu do korzenia (seria rotacji), tzw. drzewa *splay*

Samoorganizujące się drzewa BST

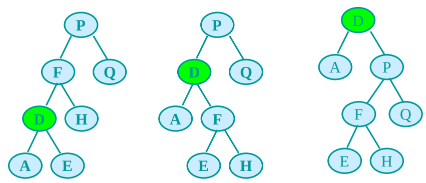
Ilustracja idei: sięgnięcie do węzła D



a) rotacja w prawo węzła D

Samoorganizujące się drzewa BST

Ilustracja idei: sięgnięcie do węzła D



b) **przesunięcie węzła D do korzenia (metodą podwójnej rotacji w prawo)**

Samoorganizujące się drzewa BST

Drzewa *splay* (1985)

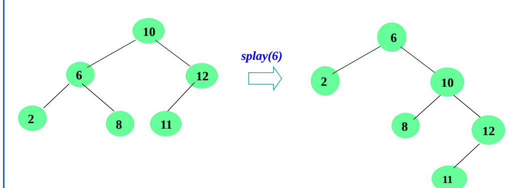
- Mechanizm równoważenia drzew AVL jest dość skomplikowany w implementacji i wymaga przechowywania w węzłach dodatkowych informacji.
- Drzewa *splay* (Sleator i Tarjan – 1985) to drzewa BST, w których wykorzystuje się rotacje do zmiany ich struktury, jednak nie trzeba przechowywać żadnych dodatkowych atrybutów w węzłach.
- Chociaż możliwe jest utworzenie niezrównoważonego drzewa *splay* i pojedyncza operacja może mieć nawet koszt liniowy względem aktualnego rozmiaru drzewa, to koszt zamortyzowany operacji w tej strukturze danych jest logarytmiczny.
- Jest to modyfikacja strategii zmiany struktury drzewa poprzez przenoszenie do korzenia elementu, do którego nastąpił dostęp.

Drzewa *splay*

- Korekta struktury drzewa następuje poprzez stosowanie pojedynczych rotacji parami, w kolejności zależnej od powiązania elementu, jego poprzednika (ojca) i poprzednika poprzednika (dziadka).
- Wszystkie operacje w drzewie *splay* są wykonywane z wykorzystaniem pomocniczej procedury *splay(T, x)* , która przekształca drzewo *T* w taki sposób, że jego korzeniem staje się węzeł z kluczem *x* albo - jeśli klucza *x* nie ma w drzewie - węzeł z kluczem *y* takim, że w *T* nie ma żadnego klucza między *min{x, y}* a *max{x, y}*.

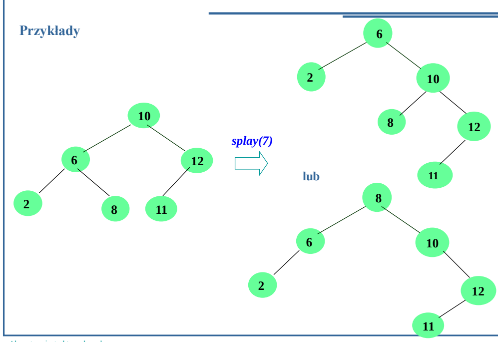
Drzewa *splay*

Przykłady



Drzewa *splay*

Przykłady



Drzewa *splay*

- Korekta struktury drzewa po dostępie do określonego węzła następuje poprzez stosowanie rotacji parami, w kolejności zależnej od powiązania elementu, jego poprzednika (ojca) i poprzednika poprzednika (dziadka).
- Procedura *splay(T, x)* jest zdefiniowana następująco:
  - najpierw szukamy węzła z kluczem ***x*** tak jak w zwykłym drzewie BST; jeśli klucza nie ma w drzewie, to bierzemy ostatni węzeł z kluczem ***x'*** na ścieżce (przed NULL);
  - następnie, dopóki ***x*** lub ***x'*** nie stanie się korzeniem, wykonujemy sekwencję rotacji, w zależności od następujących przypadków:
    1. Poprzednik węzła ***x*** jest korzeniem.
    2. Układ jednorodny (*zig-zig*): węzeł ***x*** jest lewym następnikiem swojego poprzednika, który z kolei jest lewym następnikiem swojego poprzednika (lub kiedy w obu relacjach chodzi o prawy następnik).
    3. Układ niejednorodny (*zig-zag*): węzeł ***x*** jest lewym następnikiem swojego poprzednika, który z kolei jest prawym następnikiem swojego poprzednika (lub odwrotnie).

