Breviar Laborator 9: **Heapuri**
Max-heapuri și min-heapuri. Arbori Huffman

Alexandra Maria Vieru
Facultatea de Automatică și Calculatoare

24 aprilie 2014

1 Heapuri

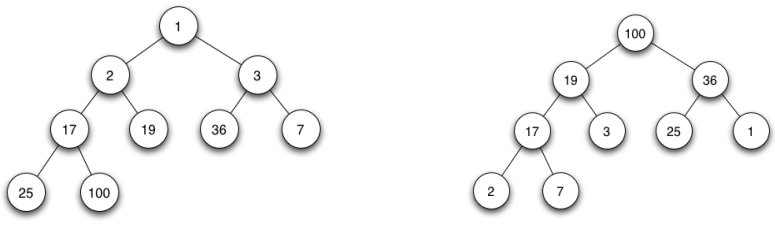
1.1 Noțiuni generale

Un `heap` este un array care poate fi privit ca un arbore binar aproape complet (toate nivelurile, eventual cu excepția ultimului, sunt complete), iar ultimul este completat de la stânga spre dreapta. Fiecărui nod dintr-un arbore heap îi corespunde un element dintr-un array care îi stochează valoarea.

Există două tipuri de heap: `min-heap` și `max-heap`. Pentru a fi heap un arbore binar trebuie să îndeplinească o proprietate heap:

- Într-un `max-heap` fiecare nod trebuie să fie mai mare decât ambii fii și ambii subarbori trebuie să fie max-heapuri.
- Într-un `min-heap` fiecare nod trebuie să fie mai mic decât ambii fii și ambii subarbori trebuie să fie min-heapuri.

În figura 1 sunt ilustrate cele 2 tipuri de heapuri.



(a) Min-heap

(b) Max-heap

Figura 1: Heapuri

În cele ce urmează ne vom referi la max-heapuri.

2 Implementare

Un heap este reprezentat printr-un array (v) în care numerotarea pozițiilor începe de la 1. Pentru a accesa părintele, respectiv fiul din stânga și pe cel din dreapta ai nodului de pe poziția i se folosesc următoarele formule:
Parent(i) = v[i/2]
Left(i) = v[2 * i]
Right(i) = v[2 * i + 1]

În figura 2 se poate vedea reprezentarea unui heap ca arbore și ca array.

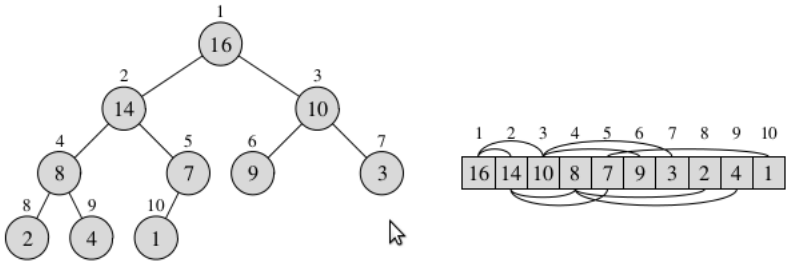


Figura 2: Max-heap (arbore și array)

2.1 Adăugarea unei valori

Valoarea care se dorește să fie adăugată la heap va fi inserată la sfârșitul vectorului. Inserarea ei va păstra prima proprietate a unui heap (cea care spune că este un arbore aproape complet, iar ultimul nivel e completat de la stânga la dreapta), dar nu o va păstra pe cea de a doua care spune că părintele trebuie să fie mai mare decât ambii fii, iar subarorii săi trebuie să fie tot max-heapuri.

Pentru a reface heapul, se va implementa o funcție recursivă `moveUp` care pornește de la noua valoare și o compară cu părintele său. Dacă aceasta e mai mare decât părintele, atunci va intershimba cele 2 valori și va repeta procesul pentru nodul părinte și părintele acestuia, până când nu mai e nevoie de interchimbare.

În figura 3 sunt ilustrate etapele prin care trece un arbore la inserarea unei noi valori.

