Algorytmy i struktury danych

Wykład 5: Drzewa częściowo uporządkowane

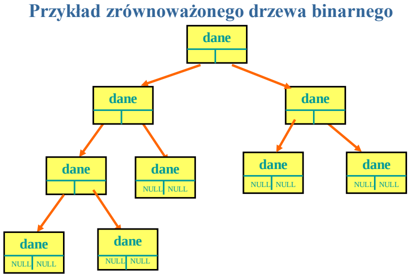
- Kopiec binarny
- Tablicowa reprezentacja kopca
- Przekształcanie tablicy w kopiec metodą wstępującą R.Floyda
- Kopiec jako kolejka priorytetowa
- Drzepiec binarny

Drzewa zrównoważone

- Drzewo (binarne) jest zrównoważone, jeżeli dla każdego węzła wysokości dwóch jego poddrzew (lewego i prawego) różnią się co najwyżej o 1 (własność tzw. drzew AVL)
- Dla drzewa zrównoważonego o liczbie węzłów równej *n* każda droga od korzenia do któregokolwiek z węzłów (w tym liści) nie jest dłuższa niż **lg n**

Drzewa zrównoważone

Przykład zrównoważonego drzewa binarnego

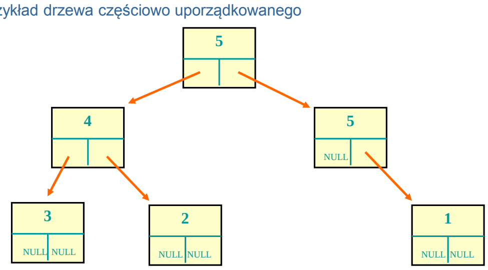


Drzewa częściowo uporządkowane

- Drzewa częściowo uporządkowane (ang. *Partially ordered tree*) są to drzewa binarne mające następującą własność:
  - Element przechowywany w węźle musi mieć co najmniej (co najwyżej) tak dużą wartość, jak wartości następników tego węzła
  - Własność ta oznacza, że element w korzeniu dowolnego poddrzewa jest zawsze największym (najmniejszym) elementem tego poddrzewa

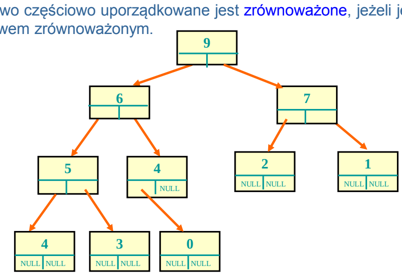
Drzewa częściowo uporządkowane

Przykład drzewa częściowo uporządkowanego



Drzewa częściowo uporządkowane

- Drzewo częściowo uporządkowane jest zrównoważone, jeżeli jest drzewem zrównoważonym.

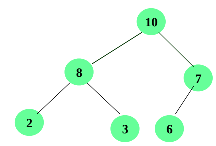


Drzewa częściowo uporządkowane

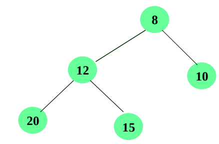
- **Kopiec**
  - Przykładem drzewa częściowo uporządkowanego może być tzw. kopiec (sterta, stóg, zwał), *ang. heap*
  - Drzewo binarne jest kopcem jeżeli:
    - wartości przechowywane w następnikach każdego węzła są mniejsze od wartości w danym węźle (tzw. kopiec maksymalny) lub jeżeli wartości przechowywane w następnikach każdego węzła są większe od wartości w danym węźle (tzw. kopiec minimalny)
    - drzewo jest zrównoważone, a wszystkie liście najniższego poziomu znajdują się na jego skrajnych, lewych pozycjach

