

Coadă cu priorități.

SD 2020/2021

Cooda cu priorități și “max-heap”

Colecții de mulțimi disjuncte și “union-find”

Coadă cu priorități – exemple

► Pasagerii unui avion

Priorități:

- clasa “business”;
- persoane călătorind cu copii / cu mobilitate redusă;
- ceilalți pasageri.

Coadă cu priorități – exemple

► Pasagerii unui avion

Priorități:

- clasa “business”;
- persoane călătorind cu copii / cu mobilitate redusă;
- ceilalți pasageri.

► Avioane care se pregătesc să aterizeze

Priorități

- urgențe;
- nivelul carburantului;
- distanța față de aeroport.

Tipul abstract Coadă cu priorități

► OBIECTE:

- structuri de date în care elementele sunt numite **atomi**;
- orice atom are un câmp *cheie* numit **prioritate**.

- Elementele sunt memorate în funcție de prioritate și nu de poziția lor.

Coadă cu priorități – operații

► citește

- intrare: o coadă cu priorități C
- ieșire: atomul din C cu cheia cea mai mare.

► elimină

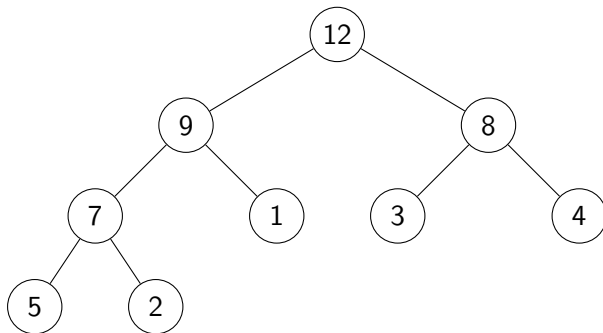
- intrare: o coadă cu priorități C
- ieșire: C din care s-a eliminat atomul cu cheia cea mai mare.

► inserează

- intrare: o coadă cu priorități C și un atom at
- ieșire: C la care s-a adăugat at .

- ▶ Implementează coada cu priorități.
- ▶ Arbori binari cu proprietățile:
 - Nodurile memorează câmpurile *cheie*;
 - Pentru orice nod, cheia din acel nod este mai mare decât sau egală cu cheile din nodurile fiu;
 - Arborele este complet. Fie h înălțimea arborelui. Atunci,
 - Pentru $i = 0, \dots, h - 1$, sunt 2^i noduri cu adâncimea i ;
 - Pe nivelul $h - 1$ nodurile interne sunt situate la stânga nodurilor externe.
 - Ultimul nod al unui maxHeap este nodul cel mai la dreapta pe nivelul h .

maxHeap – exemplu



Înălțimea unui maxHeap

Teoremă

Un maxHeap care conține n chei are înălțimea $O(\log_2 n)$.

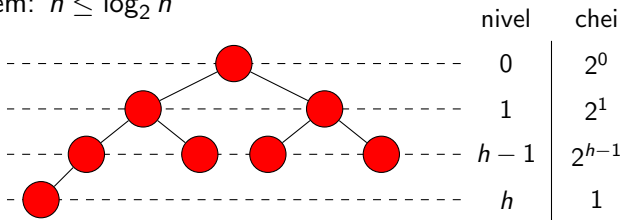
Înălțimea unui maxHeap

Teoremă

Un maxHeap care conține n chei are înălțimea $O(\log_2 n)$.

Demonstrație.

