Heap-uri de INTERVALE

Mihai Scorțaru

Principalul dezavantaj al heap-urilor îl constituie imposibilitatea de a determina rapid atât elementul minim, cât și elementul maxim al unei secvențe de elemente între care se poate stabili o relație de ordine. În cadrul acestui articol vom propune o structură de date care rezolvă acest dezavantaj.

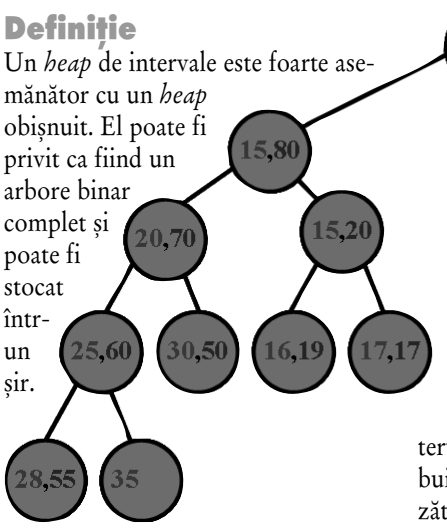
Heap-urile de intervale sunt structuri de date care permit determinarea în timp constant atât a minimului, cât și a maximului unei secvențe de elemente.

Fiind o structură de date derivată din *heap*, *heap*-ul de intervale va trebui să respecte o proprietate de *heap*, evchivalentă cu cea a *heap*-urilor obișnuite.

Pe lângă această proprietate, structura de date va trebui să permită determinarea în timp constant a elementelor minim și maxim, precum și ștergerea acestor elemente (cu păstrarea proprietății de *heap*) în timp logaritmic. Pe lângă aceste operații, *heap*-urile de intervale trebuie să permită adăugarea unui element în timp logaritmic.

Definiție

Un *heap* de intervale este foarte asemănător cu un *heap* obișnuit. El poate fi privit ca fiind un arbore binar complet și poate fi stocat într-un șir.



Totuși, există două diferențe semnificative. Prima dintre ele o constituie faptul că fiecare element al *heap*-ului va conține două valori în loc de una. Cele două valori (prima este mai mică sau egală cu a doua) reprezintă un interval. Ultimul nod, poate să conțină un singur element, în cazul în care numărul elementelor care trebuie păstrate este impar.

Ca urmare, un *heap* de intervale care trebuie să păstreze N valori, va conține un număr de elemente egal cu partea întreagă superioară a valorii $N / 2$.

Cea de-a doua diferență semnificativă este dată de proprietatea de *heap*. Pentru *heap*-urile de intervale, această proprietate specifică faptul că intervalul corespunzător unui nod trebuie să fie inclus în intervalul corespunzător părintelui nodului respectiv.

O altă formulare ar putea fi: intervalul corespunzător unui nod trebuie să includă intervalele corespunzătoare fiilor nodului respectiv.

Studiind această structură putem trage concluzia că extremitățile stângi ale intervalelor formează un *heap* obișnuit în care rădăcina conține minimul, în timp ce extremitățile drepte formează un alt *heap* obișnuit în care rădăcina conține maximul.

În imaginea de pe această pagină este prezentat un *heap* de intervale.

Observăm că, la fel ca și în cazul *heap*-urilor obișnuite, pot exista mai multe elemente cu valori egale.

Minim și maxim

Se observă imediat, pe baza proprietății de *heap* de intervale, că intervalele corespunzătoare oricărui nod sunt incluse în intervalul corespunzător rădăcinii.

Așadar, putem trage imediat concluzia că elementul minim este extremitatea stângă a intervalului corespunzător rădăcinii, iar elementul maxim este extremitatea dreaptă a acestui interval.

Ca urmare, operațiile de determinare a minimului și maximului pot fi implementate în timp constant deoarece implică doar accesarea rădăcinii arborelui (primul element al vectorului în care este stocat *heap*-ul de intervale).