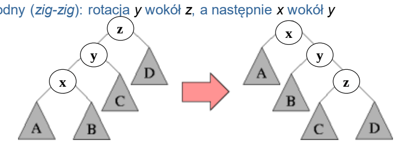
Drzewa *splay*

- Wykonanie pojedynczej pary rotacji (*zig-zig* lub *zig-zag*) nazywa się krokiem rozchylającym.
- Jeśli odległość węzła, do którego następuje dostęp od korzenia jest nieparzysta, to po serii par rotacji wykonywana jest rotacja pojedyncza.

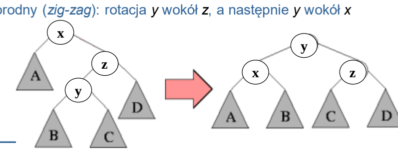
Drzewa *splay*

- **Wykonanie kroków rozchylających:**

a) jednorodny (*zig-zig*): rotacja *y* wokół *z*, a następnie *x* wokół *y*



b) niejednorodny (*zig-zag*): rotacja *y* wokół *z*, a następnie *y* wokół *x*



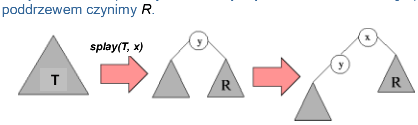
Samoorganizujące się drzewa BST

Idea algorytmu funkcji *splay(T, x)*
(przenoszenia węzła *x* do korzenia)

```
splay(T, x) {
    while (x nie jest korzeniem)
        if (poprzednik x jest korzeniem)
            wykonaj pojedynczą rotację x wokół jego poprzednika;
        else
            if (x jest ze swoimi poprzednikami w układzie jednorodnym)
                wykonaj krok rozchylający jednorodny;
            else // x jest ze swoimi poprzednikami w układzie niejednorodnym
                wykonaj krok rozchylający niejednorodny;
}
```


Drzewa *splay*

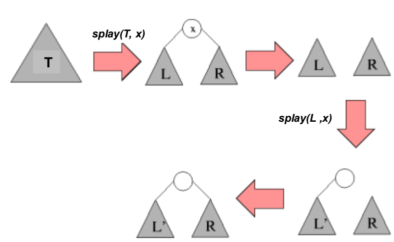
- Pozostałe operacje (*find*, *insert*, *delete*) wykonywane są z wykorzystaniem operacji *splay*
- Operacja *find(T, x)* sprowadza się do wywołania *splay(T, x)* i sprawdzenia, czy węzeł z kluczem *x* jest w korzeniu.
- W celu wykonania operacji *insert(T, x)* wywołujemy najpierw *splay(T, x)* , w wyniku czego w korzeniu znajduje się klucz *y*; bez straty ogólności możemy przyjąć, że *y*<*x*.
- Odcinamy prawe poddrzewo *R* węzła *y*, jego poprzednikiem (a zarazem nowym korzeniem) zostaje wstawiany węzeł z kluczem *x*, którego prawym poddrzewem czynimy *R*.



Drzewa *splay*

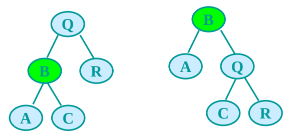
- Operację *delete(T, x)* zaczynamy od wywołania *splay(T, x)*, sprowadzając usuwany klucz *x* do korzenia.
- Niech *L* i *R* będą, odpowiednio, lewym i prawym poddrzewem uzyskanego drzewa.
- Odcinamy korzeń i - jeśli *L* jest niepuste - wywołujemy *splay(L, x)*, a następnie przyłączamy *R* jako prawe poddrzewo korzenia.