Drzewa częściowo uporządkowane

Przykłady kopców



Kopiec binarny maksymalny



Kopiec binarny minimalny

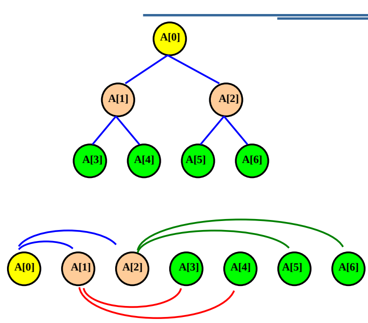
Drzewa częściowo uporządkowane

- Kopiec można zaimplementować bazując na tablicy jednowymiarowej (wektorze) o długości *n*
- Elementy są umieszczane w drzewie w kolejnych węzłach od góry do dołu i od lewej strony do prawej
- **Uwaga**: każdy kopiec jest tablicą (ale nie każda tablica jest kopcem)

Drzewa częściowo uporządkowane

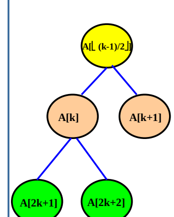
- Cechy charakterystyczne tablicy A implementującej kopiec:
  - Korzeń znajduje się w A[0]
  - Po korzeniu zapisujemy w tablicy kolejne poziomy;
  - Zatem: lewy następnik korzenia znajduje się w A[1], prawy następnik korzenia – w A[2] ;
  - Ogólnie: lewy następnik węzła zapisanego w A[i] znajduje się w A[2i +1], prawy następnik – w A[2i+2] (jeżeli następniki istnieją);

Drzewa częściowo uporządkowane – reprezentacja tablicowa



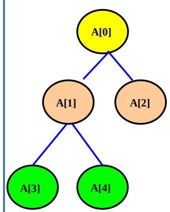
Drzewa częściowo uporządkowane

$\lfloor a \rfloor$ - największa liczba całkowita x taka, że $x \le a$
$\lceil a \rceil$ - najmniejsza liczba całkowita x taka, że $x \ge a$



- **następniki k-tego węzła mają indeksy równe:**
  2k +1 **lewy następnik**
  2k +2 **prawy następnik**
- **węzeł nadrzędny ma indeks równy** $\lfloor (k-1)/2 \rfloor$ **(dzielenie całkowitoliczbowe)**

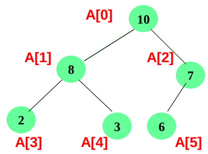
Drzewa częściowo uporządkowane



⌊a⌋ - największa liczba całkowita x taka, że x≤a
⌈a⌉ - najmniejsza liczba całkowita x taka, że x≥a

- następniki k-tego węzła mają indeksy równe:
  2k +1 lewy następnik
  2k +2 prawy następnik
- węzeł nadrzędny ma indeks równy
  ⌊*(k –1)/ 2*⌋ (dzielenie całkowitoliczbowe)

Drzewa częściowo uporządkowane



tablica A:

0	1	2	3	4	5
10	8	7	2	3	6

Zachodzi:

$A[k] \geq A[2k+1]$

oraz

$A[k] \geq A[2k+2]$

Drzewa częściowo uporządkowane

Idea algorytmu przekształcania tablicy data[ ] w kopiec metodą wstępującą R. Floyda (1964):

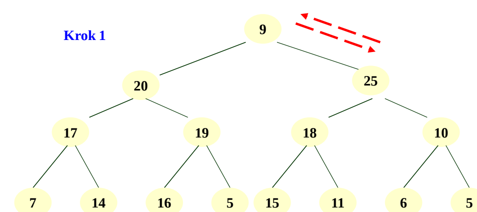
for (i=indeks ostatniego węzła-nie liścia; i>=0; i--)
odtwórz warunki kopca dla drzewa, którego korzeniem jest data[i], wywołując funkcję MoveDown(data[ ], i, n-1);

Prototyp funkcji MoveDown:
```
void MoveDown(T data[ ], int first, int last);
```


Drzewa częściowo uporządkowane

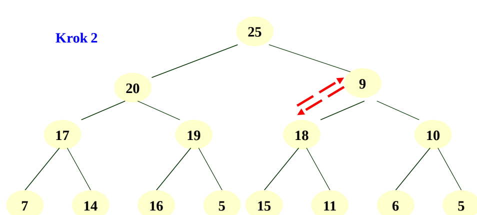
Idea działania funkcji *MoveDown* (tzw. przesiewanie)

