

Structuri de date si algoritmi

Curs, IS – An II

100101001010
01010110001010010100
01010110001010010100101001
10101100010100101001010010 100
0010101100010100101001010010100 10 10
010101100010100101001010010100111
1000101001010010100101001010
01001010010100101001010
101100
011000
01100
011000
101100
00101
110001010010011
01010110001010010100101001

Arbori Heap

Heap

Definiție

- Arbori **binari** parțial ordonați, de **înălțime minimă**, în care fiecare vârf (nod) conține un câmp **cheie** a.i. pe mulțimea constantelor tipului cheii există o relație de **ordine parțială**

Proprietati

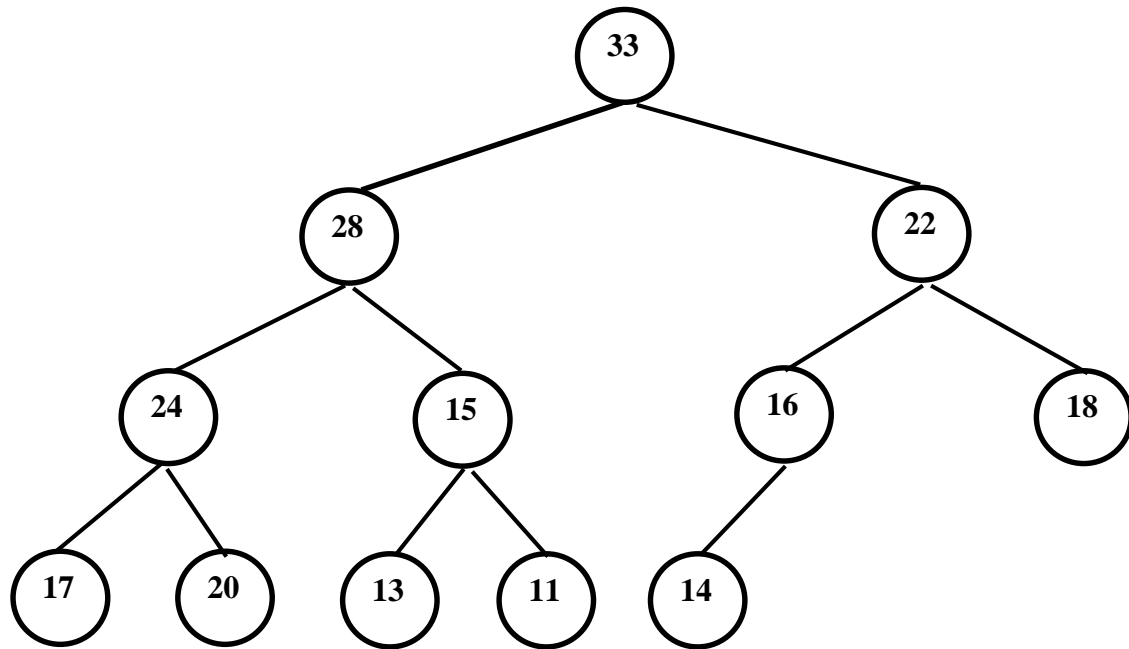
- Un arbore Heap este aproape complet, mai exact toate nivelurile sunt complete cu posibila excepție a ultimului nivel, completat de la stânga spre dreapta;
- Valoarea cheii oricărui nod este mai mare decât valorile cheilor descendenților săi (**MaxHeap**), sau, alternativ, valoarea cheii oricărui nod este mai mică decât valorile cheilor fiilor săi (**MinHeap**) .

Obs.: Pot fi mai mulți atomi cu aceeași cheie.

