Structuri de date

Laboratorul 6

Mihai Nan
mihai.nan.cti@gmail.com
Grupa 312aCC

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2016 - 2017

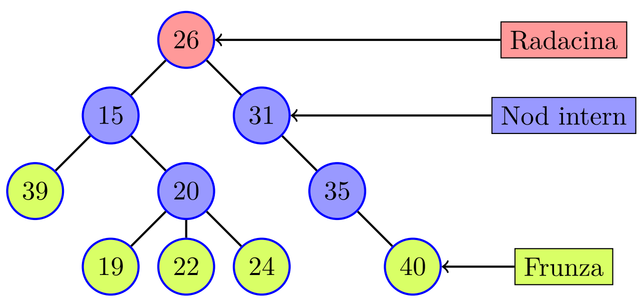
1 Arbori

1.1 Introducere

Un arbore combina avantajele oferite de alte doua structuri: tablourile si listele inlantuite. Arborii permit, pe de o parte, executarea unor cautari rapide, la fel ca si tablourile ordonate, iar, pe de alta parte, inserarea si stergerea elementelor sunt rapide, la fel ca la o lista simplu inlantuita.

Daca ne-am uitat la un arbore genealogic, sau la o ierarhie de comanda intr-o firma, am observat informatiile aranjate intr-un arbore. Un arbore este compus dintr-o colectie de **noduri**, care sunt unite prin **arce**, unde fiecare nod are asociata o anumita informatie si o colectie de copii. Vom reprezenta nodurile prin cercuri, iar arcele, prin linii care unesc cercurile. **Copiii** unui nod sunt acele noduri care urmeaza imediat sub nodul insasi. **Parintele** unui nod este acel nod care se afla imediat deasupra. **Radacina** unui arbore este acel nod unic, care nu are niciun parinte.

Nodurile reprezinta, de regula, entitati din lumea reala, cum ar fi valori numerice, persoane, parti ale unei masini, rezervari de bilete de avion. Nodurile sunt elemente obisnuite pe care le putem memora in orice alta structura de date. Arcele dintre noduri reprezinta modul in care nodurile sunt conectate. Este usor si rapid sa ajungem de la un nod la altul daca acestea sunt conectate printr- un arc. Arcele sunt reprezentate in C si C++ prin **pointeri**.



Toti arborii au urmatoarele proprietati:

- Exista o singura radacina.
- Toate nodurile, cu exceptia radacinii, au exact un parinte.
- Nu exista cicluri. Aceasta inseamna ca pornind de la un anumit nod, nu exista un anumit traseu pe care il putem parcurge, astfel incat sa ajungem inapoi la nodul de plecare.

Inaltimea unui arbore este, de fapt, valoarea maxima de pe nivelurile nodurilor terminale. Numarul de descendenti directi ai unui nod reprezinta **ordinul** sau **gradul nodului**. **Ordinul** sau **gradul arborelui** este valoarea maxima luata de gradul unui nod component al arborelui.

2 Arbore binar

Arborii binari sunt un tip aparte de arbori, in care fiecare nod are maxim 2 copii. Pentru un nod dat, intr-un arbore binar, vom avea fiul din stanga si fiul din dreapta. Deci, un nod dintr-un arbore binar poate avea 2 copii (stanga si dreapta), un singur copil (doar stanga sau doar dreapta) sau niciun copil. Nodurile care nu au niciun copil se numesc **noduri frunza**, iar cele care au 1 sau 2 copii se numesc **noduri interne**.

Intre numarul **N** de noduri al unui arbore binar si inaltimea sa **H** exista relatiile:

$$H \leq N \leq 2^H - 1 \tag{1}$$

$$\log_2 N \leq H \leq N \tag{2}$$

2.1 Arbore binar de cautare

Intr-un **arbore binar de cautare**, pentru fiecare nod cheia acestuia are o valoare mai mare decat cheile tuturor nodurilor din subarborele stang si mai mica decat cheile nodurilor ce compun subarborele drept. Arborii binari de cautare permit mentinerea datelor in ordine si o cautare rapida a unei valori, ceea ce ii recomanda pentru implementarea de multimi si dictionare ordonate.

Observatie

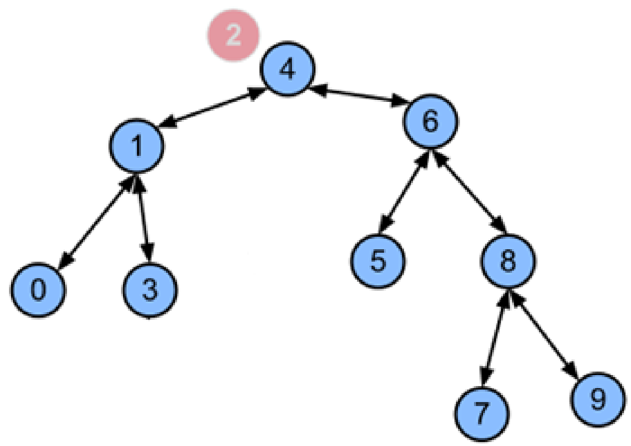
O importanta proprietate a arborilor de cautare, este aceea ca **parcurgerea in inordine** produce o secventa ordonata crescator a cheilor din nodurile arborelui.
Valoarea maxima dintr-un arbore binar de cautare se afla in nodul din **extremitatea dreapta** si se determina prin coborarea pe subarborele drept, iar **valoarea minima** se afla in nodul din **extremitatea stanga**.

