Breviar Laborator 7: **Arbori**
Arbori binari de căutare

Alexandra Maria Vieru
Facultatea de Automatică și Calculatoare

4 aprilie 2014

1 Noțiuni generale

Arborele este un tip de date abstract care e format dintr-o rădăcină și un set de subarbori. Elementul de bază al arborelui este nodul, acesta având o valoare și un set de noduri copii.
Arborii pot fi de mai multe feluri:

generali - pot avea oricât de mulți copii (Figura 1)

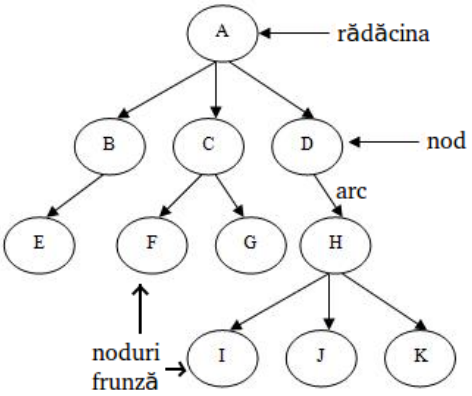

Figura 1: Arbore general

binari - fiecare nod are 0, 1 sau 2 fii (Figura 2)

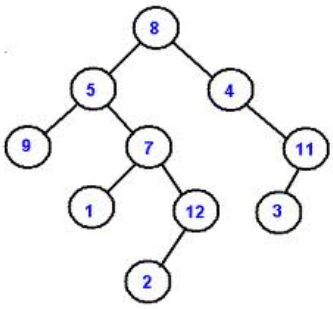

Figura 2: Arbore binar

binari de căutare (Figura 3)

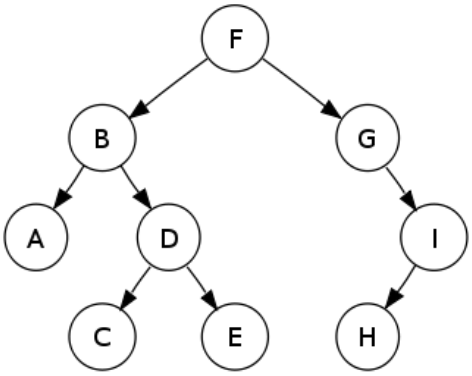

Figura 3: Arbore binar de căutare

AVL - arbori binari de căutare echilibrați

roșu-negru - arbori binari de căutare cu auto-balansare

B - arbori de căutare generalizați

`Adâncimea unui nod` reprezintă lungimea căii de la rădăcină până la acel nod.
`Un nod frunză` este un nod care nu are niciun fiu.
`Înălțimea arborelui` este lungimea celei mai lungi căi de la rădăcină la un nod frunză.

2 Arbori binari de căutare (Binary Search Trees)

2.1 Generalități

Arborii binari de căutare (sau arborii binari ordonați) sunt arbori binari care îndeplinesc următoarele condiții:

- toate nodurile din subarbolele stâng au valoare mai mică decât valoarea rădăcinii
- toate nodurile din subarborele drept au valoarea mai mare decât valoarea rădăcinii
- subarborele drept și cel stâng sunt, de asemenea, arbori de căutare
- nu conțin două noduri cu aceeași valoare

3 Nodul unui arbore binar

Un nod dintr-un arbore binar este un tip de date care e caracterizat prin 3 valori:

- informația propriu-zisă (poate să aibă orice tip și oricâte valori)
- un pointer către fiul stâng
- un pointer către fiul drept

De exemplu, pentru un arbore care conține câte o literă în fiecare nod, îl vom defini astfel:

```
struct nod {
    char val;
    nod *stanga;
    nod *dreapta;
}
```

3.1 Operații

Operațiile uzuale care se execută pe arborii binari de căutare sunt:

- inserarea unui nod
- ștergerea unui nod
- căutarea unei valori
- parcurgerea (preordine, inordine și postordine)
- aflarea minimului și a maximului