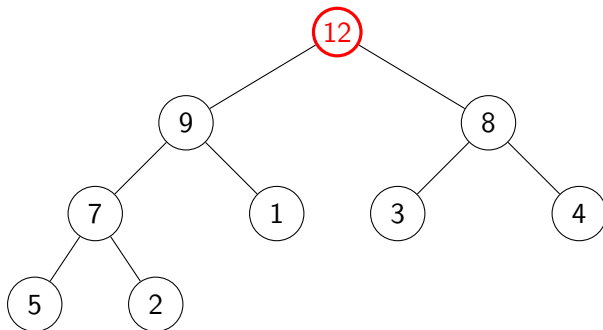
- Utilizăm proprietatea de arbore binar complet.
- Fie h înălțimea unui maxHeap cu n chei.
- Avem 2^i chei de adâncime i , pentru $i = 0, \dots, h-1$ și cel puțin o cheie de adâncime h : $\Rightarrow n \geq 2^0 + 2^1 + 2^2 + \dots + 2^{h-1} + 1 = 2^h$.
- Obținem: $h \leq \log_2 n$



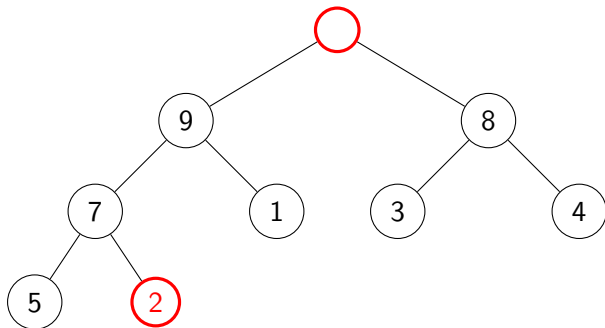
maxHeap: eliminarea

- ▶ Se elimină rădăcina heap-ului (corespunde elementului cel mai prioritar).
- ▶ Algoritmul are trei etape:
 - ▶ Se înlocuiește cheia rădăcinii cu cheia ultimului nod;
 - ▶ Se șterge ultimul nod (de pe ultimul nivel);
 - ▶ Se reface proprietatea de maxHeap.