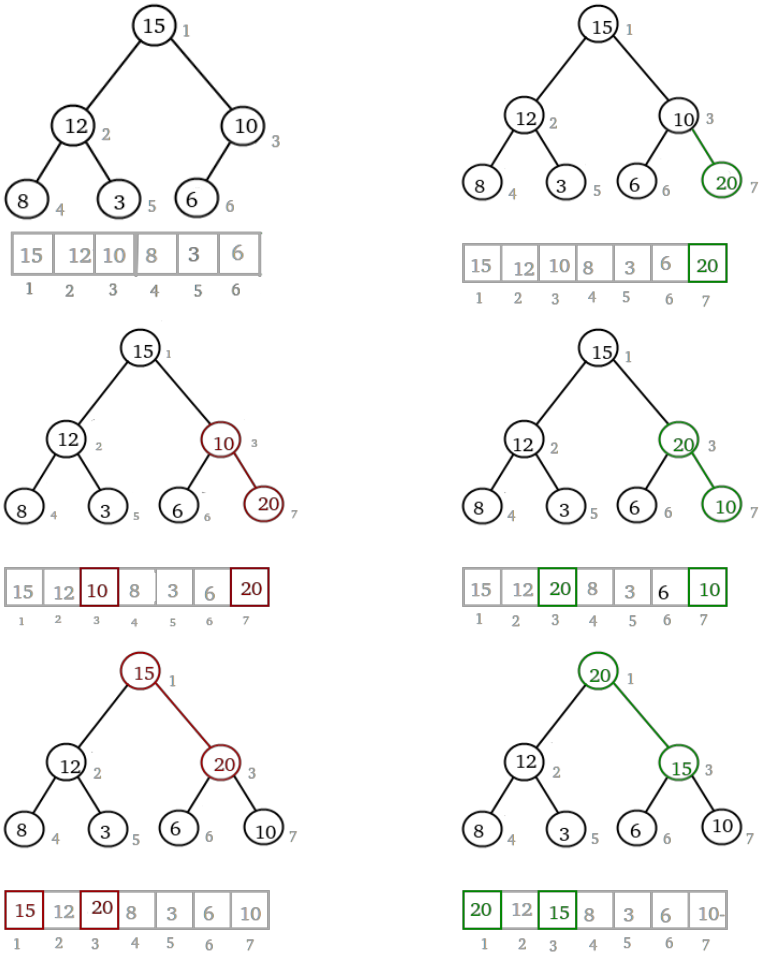


Figura 3: Inserarea valorii 20 într-un max-heap

2.2 Eliminarea maximului

Când se elimină valoarea maximă dintr-un max-heap se va șterge valoarea de pe prima poziție (rădăcina arborelui), dar vor rămâne două heapuri. Este nevoie să se reunească cele două heapuri într-unul singur. Valoarea de pe ultima poziție se va pune pe prima poziție, iar apoi se va raface ordinea valorilor astfel încât acestea să îndeplinească în continuare proprietatea de max-heap.

Pentru aceasta se va face operația inversă de la inserare, adică se va porni de la rădăcină, iar dacă unul dintre fii este mai mare decât rădăcina, aceasta se va înlocui cu valoarea maximă dintre cei doi fii. Procesul se va repeta recursiv pentru fiul cu a cărui valoare s-a făcut interchimbarea, până când nu mai este nevoie de interschimbări (numim funcția `moveDown`).

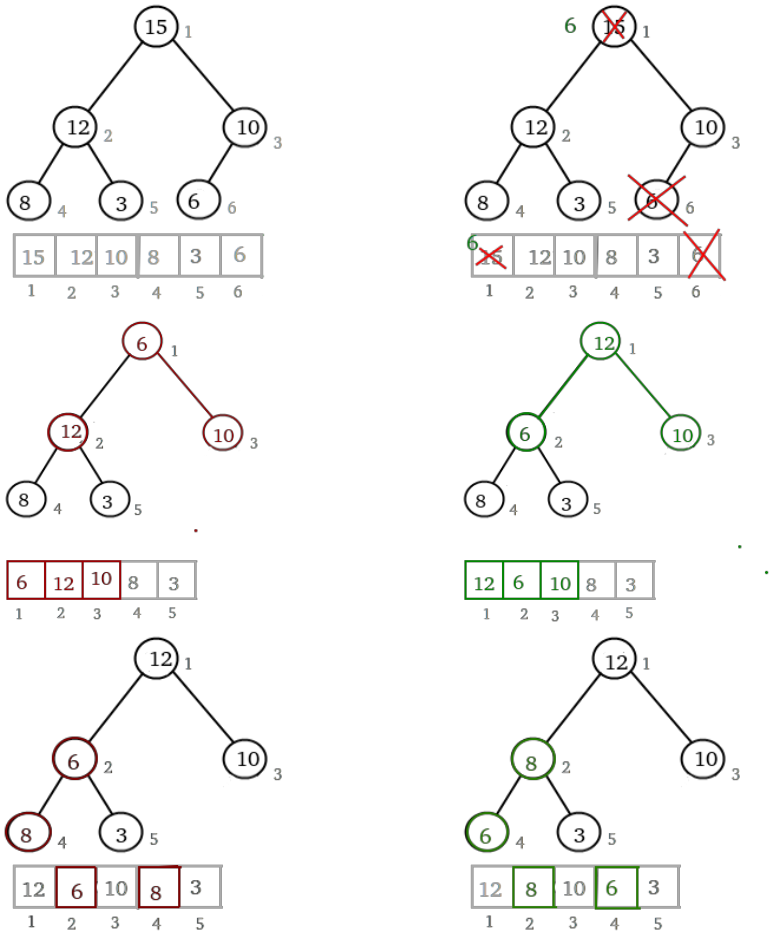


Figura 4: Ștergerea valorii maxime (15)

2.3 Aplicații

Cele mai importante dintre aplicațiile heapurilor sunt:

- Heapsort
- Cozile cu priorități

2.4 Heapsort

Algoritmul 1 heapsort(a, n)

{a=vectorul nesortat, n=numărul de elemente}
$v \leftarrow createHeapFrom(a, n)$
$i \leftarrow n$
for $i > 1$ **do**
  swap(v[1], v[i]
  moveDown(v[1])
end for

2.5 Discuție

Operațiile de inserare și ștergere au aceeași complexitate (O(lg n)) - datorită faptului că arborele este echimilbrat și complexitatea acestor două funcții este dată de înălțimea arborelui. Iar aflarea maximului se face în timp constant.