Pentru retinerea informatiei, vom folosi alocarea dinamica. Pentru fiecare nod in parte este necesar sa se retina, pe langa informatia utila, si legaturile cu nodurile copii (adresele acestora), iar pentru aceasta uzitam pointeri. In definirea de mai jos, ***left*** reprezinta adresa nodului copil din stanga, ***value*** reprezinta campul cu informatie utila, iar ***right*** este legatura cu copilul din dreapta.

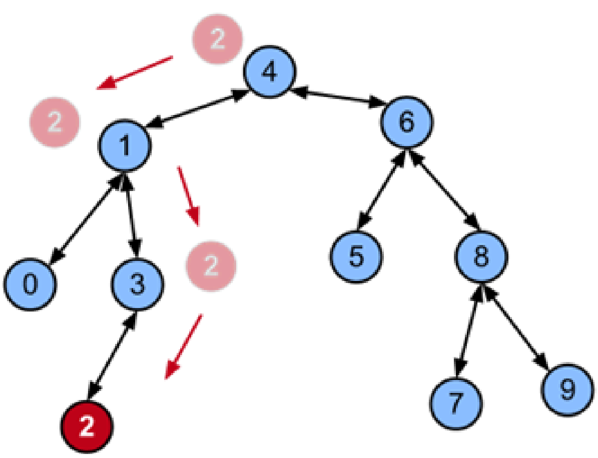
Cod sursa C

```c
typedef struct _tree {
  int value;
  struct _tree* left;
  struct _tree* right;
}*Tree;
```

2.1.1 Inserarea unui nod



Sa consideram arborele de mai sus si sa presupunem ca dorim sa inseram nodul cu valoarea 2. Acesta se va insera ca nod frunza. Pentru a-l insera va trebui sa cautam o pozitie in arbore care respecta regula de integritate a arborilor binari de cautare.

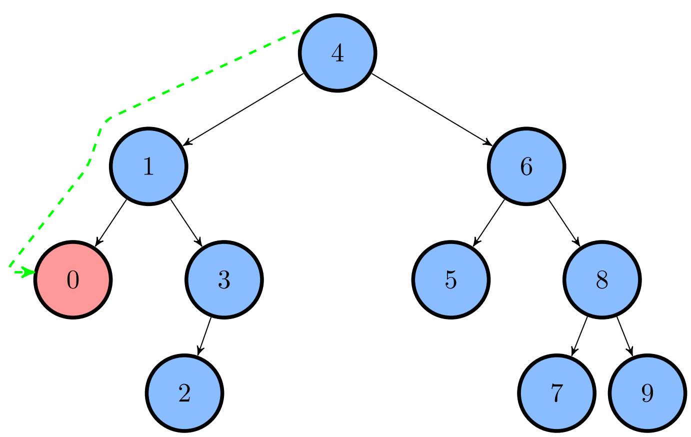


Vom incepe prin compararea nodului de inserat (2) cu radacina arborelui date (4). Observam ca este mai mic decat ea, deci va trebui inserat undeva in subarborele stang al acesteia.Vom compara apoi 2 cu 1. Din moment ce 2 este mai mare decat 1, nodul cu valoarea 2 va trebui plasat undeva in subarborele drept al lui 1. Se compara apoi 2 cu 3. Deoarece 3 este mai mare decat 2, nodul cu valoarea 2 trebuie sa se afle in subarborele din stanga al nodului 3. Dar nodul 3 nu are niciun copil in partea stanga. Asta inseamna ca am gasit locatia pentru nodul 2. Tot ceea ce mai trebuie facut este sa modificam in nodul 3 adresa catre fiul sau stang, incat sa indice spre 2. (Vezi fig. de mai sus!)

Algorithm 1 Insert

```
procedure INSERT(root, value)
    if root = NULL then
        root ← initTree(value)
        return root
    else if root → value = value then
        print(Nodul exista)
        return root
    else
        if root → value > value then
            if root → left = NULL then
                root → left ← initTree(value)
                return root
            else
                root → left ← insert(root → left, value)
                return root
        else
            if root → right = NULL then
                root → right ← initTree(value)
                return root
            else
                root → right ← insert(root → right, value)
                return root
```