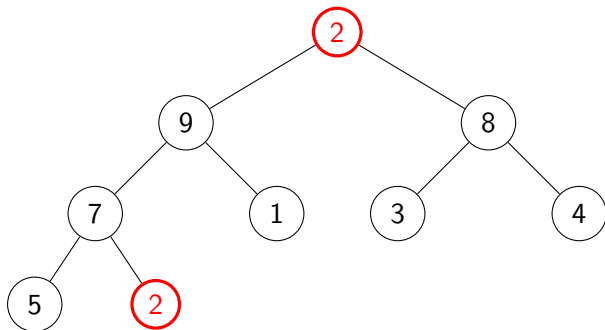
maxHeap: eliminare – exemplu



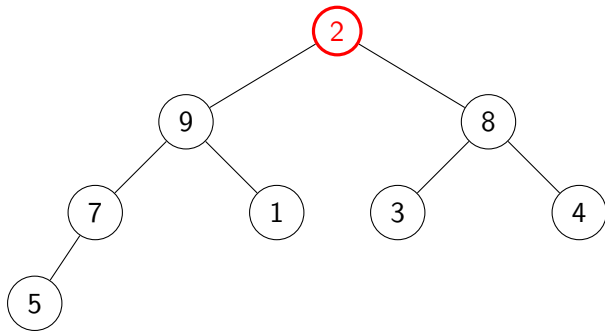
maxHeap: eliminare – exemplu



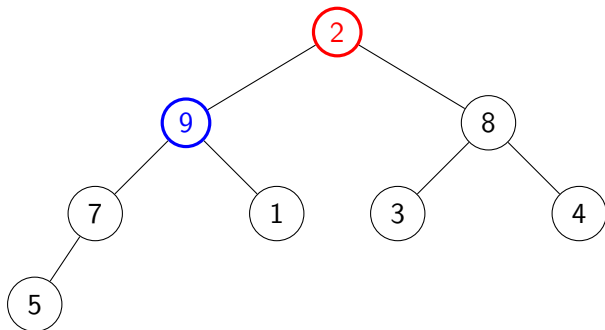
maxHeap: eliminare – exemplu



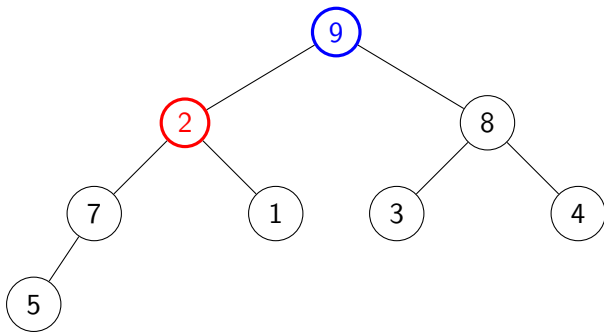
maxHeap: eliminare – exemplu



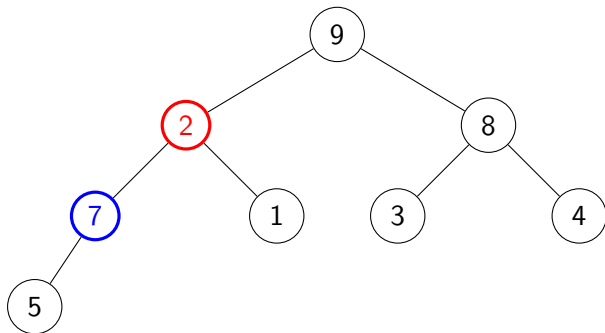
maxHeap: eliminare – exemplu



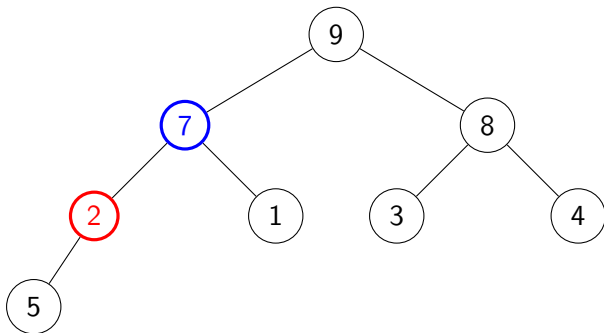
maxHeap: eliminare – exemplu



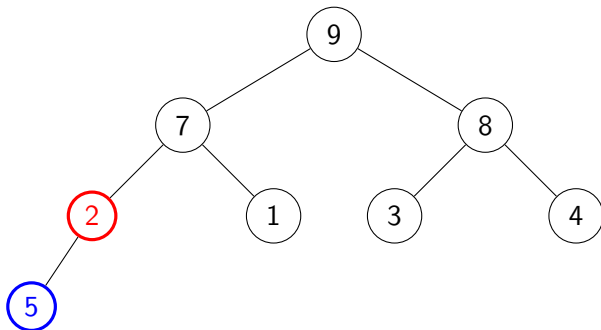
maxHeap: eliminare – exemplu



