ALGORYTMY I STRUKTURY DANYCH

Temat 5:

Drzewa zrównoważone, sortowanie drzewiaste

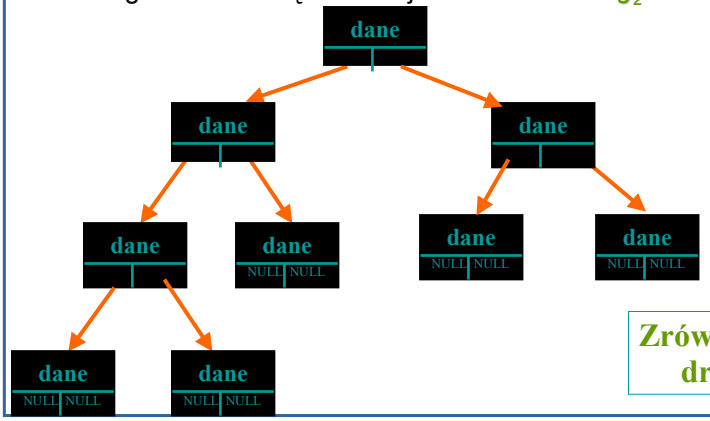
Wykładowca: dr inż. Zbigniew TARAPATA

e-mail: Zbigniew.Tarapata@isi.wat.edu.pl

http://www.tarapata.strefa.pl/p_algorytmy_i_struktury_danych/

Drzewa zrównoważone, sortowanie drzewiaste

- Drzewo (dowolne) jest zrównoważone, jeżeli na wszystkich poziomach poza najniższym zawiera wszystkie możliwe węzły oraz liście na najniższym poziomie są ułożone od lewej strony. Własność: dla drzewa o liczbie węzłów równej n żadna droga od korzenia do któregokolwiek z węzłów nie jest dłuższa niż $log_2 n$.



Zrównoważone drzewo binarne

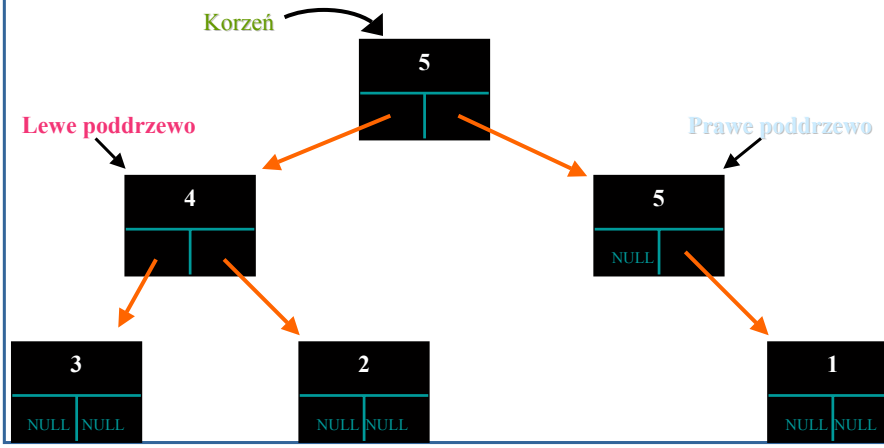
Drzewa zrównoważone, sortowanie drzewiaste

- Drzewa częściowo uporządkowane (ang. *Partially ordered tree*) są to drzewa binarne spełniające następujące własności:
- Etykietami węzłów są elementy z przypisanymi priorytetami (jedno z pól rekordu opisującego węzeł drzewa);
- Element przechowywany w węźle musi mieć co najmniej tak duży priorytet, jak element znajdujący się w dzieciach tego węzła;

- Druga własność oznacza, że element w korzeniu dowolnego poddrzewa jest zawsze największym elementem tego poddrzewa. Własność ta nosi nazwę własności drzewa częściowo uporządkowanego;

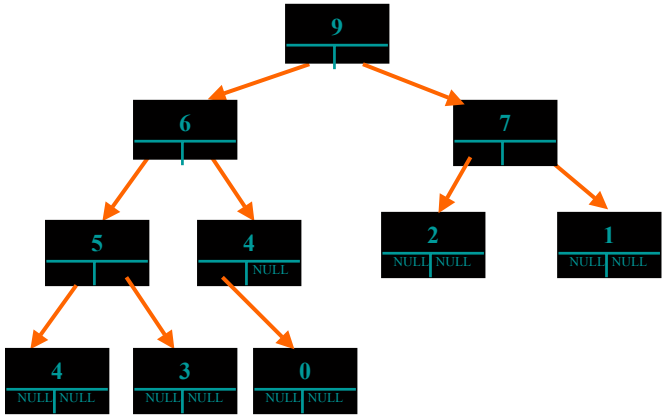
Drzewa zrównoważone, sortowanie drzewiaste

- Przykład drzewa częściowo uporządkowanego.