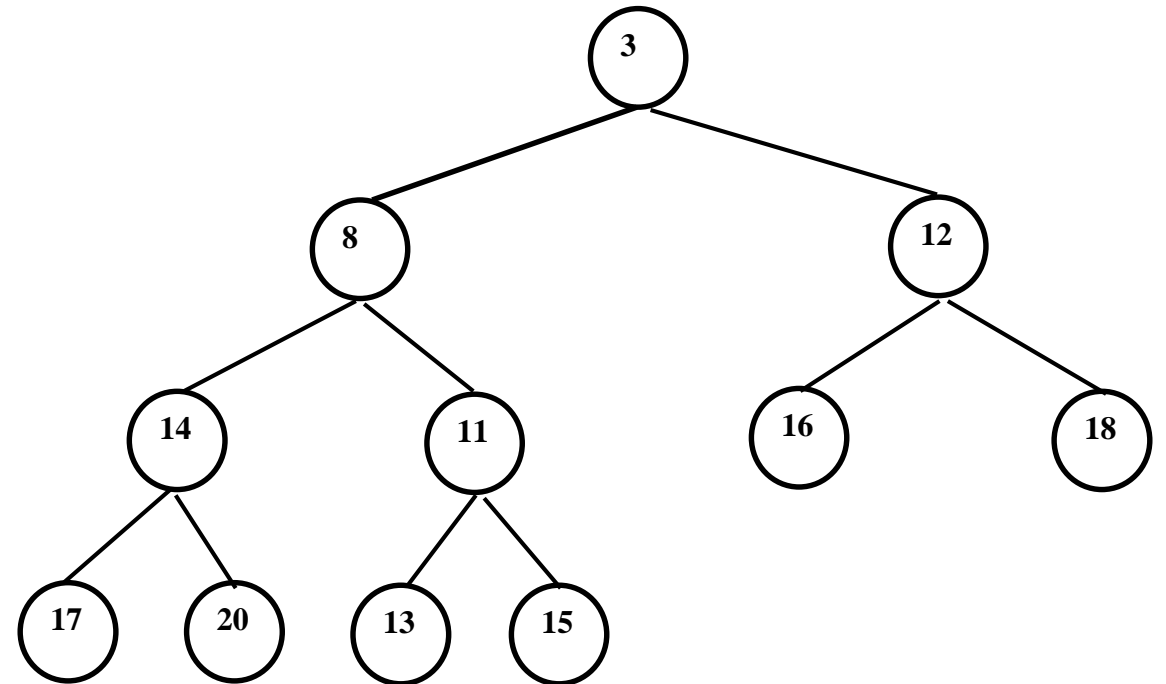
Heap

Exemple

MaxHeap



