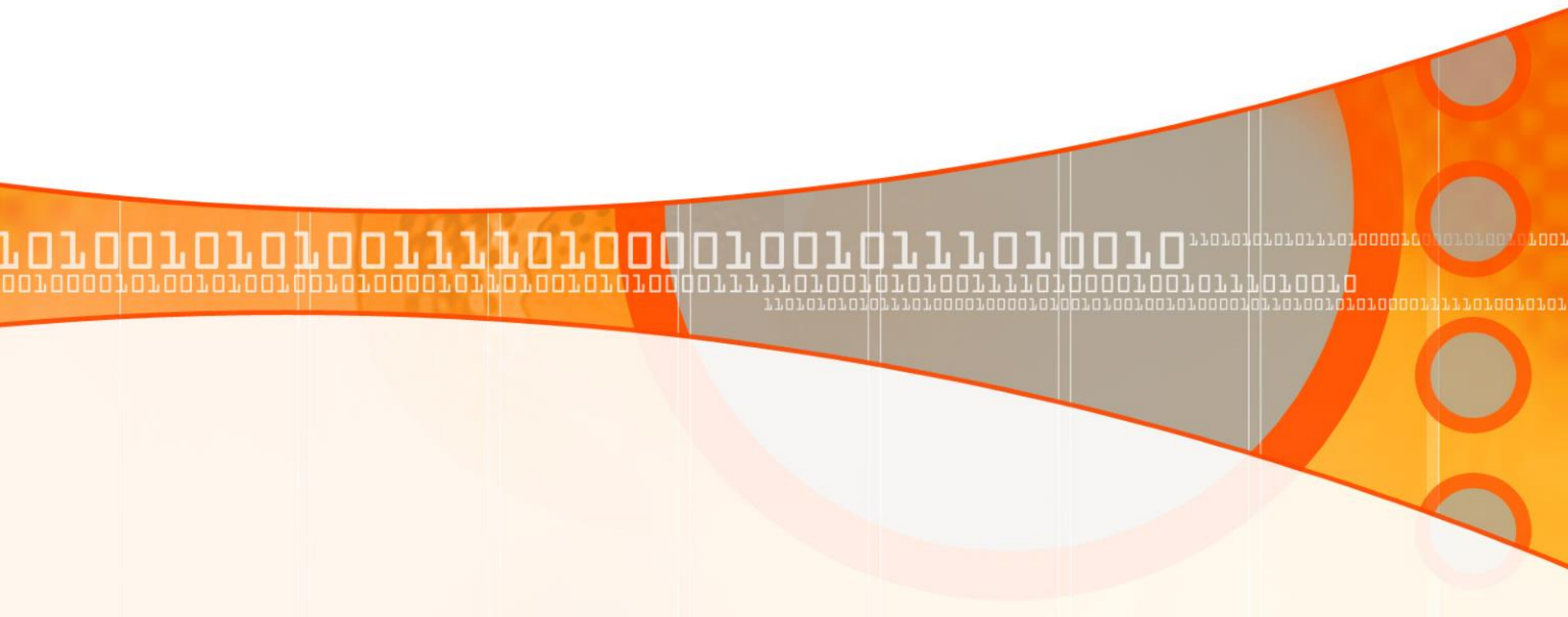
STRUCTURI DE DATE

Structura Heap



Structura Heap

Caracteristici:

- **arbore (frecvent binar – binary heap) cu proprietăţi de structură şi de ordonare;**
- **nod arbore: valoare de cheie şi, eventual, alte date suplimentare;**
- **cheia: permite definirea unei relaţii de ordine totală pe mulţimea nodurilor;**

Structura Heap

Caracteristici (continuare):

- moduri de organizare:
  - max-heap: cea mai mare cheie in radacina;
  - min-heap: cea mai mica cheie in radacina;

- Conversie max-heap in min-heap sau invers: inversarea relatiei de ordine.