Drzewa *splay*



Samoorganizujące się drzewa BST

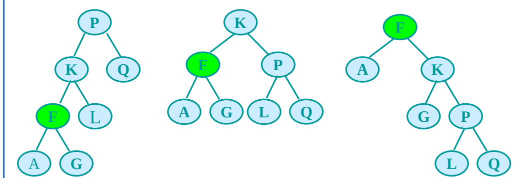
Drzewa *splay* – przykład (po sięgnięciu do węzła B)



Przypadek 1. Poprzednik węzła B jest korzeniem
(rotacja w prawo węzła B)

Samoorganizujące się drzewa BST

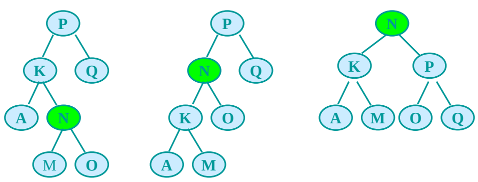
Drzewa *splay* – przykład (po sięgnięciu do węzła F)



Przypadek 2. Układ jednorodny
(rotacja w prawo węzła K oraz rotacja w prawo węzła F)

Samoorganizujące się drzewa BST

Drzewa *splay* – przykład (po sięgnięciu do węzła N)



Przypadek 2. Układ niejednorodny
(rotacja w lewo węzła N oraz rotacja w prawo węzła N)

Samoorganizujące się drzewa BST

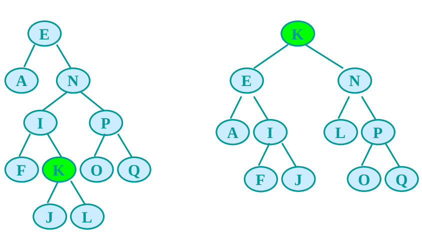
Przykład ukosowania (po dostępie do węzła E)

Stan po drugiej parze rotacji

Stan po pierwszej parze rotacji

Samoorganizujące się drzewa BST

Przykład ukosowania c.d. (dostęp do węzła *K*)



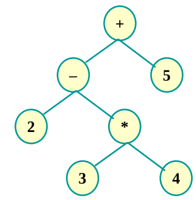
Binarne drzewa wyrażeń

Binarne drzewa wyrażeń

- Jednym z zastosowań drzew binarnych jest jednoznaczny zapis wyrażeń arytmetycznych lub logicznych (bez używania nawiasów);
- Drzewa wyrażeń konstruuje się, wykorzystując następujące założenia:
  - każdy liść zawiera operand (argument) wyrażenia;
  - każdy węzeł wewnętrzny zawiera operator np.: *, /, +, - itp.;
  - wyrażenie powstaje w wyniku realizacji procedury przechodzenia drzewa jedną z trzech metod:
    - inorder (metoda bezpośrednia),
    - preorder (metoda z wyprzedzeniem),
    - postorder (metoda z opóźnieniem).

Binarne drzewa wyrażeń

Przykład drzewa wyrażeń



Wynik przejścia drzewa metodą:
- **bezpośrednią (inorder):**
  2 – 3 * 4 + 5 = -5
- **z wyprzedzeniem (preorder):**
  + – 2 * 3 4 5 = -5
- **z opóźnieniem (postorder):**
  2 3 4 * – 5 + = -5

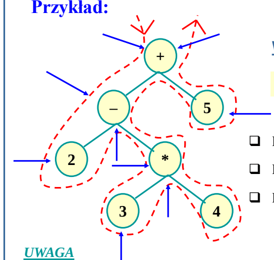
Binarne drzewa wyrażeń

Mnemotechniczna metoda przechodzenia drzewa:

- **wyruszamy z korzenia, okrążając drzewo w kierunku przeciwnym do ruchu wskazówek zegara;**
- **staramy się być jak najbliżej mijanych węzłów (niektóre węzły odwiedzamy wielokrotnie);**
- **chcą otrzymać listę węzłów odpowiadającą kolejności:**
  - ***preorder*, należy wypisywać każdy węzeł przy pierwszym jego odwiedzeniu;**
  - ***postorder*, należy wypisywać każdy węzeł przy ostatnim jego odwiedzeniu;**
  - ***inorder*, należy wypisywać każdy węzeł przy pierwszym jego odwiedzeniu jeżeli jest liściem, natomiast przy drugim odwiedzeniu, jeżeli jest węzłem wewnętrznym.**

Binarne drzewa wyrażeń

Przykład:



Wynik przejścia drzewa:

+ – 2 – * 3 * 4 * – + 5 +

- **Kolejność *preorder*:** + – 2 * 3 4 5
- **Kolejność *postorder*:** 2 3 4 * – 5 +
- **Kolejność *inorder*:** 2 – 3 * 4 + 5

UWAGA

Niezależnie od metody przechodzenia drzewa każdy liść odwiedzany jest dokładnie jeden raz (i w tej samej kolejności)

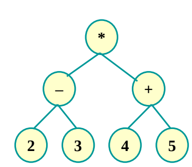
Binarne drzewa wyrażeń

Z uwagi na sposób przechodzenia drzewa wyróżniamy notacje :

- **przedrostkową (prefiksową), np. + – 2 * 3 4 5**
- **wzrostkową (infiksową), np. 2 – 3 * 4 + 5**
- **przyrostkową (postfiksową), znanej jako notacja polska odwrotna, np. 2 3 4 * – 5 +**

Binarne drzewa wyrażeń

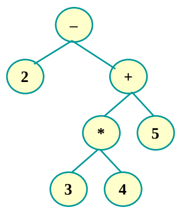
Przykłady drzew wyrażeń



Zapis wyrażenia metodą:
- **wzrostkową (inorder):**
  2 – 3 * 4 + 5 = -9
- **przedrostkową (preorder):**
  *** – 2 3 + 4 5 = -9**
- **przyrostkową (postorder):**
  2 3 – 4 5 + * = -9

Binarne drzewa wyrażeń

Przykłady drzew wyrażeń



Zapis wyrażenia metodą:

- **wzrostkową (inorder):**

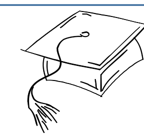
  2 – 3 * 4 + 5 = -15

- **przedrostkową (preorder):**

  – 2 + * 3 4 5 = -15

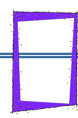
- **przyrostkową (postorder):**

  2 3 4 * 5 + – = -15



Tablice rzadkie

Tablice rzadkie



- *Tablica rzadka* – tablica, w której wykorzystany jest jedynie niewielki procent komórek („marnotrawstwo" pamięci); w takim przypadku tablicę lepiej zastąpić zestawem list
- *Tablice rzadkie* w implementacji listowej przechowują jedynie *wartości niezerowe*, ponieważ komórka tablicy z wartością zerową nie występuje (brak pozycji na liście oznacza wartość zerową)

Tablice rzadkie

Przykład – ewidencja ocen studentów w centrum kursowym

Nr studenta

Nr kursu	1	2	3	4	5	...			7999	8000
1										
2	5		4		3				3	
3										
4	3	5								
5			4							
...										
299	4								4	
300										

Tablice rzadkie

Przykład – ewidencja ocen studentów w centrum kursowym

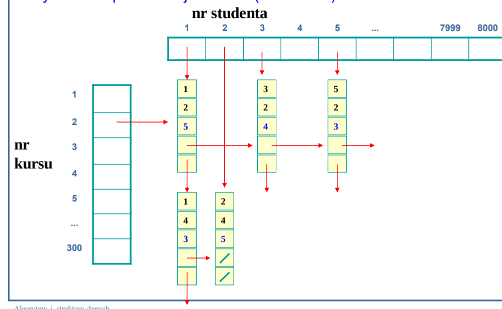
Wykorzystanie pamięci w przykładowej tablicy rzadkiej:

- Liczba studentów: 8000
- Liczba kursów: 300
- Maksymalna liczba kursów, w których może uczestniczyć student: 4

- Liczba wymaganych komórek tablicy: 8000 x 300=2 400 000 np. bajtów
- Maksymalna liczba komórek tablicy zajętych na oceny: 4 x 8000=32 000
- Wykorzystanie pamięci w tablicy: 32 000 / 2 400 000 = 0,013 =1 %

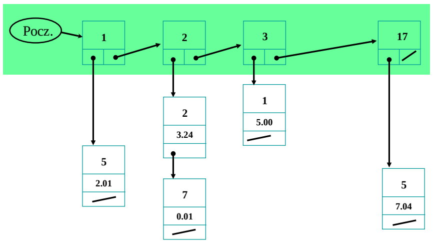
Tablice rzadkie

Przykład – implementacja listowa (zestaw list)



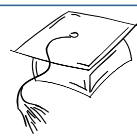
Tablice rzadkie

Przykład realizacji tablic rzadkich (lista dwupoziomowa)



Tablice rzadkie

- Tablice rzadkie mogą być wykorzystywane do implementacji macierzy koincydencji wierzchołków w grafach
- Innym zastosowaniem tablic rzadkich jest przechowywanie obrazów rastrowych (szczególnie wtedy, gdy na obrazie jest mało „zapalonych" punktów)



Temat: Tablice rozproszone

Definicja tablicy rozproszonej (z haszowaniem)

- Tablicą rozproszoną nazywamy trójkę uporządkowaną

$$T = \langle K, D, h \rangle$$

gdzie

K = {k_1, k_2, k_3,..., k_n} - zbiór kluczy,

D = {d_1, d_2, d_3,..., d_n} - zbiór adresów, $|K| > |D|$

h - tzw. funkcja mieszająca *(haszująca)*:

$$h : K \to D$$

- Tradycyjnym obszarem zastosowań tablic rozproszonych są zagadnienia związane z przetwarzaniem danych.

Tablice rozproszone, funkcja haszująca

- Zadaniem funkcji haszującej *h* jest w miarę równomierne obciążanie tablicy rozproszonej.
- Zagadnienie definiowania funkcji mieszającej jest istotne dla efektywności przetwarzania danych, realizowanego na bazie tablic rozproszonych.
- Na ogół nie można wykluczyć powstawania tzw. *konfliktów* w tablicach rozproszonych.

Konflikty w tablicach rozproszonych

- *Kolizją (konfliktem)* w tablicy rozproszonej nazywamy sytuację powstałą wtedy, gdy:

$$\exists\ k_i, k_j \in \boldsymbol{K}, k_i \neq k_j \qquad h(k_i) = h(k_j)$$

- Klucze k_i, k_j biorące udział w kolizji nazywamy *synonimami*

Metody haszowania w tablicach rozproszonych

1. Metoda dzielenia

- Adres w tablicy wyliczany jest według formuły:

  h(K) = K mod *RozmiarTablicy*

  lub

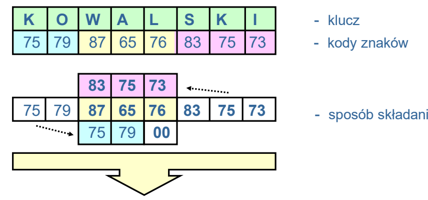
  h(K) = f(K) mod *RozmiarTablicy*

  h(K) - adres w tablicy rozproszonej
  f(K) - wartość pewnej funkcji wyliczona na podstawie klucza
  RozmiarTablicy = *sizeof(tablica)*