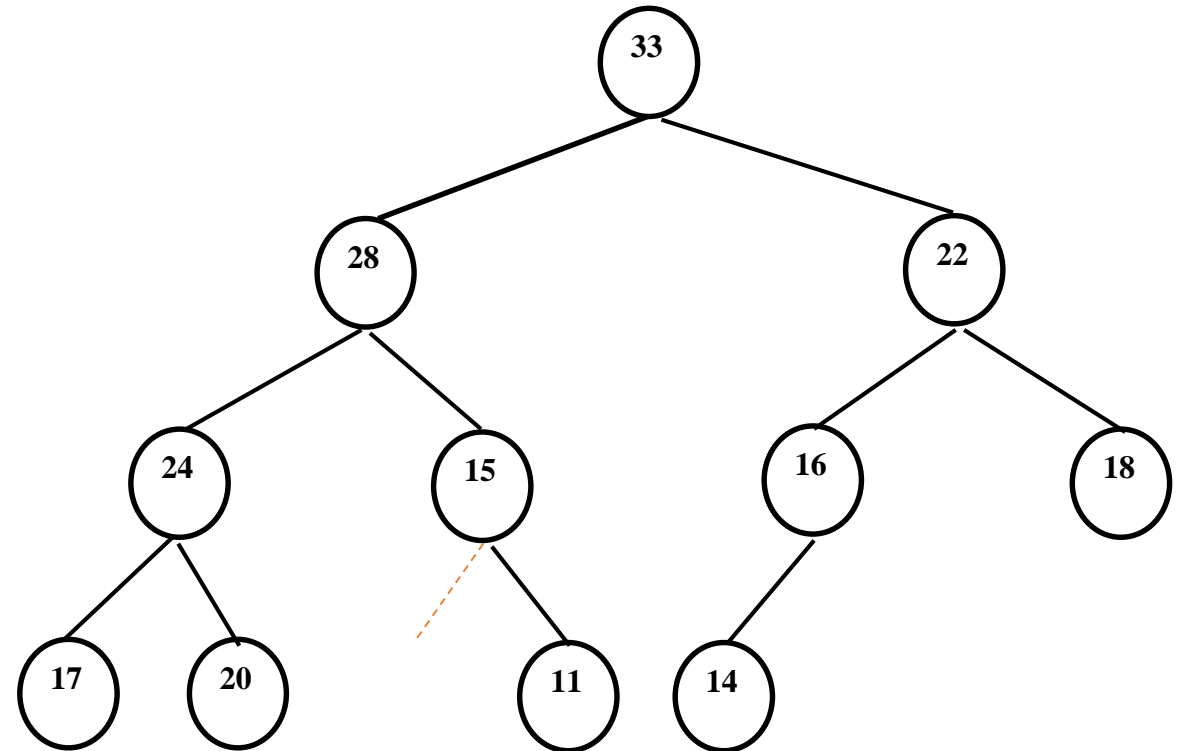
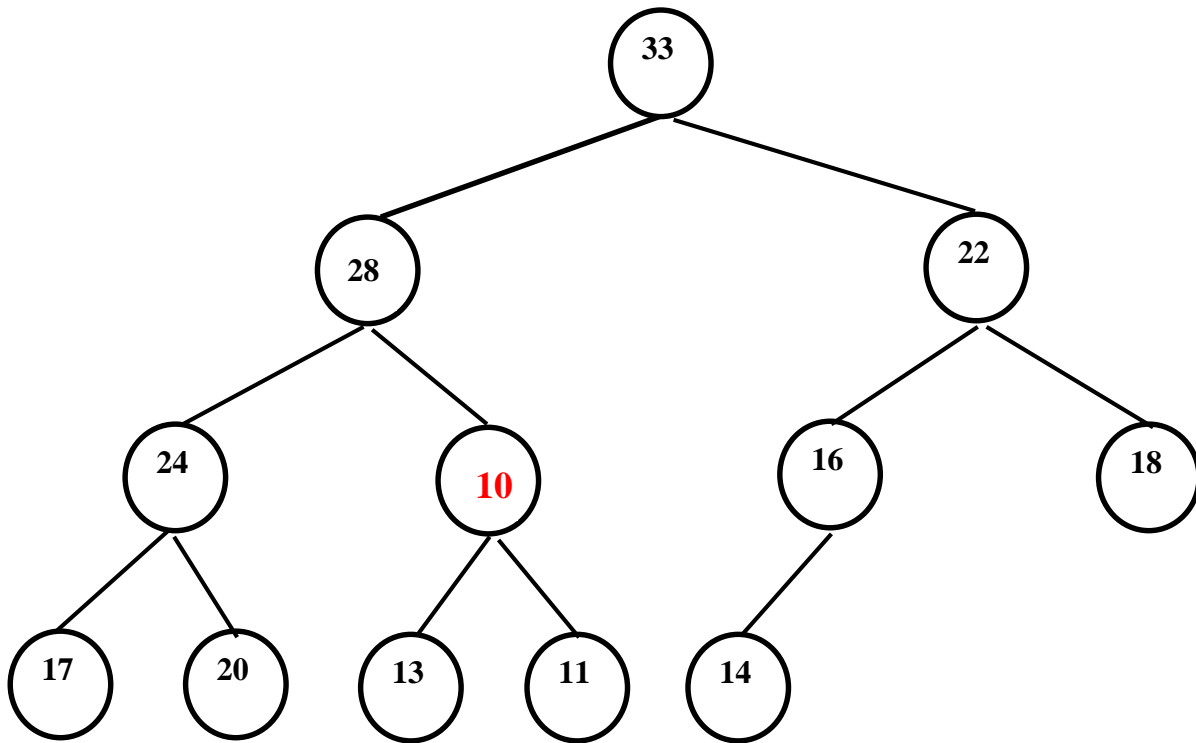
MinHeap