Krok 1



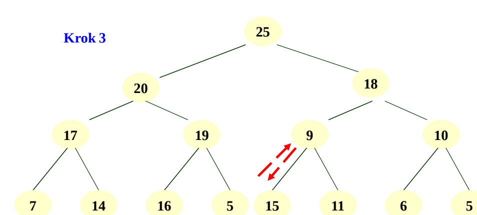
Drzewa częściowo uporządkowane

Krok 2



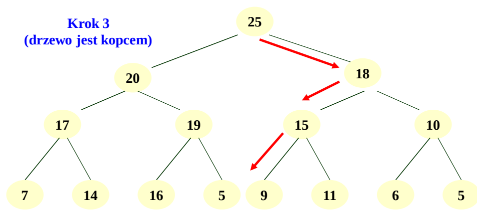
Drzewa częściowo uporządkowane

Krok 3



Drzewa częściowo uporządkowane

Krok 3
(drzewo jest kopcem)

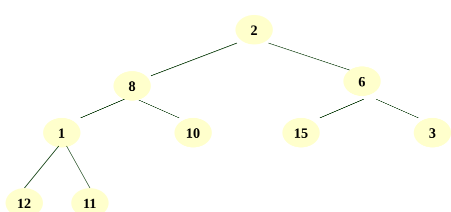


Drzewa częściowo uporządkowane

```
void MoveDown(T data[ ], int first, int last) {
int largest = 2* first +1;
 while (largest <= last)  {
    if (largest < last && data[largest] < data[largest+1] )
          largest ++ ;   /* first ma dwa następniki: lewy  w 2*first+1 oraz prawy
                            w 2*first+2, przy czym prawy jest większy od lewego  */
    if (data[first] < data[largest]) {
    //  jeśli trzeba zamień większy następnik z jego poprzednikiem
          zamien(data[first], data[largest]);
          first=largest;
          largest=2*first+1;
    }
    else largest=last+1; // nastąpi wyjście z pętli; poddrzewo jest kopcem
 }
}
```


Drzewa częściowo uporządkowane

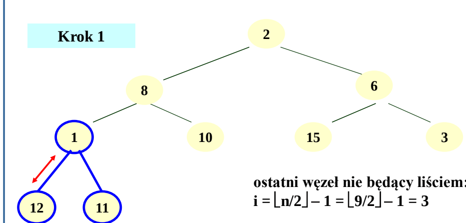
Idea przekształcania tablicy A=[2 8 6 1 10 15 3 12 11]
w kopiec metodą wstępującą R. Floyda



2	8	6	1	10	15	3	12	11

Drzewa częściowo uporządkowane

Krok 1

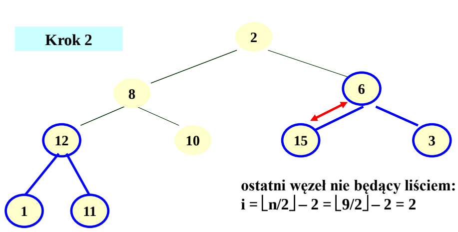


ostatni węzeł nie będący liściem:
$i = \lfloor n/2 \rfloor - 1 = \lfloor 9/2 \rfloor - 1 = 3$

2	8	6	1	10	15	3	12	11

Drzewa częściowo uporządkowane

Krok 2

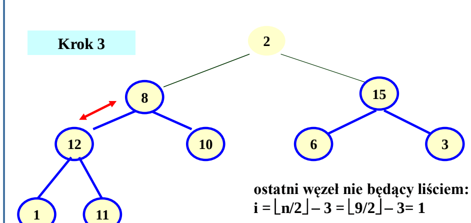


ostatni węzeł nie będący liściem:
$i = \lfloor n/2 \rfloor - 2 = \lfloor 9/2 \rfloor - 2 = 2$

