

Structuri de date – Curs 7

Prof. univ. dr. Cristian CIUREA
Departamentul de Informatică și Cibernetică Economică
Academia de Studii Economice din București
cristian.ciurea@ie.ase.ro

Agenda

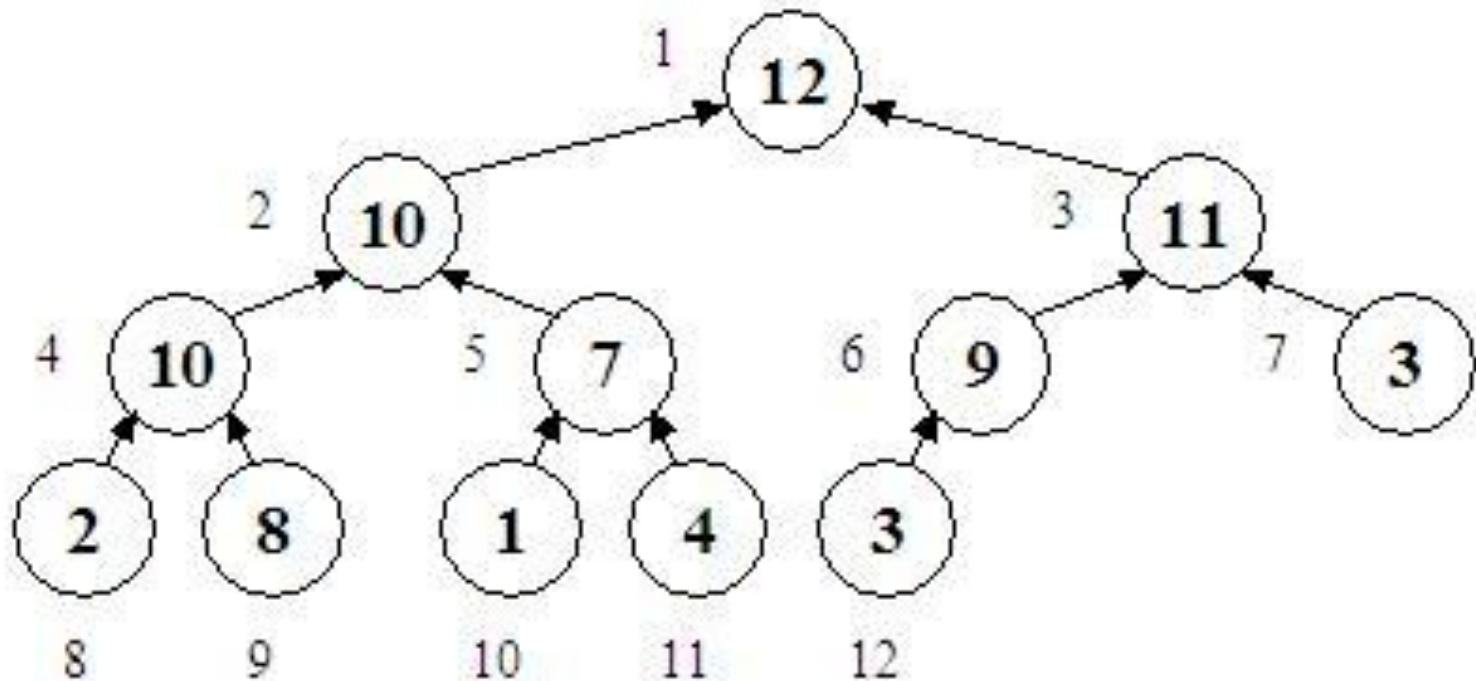
- ▶ Structura de tip *Heap*
- ▶ Cozi de prioritate
- ▶ Sortarea datelor prin *HeapSort*

Heap

- ▶ Structura de tip Heap este un arbore binar care respectă proprietățile de structură și de ordonare:
 - *proprietatea de structură* specifică faptul că elementele sunt organizate sub forma unui arbore binar complet, respectiv un arbore binar în care toate nodurile, cu excepția celor de pe ultimul nivel, au exact doi fii, iar nodurile de pe ultimul nivel sunt completate de la stânga la dreapta.
 - *proprietatea de ordonare* impune ca valoarea asociată fiecărui nod, cu excepția nodului rădăcină, să fie mai mică sau egală decât valoarea asociată nodului părinte. Spre deosebire de arborii binari de căutare, nu se impune nici o regulă referitoare la poziția sau relația dintre nodurile fiu.

Heap

- ▶ $v[] = \{12, 10, 11, 10, 7, 9, 3, 2, 8, 1, 4, 3\}$

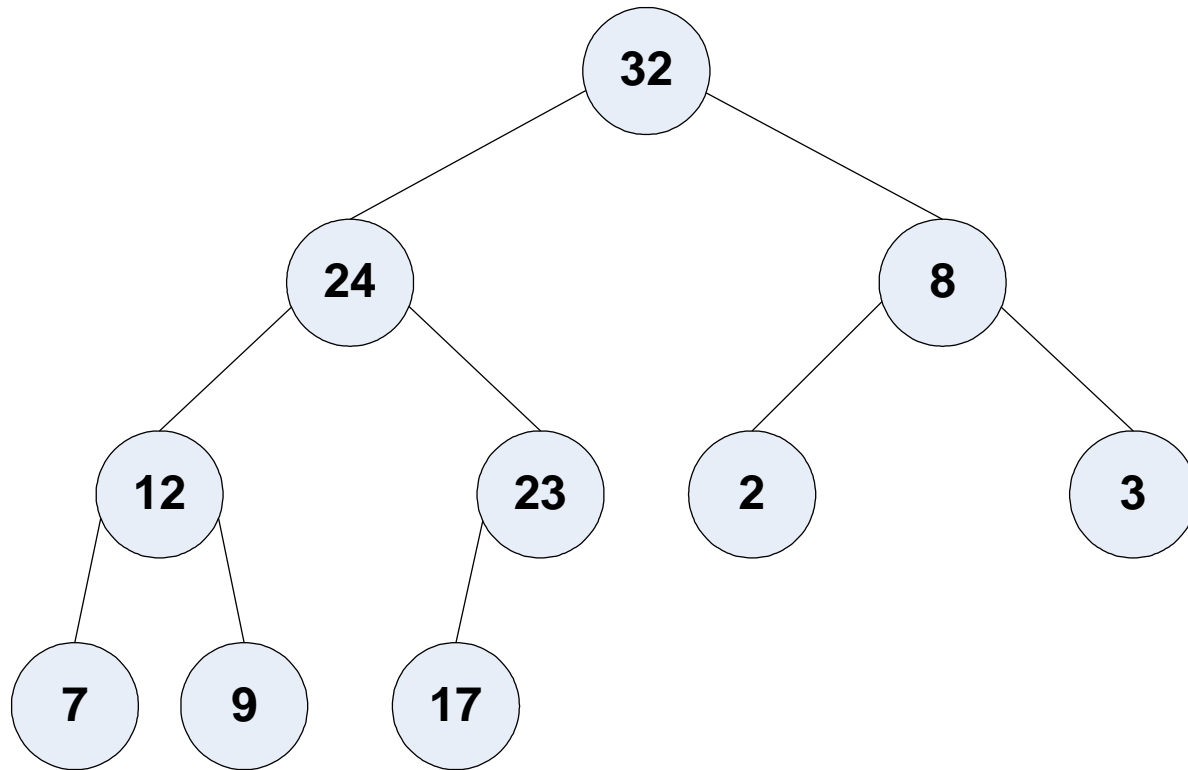


Heap

- ▶ Fiecare nod al arborelui trebuie să conțină o valoare asociată numită cheie și poate conține și alte informații suplimentare.
- ▶ Cheia trebuie să permită definirea unei relații de ordine totală pe mulțimea nodurilor.
- ▶ În funcție de obiectivul urmărit, heap-ul poate fi organizat sub formă de *max-heap* sau *min-heap*.
- ▶ Cele două tipuri de heap sunt echivalente. Transformarea unui *max-heap* în *min-heap*, sau invers, se poate realiza prin simpla inversare a relației de ordine.