Heap

Exemple

~~MaxHeap~~



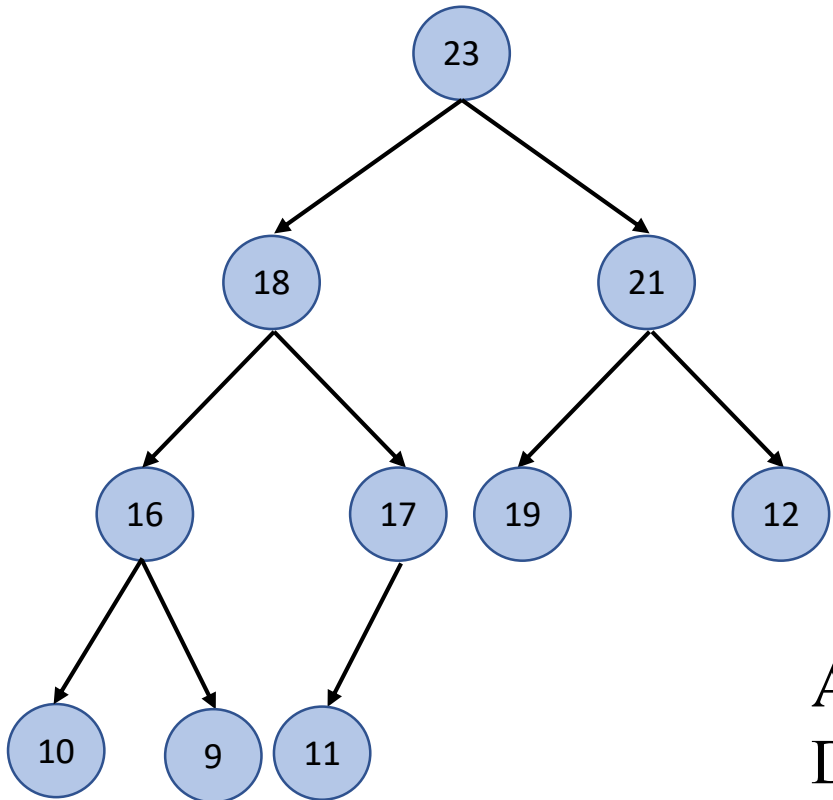
Heap

Observatii

- In continuare ne referim doar la MaxHeap
- Implementare
 - **Dinamică**: avantajul flexibilității și a posibilității de a crește sau micșora dimensiunea arborelui oricât de mult, cu un efort minim
 - **Statică**: reprezentare compactă a arborelui în memorie (de ex. stocarea într-un fișier), deoarece pointerii nu sunt valizi decât în cadrul programului curent.

Heap

Implementarea statică



Index i	0	1	2	3	4	5	6	7	8	9	10
$V[i]$	x	23	18	21	16	17	19	12	10	9	11

Rădăcina are indicele 1 (este primul element din vector).

Pentru nodul din poziția k :

- Fiul stânga în poziția $2*k$
- Fiul dreapta în poziția $2*k + 1$
- Părintele în poziția $[k/2]$ (partea întreagă)

Arborele este reprezentat ”fără goluri” în vector

Dacă sunt k niveluri complete și N noduri în total, atunci

- pe ultimul nivel sunt $N-(2^k-1)$ noduri
- pe ultimul nivel sunt $(2^{k+1}-1)-N$ locuri neocupate

Heap

Operații

- Inserare
- Extragerea elementului de valoare maximă (MaxHeap), respectiv minimă (MinHeap)

Aplicații

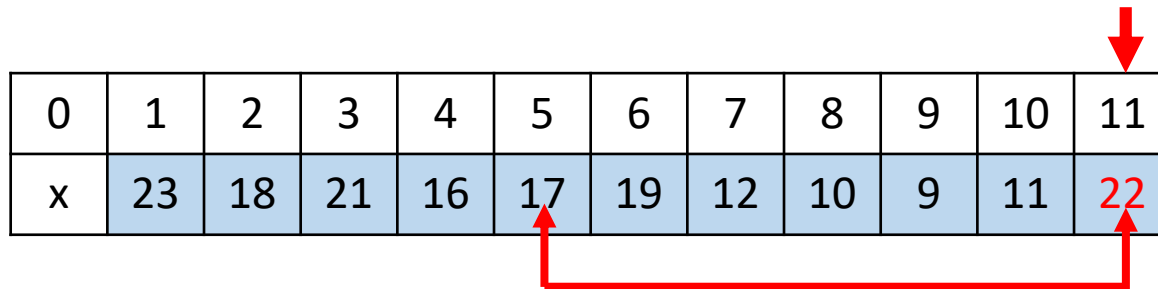
- Implementare eficientă a tipului "Coadă cu priorități" (Priority Queue)
- Implementarea algoritmului de sortare HeapSort

Heap

Inserarea unui nod

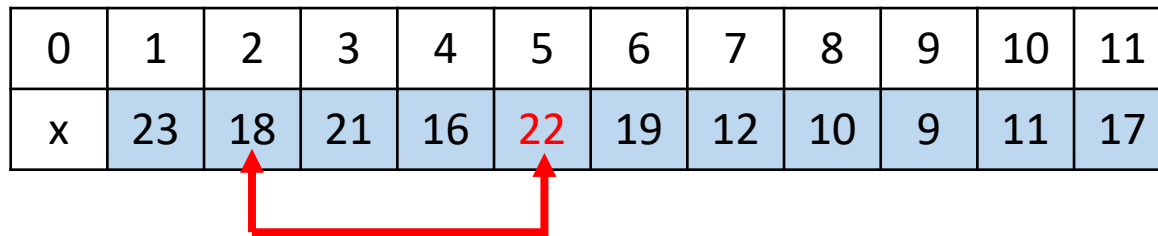
- Se adaugă nodul la nivelul incomplet (la sfârșitul vectorului)
- Se reorganizează structura a.î. să se păstreze proprietatea heap