2	8	6	12	10	15	3	1	11

Drzewa częściowo uporządkowane

Krok 3

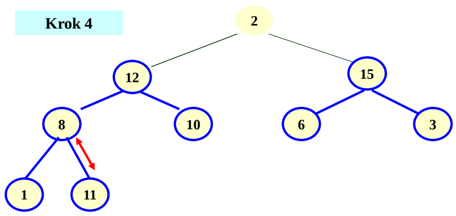


ostatni węzeł nie będący liściem:
$i = \lfloor n/2 \rfloor - 3 = \lfloor 9/2 \rfloor - 3 = 1$

2	8	15	12	10	6	3	1	11

Drzewa częściowo uporządkowane

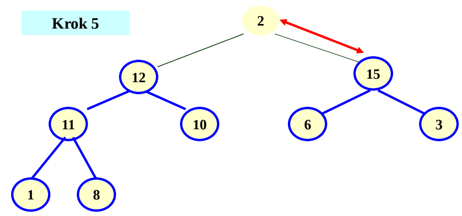
Krok 4



2	12	15	8	10	6	3	1	11

Drzewa częściowo uporządkowane

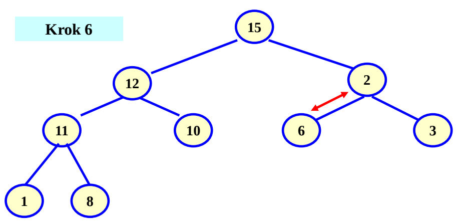
Krok 5



2	12	15	11	10	6	3	1	8

Drzewa częściowo uporządkowane

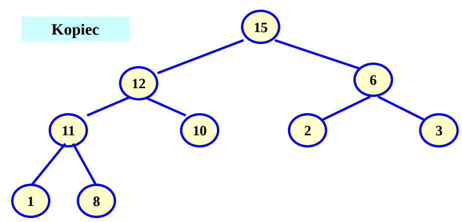
Krok 6



15	12	2	11	10	6	3	1	8

Drzewa częściowo uporządkowane

Kopiec



15	12	6	11	10	2	3	1	8

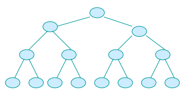
Przekształcanie tablicy w kopiec
(algorytmem wstępującym Floyda)

```
void Alg_Wst_Floyda( T data[ ], int n)
{
    int i;
    for (i=n/2-1;  i>=0;  i--)          // utworzenie kopca
          MoveDown(Tab, i, n-1);
}
```

Program 5.1


```
void MoveDown(T data[ ], int first, int last) {
int largest = 2* first +1;
 while (largest <= last)  {
   if (largest < last && data[largest] < data[largest+1] )
         largest ++ ;   /* first ma dwa następniki: lewy  w 2*first+1 oraz prawy
                           w 2*first+2, przy czym prawy jest większy od lewego  */
   if (data[first] < data[largest]) {
   //  jeśli trzeba zamień większy następnik z jego poprzednikiem
         zamien(data[first], data[largest]);
         first=largest;
         largest=2*first+1;
   }
   else largest=last+1; // nastąpi wyjście z pętli; poddrzewo jest kopcem
 }
}
```

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



dla np. n=10 000

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

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

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}	$n=2^h-1$

Drzewa częściowo uporządkowane

Analiza złożoności algorytmu przekształcania tablicy w kopiec *(algorytmem wstępującym Floyda)*

- Rozpatrujemy drzewo binarne o **n** węzłach i wysokości **h**
- Zachodzi:

  $n = 2^h – 1$ czyli **$h = \lg(n+1)$**
- Liczba węzłów na ostatnim (h-tym) poziomie drzewa

  $n_h = 2^{h-1}$
- Związek pomiędzy liczba węzłów na **i**-tym od dołu poziomie drzewa a liczbą węzłów drzewa **n, i=0, 1, 2, ..., h-1**

$$n_{h-i} = 2^{h-i-1} = 2^{\lg(n+1)-i-1}$$
$$\lg n_{h-i} = \lg 2^{\lg(n+1)-i-1} = \lg(n+1) - i - 1 =$$
$$= \lg(n+1) - \lg 2^{i+1} = \lg\frac{n+1}{2^{i+1}}$$