Drzewa zrównoważone, sortowanie drzewiaste

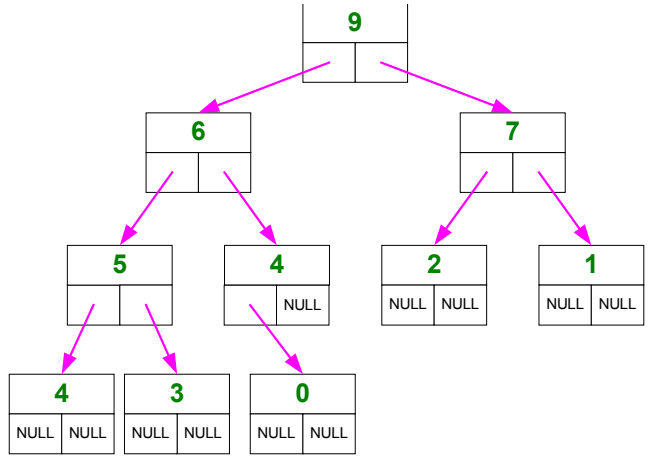
- Drzewo częściowo uporządkowane jest zrównoważone, jeżeli jest drzewem zrównoważonym.



Drzewa zrównoważone, sortowanie drzewiaste

- Zrównoważone drzewa częściowo uporządkowane można implementować za pomocą tablicy A zwanej stogiem (ang. *heap*). Cechy charakterystyczne:
- Korzeń znajduje się w A[1]; nie wykorzystujemy A[0] !!!;
- Po korzeniu zapisujemy w tablicy kolejne poziomy;
- Na każdym poziomie węzły porządkujemy od lewej do prawej;
- Zatem: lewe dziecko korzenia znajduje się w A[2], prawe dziecko korzenia – w A[3] ;
- Ogólnie: lewe dziecko węzła zapisanego w A[i] znajduje się w A[2i] , prawe dziecko – w A[2i+1] (jeśli dzieci istnieją);

Drzewa zrównoważone, sortowanie drzewiaste –
przykład stogu w postaci drzewa i tablicy



1	2	3	4	5	6	7	8	9	10
9	6	7	5	4	2	1	4	3	0

Drzewa zrównoważone, sortowanie drzewiaste

- Przykład 1

Napisać program, który dodaje nowy element stogu oraz umieszcza go na właściwej dla niego pozycji.

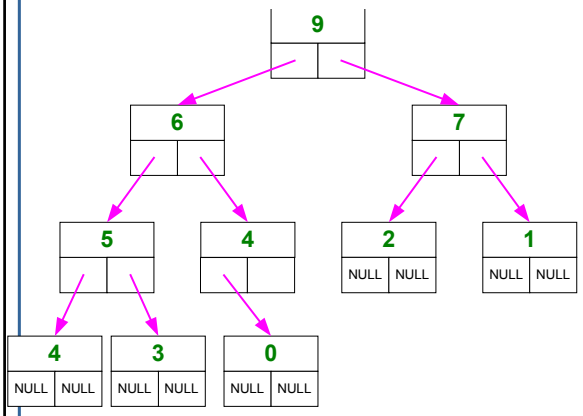
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - rozwiązanie

```
void swap(int A[], int i, int j) /*wymienia pozycjami elementy tablicy */
{
        int temp;
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
}
void bubbleUp(int A[], int i) /*umieszcza nowy element stogu na właściwej dla niego pozycji*/
{
        if (i > 1 && A[i] > A[i/2]) {
                swap(A, i, i/2);
                bubbleUp(A, i/2);
        }
}
```


Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie

```
void insert(int A[], int x, int *pn) /*wstawia nowy element do stogu */
{
        (*pn)++;
        A[*pn] = x;
        bubbleUp(A, *pn);
}
```