In arborele anterior se inserează cheia **22** pe poziția **11**



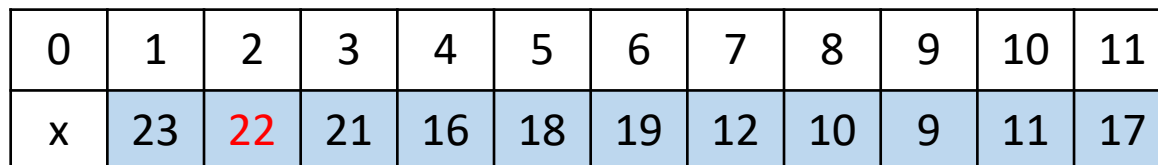
0	1	2	3	4	5	6	7	8	9	10	11
x	23	18	21	16	17	19	12	10	9	11	22

Se face schimb de poziții între nodul de cheie **22** și parintele său care este pe poziția **5** și are cheia **17**



0	1	2	3	4	5	6	7	8	9	10	11
x	23	18	21	16	22	19	12	10	9	11	17

Se face schimb de poziții între nodul de cheie **22** de pe poziția **5** și parintele său care este pe poziția **2** și are cheia **18**

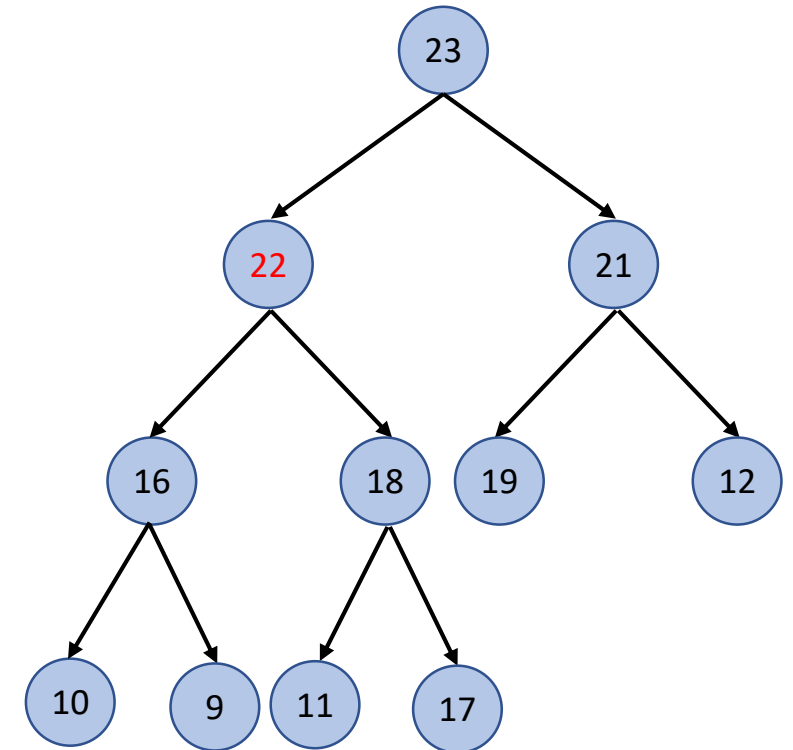
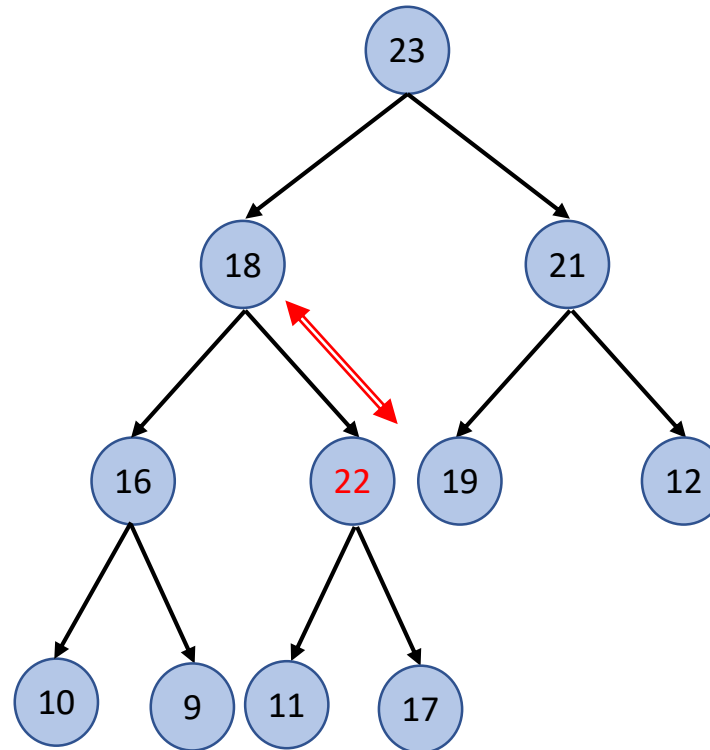
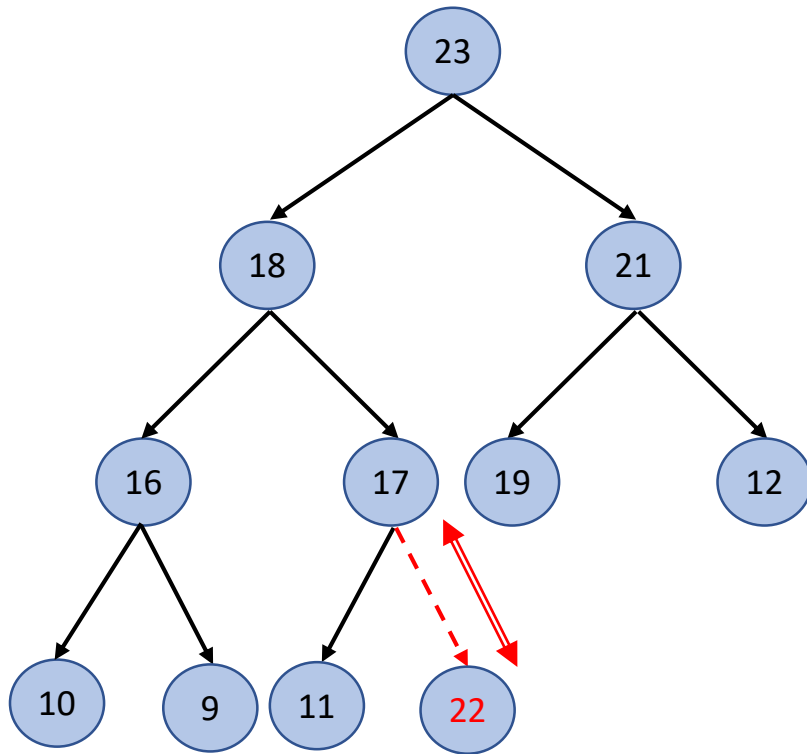