2.1.2 Determinarea elementului minim

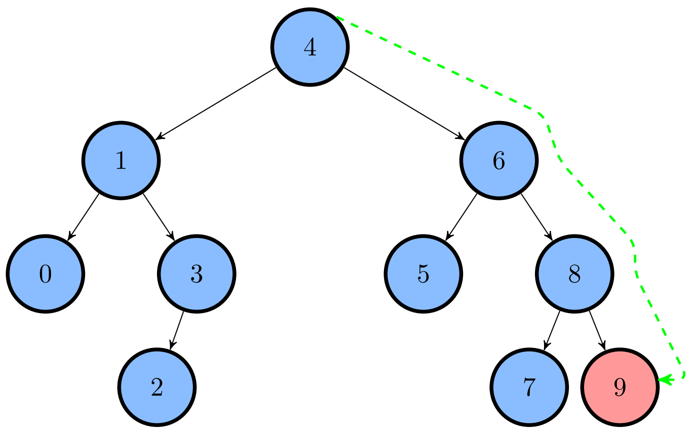


Pentru determinarea valorii minime, ne deplasam in fiul stang al radacinii. De acolo, in fiul stang al acelui fiu s. a. m. d. , pana cand ajungem la un nod care nu mai are niciun fiu stang. Acest nod contine valoarea minima din arbore. (Vezi fig. de mai sus)

Algorithm 2 FindMinimum

procedure FINDMINIMUM(root)
  if $root \neq$ **NULL then**
    while $root \rightarrow left \neq$ **NULL do**
      $root \leftarrow root \rightarrow left$
    return root $\rightarrow$ **value**

2.1.3 Determinarea elementului maxim

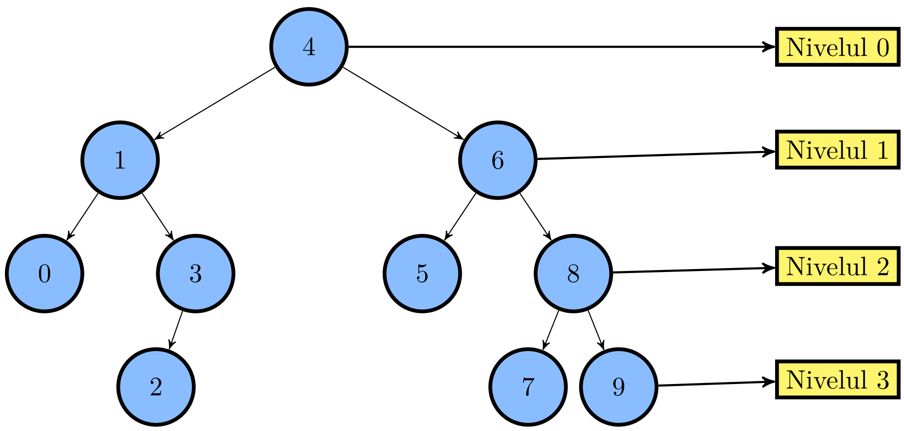


Pentru determinarea nodului cu valoarea maxima procedam similar, ca si in cazul determinarii minimului, avansand mereu spre fiul drept, pana cand gasim un nod care nu mai are niciun fiu drept. Acest nod va contine valoarea maxima. (Vezi fig. de mai sus)

Algorithm 3 FindMaximum

procedure FINDMAXIMUM(root)
  if $root \neq$ **NULL then**
    while $root \rightarrow right \neq$ **NULL do**
      $root \leftarrow root \rightarrow right$
    return root $\rightarrow$ **value**

2.1.4 Inaltimea arborelui



Algorithm 4 Height

procedure HEIGHT(root)
    if $root = $ **NULL then**
        return 0
    else
        $tmp \leftarrow root$
        $left \leftarrow$ ***height***$(tmp \rightarrow left)$
        $right \leftarrow$ ***height***$(tmp \rightarrow right)$
        if $left \geq right$ **then**
            $left \leftarrow left + 1$
            return left
        else
            $right \leftarrow right + 1$
            return right

2.1.5 Stergerea unui element

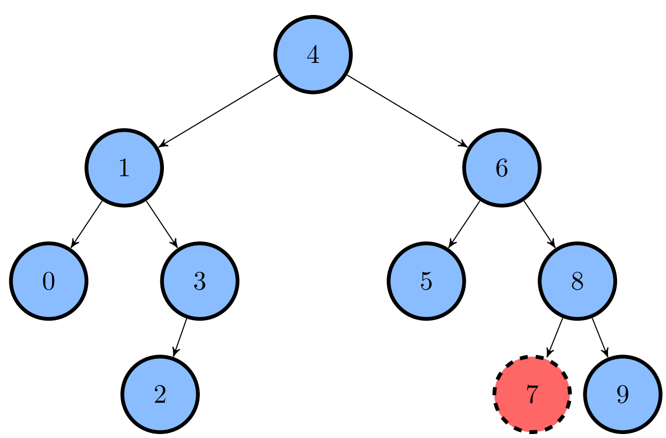
Operatia de stergere a unui nod presupune urmatoarele etape: mai intai, se cauta nodul care va fi sters, iar dupa ce am gasit nodul apar trei cazuri pe care le analizam separat. Cele trei cazuri posibile sunt urmatoarele: nodul care urmeaza sa fie sters este o frunza (nu are fii), nodul are un singur fiu sau nodul are doi fii.

