Arbori

1. Obiectivele lucrării

În această lucrare se vor studia principalele modalități de generare, reprezentare, parcurgere, afişare şi ştergere a arborilor.

2. Breviar teoretic

Printr-un **arbore A** se înțelege o mulțime finită de obiecte de date numite noduri cu următoarele particularități:

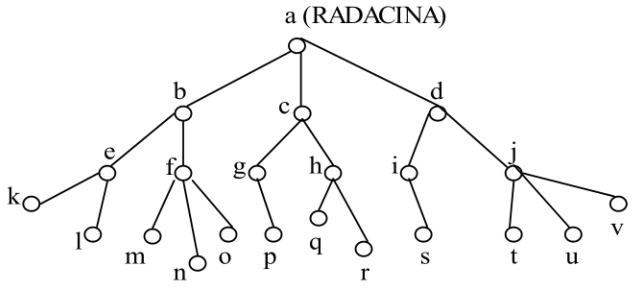
- Există un nod particular numit **rădăcină**,
- Celelalte noduri ale arborelui formează o mulțime finită m ≥1 sau m ≥ 0 de subarbori ale arborelui definit.

Arborele descrie o structură ierarhică. Forma cea mai generală de descriere a unui arbore este următoarea:

A = (N, R), unde:

N - reprezintă mulțimea nodurilor;
R - reprezintă mulțimea relațiilor dintre noduri sau a arcelor.

O formă generală de arbore este următoarea:



Definiții:

- Numărul de subarbori ai unui nod oarecare este **gradul** acelui nod.
- Nodurile de grad 0 se numesc **frunze** sau **noduri terminale (sunt acele noduri din care nu mai rezultă nimic)**.
- Nodurile de grad mai mare sau egal cu 1 se **numesc noduri** interne sau **noduri neterminale**.

- Nodurile care sunt fii ai aceluiaşi nod X (au acelaşi părinte) se numesc **fraţi (noduri înrudite)**.
- **Gradul unui arbore** este gradul maxim al unuia din nodurile sale.
- O **cale** de la un nod a la un nod b al aceluiaşi arbore este definită ca o secvenţă de noduri n_0=a, n_1,...,n_k = b alese astfel încât nodul n_i să fie părintele nodului n_{i+1}.
- Strămoşii unui nod X sunt nodurile aflate pe calea de la X la rădăcina arborelui.
- **Nivelul** unui nod (X) este:
  - 1 în cazul rădăcinii arborelui,
  - n + 1 în cazul unui nod fiu al unui nod plasat pe nivelul n
- Un arbore cu ordinul mai mic sau egal cu 2 se numeşte **arbore binar(deci un arbore la care fiecare nod tată are maxim 2 noduri fiu)**. În caz contrar, arborele se numeşte **multicăi.**

Arborii binari sunt cei mai simpli arbori, motiv pentru care sunt cei mai utilizaţi.

2.1. Modalităţi de reprezentare a arborilor

Toate metodele de reprezentare ale unui arbore încearcă să pună în evidenţă fiii sau părintele unui nod oarecare al arborelui.

Reprezentarea unui arbore binar cu celule alocate dinamic

Un nod al arborelui (structura de memorare asociată acestui nod) este descris prin tipul:

```
typedef struct nod {
        Tip_elem info;
        struct nod *fst, *fdr; //referinţe  la fiii stâng, respectiv drept ai
                               //nodului curent
                } NOD, *ARBORE;
```

Conform acestei modalităţi de reprezentare, arborele este specificat printr-o referinţă la nodul rădăcină.

Reprezentarea cu tablouri

Această reprezentare este utilizată de obicei în aplicaţii dezvoltate sub limbajele care nu permit lucrul cu pointeri. De cele mai multe ori tablourile sunt alocate static.