3.1.1 Inserarea unui nod

Algoritmul 1 insereaza(radacina, x)

```
if radacina = null then
  radacina = nod_nou(x)
end if
if radacina.val = x then
  {valoarea există deja}
  return
else
  if radacina.val > x then
    if radacina.stanga = null then
      radacina.stanga ← nod_nou(x)
    else
      insereaza(radacina.stanga, x)
    end if
  else
    if radacina.dreapta = null then
      radacina.dreapta ← nod_nou(x)
    else
      insereaza(radacina.dreapta, x)
    end if
  end if
end if
```

3.1.2 Ștergerea unui nod

Există 3 situații diferite de ștergere a unui nod:

1. nodul de șters este o frunză: se șterge nodul și părintele va indica spre null (Figura 4)

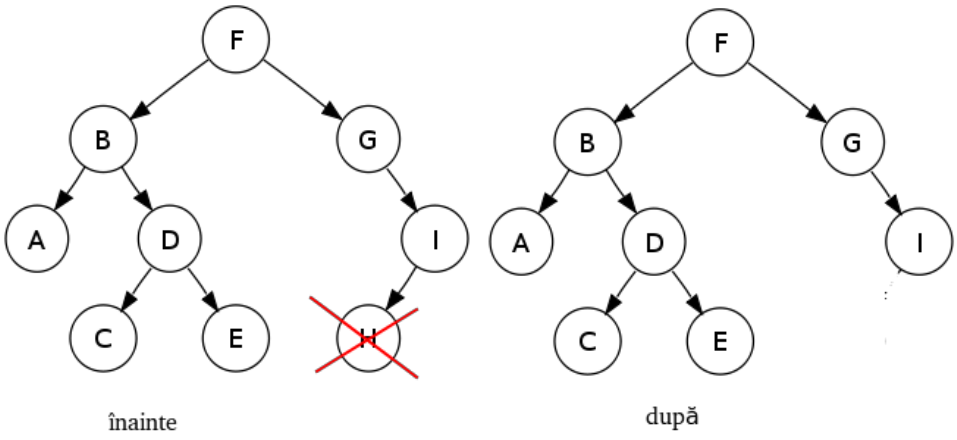

Figura 4: Ștergerea valorii H

2. nodul de șters are un singur fiu: se șterge nodul, iar fiul său va deveni fiul părintelui său (Figura 5)

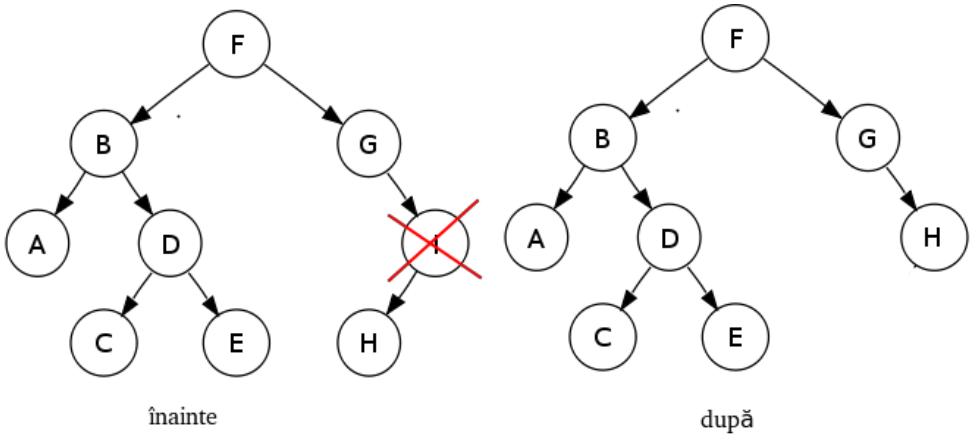

Figura 5: Ștergerea valorii I

3. nodul de șters are 2 fii:
   (a) se găsește cel mai din stânga nod al fiului din dreapta, se șterge acesta și se pune valoarea lui în locul nodului care trebuia șters (Figura 6)
   (b) se găsește cel mai din dreapta nod al fiului din stânga, se șterge acesta și se pune valoarea lui în locul nodului care trebuia șters (Figura 7)