0	1	2	3	4	5	6	7	8	9	10	11
x	23	22	21	16	18	19	12	10	9	11	17

Proprietatea MaxHeap este îndeplinită

Heap

Inserarea



Are loc o parcurgere de la frunze spre rădăcină și se realizează rocada între fiu și părinte până se obține proprietatea heap.

Heap

Inserare

N – dimensiunea vectorului V in care se insereaza

a – nodul care se insereaza

```
Insert(V,N,a)
```

```
    N := N+1;
```

```
    V[N] := a;
```

```
    child:=N;
```

```
    parent:=[N/2];
```

```
    while( parent>=1) Do
```

```
        if(key(V[child]>key(V[parent])) Then
```

```
            swap(V[child],V[parent]);
```

```
            child:=parent;
```

```
            parent:=child/2;
```

```
        else parent:=0;
```

```
        endif
```

```
    endwhile
```

```
end
```

Arborele este de înălțime minimă!

Complexitate: $O(h) \equiv O(\log(n))$

Heap

Extragerea nodului de cheie maximă

0	1	2	3	4	5	6	7	8	9	10	11
x	23	22	21	16	18	19	12	10	9	11	17

0	1	2	3	4	5	6	7	8	9	10
x	17	22	21	16	18	19	12	10	9	11

0	1	2	3	4	5	6	7	8	9	10
x	22	17	21	16	18	19	12	10	9	11

0	1	2	3	4	5	6	7	8	9	10
x	22	18	21	16	17	19	12	10	9	11

- Dintr-un **MaxHeap** se extrage întotdeauna doar **nodul de cheie maximă**, care este pe poziția 1!
- În locul său se aduce nodul de pe ultima poziție