Drzewa częściowo uporządkowane

- Mamy zatem

$$\lg n_{h-i} = \lg\frac{n+1}{2^{i+1}}$$

czyli

$$n_{h-i} = \frac{n+1}{2^{i+1}}$$

- np. dla i=1 (przedostatni poziom); dla i=2 (drugi od dołu poziom)

$$n_{h-1} = \frac{n+1}{4} \qquad n_{h-2} = \frac{n+1}{8}$$

Drzewa częściowo uporządkowane

Analiza złożoności algorytmu przekształcania tablicy w kopiec *(algorytmem wstępującym Floyda)*

- **Funkcja *MoveDown* przenosi (w najgorszym razie) dane z przedostatniego poziomu zawierającego (n+1)/4 węzłów o jeden poziom w dół, przeprowadzając (n+1)/4 zamiany**
- **Dane z drugiego od końca poziomu, który ma (n+1)/8 węzłów przenoszone są o dwa poziomy w dół, co oznacza 2 (n+1)/8 przesunięć itd. aż do korzenia**
- **Korzeń może być (w najgorszym razie) przeniesiony o lg(n+1) – 1 = lg[(n+1)/2] poziomy**
- **Łączna liczba przesunięć (zamian wartości):**

$$\sum_{i=1}^{\lg(n+1)} \frac{n+1}{2^{i+1}} i = \sum_{j=2}^{\lg(n+1)} \frac{n+1}{2^j}(j-1) = (n+1)\sum_{j=2}^{\lg(n+1)} \frac{j-1}{2^j} =$$
$$= (n+1)\left[\sum_{j=2}^{\lg(n+1)} \frac{j}{2^j} - \sum_{j=2}^{\lg(n+1)} \frac{1}{2^j}\right] \le (n+1)(1{,}5-0{,}5) = n+1 = O(n)$$

Kopiec jako kolejka priorytetowa

- Kopiec może być podstawą bardzo efektywnej implementacji kolejki priorytetowej;
- Aby wstawić element do kolejki dodaje się go na koniec jako ostatni liść (należy wówczas najczęściej odtworzyć własność kopca);
- Pobieranie elementu z kolejki polega na pobraniu wartości z korzenia; na jego miejsce przesuwany jest ostatni liść (najczęściej trzeba potem odtworzyć własność kopca);

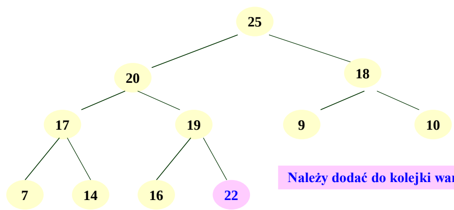
Kopiec jako kolejka priorytetowa

Idea algorytmu wstawiania elementu do kolejki:

```
Wstaw_Do_Kolejki_Kopca(T elm) {
    wstaw elm na koniec kopca;
    while (elm nie jest korzeniem && elm > poprzednik(elm))
            zamień miejscami elm i poprzednik(elm);
}
```

Kopiec jako kolejka priorytetowa

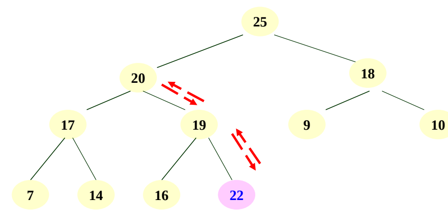
Idea algorytmu wstawiania elementu do kolejki-kopca