- W zastosowaniach praktycznych metoda ta jest bardzo efektywna

Metody haszowania w tablicach rozproszonych

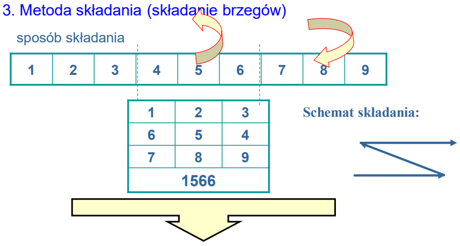
2. Metoda składania (składanie z przesuwaniem)



f(KOWALSKI) = 837573 + 3876576 + 757900 = 5472049

Metody haszowania w tablicach rozproszonych

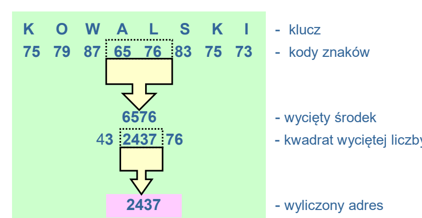
3. Metoda składania (składanie brzegów)



f(123456789) = 1566 - wyliczony adres

Metody haszowania w tablicach rozproszonych

4. Metoda kwadratu środka



f(KOWALSKI)=2437

Metody haszowania w tablicach rozproszonych, cd.

Przykłady zastosowania metody dzielenia (tablica ma 1000 pozycji):

1) **f(KOWALSKI)= 913**
   913 mod 1000 = 913

2) **f(KOWALSKI) = 834741**
   834741 mod 1000 = 741

2) **f(123456789) = 1566**
   1566 mod 1000 = 566

Organizacja tablic rozproszonych

- Problemy kolizji mogą być rozwiązywane dwiema metodami:
  - Tablice rozproszone bez obszaru nadmiarowego - dane znajdują się wyłącznie w *obszarze bazowym tablicy*
  - Tablice rozproszone z obszarami nadmiarowymi:
    - z listami synonimów
    - rozproszone tablice indeksowe

Usuwanie konfliktów w tablicach rozproszonych

Bez obszarów nadmiarowych – adresowanie otwarte

- Jeśli wyznaczony klucz koliduje z innym kluczem, znajdowana jest w tablica inna, dostępna komórka
- Stosuje się tutaj technikę próbkowania (aż do znalezienia wolnej komórki):

$$h_1(K)=[h(K)+p(1)] \bmod R$$
$$h_2(K)=[h(K)+p(2)] \bmod R$$
$$\ldots$$
$$h_i(K)=[h(K)+p(i)] \bmod R$$

gdzie p(i) jest tzw. funkcją próbkującą a R jest rozmiarem tablicy (liczba komórek)

Usuwanie konfliktów w tablicach rozproszonych

Bez obszarów nadmiarowych – próbkowanie liniowe

- Funkcja próbkująca ma postać:

$$p(i)=i$$

wówczas:

$$h_i(K)=[h(K)+i] \bmod R$$

- Wada: tendencja do „grupowania” zajętych komórek w tablicy (wówczas można stosować tzw. próbkowanie kwadratowe $p(i)=i^2$)

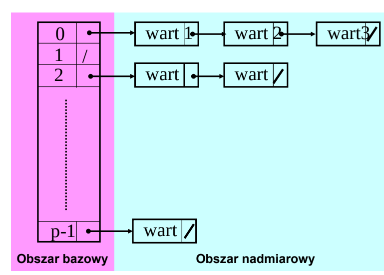
Usuwanie konfliktów w tablicach rozproszonych

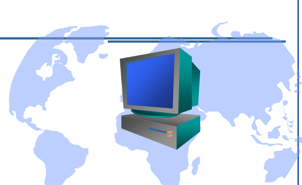
Z wykorzystaniem obszarów nadmiarowych

- **lista synomimów:** pierwsze wstawienie następuje do wolnego miejsca w obszarze bazowym; kiedy wyliczona funkcją haszującą pozycja z obszaru bazowego jest zajęta, to wstawiamy nowy element do listy synonimów przypisanych do tej pozycji w obszarze bazowym; listy synonimów tworzą obszary nadmiarowe;
- **rozproszona tablica indeksowa** - wszystkie dane są wstawiane do obszaru nadmiarowego

Rozproszone tablice indeksowe

- Idea rozproszonej tablicy indeksowej





Dziękuję za uwagę