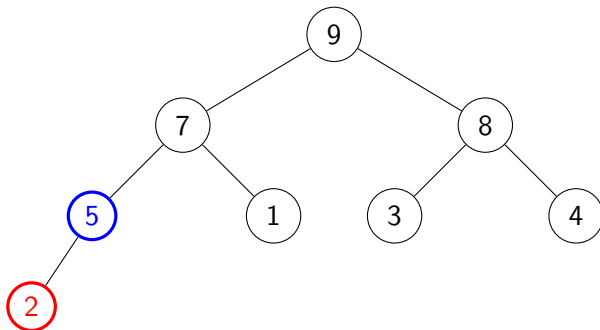
maxHeap: eliminare – exemplu



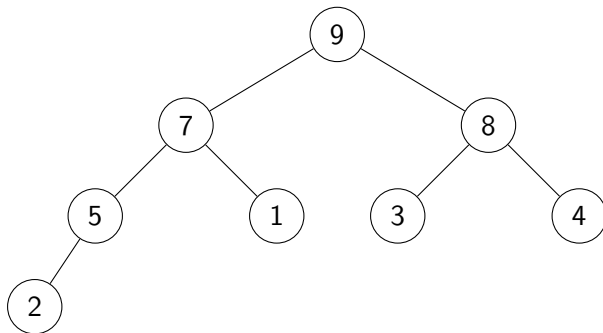
maxHeap: eliminare – exemplu



maxHeap: eliminare – exemplu

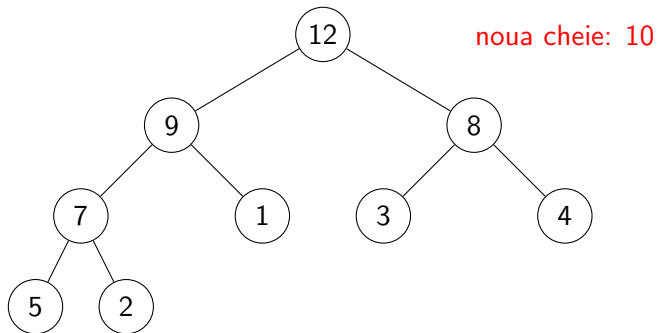


maxHeap: eliminare – exemplu

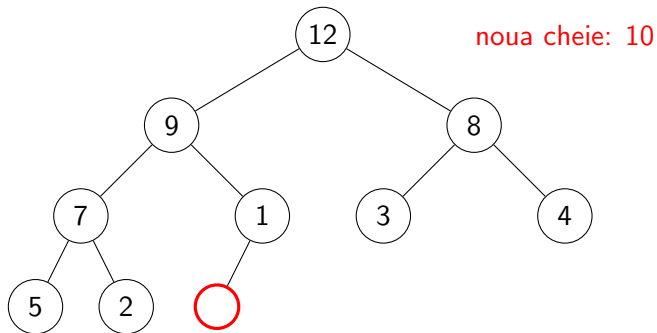


- ▶ Se inserează noua cheie într-un nou nod.
- ▶ Algoritmul are trei etape:
 - ▶ Se adaugă noul nod ca cel mai din dreapta pe ultimul nivel;
 - ▶ Se inserează noua cheie în acest nod;
 - ▶ Se reface proprietatea de maxHeap.