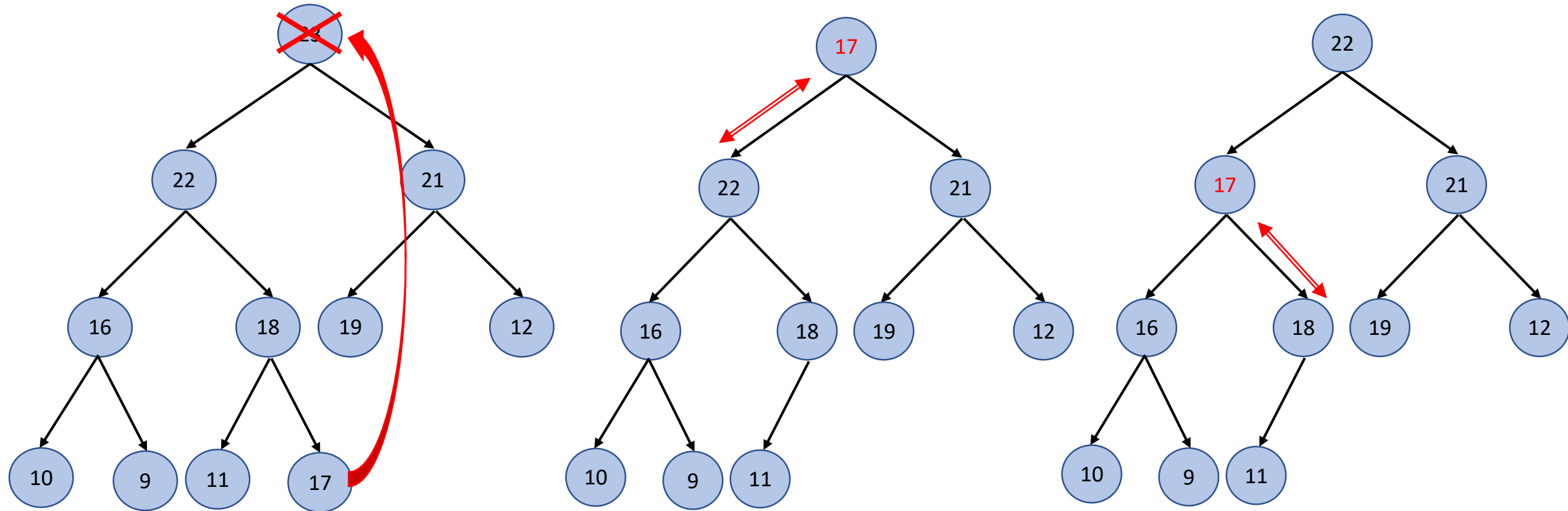
swap ($V[1], V[2]$), deoarece $V[2] > V[3]$

swap ($V[2], V[5]$), deoarece $V[5] > V[4]$

S-a refăcut structura Heap

Heap

Extragerea nodului de cheie maximă



Arborele este parcurs de la rădăcină spre frunze, până se reface structura Heap.

Heap

Extragerea nodului de cheie maximă

```
Remove(V, N)
  a := V[1]
  V[1]:=V[N]
  parent:=1; child:=2; N:=N-1;
  while(child<=N) do
    if(child+1 <= N and key(V[child+1])>key(V[child]))
      child := child+1
    endif
    if(key(V[parent])<key(V[child]))
      swap(V[child],V[parent])
      parent := child
      child:=2*parent
    elseif
      child:=N+1
    endif
  endwhile
  return a
end
```

Complexitate: $O(h) \equiv O(\log(N))$

HeapSort

Realizează sortarea în doi pași:

1. Dacă secvența de sortat se află într-un vector, vectorul se transformă în Heap
2. Se golește Heap-ul prin extrageri succesive iar nodul extras se inserează în Heap în locurile rămase libere

Observații:

- Sortarea se realizează ”in place” (fără memorie suplimentară)
- HeapSort “împacă” viteza cu consumul mic de memorie
- Este nerecursiv
- Algoritmii recursivi rulează rapid, dar consumă o mare cantitate de memorie, ceea ce nu le permite să sorteze tablouri de dimensiuni oricât de mari

HeapSort

Generarea Heap-ului

Prima metodă

V – vector cu N elemente

Hepify1(V,N)

 for i=2 to N do

 Insert(V,i-1,V[i])

 endfor

end

Pentru a adăuga în Heap elementul de pe poziția **i**, se presupune că toate nodurile de la **1** la **i-1** formează un Heap.