Cazul I: Pentru a sterge un nod frunza se modifica pointerul catre nodul parinte, atribuindu-i acestuia valoarea ***NULL***. Trebuie sa eliminam explicit nodul din memorie, apeland functia ***free***.

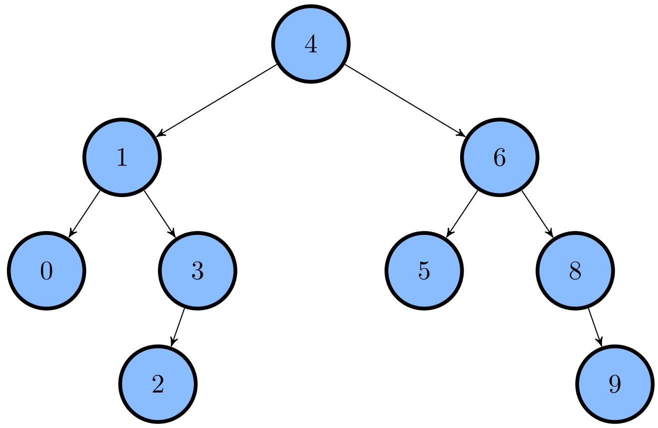
Cazul II: Nodul sters are doua legaturi: una catre parinte si cealalta catre unicul descendent.Vom taia nodul din aceasta secventa, conectand direct singurul fiu al nodului cu nodul parinte. Pointerul corespunzator al parintelui va fi modificat, astfel incat sa indice spre fiul nodului sters.

Cazul III: Daca nodul sters are doi fii, nu il putem inlocui pur si simplu cu unul dintre acestia, cel putin in cazul in care fiul are la randul sau alti fii. Intr-un arbore binar de cautare, nodurile sunt in ordinea crescatoare a cheilor. Pentru un anumit nod, nodul cu cheia imediat superioara se numeste ***succesorul in inordine*** al nodului, sau, pur si simplu, ***succesorul*** nodului. Pentru a sterge un nod cu doi fii, vom inlocui nodul sters cu succesorul sau. Aceasta operatie pastreaza ordinea elementelor. Operatia se complica daca succesorul nodului sters are la randul sau succesori. Algoritmul se deplaseaza in fiul drept al nodului sters, a carui cheie este mai mare. In urmatorul pas, se efectueaza o deplasare in fiul stang al fiului drept (daca exista un astfel de fiu), continuandu-se pe cat posibil deplasarea pe o cale alcatuita numai din fii stangi. Ultimul nod din aceasta cale este succesorul nodului initial. Nodul pe care il cautam este minimul multimii de noduri care sunt mai mari decat nodul original. Cand ne deplasam in subarborele drept, toate nodurile de acolo sunt mai mari decat cel initial, aceasta rezultand din modul de definire a unui arbore binar de cautare. Dorim sa determinam cea mai mica valoare din acest subarbore. Valoarea minima dintr-un subarbore se poate determina urmand calea care porneste de la radacina si merge numai prin fiii stangi. Algoritmul va determina valoarea minima care este mai mare decat nodul original. Aceasta valoare este chiar succesorul cautat. Daca fiul drept al nodului nu are fii stangi, succesorul este el insusi.