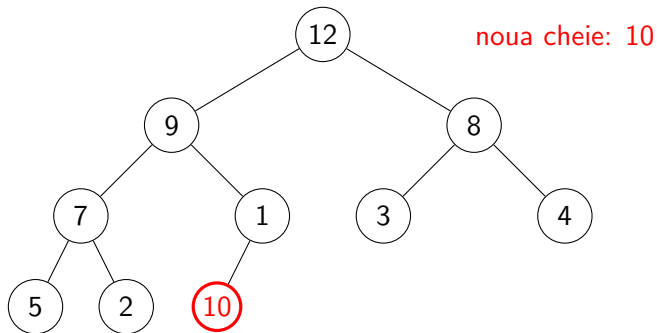
maxHeap: inserare – exemplu



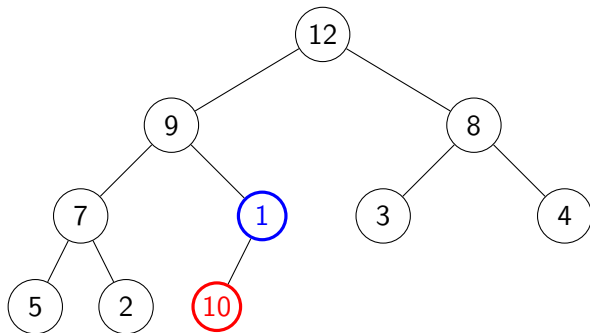
maxHeap: inserare – exemplu



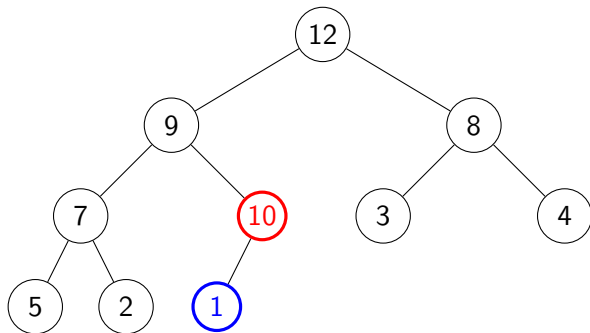
maxHeap: inserare – exemplu



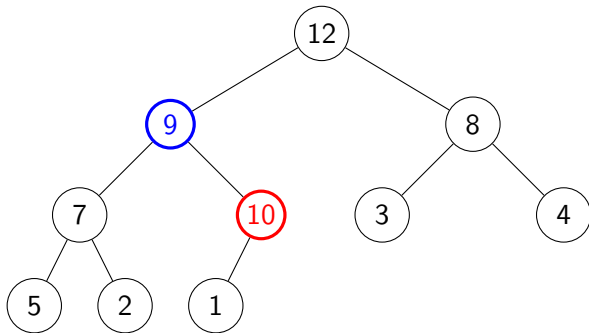
maxHeap: inserare – exemplu



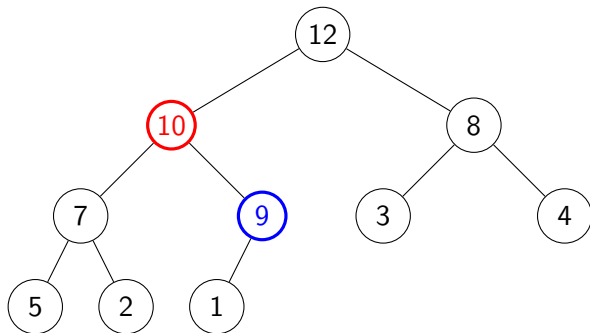
maxHeap: inserare – exemplu



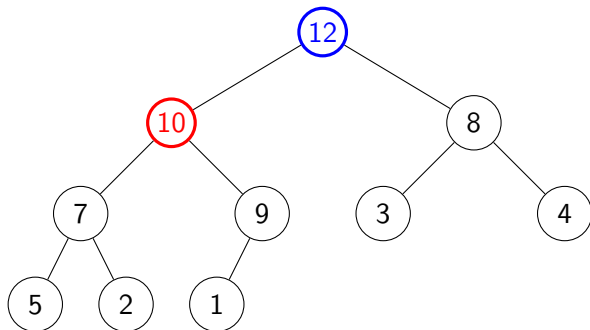
maxHeap: inserire – esemplu



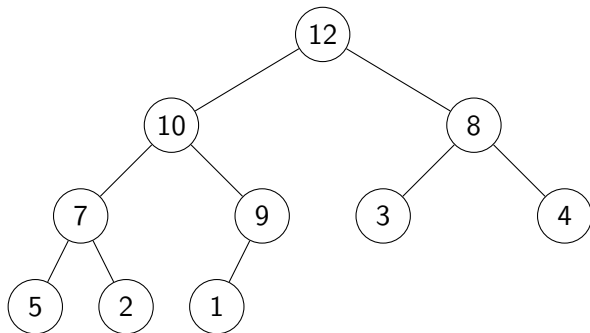
maxHeap: inserare – exemplu



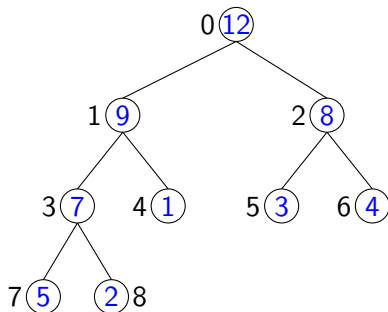
maxHeap: inserare – exemplu



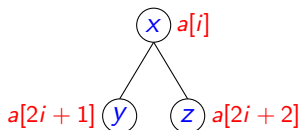
maxHeap: inserare – exemplu



maxHeap: implementarea cu tablouri



12	9	8	7	1	3	4	5	2
0	1	2	3	4	5	6	7	8



$$\forall k : 1 \leq k \leq n-1 \Rightarrow a[k] \leq a[(k-1)/2]$$

maxHeap: inserare

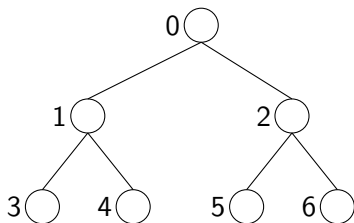
```
procedure insereaza(a, n, cheie)
begin
    n  $\leftarrow$  n+1
    a[n-1]  $\leftarrow$  cheie
    j  $\leftarrow$  n-1
    heap  $\leftarrow$  false
    while (j > 0 and not heap) do
        k  $\leftarrow$  [(j-1)/2]
        if (a[j] > a[k]) then
            swap(a[j], a[k])
            j  $\leftarrow$  k
        else
            heap  $\leftarrow$  true
end
```

maxHeap: elimina

```
procedure elimina(a, n)