Należy dodać do kolejki wartość 22

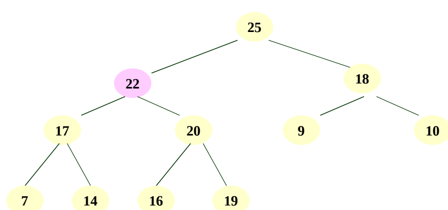
Kopiec jako kolejka priorytetowa

Przywracanie własności kopca



Kopiec jako kolejka priorytetowa

Kopiec z dodanym elementem



Kopiec jako kolejka priorytetowa

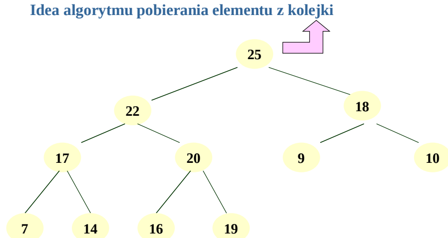
Idea algorytmu pobierania elementu z kolejki:

```
Pobierz_Z_Kolejki_Kopca() {
    pobierz element z korzenia;
    przenieś do korzenia element z ostatniego liścia;
    usuń ostatni liść;  // lewe i prawe poddrzewo są kopcami
    p=korzeń;
    while (p nie jest liściem && p < którykolwiek następnik(p))
            zamień miejscami p i większy następnik(p);
}
```

Ostatnie trzy wiersze algorytmu to przywracanie drzewu własności kopca (realizuje to funkcja MoveDown)

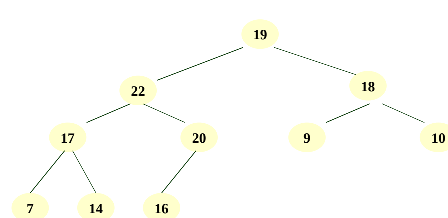
Kopiec jako kolejka priorytetowa

Idea algorytmu pobierania elementu z kolejki



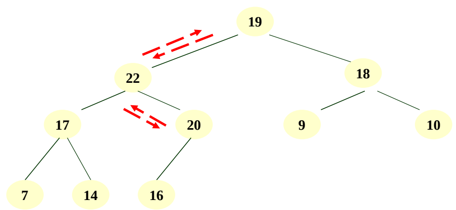
Kopiec jako kolejka priorytetowa

Zastąpienie korzenia ostatnim liściem



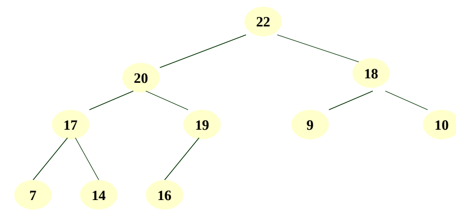
Kopiec jako kolejka priorytetowa

Przywrócenie własności kopca



Kopiec jako kolejka priorytetowa

Kolejka po usunięciu elementu o najwyższym priorytecie

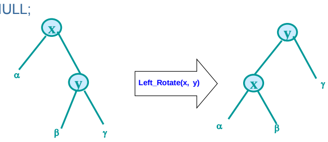


Rotacja węzła w drzewie binarnym

- **Rotacja:**
  - zmiana konfiguracji węzłów;
  - cel: przywrócenie struktury drzewa AVL;
  - podstawowa własność rotacji: po jej wykonaniu drzewo jest nadal drzewem BST;
  - rodzaje rotacji:
    - w lewo i w prawo,
    - pojedyncze i podwójne;

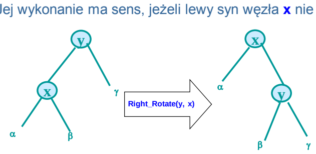
Rotacja węzła w drzewie binarnym

- Lewa rotacja (lub „rotacja w lewo") węzła **y** wokół węzła *x*
  - Polega na obrocie węzła **y** wokół wyróżnionego węzła **x** przeciwnie do ruchu wskazówek zegara;
  - W wyniku rotacji węzeł **y** staje się nowym korzeniem poddrzewa, węzeł **x** zostaje jego lewym następnikiem, a lewy następnik węzła **y** zostaje prawym następnikiem węzła **x**;
  - Jej wykonanie ma sens, jeżeli prawy następnik węzła **y** nie jest NULL;