Vectorul se parcurge de la stânga la dreapta care corespunde unei deplasări în arbore de sus în jos

Complexitate: **$O(n \cdot \log(n))$**

HeapSort

Generarea Heap-ului

Exemplu: se aplică Heapify1 vectorului urmator

2,4,6,8,5,7,9,11,10

Algoritmul începe cu inserarea lui 4 în heap-ul cu nodul 2

Insert 4:	4,2, 6 ,8,5,7,9,11,10
Insert 6:	6,2,4, 8 ,5,7,9,11,10
Insert 8:	8,6,4,2, 5 ,7,9,11,10
Insert 5:	8,6,4,2,5, 7 ,9,11,10
Insert 7:	8,6,7,2,5,4, 9 ,11,10
Insert 9:	9,6,8,2,5,4,7, 11 ,10
Insert 11:	11,9,8,6,5,4,7,2, 10
Insert 10:	11,10,8,9,5,4,7,2,6

HeapSort

Generarea Heap-ului

Metoda a doua: se consideră că toate nodurile de la $N/2$ la N respectă condiția de Heap (sunt frunze)

```

Heapify2(V,N)
  for i=N/2 to 1 do
    Retrogradare(V,N,i)
  endfor
end

Retrogradare(V, N, i)
  parent:=i; child:=2*i;
  while(child<=N) do
    if(child+1 <= N and key(V[child+1])>key(V[child]))
      child := child+1
    endif
    if(key(V[parent])<key(V[child]))
      swap(V[child],V[parent])
      parent := child
      child:=2*parent
    elseif
      child:=N+1
    endif
  endwhile
end
```

HeapSort

Metoda Heapify2

- Vectorul este parcurs de la dreapta la stânga care corespunde unei parcurgeri a arborelui de jos în sus;
- Cele mai multe noduri sunt frunze (la bază) și acestea parcurg cel mai lung drum în cazul Heapfy1
- Funcția Retrogradare seamănă cu Remove
- Se poate demonstra că ordinul de complexitate pentru Heapify2 este: $O(n)$

HeapSort

Pasul 2: sortarea heap-ului

- Dintr-un MaxHeap nodurile se extrag în ordine descrescătoare și la fiecare extragere dimensiunea vectorului (N) se decrementează.
- Valorile extrase pot fi depuse în vector în locurile libere de la dreapta la stânga.

```
While (N>1) do  
    aux:=Remove(V,N)  
    V[N+1]:=aux  
endwhile
```

Complexitate: $O(n \cdot \log(N))$

HeapSort

Exemplu:

Se sortează crescător MaxHeap-ul 11,10,8,9,7,5,4,2,6 (N=9)

Pas 1	aux:=11, N:=8, V: 10,9,8,6,7,5,4,2,11
Pas 2	aux:=10, N:=7, V: 9,7,8,6,2,5,4,10,11
Pas 3	aux:=9, N:=6, V: 8,7,5,6,2,4,9,10,11
Pas 4	aux:=8, N:=5, V: 7,6,5,4,2,8,9,10,11

Coadă cu priorități

Coadă a carei atomi conține un câmp **cheie**

Cheia se numește **prioritate**

Se implementează cu MaxHeap-uri/MinHeap-uri

Operații

Extragerea unui atom (Remove \equiv Get) – se extrage întotdeauna atomul de prioritate maximă (MaxHeap) sau minimă (MinHeap) aflat pe prima poziție

Inserarea unui atom (Insert \leftrightarrow Put) – inserarea se realizează pe o poziție care să asigure proprietatea Heap din punct de vedere a priorității

Modificarea priorității unui atom – presupune localizarea atomului și reorganizarea cozii a.î. să se asigure proprietatea Heap

Obs: o astfel coadă nu mai este FIFO în sensul discutat anterior

Coada cu priorități

Aplicații

- CPU Scheduling: fiecărei surse de intrerupere i se asociază un nivel de prioritate de la 0 la n (MinHeap);
- Algoritmul Dijkstra pentru determinarea drumurilor minime de la un nod sursă la toate celelalte noduri dintr-un graf;
- Algoritmul lui Prim;
- Algoritmul A^* ;
- Compresia datelor care utilizeaza coduri Huffman;
- Problema rutării unui mesaj într-o rețea complex de comunicații.

Exerciții

1. Se considera un MaxHeap cu 21 de noduri. Câte locuri libere și câte noduri sunt pe ultimul nivel?
2. Se consideră următorul MinHeap:

2, 5, 7, 9, 7, 11, 9, 10, 12, 11

Se extrag doi atomi. Care va fi rădăcina arborelui? Reprezentați arborele rezultat.