begin
  a[0]  $\leftarrow$  a[n-1]
  n  $\leftarrow$  n-1
  j  $\leftarrow$  0
  heap  $\leftarrow$  false
  while ( $2*j+1 < n$  and not heap) do
    k  $\leftarrow$   $2*j+1$ 
    if ( $k < n-1$  and  $a[k] < a[k+1]$ ) then
      k  $\leftarrow$  k+1
    if ( $a[j] < a[k]$ ) then
      swap(a[j], a[k])
      j  $\leftarrow$  k
    else
      heap  $\leftarrow$  true
end
```

- ▶ Operațiile de inserare / eliminare au clasa de complexitate

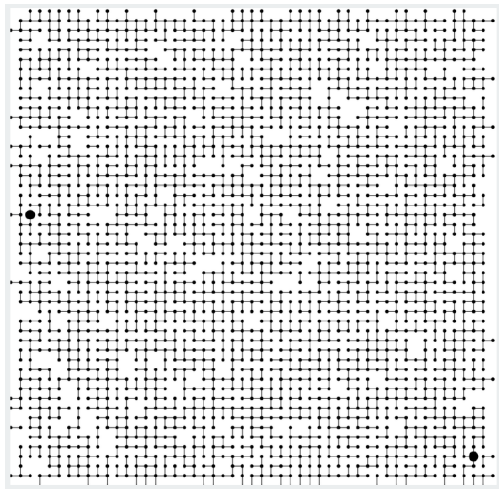
$$O(h) = O(\log n)$$



Coadă cu priorități și “max-heap”

Colecții de mulțimi disjuncte și “union-find”

Colecții de mulțimi disjuncte



Aplicații:

- ▶ Rețele de calculatoare
- ▶ Pagini web (Internet)
- ▶ Pixeli într-o imagine digitală

Tipul abstract Colecții de mulțimi disjuncte

► OBIECTE:

Colecții de submulțimi disjuncte (partiții) ale unei mulțimi univers.

► OPERAȚII:

► find

- intrare: o colecție C și un element i din mulțimea univers;
- ieșire: submulțimea din C căreia aparține i .

► union

- intrare: o colecție C și două elemente i și j din mulțimea univers;
- ieșire: C în care s-au reunit componentele lui i și j .

► singleton

- intrare: o colecție C și un element i din mulțimea univers;
- ieșire: C la care componenta lui i are pe i ca unic element.