Heap

- ▶ Exemplu reprezentarea grafică a structurii heap:



Heap

- ▶ Operațiile principale care se execută pe o structură heap sunt:
 - construirea heap-ului, pornind de la un masiv unidimensional oarecare;
 - inserarea unui element în structură;
 - extragerea elementului maxim sau minim.

Heap

- ▶ *Construirea heap-ului* se face utilizând o procedură ajutătoare numită procedură de filtrare.
- ▶ Rolul acesteia este de a transforma un arbore în care doar subarborii rădăcinii sunt heap-uri, ale căror înălțimi diferă cu cel mult o unitate, într-un heap prin coborârea valorii din rădăcină pe poziția corectă.
- ▶ Structura rezultată în urma aplicării procedurii de filtrare este un heap (respectă proprietățile de structură și ordonare).

Heap

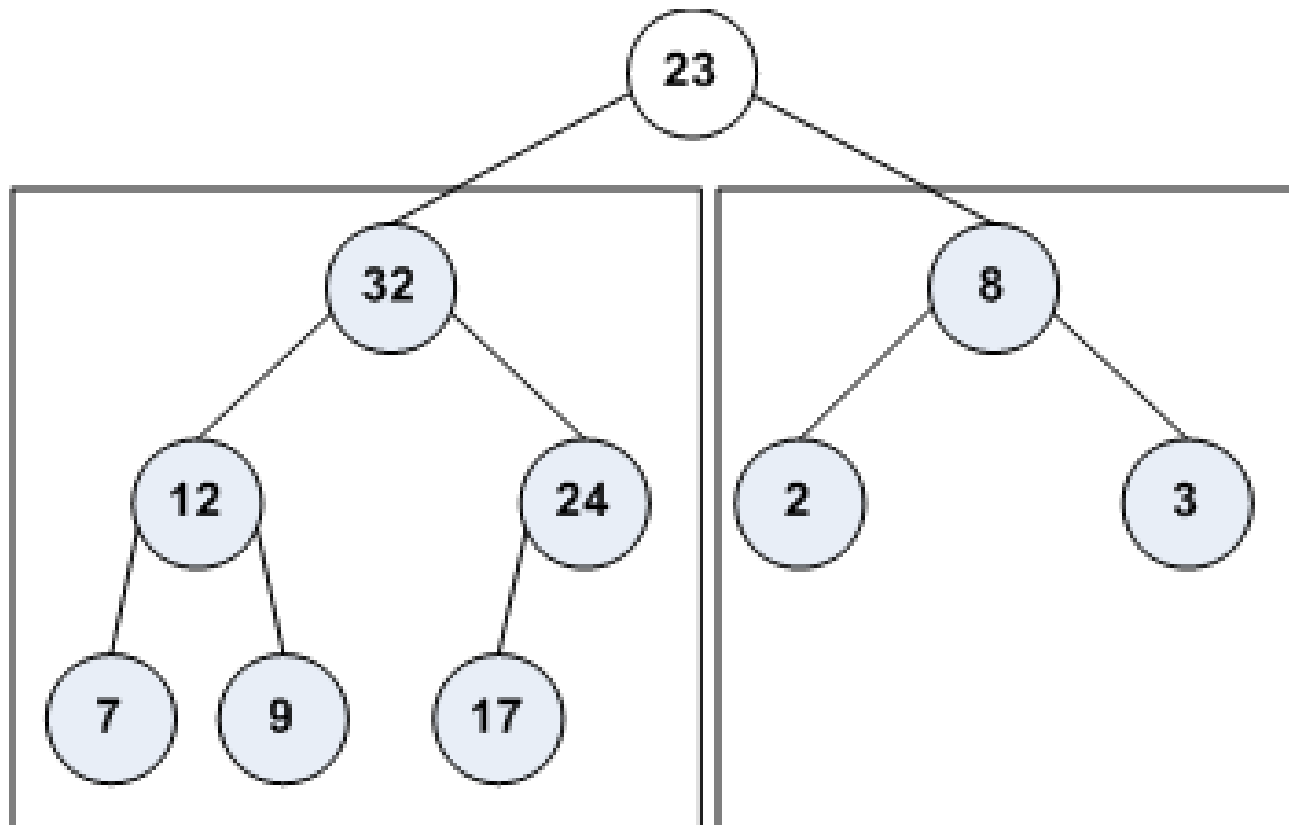
- ▶ Algoritmul de filtrare presupune parcurgerea următoarelor etape, începând cu nodul rădăcină:
 - se determină maximul dintre nodul curent, fiul stânga și fiul dreapta (dacă există);
 - dacă maximul se află în nodul curent, atunci algoritmul se oprește;
 - dacă maximul se află într-unul dintre fii, atunci se interschimbă valoarea din nodul curent cu cea din fiu și se continuă execuția algoritmului cu nodul fiu.

Heap

- ▶ Construirea heap-ului pornind de la un arbore binar, care respectă doar proprietatea de structură, se face aplicând procedura de filtrare pe nodurile non-frunză ale arborelui, începând cu nodurile de la baza arborelui și continuând în sus, până când se ajunge la nodul rădăcină.
- ▶ Corectitudinea algoritmului este garantată de faptul că, la fiecare pas, subarborii nodului curent sunt heap-uri (deoarece sunt noduri frunză sau sunt noduri pe care a fost aplicată procedura de filtrare).