3 Arbori Huffman

3.1 Noțiuni generale

Arborii Huffman sunt folosiți pentru compresie, atribuindu-se fiecărui element distinct dintr-un șir o reprezentarea binară care duce la reducerea numărului de biți folosiți față de reprezentarea inițială. Ideea este de a asigna caracterelor mai frecvente (ex: a, e) coduri formate din mai puțini biți, iar celor mai rare (ex: z, q), coduri mai lungi. Astfel caracterul `a` va ocupa mai puțini biți deoarece el apare de mai multe ori în text.

3.2 Crearea arborelui Huffman

Codificarea Huffman constă în construirea unui arbore bazându-se pe caracterele ce apar într-un text și pe frecvența de apariție a acestora. Un exemplu de arbore Huffman se găsește în figura 5.

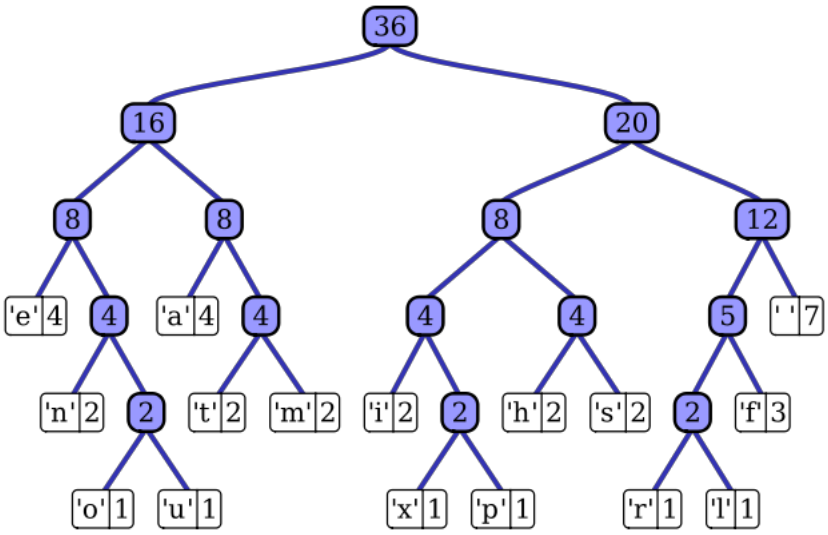


Figura 5: Arbore Huffman

Caracterele se introduc în arbore în ordinea crescătoare a frecvenței de apariție. Se folosește o coadă de priorități pentru a păstra nodurile ordonate. Astfel, algoritmul de creare a unui arbore Huffman este următorul:

Algoritmul 2 buildHuffman(characters, frequencies)

creează un nod frunză pentru fiecare caracter din text
while există mai mult de un nod în coada cu priorități **do**
  extrage cele două noduri cu frecvențele de apariție cele mai mici
  creează un nou nod intermediar având cele 2 noduri extrase drept fii, și a cărui frecvență să fie suma celor două frecvențe
  adaugă noul nod în coada cu priorități
end while
nodul rămas este rădăcina arborelui Huffman complet

Arborele va conține în fiecare frunză unul dintre caracterele ce trebuia codificate, iar celelalte noduri din arbore vor fi doar niște noduri intermediare.

Se parcurge arborele de la rădăcină spre frunze și fiecărei muchii i se asociază o valoare 0 sau 1. De exemplu, muchiei ce duce la fiul stâng i se asociază valoarea 0, iar celei care duce la fiul drept i se asociază valoarea 1. Convenția se păstrează pentru întreg arborele. La finalul parcurgerii, fiecare nod va avea asociat un cod Huffman. După această parcurgere fiecărui caracter îi va fi asociat un cod, această mapare va fi reținută, pe baza ei codificându-se textul.

3.3 Decodificarea

Pentru a decodifica un text care a fost compresat folosind un arbore Huffman este nevoie ca arborele să fi fost salvat, altfel nu se poate face decodificarea.

Decodificarea se face parcurgând textul codificat cifră cu cifră și în funcție de cifra citită se va parcurge arborele, dacă se întâlnește un 0 se merge pe ramura stângă, iar dacă se întâlnește un 1, pe ramura dreaptă. În momentul în care se ajunge la o frunză, înseamnă că un caracter a fost decodificat și urmează un altul.