O versiune a algoritmului de determinare a minimului, care prevede toate situațiile care pot apărea, este următoarea:

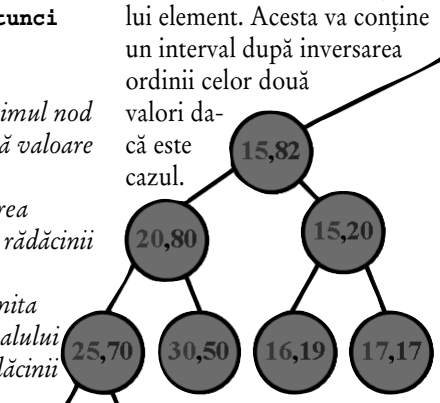
```
algoritm determinăMinim
  dacă heap-ul este vid atunci
    semnalează eroare
  altfel
    dacă rădăcina este ultimul nod
         și conține o singură valoare
    atunci
      returnează valoarea
          corespunzătoare rădăcinii
    altfel
      returnează extremita
        tea stângă a intervalului
         corespunzător rădăcinii
    sfârșit dacă
  sfârșit dacă
sfârșit algoritm
```

Algoritmul pentru determinarea maximului este foarte asemănător:

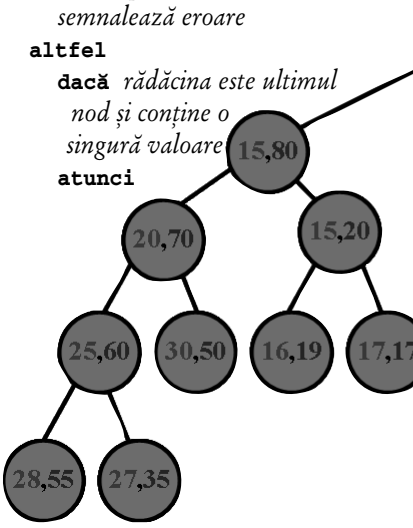
```
algoritm determinăMaxim
  dacă heap-ul este vid atunci
    semnalează eroare
  altfel
    dacă rădăcina este ultimul
      nod și conține o
      singură valoare
    atunci
      returnează valoarea
          corespunzătoare rădăcinii
    altfel
      returnează extremitatea
          dreaptă a intervalului
          corespunzător rădăcinii
    sfârșit dacă
  sfârșit dacă
sfârșit algoritm
```

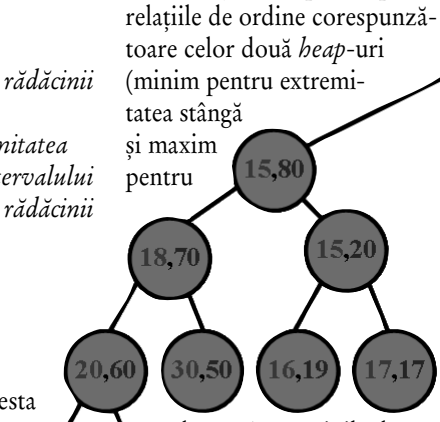
Inserarea

Pentru a insera o valoare, acesta este adăugat la sfârșitul vectorului. În cazul în care înaintea inserării, ultimul element conținea o singură valoare, atunci valoarea este adăugată ultimului element. Acesta va conține un interval după inversarea ordinii celor două valori dacă este cazul.



În cazul în care înaintea inserării ultimul element conținea două valori, atunci se creează un nou element care va conține o singură valoare. După adăugarea valorii, aceasta trebuie să "urce" în arbore într-un mod asemănător celui de la *heap*-urile obișnuite. Dacă noul element a devenit o extremitate stângă, atunci elementul va "urca" în *heap*-ul corespunzător extremităților stângi, iar în caz contrar va urca în *heap*-ul corespunzător extremităților drepte, respectându-se relațiile de ordine corespunzătoare celor două *heap*-uri (minim pentru extremitatea stângă și maxim pentru cea dreaptă). Imaginile de pe această pagină prezintă configurația *heap*-ului după ce au fost inserate elemente cu valorile 18, 27 și 82 (nu sunt inserări succesive, ci inserări pornind de la *heap*-ul prezentat anterior). Prezentăm în continuare algoritmul care realizează inserarea unui element.