Heap

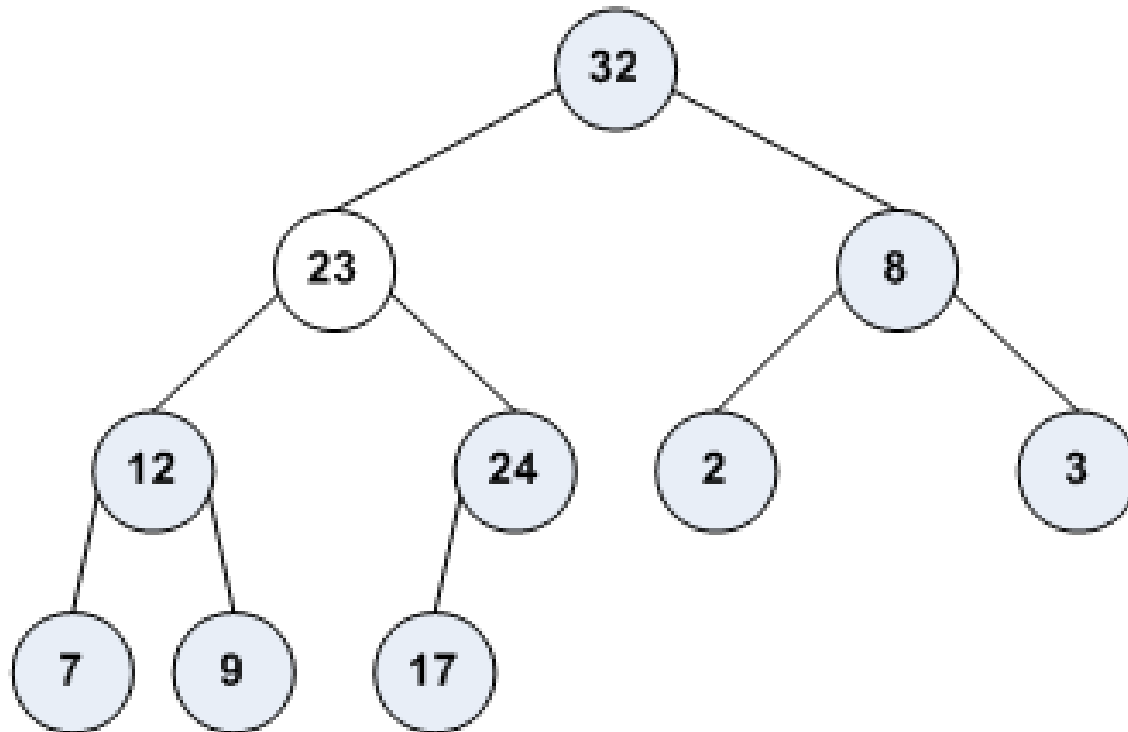
- ▶ Aplicarea algoritmului de filtrare:



a) Situația inițială

Heap

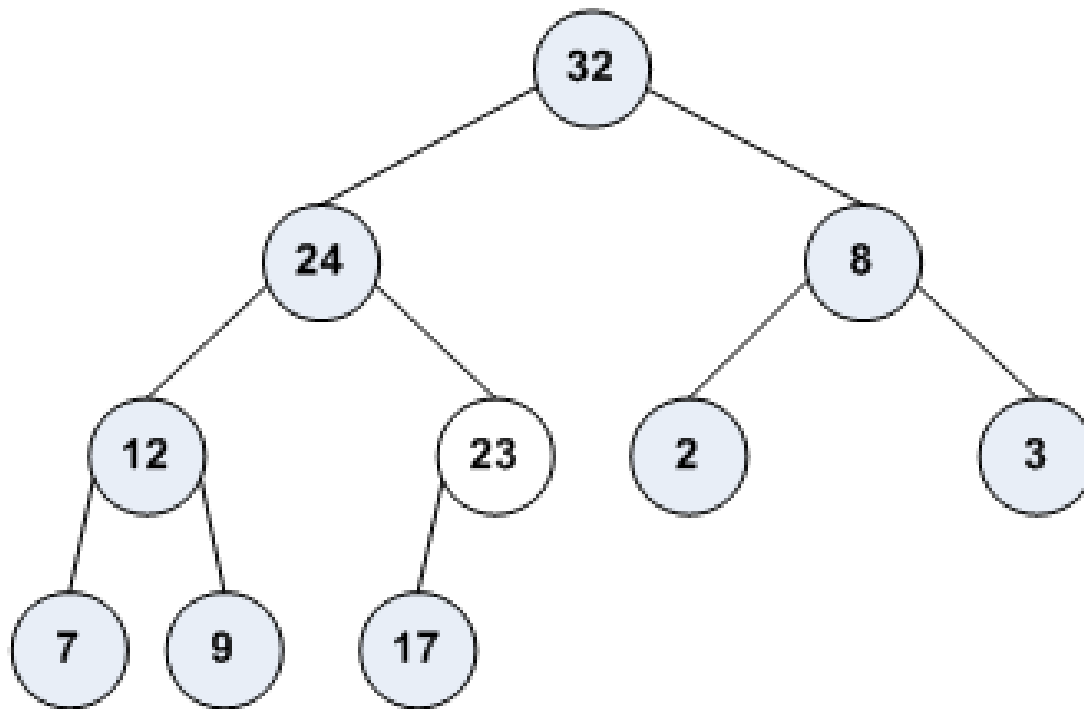
- ▶ Aplicarea algoritmului de filtrare:



b) Arborele după aplicarea primului pas

Heap

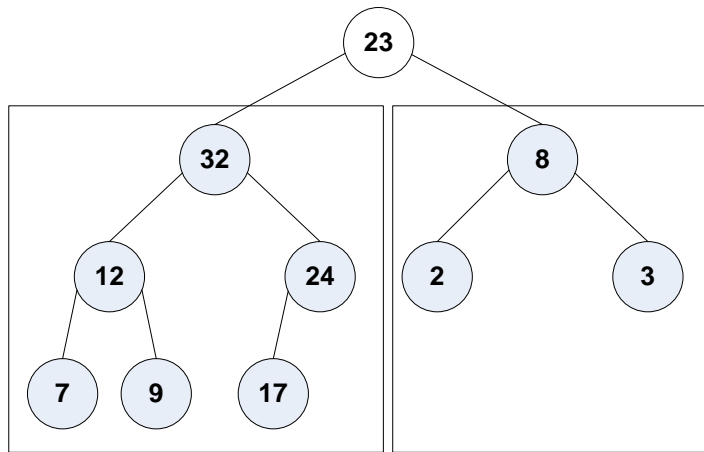
- ▶ Aplicarea algoritmului de filtrare:



c) Arborele la sfârșitul procedurii de filtrare

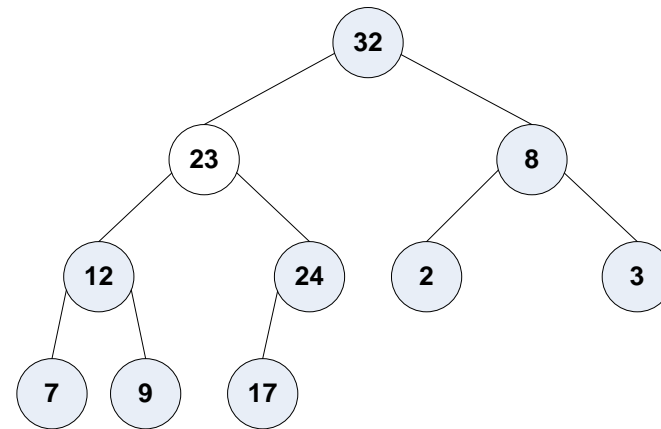
Heap

► Aplicarea algoritmului de filtrare:

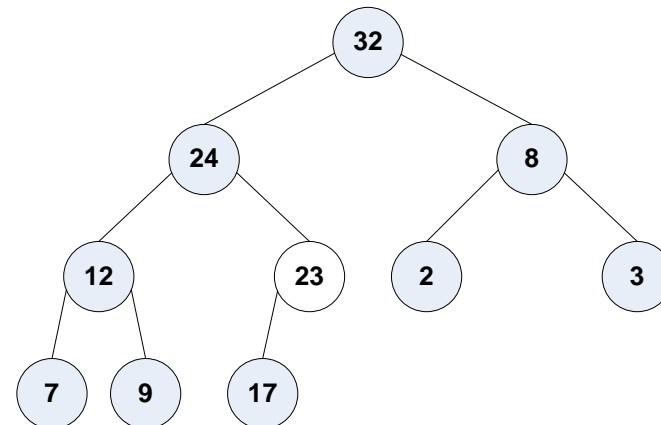


a) Situația inițială

Subarbori organizați sub formă de heap (respectă proprietățile de structură și ordonare)



b) Arborele după aplicarea primului pas



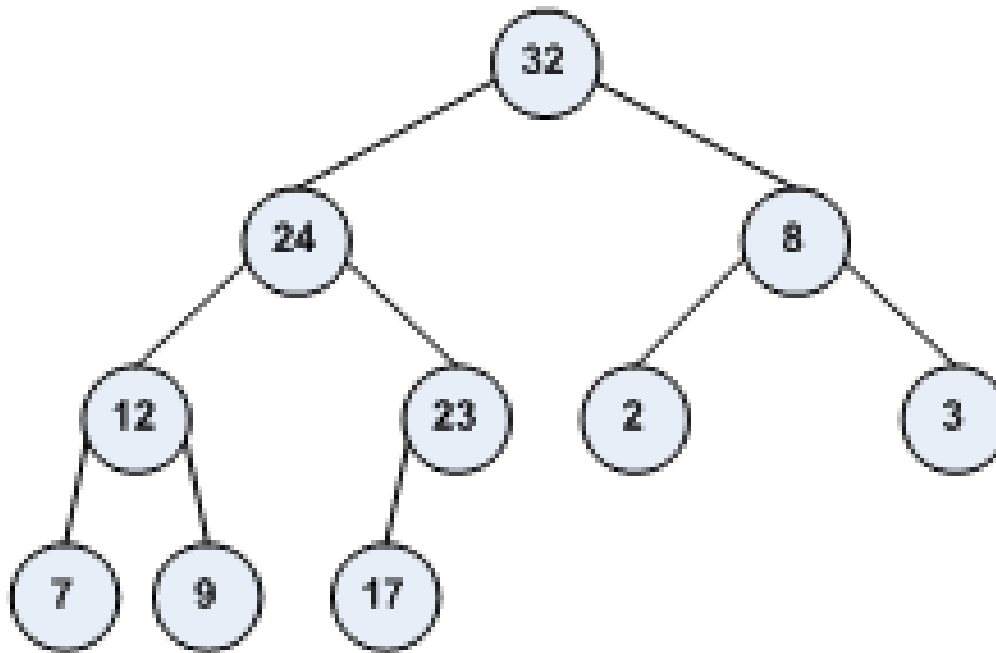
c) Arborele la sfârșitul procedurii de filtrare

Heap

- ▶ *Inserarea elementelor* într-un heap se poate face și după etapa inițială de construcție. Adăugarea elementului nou trebuie făcută astfel încât structura rezultată să păstreze proprietatea de ordonare.
- ▶ Inserarea unui element în heap presupune parcurgerea următoarelor etape:
 - se adaugă elementul ca nod frunză la sfârșitul arborelui pentru a păstra proprietatea de structură;
 - se compară nodul curent cu nodul părinte;
 - dacă nodul părinte este mai mic, se interschimbă nodul curent cu nodul părinte;
 - dacă nodul părinte este mai mare sau egal, atunci algoritmul se oprește.

Heap

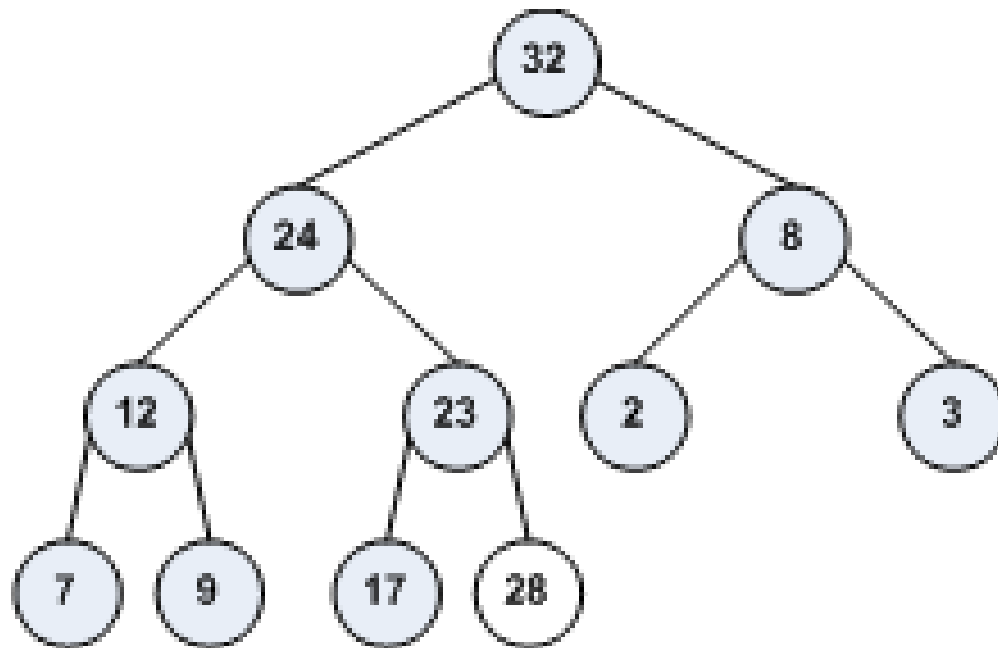
- ▶ Aplicarea algoritmului de inserare:



a) Heap-ul înainte a inserării elementului 28

Heap

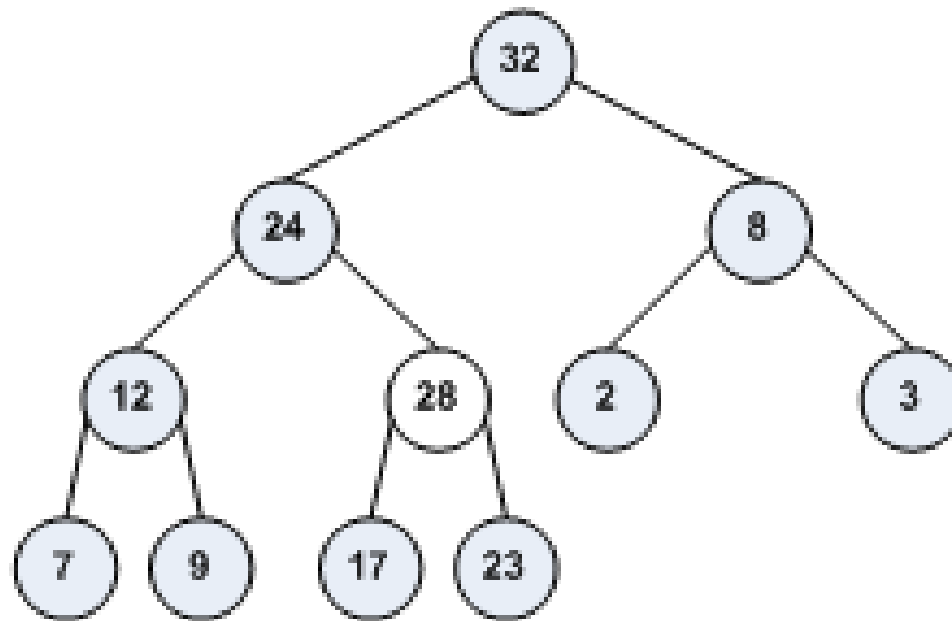
- ▶ Aplicarea algoritmului de inserare:



b) Elementul este inserat la sfârșitul structurii

Heap

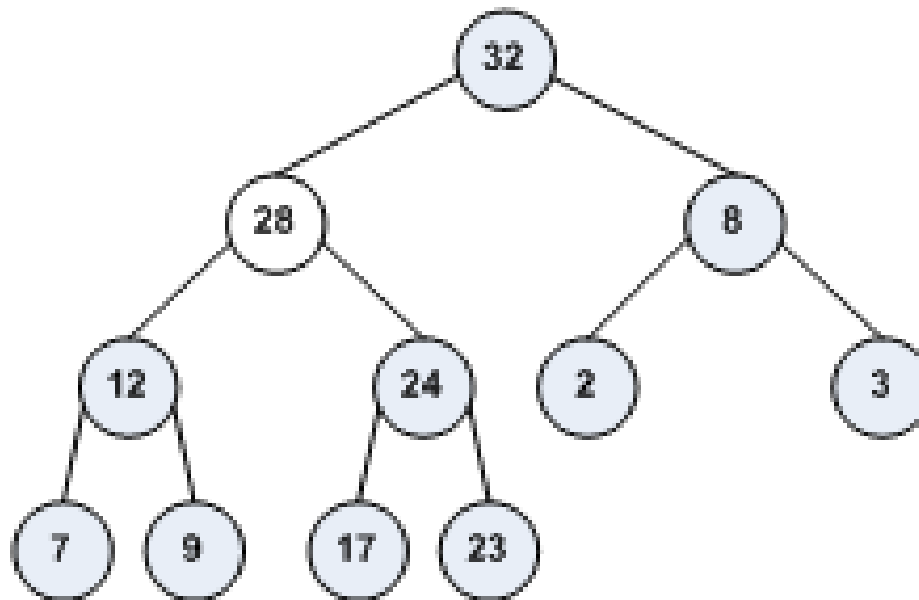
- ▶ Aplicarea algoritmului de inserare:



c) Elementul ridicat în arbore deoarece nu se respectă proprietatea de ordonare

Heap

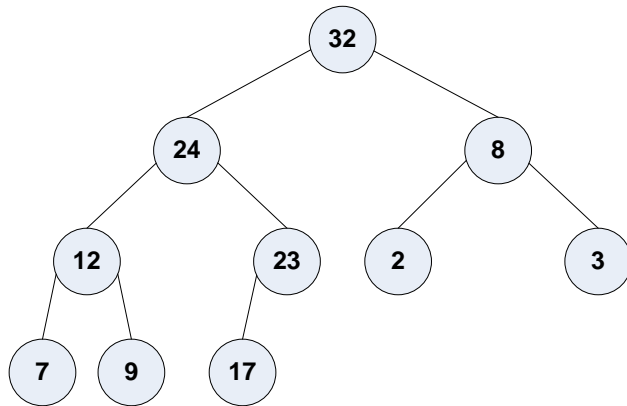
- ▶ Aplicarea algoritmului de inserare:



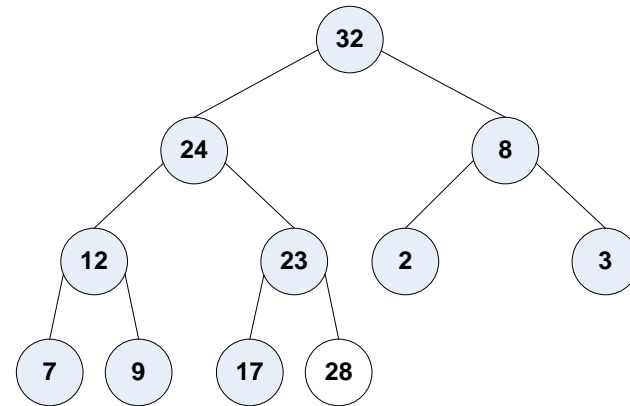
d) Algoritmul este încheiat deoarece valoarea nodului inserat este mai mică decât valoarea nodului părinte

Heap

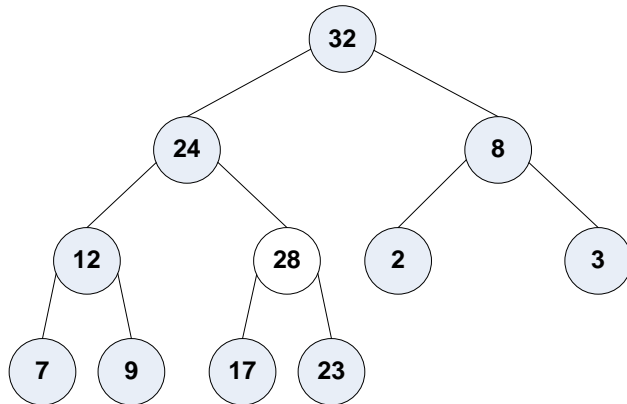
► Aplicarea algoritmului de inserare:



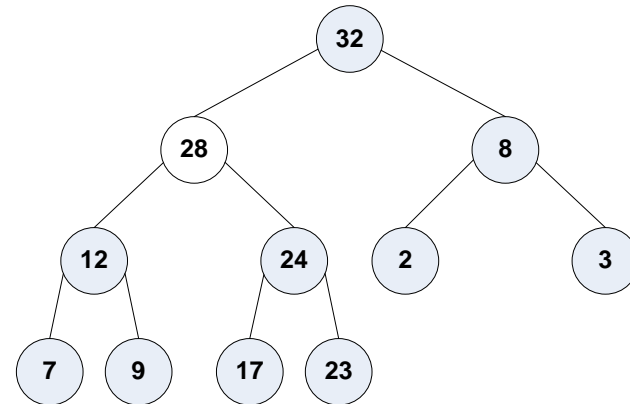
a) Heap-ul înainte a inserării elementului 28