Structura Heap

Proprietatea de structură:

- **elemente organizate ca arbore binar complet;**
- **arbore binar complet: toate nodurile (nivelurile 1 … h-2) au exact doi fii, iar nodurile de pe h-1 pot face exceptie.**

Structura Heap

Proprietatea de ordonare:

- **valoarea de cheie, cu excepţia nodului rădăcină, mai mică sau egală decât cea a nodului părinte;**

- **nu se impune nici o regulă referitoare la poziţia sau relaţia dintre nodurile fiu (nu este arbore binar de cautare).**

Structura Heap

Utilizări importante:

- **implementare cozi de prioritate: simulare pe bază de evenimente, algoritmi de alocare a resurselor;**
- **implementare selecţie algoritmi de tip greedy: algoritmul Prim (arbore de acoperire minimă), algoritmul Dijkstra (determinare drum minim);**
- **sortare masive utilizând algoritmul HeapSort.**

Structura Heap

Operaţii principale:

- **construire heap pornind de la un masiv unidimensional oarecare;**
- **inserare element în structură;**
- **extragere element maxim sau minim.**

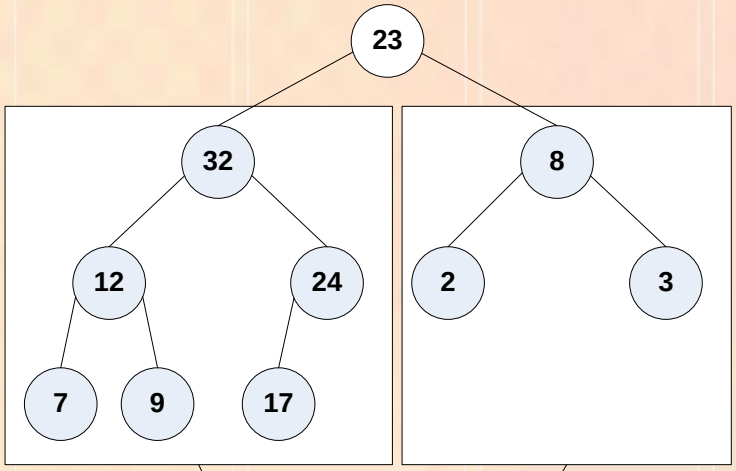
Structura Heap

Construire heap:

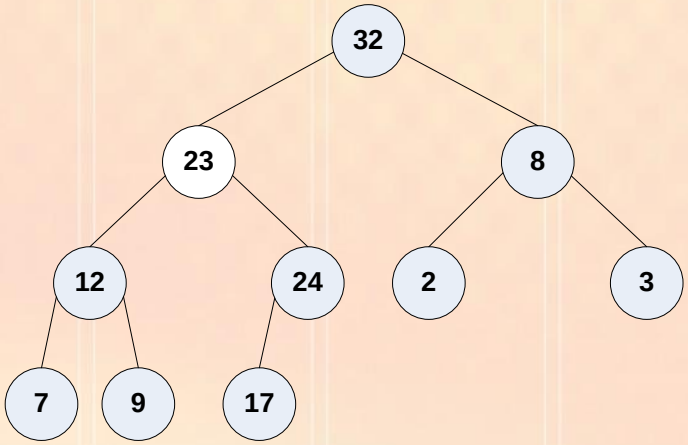
- Implementare procedură de filtrare: transforma un arbore în care doar subarborii rădăcinii sunt heap-uri ale căror înălţime diferă cu cel mult o unitate într-un heap prin coborârea valorii din rădăcină pe poziţia corectă;

Structura Heap

Construire heap (continuare):

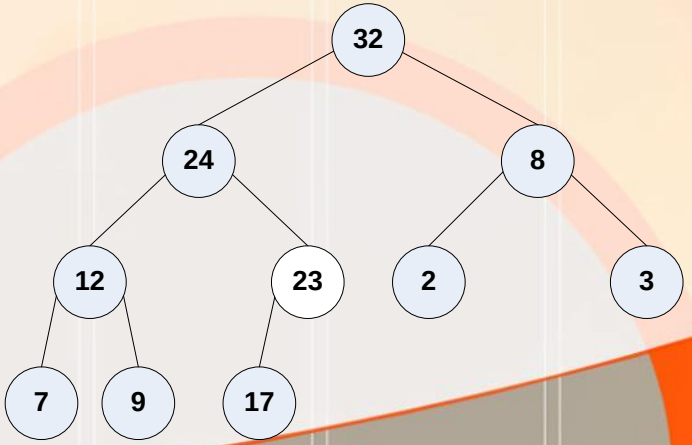


a) Situaţia iniţială



b) Arborele după aplicarea primului pas

Subarbori organizaţi sub formă de heap (respectă proprietăţile de structură şi ordonare)



c) Arborele la sfârşitul procedurii de filtrare

Structura Heap

Construire heap (continuare):

