# ПИРАМИДАЛНО СОРТИРАНЕ

ВТУ, 27.10.2008 г.

# ДЪРВЕТА – ОБЩИ ПОНЯТИЯ

Дефиниция (неформална) за **дърво** – множество от върхове, свързани с ребра, като тръгвайки от даден връх и движейки се по ребрата (без да повтаряме ребро), не може да се стигне до началния връх (т.е., няма цикли)

**Двоично дърво** – състои се от корен, ляво поддърво и дясно поддърво (като е възможно поддърветата да са празни – без елементи). Връх, на който поддърветата са празни (т.е., няма **наследници**), наричаме **листо**.

**Двоично дърво** – дърво, на което всеки връх има най-много два наследника.

**Пълно двоично дърво (ПДД)** – дърво, на което всеки връх има точно 2 наследника, като последното ниво от листа може да не е запълнено.

Свойства на ПДД:

1. Броят на върховете на ПДД на всяко ниво (без последното) е степен на двойката;

2. Броят на листата в ПДД с $n$ върха е $n/2$.

# ПРЕДСТАВЯНЕ НА ПДД

Всяко ПДД с $n$ върха може да се представи като едномерен $n$-елементен масив по следния начин:

- Пръв елемент на масива е коренът на ПДД;
- В масива последователно добавяме всяко ниво на ПДД, поставяйки елементите от ляво на дясно;
- Ако даден връх е разположен на $i$-та позиция, то неговите наследници са на $(2i + 1)$-ва и $(2i + 2)$-ра позиция;

**Пирамида (heap)** – пълно двоично дърво, като стойността на даден връх е по-голяма или равна на стойностите на наследниците му – **свойство на пирамидата**.

– В корена на пирамидата е разположен най-големият елемент (следствие от дефиницията)

# АЛГОРИТЪМ НА ПИРАМИДАЛНОТО СОРТИРАНЕ

Алгоритъмът използва структурата от данни пирамида и се състои от два етапа:

1. Функция *Sift* (отсяване) която се извиква за връх $i$, който нарушава свойството на пирамидата. Тя предполага, че лявото и дясното поддърво на $i$ са пирамиди (удовлетворяват свойството на пирамидата). Като резултат функцията *Sift* прави пълното двоично поддърво, с корен връх $i$, на пирамида;

2. Функция *HeapSort*, която прави произволен масив на пирамида, като използва функцията *Sift* и извършва последната фаза на сортирането, като използва *Sift* и свойството на пирамидата.

# ФУНКЦИЯ ЗА 'ОТСЯВАНЕ' НА ВРЪХ

Функцията за 'отсяване' ($Sift$) на даден връх $i$ се извиква, когато за върха $i$ е нарушено свойството на пирамидата. Тя се изпълнява бързо, тъй като работи върху $\log_2 n$ елемента на пирамидата.

Функцията предполага, че поддърветата с корени левия и десен наследник на върха $i$ са пирамиди.

За да се удовлетвори свойството на пирамидата, е достатъчно да се разменят стойността на върха $i$ с по-голямата от двете стойности на левия и десния наследник на $i$.

Но тъй като по този начин може да се наруши свойството на пирамидата за наследника на върха $i$, с който е станала размяната, то трябва да се извършват проверки и размени, докато върхът $i$ 'си намери мястото' (т.е., наследниците на върха $i$ да не са по-големи от него).

# ФУНКЦИЯ $Sift$

$l$ – индексът на върха, който ще се 'отсява'

$r$ – индексът на последния елемент на пирамидата

```
void Sift(int A[ ], int l, int r) {
    int i = l, j = 2 * i + 1;
    int x = A[i];
    if(j < r && A[j + 1] > A[j])
        j++;
    while(j <= r && A[j] > x) {
        A[i] = A[j]; i = j; j = 2 * i + 1;
        if(j < r && A[j + 1] > A[j])
            j++;
    }
    if(A[i] != x)
        A[i] = x;
}
```

# ФУНКЦИЯ *HEAPSORT*

Построяваме пирамида, проверявайки в обратен ред първите $n/2$ върха (останалите са листа).

След това разменяме първия (най-големия) и последния елемент и намаляваме с един броя на елементите на масива, от които се 'изгражда' пирамидата. Повтаряме това действие, докато остане само един елемент.

```
void HeapSort (int A[ ], int n) {
    for (int i = n/2; i >= 0; i − −)
        Sift (A, i, n);
    i = n − 1;
    while (i > 0) {
        int x = A[0]; A[0] = A[i]; A[i] = x;
        i − −;
        Sift (A, 0, i);
    }
}
```