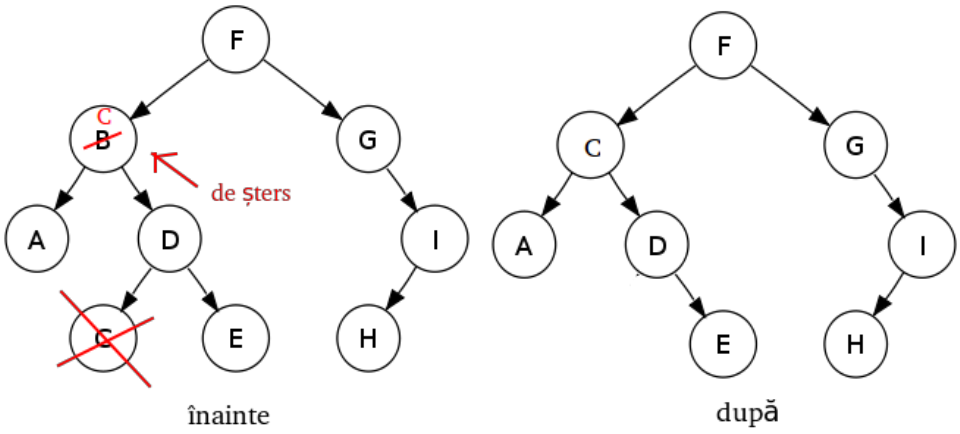

Figura 6: Ștergerea valorii B (varianta 3a)

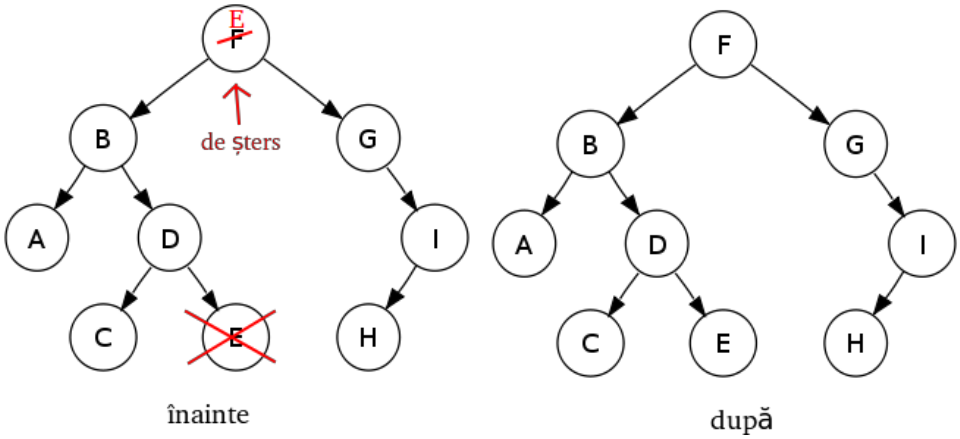

Figura 7: Ștergerea valorii F (varianta 3b

Algoritmul 2 sterge(radacina, x)

```
if radacina = null then
  {nodul nu a fost găsit}
  return r
end if
if radacina.val > x then
  radacina.stanga ← sterge(radacina.stanga, x)
else
  if radacina.val < x then
    radacina.dreapta ← sterge(radacina.dreapta, x)
  else
    {am găsit nodul căutat}
    if radacina.stanga and radacina.dreapta then
      {nodul are 2 fii, folosim varianta 3a de ștergere}
      tmp ← minim(radacina.dreapta)
      radacina.val ← tmp.val
      radacina.dreapta ← sterge(radacina.dreapta, tmp.val)
    else
      {nodul are 1 fiu sau niciunul}
      tmp ← radacina
      if radacina.stanga ≠ null then
        radacina ← radacina.stanga
      else
        radacina ← radacina.dreapta
      end if
      eliberează tmp
    end if
  end if
end if
return radacina
```

3.1.3 Căutarea unei valori

Algoritmul 3 cauta(radacina, x)

```
if radacina = null then
  return null
end if
if radacina.val = x then
  return radacina
else
  if radacina.val > x then
    return cauta(radacina.stanga, x)
  else
    return cauta(radacina.dreapta, x)
  end if
end if
```

3.1.4 Parcurgerea

Algoritmul 4 preordine(radacina)

```
{rădăcină - stânga - dreapta}
if radacina = null then
  return
end if
afiseaza(radacina.val)
preordine(radacina.stanga)
preordine(radacina.dreapta)
```