- **algoritm de filtrare – etape incepand cu radacina:**

1. **se determină maximul dintre nodul curent, fiul stânga şi fiul dreapta;**
2. **dacă maximul se află în nodul curent, atunci algoritmul se opreşte;**
3. **dacă maximul de află într-unul dintre fii, atunci se interschimbă valoarea din nodul curent cu cea din fiu şi se continuă execuţia algoritmului cu nodul fiu.**

Structura Heap

Inserare elemente:

- **poate succede etapa initiala de constructie;**
- **structura rezultată trebuie să păstreze proprietatea de ordonare.**

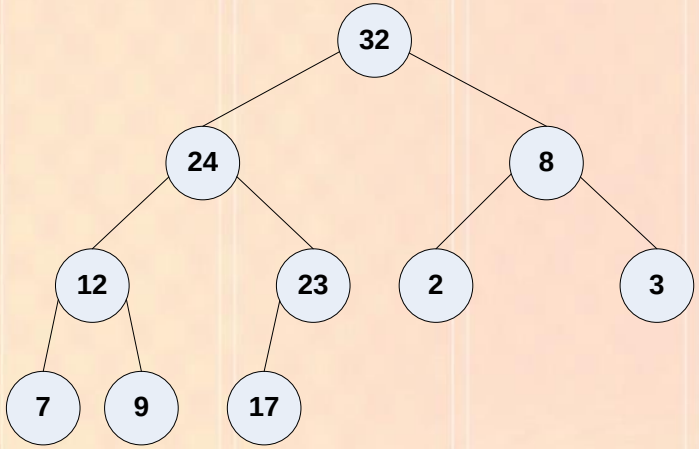
Structura Heap

Inserare elemente (continuare):

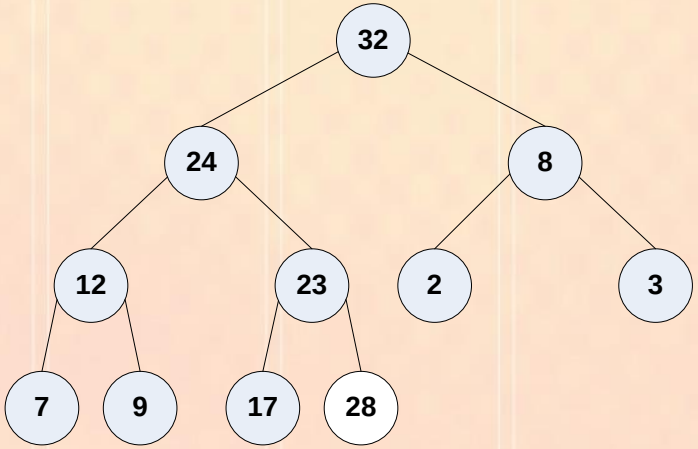
- Etape:

1. se adaugă elementul ca nod frunză pentru a păstra proprietatea de structură;
2. se compară cheia din nodul curent cu cea din nodul părinte;
3. dacă valoarea de cheie din nodul părinte este mai mica, se interschimbă nodul curent cu nodul părinte;
4. dacă nodul părinte are cheie mai mare sau egala atunci algoritmul se opreşte.