Exemplu - Cazul I

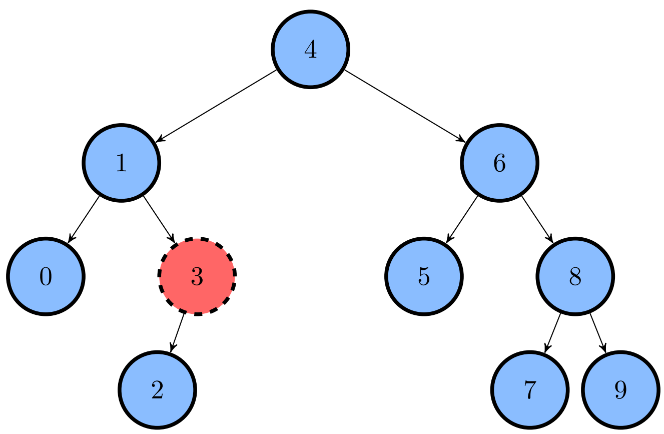


Eliminarea nodului cu valoare 7

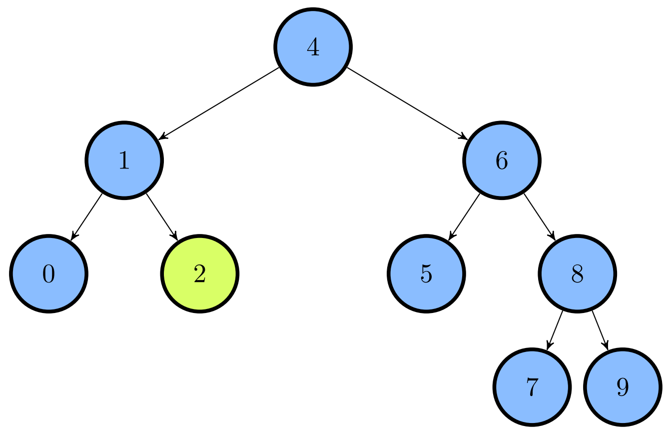


Arborele rezultat in urma operatiei de stergere

Exemplu - Cazul II

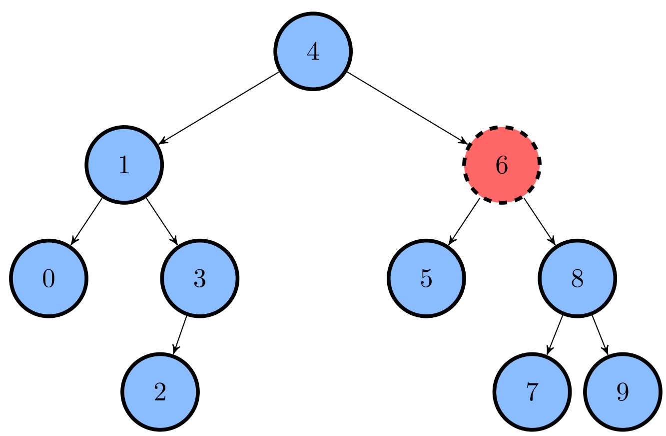


Eliminarea nodului cu valoare 3

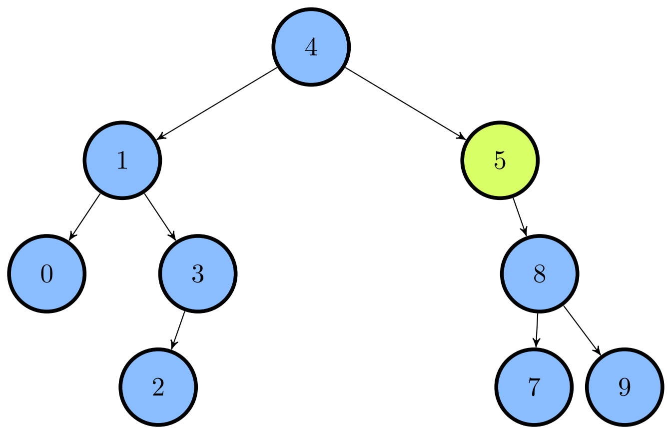


Arborele rezultat in urma operatiei de stergere

Exemplu - Cazul III



Eliminarea nodului cu valoare 6



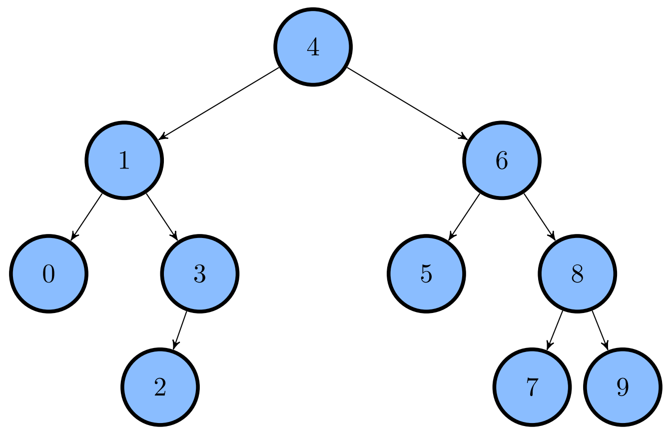
Arborele rezultat in urma operatiei de stergere

Algorithm 5 Delete

```
procedure DELETE(root, x)
    if root = NULL then
        print(Nodul nu a fost gasit)
        return root
    if root → value > value then
        root → left ← delete(root → left, x)
    else if root → value < value then
        root → right ← delete(root → right, x)
    else                                                // Am gasit nodul cautat
        if root → left ≠ NULL and root → right ≠ NULL then
                                                        // Nodul are 2 fii - cazul III
            tmp ← findMinimum(root → right)
            root → value ← tmp → value
            root → right ← delete(root → right, tmp → value)
        else                // Nodul are un fiu sau este frunza - cazurile I si II
            tmp ← root
            if root → left ≠ NULL then
                root ← root → left
            else
                root ← root → right
            free(tmp)
    return root
```

2.1.6 Parcurgerea arborelui

Exista 3 tipuri de parcurgere a unui arbore: **inordine**, **preordine** si **postordine**. Aceste denumiri corespund modului cum este vizitata radacina.