b) Elementul este inserat la sfârșitul structurii



c) Elementul ridicat în arbore deoarece nu se respectă proprietatea de ordonare



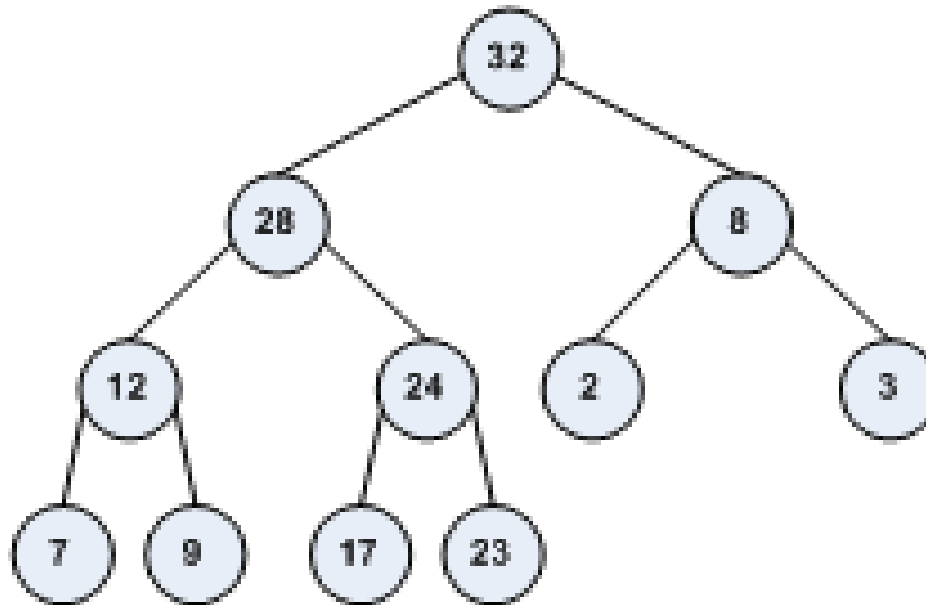
d) Algoritmul este încheiat deoarece valoarea nodului inserat este mai mică decât valoarea nodului părinte

Heap

- ▶ *Ștergerea elementelor* dintr-un heap se poate efectua doar prin extragerea elementului maxim sau minim. Pentru păstrarea structurii de heap se utilizează procedura de filtrare prezentată anterior. Algoritmul de extragere a elementului maxim presupune parcurgerea etapelor:
 - se interschimbă valoarea din nodul rădăcină cu valoarea din ultimul nod al arborelui;
 - se elimină ultimul nod din arbore;
 - se aplică procedura de filtrare pe nodul rădăcină pentru a păstra proprietatea de ordonare;
 - se returnează valoarea din nodul eliminat.

Heap

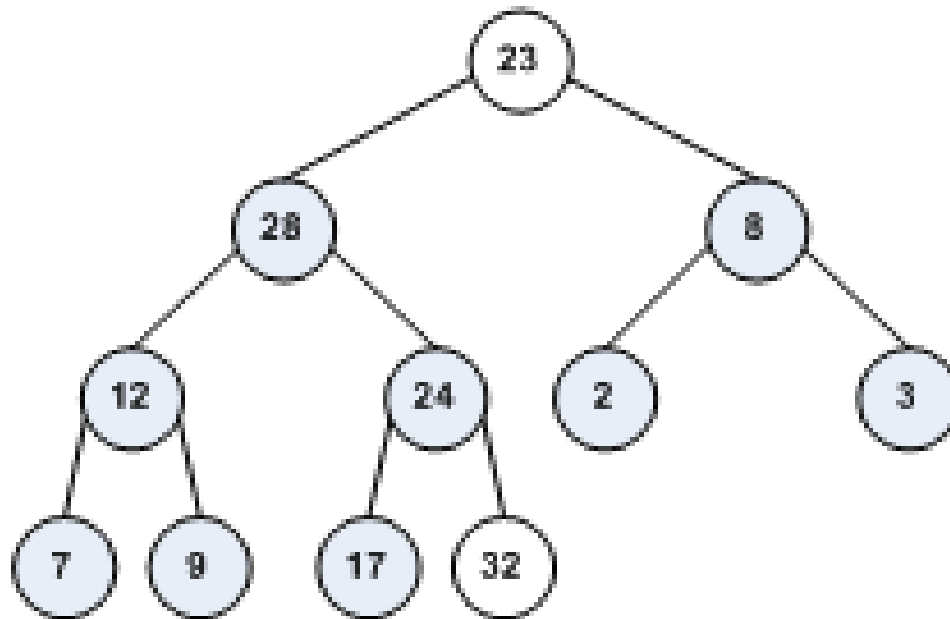
- ▶ Aplicarea algoritmului extragere element maxim:



a) Heap-ul înainte extragerii elementului maxim

Heap

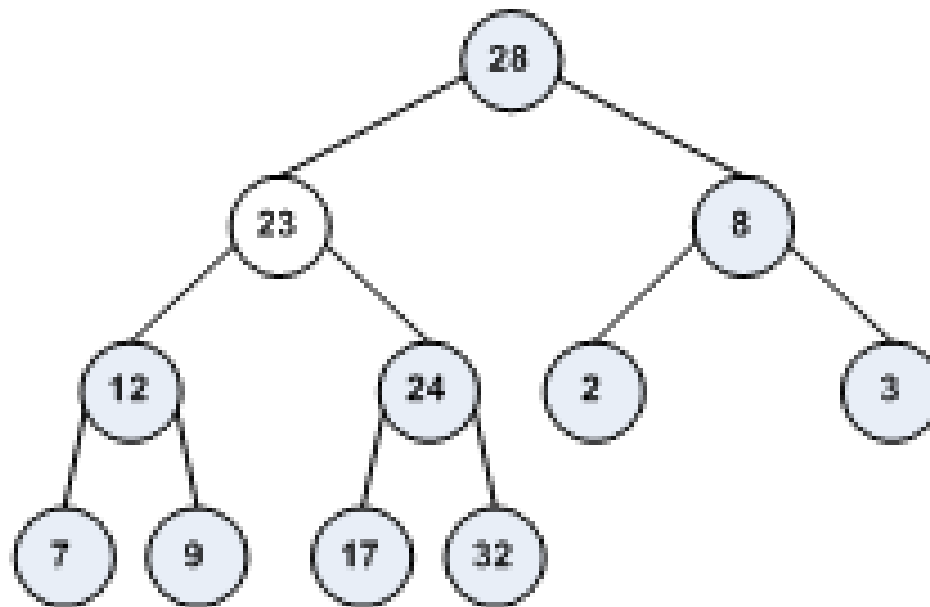
- ▶ Aplicarea algoritmului extragere element maxim:



b) Se interschimbă rădăcina cu ultimul nod

Heap

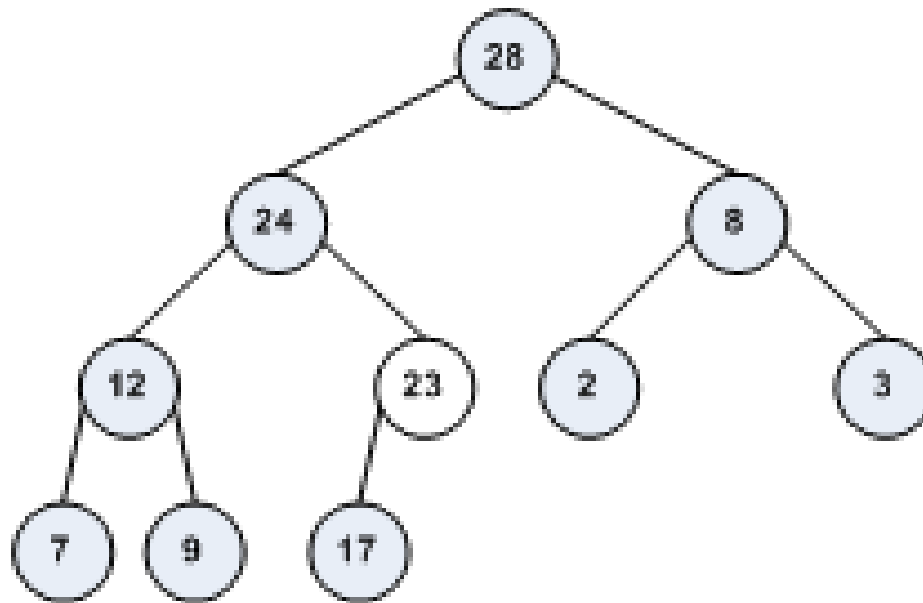
- Aplicarea algoritmului extragere element maxim:



c) Se aplică procedura de filtrare pentru coborârea nodului pe poziția corectă

Heap

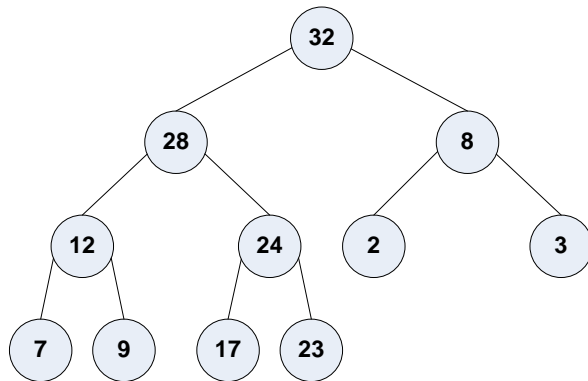
- ▶ Aplicarea algoritmului extragere element maxim:



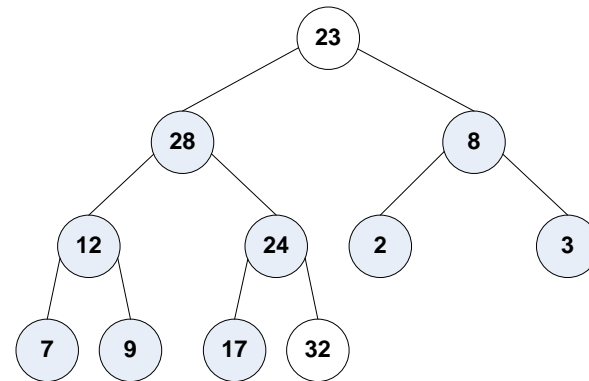
d) După încheierea procedurii de filtrare se elimină ultimul nod din structură

Heap

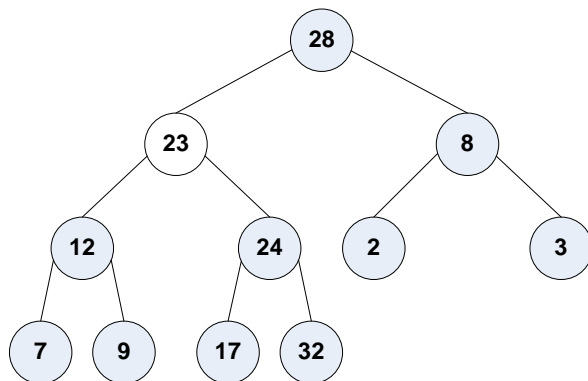
► Aplicarea algoritmului extragere element maxim:



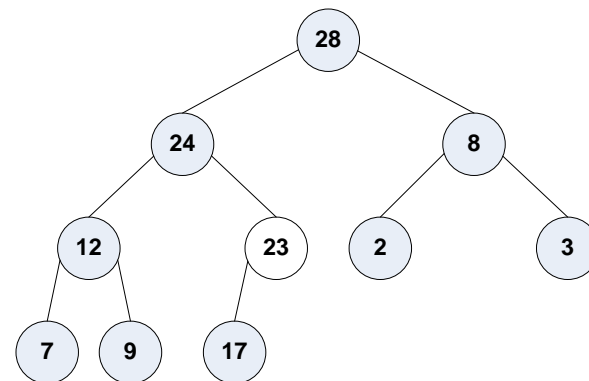
a) Heap-ul înaintea extragerii elementului maxim



b) Se interschimbă rădăcina cu ultimul nod



c) Se aplică procedura de filtrare pentru coborârea nodului pentru a-l aduce în poziția corectă



d) După încheierea procedurii de filtrare se elimină ultimul nod din structură

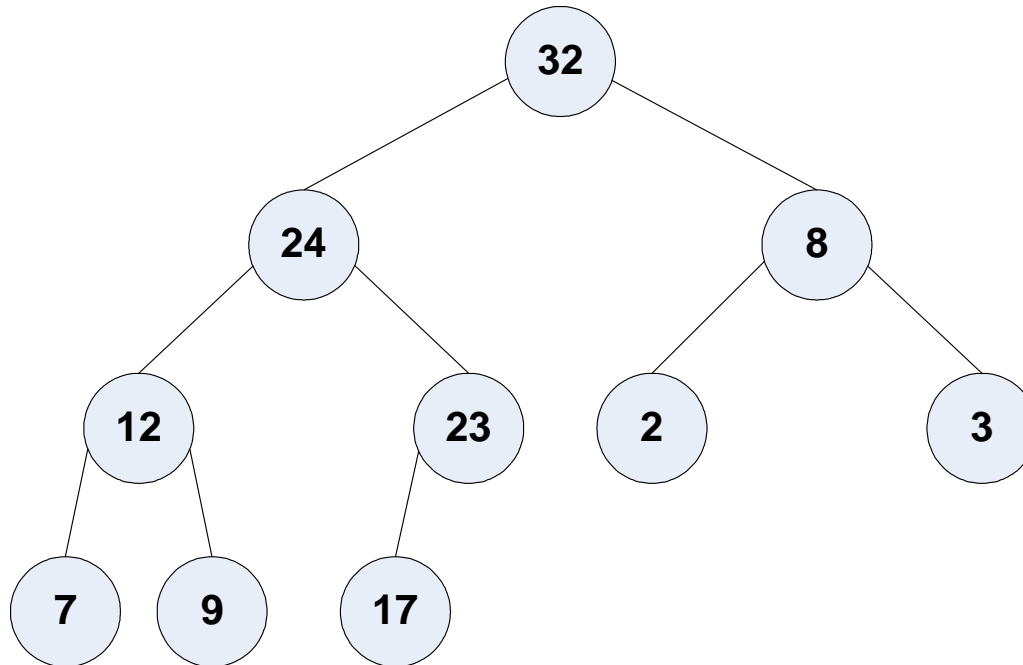
Heap

- ▶ Implementarea structurii *Heap*:
 - deși este posibilă implementarea folosind arbori binari, stocarea eficientă a structurii heap poate fi realizată folosind un masiv unidimensional;
 - elementele arborelui se stochează în masiv începând cu nodul rădăcină și continuând cu nodurile de pe nivelurile următoare, preluate de la stânga la dreapta.