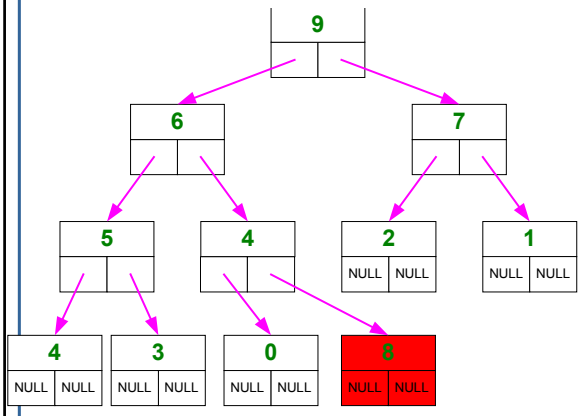
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
void insert(int A[], int x, int *pn) {
    (*pn)++;
    A[*pn] = x;
    bubbleUp(A, *pn);
}
```

1	2	3	4	5	6	7	8	9	10
9	6	7	5	4	2	1	4	3	0

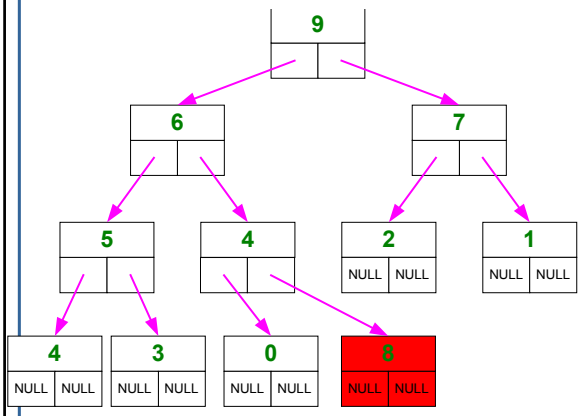
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
void insert(int A[], int x, int *pn) {
    (*pn)++;          *(pn)=11
    A[*pn] = x;       A[*pn]=8
    bubbleUp(A, *pn); bubbleUp(A, 11
}
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	4	2	1	4	3	0	8

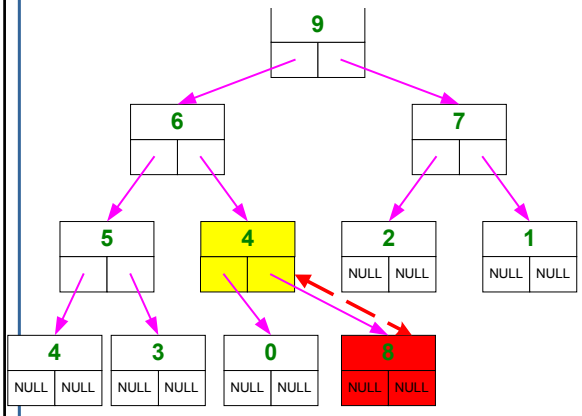
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
void bubbleUp(int A[], int i) {
    if (i > 1 && A[i] > A[i/2]) {
        swap(A, i, i/2);
        bubbleUp(A, i/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	4	2	1	4	3	0	8

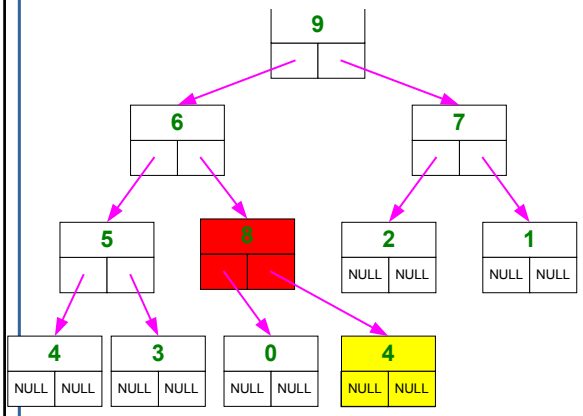
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A, 11) {
    if (11 > 1 && A[11] > A[11/2]) {
        swap(A, 11, 11/2);
        bubbleUp(A, 11/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	4	2	1	4	3	0	8

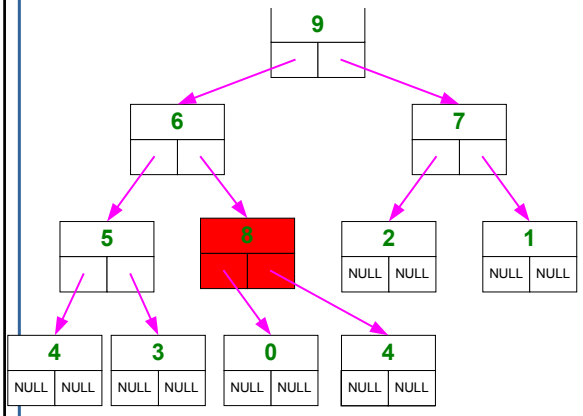
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A, 11) {
    if (11 > 1 && A[11] > A[11/2]) {
        swap(A, 11, 11/2);