Parcurgerea in inordine

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Algorithm 6 Inordine

---

procedure INORDINE(root)
    if $root \neq$ **NULL then**
        $\boldsymbol{inordine}(root \rightarrow left)$
        $\boldsymbol{print}(root \rightarrow value)$
        $\boldsymbol{inordine}(root \rightarrow right)$

Parcurgerea in preordine

4	**1**	**0**	**3**	**2**	**6**	**5**	**8**	**7**	**9**

Algorithm 7 Preordine

---

procedure PREORDINE(root)
    if $root \neq$ **NULL then**
        $\boldsymbol{print}(root \rightarrow value)$
        $\boldsymbol{preordine}(root \rightarrow left)$
        $\boldsymbol{preordine}(root \rightarrow right)$

Parcurgerea in postordine

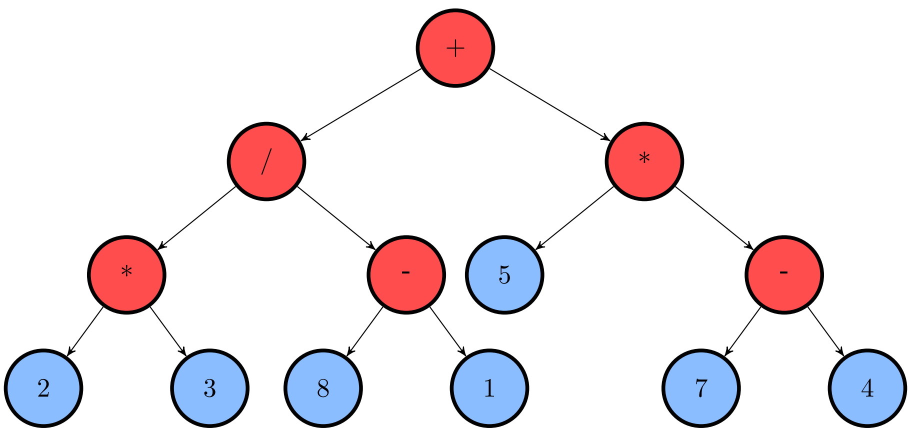
0	**2**	**3**	**1**	**5**	**7**	**9**	**8**	**6**	**4**

Algorithm 8 Postordine

```
procedure POSTORDINE(root)
    if root ≠ NULL then
        postordine(root → left)
        postordine(root → right)
        print(root → value)
```

Observatie

Parcurgerile in ***preordine*** si ***postordine*** sunt utile pentru programe care analizeaza expresii aritmetice. O expresie aritmetica poate fi reprezentata printr-un arbore binar. Nodul radacina contine un operator, iar fiecare din subarborii sai reprezinta fie numele unui variabile, fi o alta expresie.



2*3/(8 - 1)+5*(7-4)

IMPORTANT !

Din exemplul prezentat mai sus, se observa faptul ca parcurgerea in inordine va afisa, de fapt, arborele ordonat crescator.

3 Probleme de laborator

Observatie

In arhiva laboratorului, gasiti un schelet de cod de la care puteti porni implementarea functiilor propuse, avand posibilitatea de a le testa functionalitatea.

3.1 Probleme standard

Problema 1 - 4 punct

Pornind de la definitia pusa la dispozitie in scheletul de cod, implementati umratoarele operatii pentru un arbore binar de cautare:

- inserarea unui nod in arbore si verificarea existentei unui nod;
- parcurgerea arborelui in inordine, preordine si postordine;
- determinarea elementului maxim si a celui minim.

Problema 2 - 2 puncte

Definiti o functie care sa calculeze inaltimea unui arbore binar de cautare.

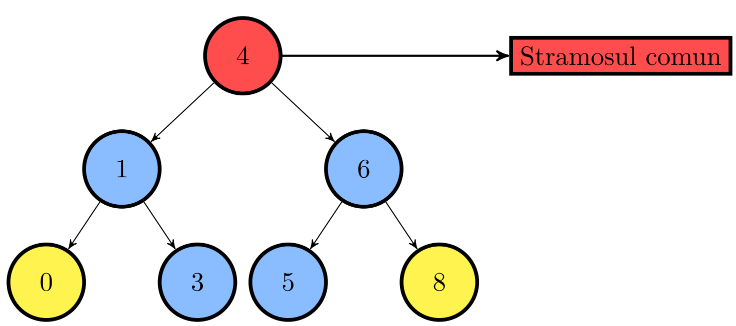
Problema 3 - 2 puncte

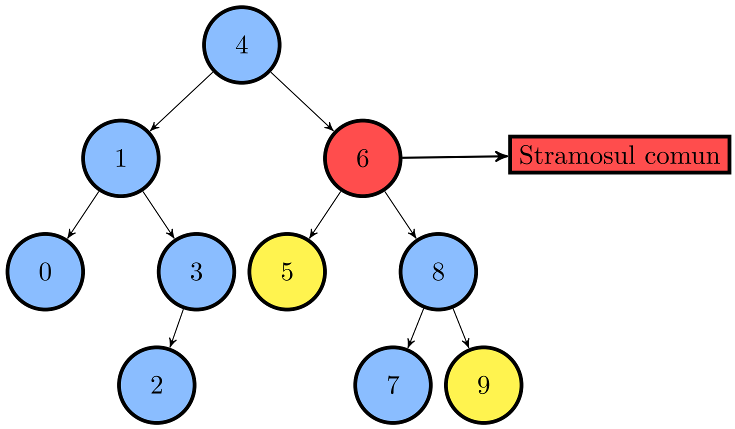
Sa se implementeze operatia de stergere a unui nod din arborele binar de cautare.