Heap

- ▶ Reprezentarea în memorie pentru structura heap:

32	24	8	12	23	2	3	7	9	17
0	1	2	3	4	5	6	7	8	9



Heap

- ▶ Navigarea între elementele arborelui se poate face în ambele direcții folosind următoarele formule:

$$\text{Parinte}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor, \text{Stânga}(i) = 2 \cdot i + 1, \text{Dreapta}(i) = 2 \cdot i + 2$$

Heap

- ▶ Structura heap este preferată pentru multe tipuri de aplicații. Cele mai importante utilizări sunt:
 - implementarea cozilor de prioritate, utilizate pentru simularea pe bază de evenimente sau algoritmi de alocare a resurselor;
 - implementarea selecției în algoritmi de tip greedy, cum ar fi algoritmul lui Prim pentru determinarea arborelui de acoperire minimă sau algoritmul lui Dijkstra pentru determinarea drumului minim;
 - sortarea masivelor utilizând algoritmul HeapSort.

Heap

- ▶ Cozile de prioritate sunt structuri de date care suportă următoarele două operații de bază:
 - inserarea unui element cu o prioritate asociată;
 - extragerea elementului cu prioritate maximă.
- ▶ Cele mai importante aplicații ale cozilor de prioritate sunt: simularea bazată pe evenimente, gestionarea resurselor partajate (lățime de bandă, timp de procesare) și căutarea în spațiul soluțiilor.

Heap

- ▶ Structura de date de tip Heap este una dintre cele mai eficiente modalități de implementare a cozilor de prioritate.
- ▶ Prioritatea elementelor este dată de relația de ordine existentă între valorile asociate nodurilor.

Heap

- ▶ Într-un simulator discret pentru o coadă de așteptare la un magazin, modul de operare este reprezentat sub forma unei secvențe de evenimente ordonate cronologic.
- ▶ Evenimentele sunt sosirile clienților în coada de așteptare și servirea clienților. Simulatorul conține o coadă de evenimente. Evenimentele sunt adăugate în coadă pe măsură ce timpul lor de producere poate fi determinat și sunt extrase din coadă pentru procesare în ordine cronologică.

Heap

- ▶ Un simulator discret pe bază de evenimente are următoarele componente:
 - *coada de evenimente* – o coadă de prioritate care conține lista evenimentelor care se vor petrece în viitor;
 - *starea simulatorului* – conține un contor pentru memorarea timpului curent, informațiile referitoare la starea actuală a sistemului simulat (clienții aflați în coadă și starea stației de servire) și indicatori;
 - *logica de procesare* – extrage din coadă evenimentele în ordine cronologică și le procesează; procesarea unui eveniment determină modificarea stării sistemului și generarea de alte evenimente.

Heap

- ▶ Pentru simularea propusă trebuie luate în considerare următoarele ipoteze:
 - există o singură stație de servire cu un timp de servire distribuit normal, cu o medie și dispersie cunoscute;
 - există o singură coadă pentru clienți, iar intervalul de timp dintre două sosiri este distribuit uniform într-un interval dat;
 - durata simulării este stabilită de către utilizator.

Heap

- ▶ O altă aplicație a structurii heap este implementarea algoritmului de sortare Heapsort.
- ▶ Sortarea presupune extragerea elementelor din heap și stocarea acestora, în ordine inversă, la sfârșitul masivului utilizat pentru memorarea structurii.
- ▶ Algoritmul poate fi utilizat pentru sortarea oricărui masiv unidimensional pentru care a fost definită o relație de ordine.

Heap

- ▶ Ideea algoritmului de sortare vine de la sine. Se începe prin a construi un heap. Apoi se extrage maximul (adica vârful heap-ului) și se reface heap-ul.
- ▶ Se extrage din nou maximul (care va fi al doilea element ca mărime din vector) și se reface din nou heap-ul.
- ▶ Dacă aceste operații se fac de n ori, se obține vectorul sortat.

Heap

- ▶ Partea frumoasă a algoritmului este că el nu folosește deloc memorie suplimentară.
- ▶ Când heap-ul are n elemente, se extrage maximul și se reține undeva în memorie.
- ▶ În locul maximului (adică în rădăcina arborelui) trebuie adus ultimul element al vectorului, adică $v[n]$.

Heap

- ▶ După această operație, heap-ul va avea $n-1$ noduri, al n -lea rămânând liber. Acest al n -lea nod se folosește pentru a stoca maximumul.
- ▶ Practic, se interschimbă rădăcina, adică pe $v[1]$ cu $v[n]$.
- ▶ Același lucru se face la fiecare pas, ținând cont de micșorarea permanentă a heap-ului.

Heap

Heap	swap elements	delete element	sorted array	details
8, 6, 7, 4, 5, 3, 2, 1	8, 1			swap 8 and 1 in order to delete 8 from heap
1, 6, 7, 4, 5, 3, 2, 8		8		delete 8 from heap and add to sorted array
1, 6, 7, 4, 5, 3, 2	1, 7		8	swap 1 and 7 as they are not in order in the heap
7, 6, 1, 4, 5, 3, 2	1, 3		8	swap 1 and 3 as they are not in order in the heap
7, 6, 3, 4, 5, 1, 2	7, 2		8	swap 7 and 2 in order to delete 7 from heap
2, 6, 3, 4, 5, 1, 7		7	8	delete 7 from heap and add to sorted array
2, 6, 3, 4, 5, 1	2, 6		7, 8	swap 2 and 6 as they are not in order in the heap
6, 2, 3, 4, 5, 1	2, 5		7, 8	swap 2 and 5 as they are not in order in the heap
6, 5, 3, 4, 2, 1	6, 1		7, 8	swap 6 and 1 in order to delete 6 from heap
1, 5, 3, 4, 2, 6		6	7, 8	delete 6 from heap and add to sorted array
1, 5, 3, 4, 2	1, 5		6, 7, 8	swap 1 and 5 as they are not in order in the heap
5, 1, 3, 4, 2	1, 4		6, 7, 8	swap 1 and 4 as they are not in order in the heap
5, 4, 3, 1, 2	5, 2		6, 7, 8	swap 5 and 2 in order to delete 5 from heap
2, 4, 3, 1, 5		5	6, 7, 8	delete 5 from heap and add to sorted array
2, 4, 3, 1	2, 4		5, 6, 7, 8	swap 2 and 4 as they are not in order in the heap
4, 2, 3, 1	4, 1		5, 6, 7, 8	swap 4 and 1 in order to delete 4 from heap
1, 2, 3, 4		4	5, 6, 7, 8	delete 4 from heap and add to sorted array
1, 2, 3	1, 3		4, 5, 6, 7, 8	swap 1 and 3 as they are not in order in the heap
3, 2, 1	3, 1		4, 5, 6, 7, 8	swap 3 and 1 in order to delete 3 from heap
1, 2, 3		3	4, 5, 6, 7, 8	delete 3 from heap and add to sorted array
1, 2	1, 2		3, 4, 5, 6, 7, 8	swap 1 and 2 as they are not in order in the heap
2, 1	2, 1		3, 4, 5, 6, 7, 8	swap 2 and 1 in order to delete 2 from heap
1, 2		2	3, 4, 5, 6, 7, 8	delete 2 from heap and add to sorted array
1		1	2, 3, 4, 5, 6, 7, 8	delete 1 from heap and add to sorted array
			1, 2, 3, 4, 5, 6, 7, 8	completed

Bibliografie

- ▶ Marius Popa, Cristian Ciurea, Mihai Doinea, Alin Zamfiroiu – *Structuri de date: teorie și practică*, Editura ASE, București, 2023, 280 pg.
 - Cap. 6. Structura coadă de priorități