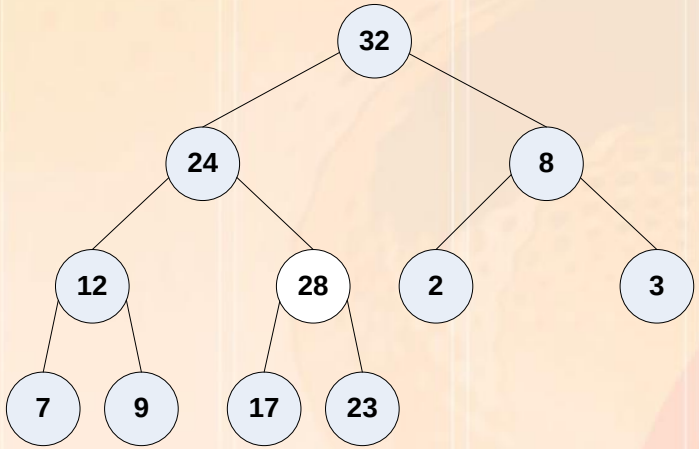
Structura Heap



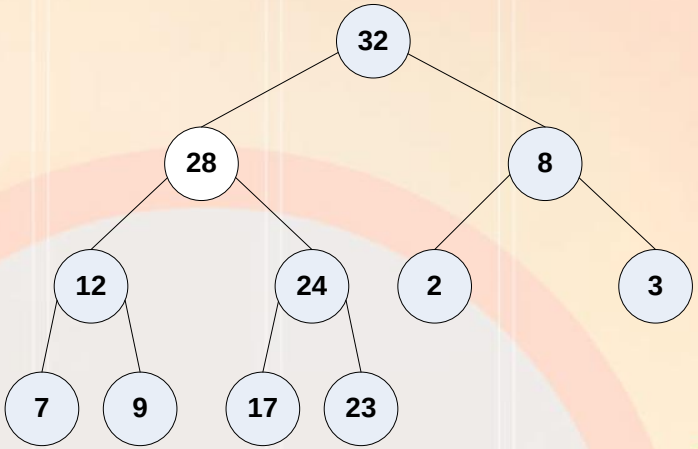
a) Heap-ul înaintea inserării elementului 28



b) Elementul este inserat la sfârşitul 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

Structura Heap

Stergere elemente:

- **extragere element maxim (max-heap) sau minim (min-heap);**
- **păstrare structurii heap: utilizare procedura de filtrare prezentată anterior.**

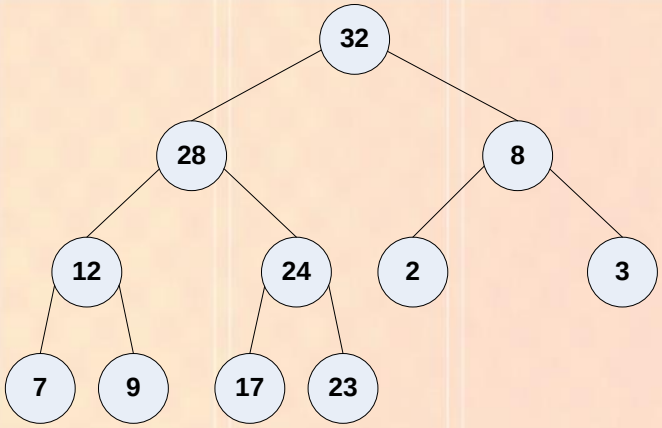
Structura Heap

Stergere elemente (continuare):

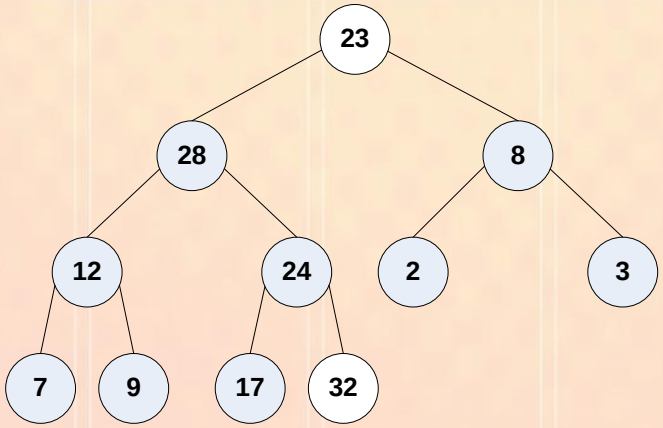
- Etape:

1. interschimb cheie din rădăcină cu cheia din ultimul nod de pe ultimul nivel;
2. eliminare ultim nod din arbore;
3. aplicare procedura de filtrare pe nodul rădăcină pentru a păstra proprietatea de ordonare;
4. retinere cheie din nodul eliminat.