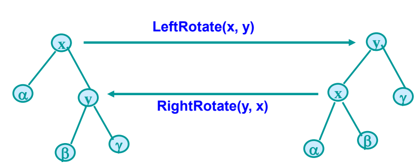
Rotacja węzła w drzewie binarnym

- Prawa rotacja (lub „rotacja w prawo") węzła **x** wokół węzła **y**:
  - Polega na obrocie węzła **x** wokół wyróżnionego węzła **y** zgodnie z ruchem wskazówek zegara;
  - W wyniku rotacji węzeł **x** staje się nowym korzeniem poddrzewa, węzeł **y** zostaje jego prawym następnikiem, a prawy następnik węzła **x** zostaje lewym następnikiem węzła **y**;
  - Jej wykonanie ma sens, jeżeli lewy syn węzła **x** nie jest NULL;



Rotacja węzła w drzewie binarnym

- ***Rotacja:***
  - Lewa i prawa rotacja działają symetrycznie



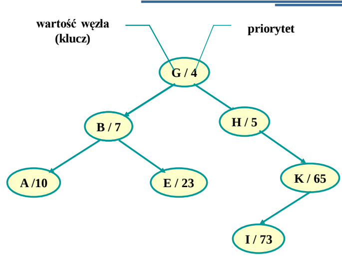
Drzepce

- Drzepiec (*ang. treap*) – drzewo BST, w którym porządek wstawiania elementów określany jest w pewien specjalny sposób;
- W każdym węźle drzepca, oprócz pola wartości (klucza) występuje pole priorytet, którego wartością jest losowo określana liczba, niezależnie dla każdego węzła (przy założeniu, że wszystkie wartości i wszystkie priorytety są różne);
- Elementy (węzły) w drzepcu są uporządkowane w taki sposób, że wartości (klucze) spełniają kryteria drzewa BST, natomiast priorytety spełniają własność kopca minimalnego, tj. z najmniejszym priorytetem w korzeniu;
- Drzepiec jest zatem połączeniem drzewa BST i kopca (stąd nazwa: drzewo BST + kopiec)

Drzepce

- Pomocne jest, aby myśleć o drzepcach w następujący sposób:
  - załóżmy, że wstawiamy do drzepca węzły x_1, x_2, ..., x_n wraz ze związanymi z nimi wartościami (kluczami);
  - aby otrzymać drzepiec wstawiamy te węzły w kolejności wyznaczonej przez ich (losowo ustalone) priorytety, tzn. x_i jest wstawiany przed x_j, jeżeli *priorytet*(x_i) < *priorytet*(x_j)
  - w efekcie otrzymamy kopiec minimalny (z uwagi na priorytety)

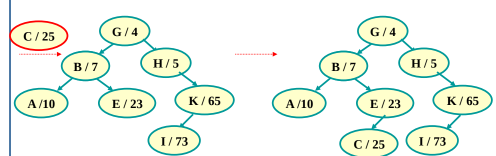
Drzepce



Przykład drzepca (będącego drzewem BST i kopcem minimalnym)

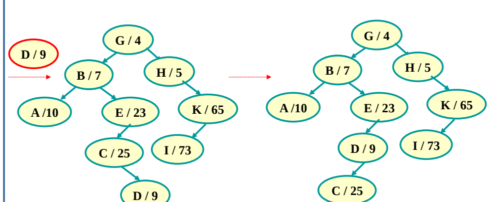
Drzepce

Idea algorytmu wstawiania węzła do drzepca



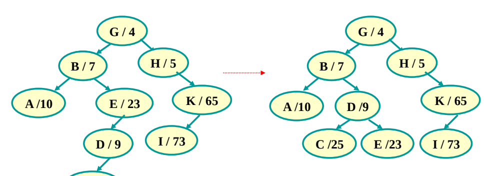
Drzepce

Idea algorytmu wstawiania węzła do drzepca



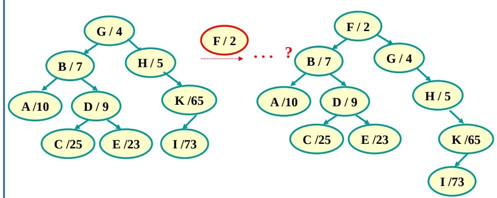
Drzepce

Idea algorytmu wstawiania węzła do drzepca



Drzepce

Idea algorytmu wstawiania węzła do drzepca



Dziękuję za uwagę