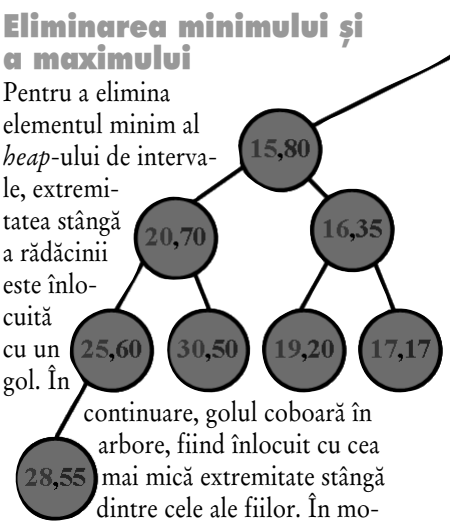





```
algoritm inserare(x):
  dacă heap-ul este vid atunci
    creează un nou nod care
      conține doar valoarea x și
      adaugă acest nod heap-ului
      ca ultim element
  altfel
    dacă ultimul element al
        heap-ului conține o
        singură valoare y
    atunci
      inserează valoarea în
        ultimul element al heap-ului
      dacă x < y atunci
        intervalul corespunzător ul-
        timului element este [x,y]
      altfel
        intervalul corespunzător ul-
        timului element este [y,x]
      sfârșit dacă
      sfârșit dacă
      cât timp x nu este în
      intervalul [a,b] corespunzător
      părintelui și x nu face parte din
        elementul rădăcină execută
        dacă x < a atunci
          interschimbă x cu a
        altfel
          interschimbă x cu b
        sfârșit dacă
      sfârșit cât timp
    sfârșit dacă
  sfârșit algoritm
```

Eliminarea minimului și a maximului

Pentru a elimina elementul minim al *heap*-ului de intervale, extremitatea stângă a rădăcinii este înlocuită cu un gol. În continuare, golul coboară în arbore, fiind înlocuit cu cea mai mică extremitate stângă dintre cele ale fiilor. În momentul în care golul ajunge la o frunză, acesta este înlocuit cu una dintre extremitățile ultimului element și apoi valorile păstrate în frunza respectivă sunt ordonate.

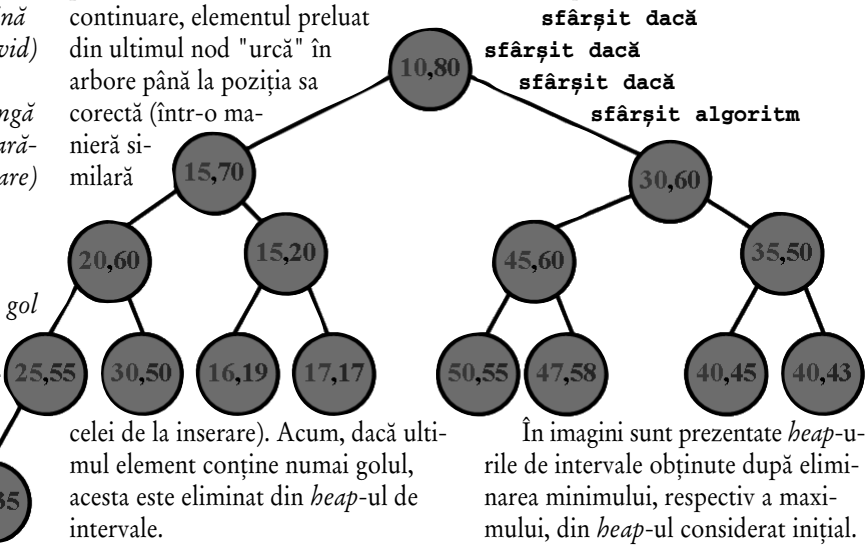


În continuare, elementul preluat din ultimul nod "urcă" în arbore până la poziția sa corectă (într-o manieră similară celei de la inserare). Acum, dacă ultimul element conține numai golul, acesta este eliminat din *heap*-ul de intervale.