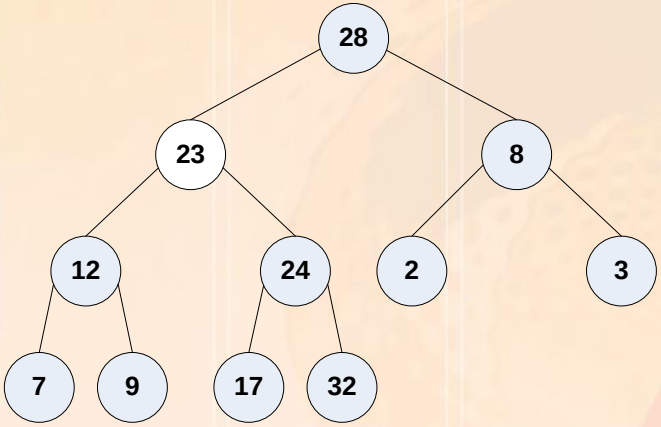
Structura Heap



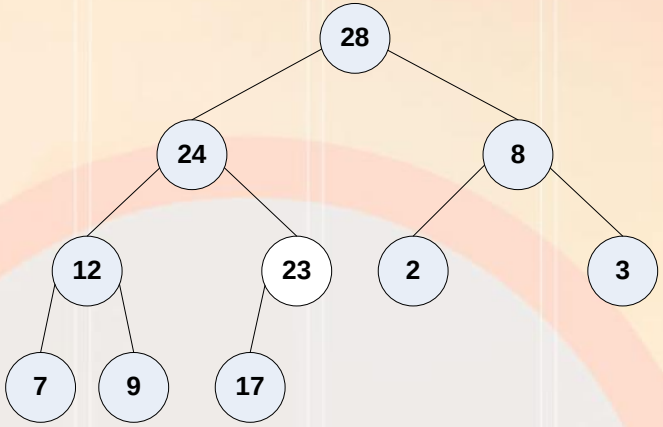
a) Heap-ul înaintea extragerii elementului maxim



b) Se interschimbă rădăcina cu ultimul nod



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



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

Structura Heap

Implementare structura Heap:

- **Stocare eficienta: masiv unidimensional (nu se utilizeaza pointeri);**
- **Dispunerea nodurilor in masiv: elementele arborelui începând cu nodul rădăcină şi continuând cu nodurile de pe nivelurile următoare preluate de la stânga la dreapta.**

Structura Heap

Implementare structura Heap (continuare):

- **Navigarea între elementele arborelui: în ambele direcţii, astfel:**

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

Cozi de prioritate

Caracteristici:

- **implementare prin structura Heap;**
- **prioritate elemente: dată de relaţia de ordine existentă între valorile asociate nodurilor.**

Cozi de prioritate

Operaţii de bază:

- inserare element cu o prioritate asociată;
- extragere element cu prioritate maximă.

Aplicaţii ale cozilor de prioritate:

- simulare bazată pe evenimente;
- gestionare resurselor partajate (lăţime de bandă, timp de procesare);
- căutare în spaţiul soluţiilor.

Cozi de prioritate

Simulare discreta:

- operare sistem sub forma unei secvenţe de evenimente ordonate cronologic;
- evenimente: sosiri clienţi în coada de aşteptare şi servire clienţi;
- simulatorul: conţine o coadă de evenimente;
- evenimentele sunt adăugate în coadă pe măsură ce timpul lor de producere poate fi determinat şi sunt extrase din coadă pentru procesare în ordine cronologică.

Cozi de prioritate

Simulare discreta – componente:

- **coada de evenimente: coadă de prioritate cu lista evenimentelor care se vor produce;**
- **starea simulatorului: contor pentru memorarea timpului curent, starea actuală a sistemului simulat (clienţii aflaţi în coadă şi starea staţiei de servire) şi indicatori;**
- **logica de procesare: extrage din coadă evenimentele în ordine cronologică şi le procesează; procesarea determină modificarea stării sistemului şi generarea de alte evenimente.**

Cozi de prioritate

Simulare discreta – ipoteze:

- există o singură staţie de servire cu un timp de servire distribuit normal, cu o medie şi dispersie cunoscută;
- există o singură coadă pentru clienţi, iar intervalul de timp dintre două sosiri este distribuit uniform într-un interval dat;
- durata simulării este stabilită de către utilizator.

Cozi de prioritate

Simulare discreta – functionare:

- extragere evenimente din heap şi procesarea acestora pe bază de reguli;
- evenimentele de tip sosire determină generarea evenimentului pentru sosirea următoare şi a unui eveniment de servire daca staţia este liberă la momentul curent;

Cozi de prioritate

Simulare discreta – functionare (continuare):

- **in cazul evenimentelor de tip servire, se generează următorul eveniment de tip servire dacă mai există clienţi în coadă;**

- **pe măsură ce sunt procesate evenimentele, sunt reţinute şi informaţiile necesare pentru calcularea indicatorilor de performanţă aferenţi sistemului simulat.**

Algoritm HeapSort

Sortare date:

- extragere element din structura heap ;
- stocare elemente extrase, în ordine inversă, intr-un masiv distinct sau la sfârşitul masivului utilizat pentru memorarea structurii.