Algoritmul 5 inordine(radacina)

```
{stânga - rădăcină - dreapta}
if radacina = null then
  return
end if
inordine(radacina.stanga)
afiseaza(radacina.val)
inordine(radacina.dreapta)
```

Algoritmul 6 postordine(radacina)

```
{stânga - dreapta - rădăcină}
if radacina = null then
  return
end if
postordine(radacina.stanga)
postordine(radacina.dreapta)
afiseaza(radacina.val)
```

Parcurgând arborele din Figura 3 în cele 3 moduri obținem următoarele secvențe:

```
Preordine:
F B A D C E G I H

Inordine:
A B C D E F G H I

Postordine:
A C E D B H I G F
```

3.1.5 Aflarea minimului și a maximului

Valoarea minimă dintr-un arbore de căutare se găsește în cel mai din stânga nod, iar valoarea maximă se găsește în cel mai din dreapta nod (Figura 8).

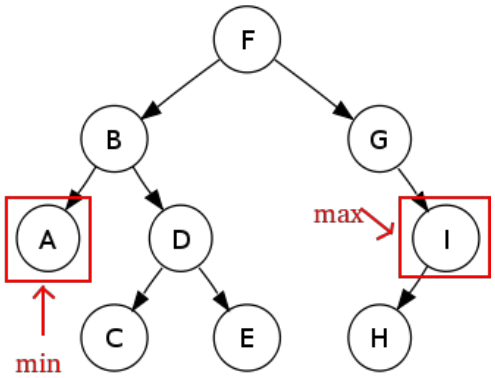

Figura 8: Minimul și maximul

Algoritmul 7 minim(radacina)

```
{cel mai din stânga nod}
if radacina = null then
  afișează: arborele e vid
  return null
end if
if radacina.stanga = null then
  return radacina
else
  return minim(radacina.stanga)
end if
```

Algoritmul 8 maxim(radacina)

```
{cel mai din dreapta nod}
if radacina = null then
  afișează: arborele e vid
  return null
end if
if radacina.dreapta = null then
  return radacina
else
  return minim(radacina.dreapta)
end if
```

4 Discuție

Căutarea unei valori într-un arbore binar de căutare este mai eficientă decât în cazul unei liste înlănțuite, de asemenea, și maximul, și minimul se află mai rapid (bineînțeles, dacă arborele nu este foarte dezechilibrat).
Din punctul de vedere al spațiului de stocare, arborii au nevoie de mai multă memorie decât listele simplu înlănțuite deoarece rețin 2 pointeri în plus față de valoarea propriu-zisă din nod, spre deosebire de listele simplu înlănțuite care au nevoie doar de un singur pointer în plus. O problemă a arborilor binari de căutare este faptul că forma lor este dependentă de oridinea în care sunt inserate nodurile, iar într-un arbore neechilibrat unele operații vor deveni ineficiente. În Figura 9 sunt ilustrați doi arbori: unul neechilibrat (diferența de nivel dintre doi subarbori >= 2) și unul echilibrat (diferența de nivel dintre doi subarbori < 2).

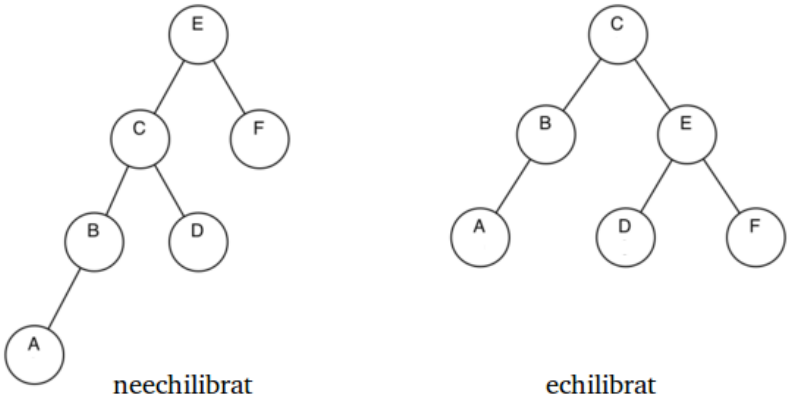

Figura 9: Arbori neechilibrați și echilibrați