► Structura union-find:

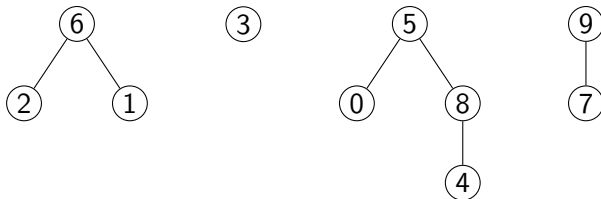
- mulțimea univers este $\{0, 1, \dots, n - 1\}$;
- o submulțime este reprezentată printr-un arbore;
- o colecție (partiție) este o colecție de arbori (“pădure”);
- reprezentarea unei “păduri” se face prin relația “părinte”.

union-find: exemplu

► $n = 10$, $C = \{\{1, 2, 6\}, \{3\}, \{0, 4, 5, 8\}, \{7, 9\}\}$

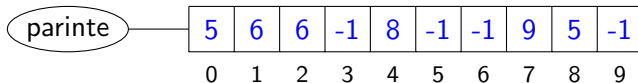
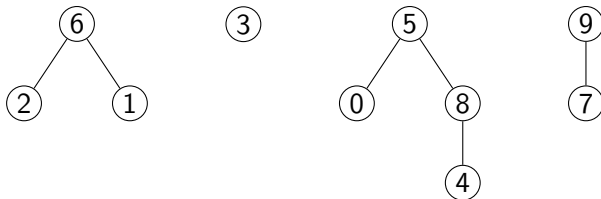
union-find: exemplu

► $n = 10$, $C = \{\{1, 2, 6\}, \{3\}, \{0, 4, 5, 8\}, \{7, 9\}\}$



union-find: exemplu

► $n = 10$, $C = \{\{1, 2, 6\}, \{3\}, \{0, 4, 5, 8\}, \{7, 9\}\}$



union-find: singleton

```
procedure singleton(C, i)
begin
    C.parinte[i]  $\leftarrow$  -1
end
```

union-find: find

```
procedure find(C, i)
begin
    temp  $\leftarrow$  i
    while (C.parinte[temp]  $\geq$  0) do
        temp  $\leftarrow$  C.parinte[temp]
    return temp
end
```

union-find: union

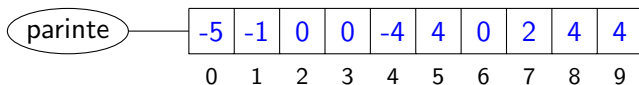
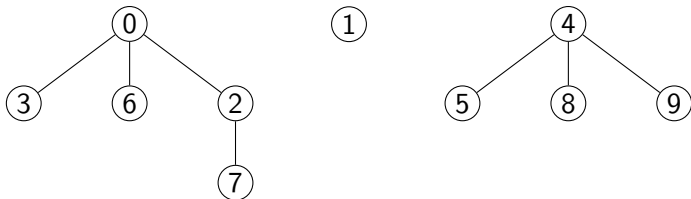
```
procedure union(C, i, j)
begin
     $r_i \leftarrow \text{find}(i)$ 
     $r_j \leftarrow \text{find}(j)$ 
    if  $r_i \neq r_j$  then
         $C.\text{parinte}[r_j] \leftarrow r_i$ 
end
```

Structură union-find ponderată

- ▶ Soluție la problema arborilor dezechilibrați.
- ▶ Mecanism:
 - Memorarea numărului de vârfuri din arbore (cu semn negativ).
 - Aplatizarea arborilor.

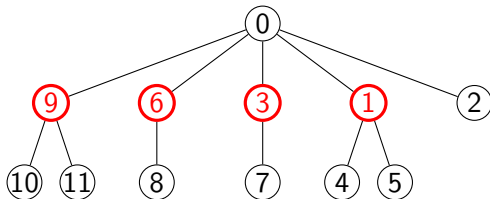
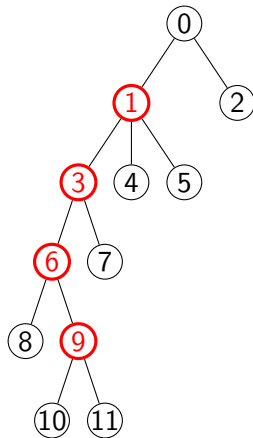
Structură union-find ponderată: exemplu