        bubbleUp(A, 11/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	8	2	1	4	3	0	4

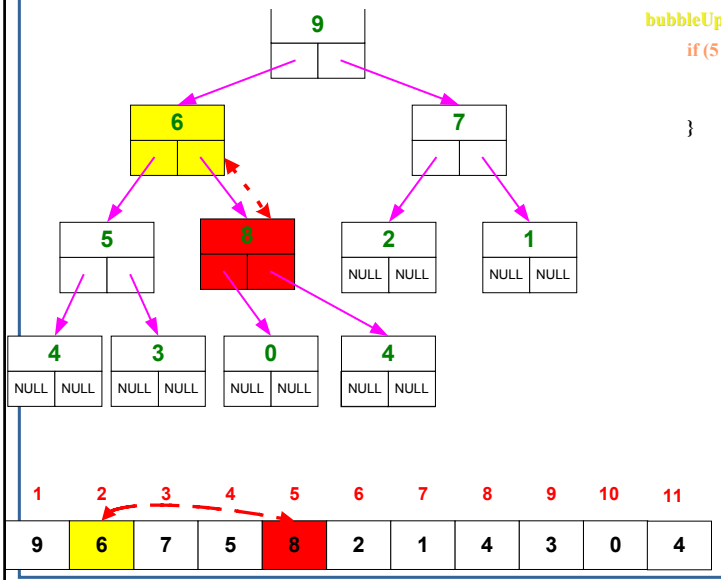
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A, 11) {
    if (11 > 1 && A[11] > A[11/2]) {
        swap(A, 11, 11/2);
        bubbleUp(A, 11/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	8	2	1	4	3	0	4

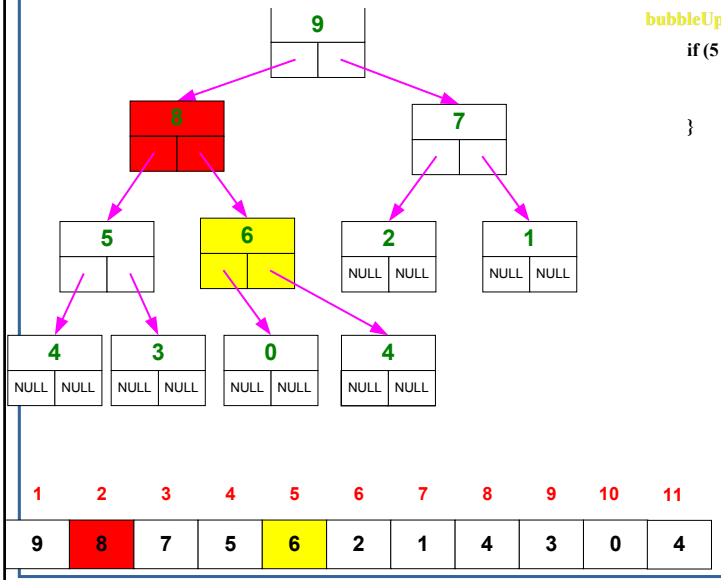
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A,5) {
    if (5 > 1 && A[5] > A[5/2]) {
        swap(A, 5, 2);
        bubbleUp(A, 5/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	6	7	5	8	2	1	4	3	0	4

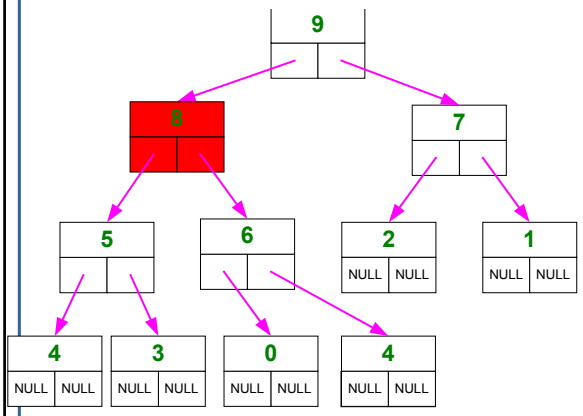
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A,5) {
    if (5 > 1 && A[5] > A[5/2]) {
        swap(A, 5, 2);
        bubbleUp(A, 5/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	8	7	5	6	2	1	4	3	0	4

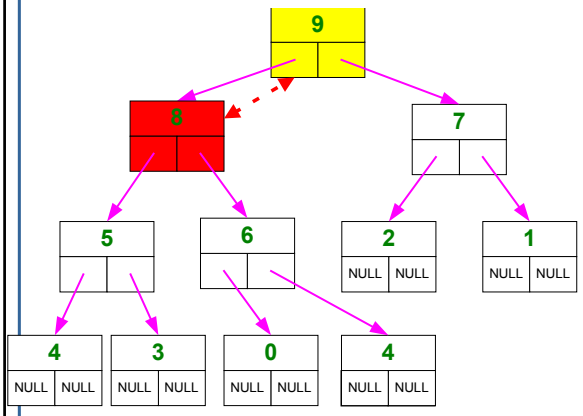
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A,5) {
    if (5 > 1 && A[5] > A[5/2]) {
        swap(A, 5, 5/2);
        bubbleUp(A, 5/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	8	7	5	6	2	1	4	3	0	4

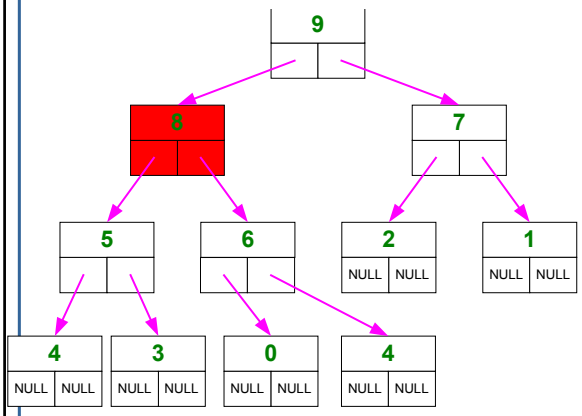
Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A,2) {
    if (2 > 1 && A[2] > A[2/2]) {
        swap(A, 2, 2/2);
        bubbleUp(A, 2/2);
    }
```