Problema 4 - 2 puncte

Implementati o functie care determina cel mai apropiat stramos comun a doua noduri date.

Exemplu



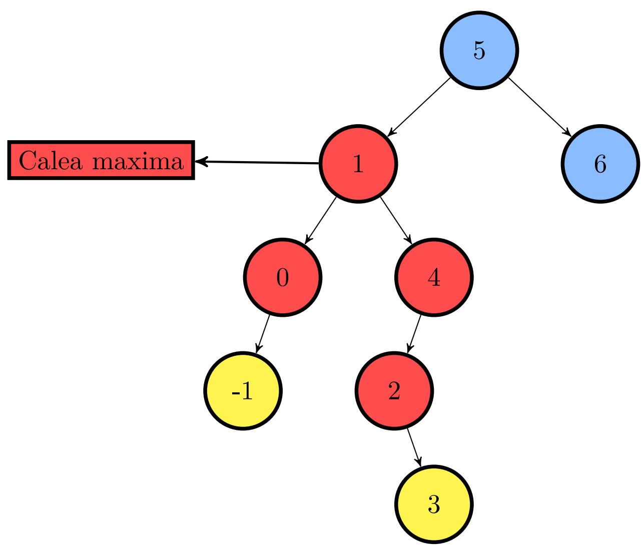


3.2 Problema bonus

Problema 5 - **2 puncte**

Sa se implementeze o functie pentru calculul ***diametrului*** unui arbore binar de cautare. ***Diametrul*** unui arbore binar se poate defini ca distanta maxima dintre doua frunze (numarul de noduri din cea mai lunga cale dintre doua frunze).

Exemplu



Diametrul arborelui este 6

4 Interviu

Observatie

Aceasta sectiune este una optionala si incearca sa va familiarizeze cu o serie de intrebari ce pot fi adresate in cadrul unui interviu tehnic. De asemenea, aceasta sectiune poate fi utila si in pregatirea pentru examenul final de la aceasta disciplina.

Intrebari interviu

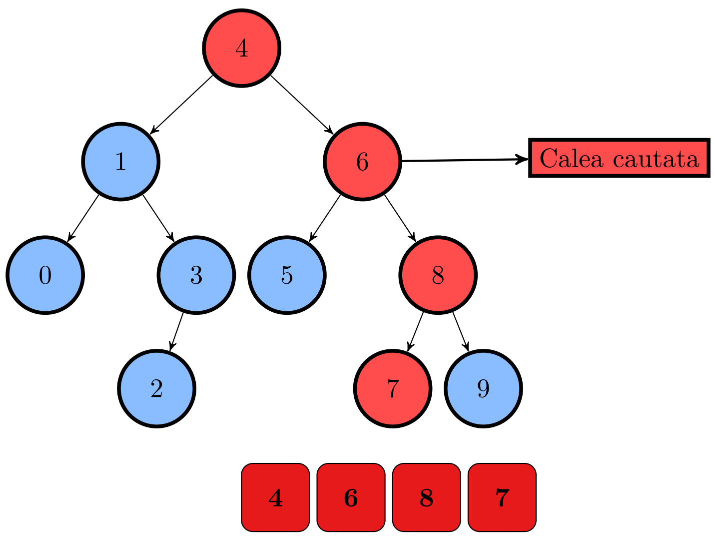
1. Consideram un arbore de cautare initial vid. Desenati arborele de cautare obtinut prin inserarea pe rand a cheilor ***P,R,O,B,L,E,M,A,F,O,A,R,T,E,U,S,O,A,R,A***.
2. Consideram un arbore binar reprezentat prin vectorii ***St = (2, 4, 0, 5, 0, 0)***, ***Dr = (3, 0, 0, 6, 0, 0)***, in care radacina este nodul cu cheia 1. Cate frunze are acest arbore? Ce puteti spune despre acest arbore?
3. Intr-un arbore binar cu 9 varfuri, in care toate varfurile ce nu sunt frunze au exact 2 fii, care este adancimea maxima?
4. Cu cat este egala lungimea prefixului in cazul unui arbore de regasire?
5. Ce avantaje prezinta un arbore, comparativ cu restul structurilor de date studiate?
6. Ce formule exista intre numarul ***N*** de noduri al unui arbore binar si inaltimea sa ***H***?
7. Ce reprezinta succesorul in inordine al unui nod intr-un arbore binar de cautare?
8. Ce se intampla daca intr-un arbore binar de cautare introducem un sir, de numere intregi, sortat crescator? Dar in cazul unui sir sortat descrescator? Argumentati!