► $n = 10$, $C = \{\{0, 2, 3, 6, 7\}, \{1\}, \{4, 5, 8, 9\}\}$



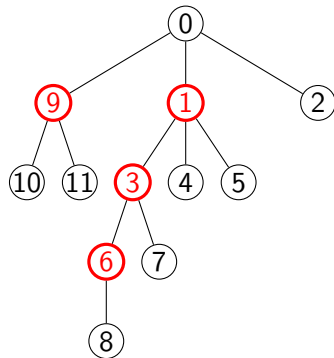
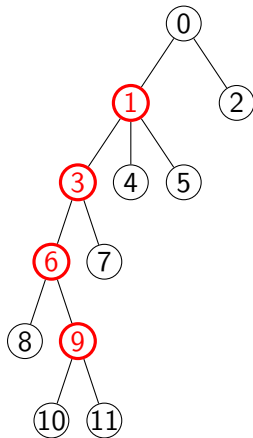
Aplatizarea arborilor – exemplu

► find(9)



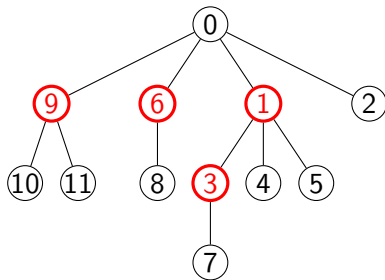
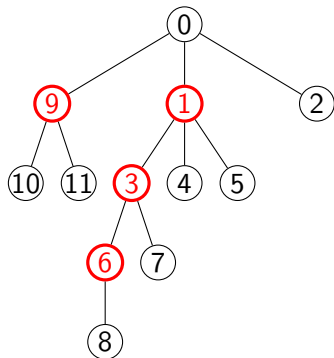
Aplatizarea arborilor – exemplu

► find(9)



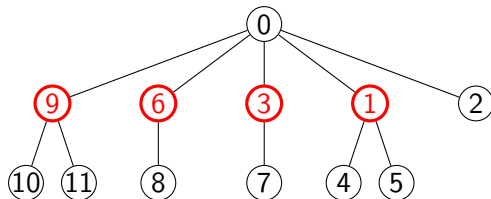
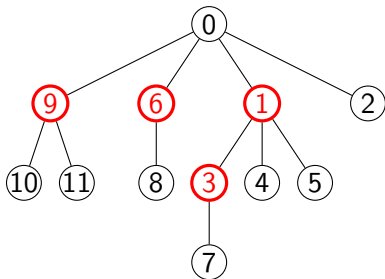
Aplatizarea arborilor – exemplu

► find(9)



Aplatizarea arborilor – exemplu

► find(9)



Structură union-find ponderată

```
procedure union(C, i, j)
begin
    ri  $\leftarrow$  find(i);    rj  $\leftarrow$  find(j)

    while (C.parinte[i]  $\geq$  0) do
        temp  $\leftarrow$  i;    i  $\leftarrow$  C.parinte[i];    C.parinte[temp]  $\leftarrow$  ri
    while (C.parinte[j]  $\geq$  0) do
        temp  $\leftarrow$  j;    j  $\leftarrow$  C.parinte[j];    C.parinte[temp]  $\leftarrow$  rj

    if C.parinte[ri] > C.parinte[rj] then
        C.parinte[rj]  $\leftarrow$  C.parinte[ri] + C.parinte[rj]
        C.parinte[ri]  $\leftarrow$  rj
    else
        C.parinte[ri]  $\leftarrow$  C.parinte[ri] + C.parinte[rj]
        C.parinte[rj]  $\leftarrow$  ri
end
```

Structură union-find ponderată

Teoremă

Pornind de la o colecție vidă, orice secvență de m operații union și find asupra n elemente are complexitatea $O(n + m \log^ n)$.*

Observatie: $\log^* n$ este numărul de logaritmări până se obține valoarea 1.