Prezentăm în continuare algoritmul de ștergere a elementului minim.

```
algoritm eliminăMinim
  dacă heap-ul este vid atunci
    semnalează eroare
  altfel
    dacă rădăcina este ultimul nod
    atunci
      dacă rădăcina conține o
           singură valoare atunci
        șterge valoarea și elimină
        nodul (heap-ul devine vid)
      altfel
        șterge extremitatea stângă
          a rădăcinii (rădăcinarămâne cu o singură valoare)
      sfârșit dacă
    altfel
      înlocuiește extremitatea
        stângă a rădăcinii cu un gol
      cât timp nodul curent
        nu este frunză executăinterschimbă golul cu
          cea mai mică extremitate stângă dintre
              cele ale fiilor
      sfârșit cât timp
      dacă frunza nu este
           ultimul element al
                arborelui
        atunci
          interschimbă golul cu o
            extremitate a intervalului
                din ultimul nod
          ordonează extremitățile
            urcă în arbore valoarea
          preluată din ultimul element
        sfârșit dacă
        dacă ultimul element al
             arborelui este vid atunci
          elimină ultimul element
        sfârșit dacă
      sfârșit dacă
    sfârșit dacă
  sfârșit dacă
sfârșit algoritm
```

Pentru a elimina elementul maxim al *heap*-ului de intervale, extremitatea dreaptă a rădăcinii este înlocuită cu un gol. În continuare, golul coboară în arbore, fiind înlocuit cu cea mai mare extremitate dreaptă dintre cele ale fiilor. În momentul în care golul ajunge la o frunză, acesta este înlocuit cu una dintre extremitățile ultimului element și apoi valorile păstrate în frunză sunt ordonate. În continuare, elementul preluat din ultimul nod "urcă" în arbore până la poziția sa corectă (într-o manieră similară celei de la inserare). Acum, dacă ultimul element conține numai golul, acesta este eliminat din *heap*-ul de intervale.



Prezentăm în continuare algoritmul de ștergere a elementului maxim.

```
algoritm eliminăMaxim
dacă heap-ul este vid atunci
    semnalează eroare
altfel
  dacă rădăcina este
       ultimul nod atunci
    dacă rădăcina conține o
         singură valoare atunci
      șterge valoarea și elimină
      nodul (heap-ul devine vid)
    altfel
      șterge extremitatea
      dreaptă a rădăcinii (rădăcina
      rămâne cu o singură valoare)
    sfârșit dacă
  altfel
    înlocuiește extremitatea
      dreaptă a rădăcinii cu un gol
    cât timp nodul curent nu
          este frunză executăinterschimbă golul cu cea
            mai mare extremitate
          dreaptă dintre cele ale fiilor
    sfârșit cât timp
    dacă frunza nu este ultimul
      element al arborelui atunci
      interschimbă golul cu o
        extremitate a intervalului
              din ultimul nod
      ordonează extremitățile
    sfârșit dacă
    dacă ultimul element al
         arborelui este vid atunci
      elimină ultimul element
      urcă în arbore valoarea
      preluată din ultimul element
    sfârșit dacă
  sfârșit dacă
sfârșit dacă
sfârșit algoritm
```

În imagini sunt prezentate *heap*-urile de intervale obținute după eliminarea minimului, respectiv a maximului, din *heap*-ul considerat inițial.