4.1 Probleme interviu

Problema 1: Implementati o functie care verifica daca intr-un arbore binar de cautare exista o cale compusa din noduri ale caror chei au suma **K**. In cazul in care exista o astfel de cale de la radacina la oricare nod ar alborelui, afisati cheile nodurilor care compun aceasta cale, in ordinea in care acestea apar in reprezentarea arborelui.

Exemplu

K = 25



Problema 2: Sa se implementeze o functie pentru inserarea unui nod intr-un arbore binar de cautare, fara a uzita recursivitatea.

Problema 3: Implementati un program care sa construiasca un arbore binar de cautare pornind de la sirurile rezultate in urma parcurgerii sale in preordine si inordine.

Exemplu

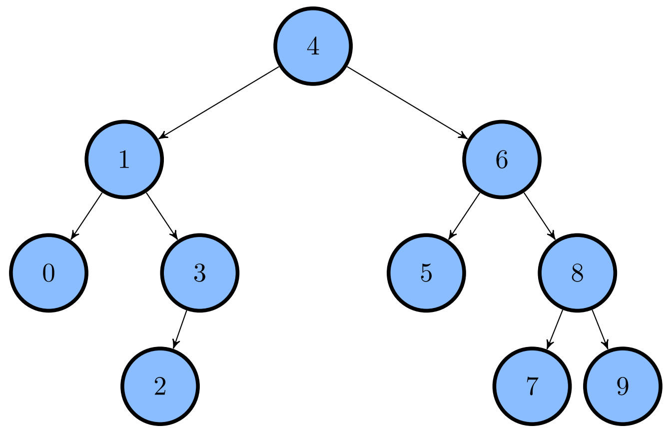
Parcurgerea in inordine

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Parcurgerea in preordine

4	**1**	**0**	**3**	**2**	**6**	**5**	**8**	**7**	**9**

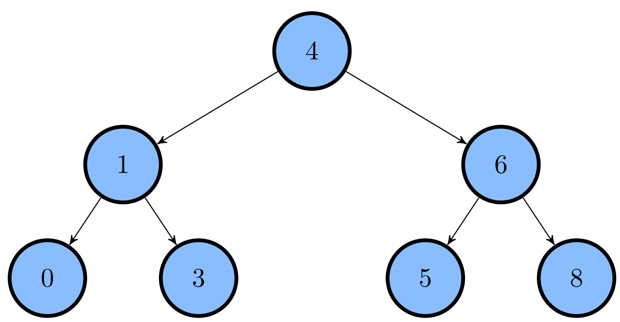
Arborele rezultat



Problema 4: Dandu-se un arbore binar de cautare si valoarea unei chei a unui nod, implementati o functie care determina toti stramosii acelui nod din arbore. In cazul in care nodul este chiar radacina arborelui, se va afisa mesajul ***Radacina arborelui***.

Problema 5: Implementati o functie ce realizeaza parcurgerea unui arbore binar de cautare in **Zig - Zag**.

Exemplu

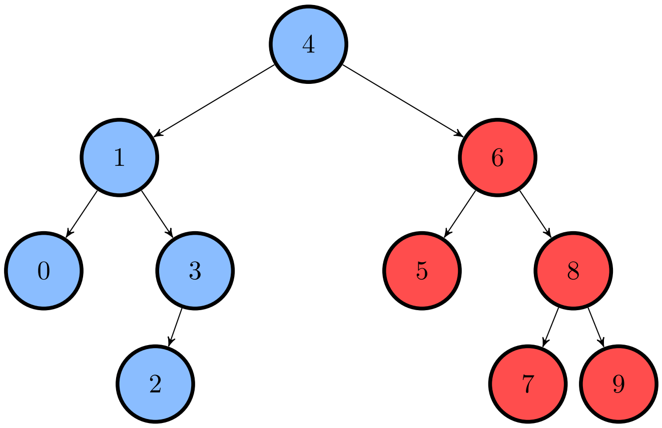


Parcurgerea in Zig - Zag

4	**6**	**1**	**0**	**3**	**5**	**8**

Problema 6: Definiti o functie care primeste ca argument un arbore binar de cautare si determina subarborele maximal al acestuia, ca numar de noduri. Pentru a se testa functionalitatea acestei functii, se va afisa parcurgerea in postordine a subarborelui gasit. In cazul in care exista doi subarbori, avand acelasi numar de noduri in componenta, se va alege subarborele de suma maxima.

Exemplu



Problema 7: Implementati o functie care imbina doi arbori binari de cautare, afisand parcurgerea in inordine a arborelui binar rezultat prin imbinarea acestora.

⚠ IMPORTANT !

⚠ Aceasta functie se poate uzita in cazul operatiei de stergere a radacinei unui arbore binar de cautare.

Feedback

Pentru imbunatatirea constanta a acestui laborator, va rog sa completati formularul de feedback disponibil aici.

De asemenea, va rog sa semnalati orice greseala / neclaritate depistata in laborator pentru a o corecta.

Va multumesc anticipat!