1	2	3	4	5	6	7	8	9	10	11
9	8	7	5	6	2	1	4	3	0	4

Drzewa zrównoważone, sortowanie drzewiaste – przykład 1 - zobrazowanie




```
bubbleUp(A,2) {
    if (2 > 1 && A[2] > A[2/2]) {
        swap(A, 2, 2/2);
        bubbleUp(A, 2/2);
    } /* KONIEC */
```

1	2	3	4	5	6	7	8	9	10	11
9	8	7	5	6	2	1	4	3	0	4

Drzewa zrównoważone, sortowanie drzewiaste

- Przypomnijmy, że w drzewie częściowo uporządkowanym element maksymalny Max znajduje się ZAWSZE w korzeniu drzewa (czyli przy reprezentacji za pomocą tablicy A, znajduje się na pozycji 1, tzn. Max=A[1]).
- Aby usunąć element maksymalny z takiego drzewa (stogu) postępujemy dwuetapowo:
- Najpierw zamieniamy pierwszy element drzewa z ostatnim (tzn. wykonujemy swap(A, 1, n) , gdzie n to rozmiar tablicy) i zmniejszamy rozmiar tablicy, tzn. n=n-1;
- Przywracamy naruszoną (być może) w ten sposób własność drzewa „przebąbelkowując" w dół elementy zmodyfikowanego drzewa poczynając od korzenia;

Drzewa zrównoważone, sortowanie drzewiaste

- „Przebąbelkowanie” w dół polega na tym, że :
- sprawdzamy korzeń A[i] z jego dziećmi (A[2i] , A[2i+1]) ;
- jeżeli wartość w korzeniu A[i] jest mniejsza niż NAJWIĘKSZA z wartości jego dzieci, tzn. zachodzi
  A[i] < max {A[2i] , A[2i+1] }, to
- zamieniamy miejscami wartość A[i] z wartością max {A[2i] , A[2i+1] };
- Wywołujemy rekurencyjnie naszą funkcję „bąbelkowania” w dół podając jako korzeń 2i lub 2i+1 (w zależności od tego, na której z tych pozycji była wartość większa);
- Postępowanie kontynuujemy dopóki nie trafimy na koniec drzewa (tablicy A).

Drzewa zrównoważone, sortowanie drzewiaste

- Sortowanie drzewiaste (przez kopcowanie, ang. *heapsort*) dowolnej tablicy A składa się z dwóch etapów:
- Najpierw tablicę A przetwarzamy na tablicę reprezentującą stóg (patrz Zadanie 4) – po wykonaniu A reprezentuje stóg;
- Następnie w pętli od 1..n wywołujemy funkcję usuwania ze stogu A elementu największego;
- Po wykonaniu ostatniego punktu tablica A posortowana jest rosnąco !
  - DLACZEGO ??? ☺ - zadanie domowe

Dziękuję za uwagę