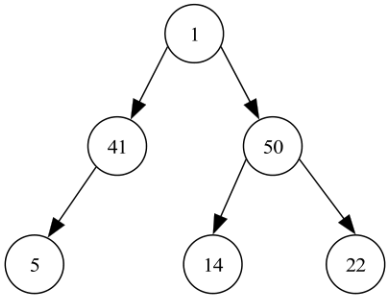


Figura 1. Arbore binar

Se poate folosi un singur tablou global aplicației, ale cărui elemente sunt structuri sau mai multe tablouri "paralele" care conțin respectiv informațiile din nodurile arborelui şi indicii nodurilor fii ai nodului curent.

```
#define NR_MAX_NOD 50

typedef {
        tip_elem info;
        int fst, fdr;
       } NOD;
typedef  NOD ARBORE[NR_MAX_NOD];
```

Arborele binar din figura 1 are reprezentarea cu tablouri prezentata in cele ce urmeaza:

	0	**1**	**2**	**3**	**4**	**5**
info	1	41	50	5	14	22
fst	1	3	4	-1	-1	-1
fdr	2	-1	5	-1	-1	-1

Reprezentarea cu legături fiu-tată

Un arbore poate fi reprezentat prin legături de la fiu la tată. Această reprezentare este adecvată reprezentării colecțiilor de mulțimi disjuncte folosind arbori. Ea permite implementarea eficientă a unor operații cum ar fi reunirea a două mulțimi ale colecției sau găsirea mulțimii ce conține o valoare specificată.

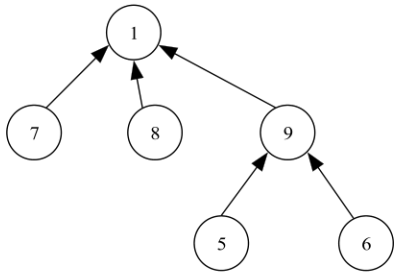


Figura 2. Reprezentarea cu legături fiu-tată

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

Reprezentarea poate fi folosită cu success pentru arborii multicăi. Figura 2 indică o astfel de reprezentare pentru care structura de memorare este un tablou.

```
typedef  struct  {
                tip_elem info;    // informatia asociata nodului curent
                int tata;         // indicele elementului de tablou care
                                  // contine tatal nodului curent
                }NOD;
 typedef  NOD ARBORE[20];
```

Reprezentarea cu liste

Un arbore se poate reprezenta folosind liste in felul urmator:

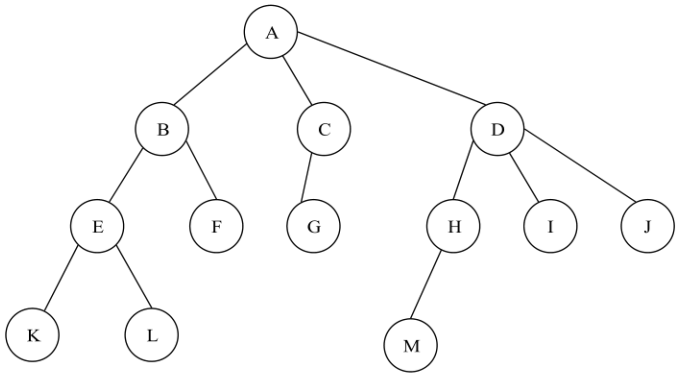


Figura 3. Reprezentarea prin liste

O reprezentare sub forma de lista a arborelui din figura 3 este:

A(B(E(K,L),F),C(G),D(H(M),I,J))

Daca gradul unui arbore este cunoscut, o reprezentare a sa ar putea avea urmatoarea forma:

```
typedef struct {
                tip_elem info;
                int grad_nod;
                struct nod **fii;  // pointer la primul element al
                                   // tabloului de referinte la fii
            } *ARBORE;
```

Reprezentarea fiu stang-frate drept

Reprezentarea arborilor multicai se poate face si sub forma de arbori binari dupa modelul prezentat in figura 4.

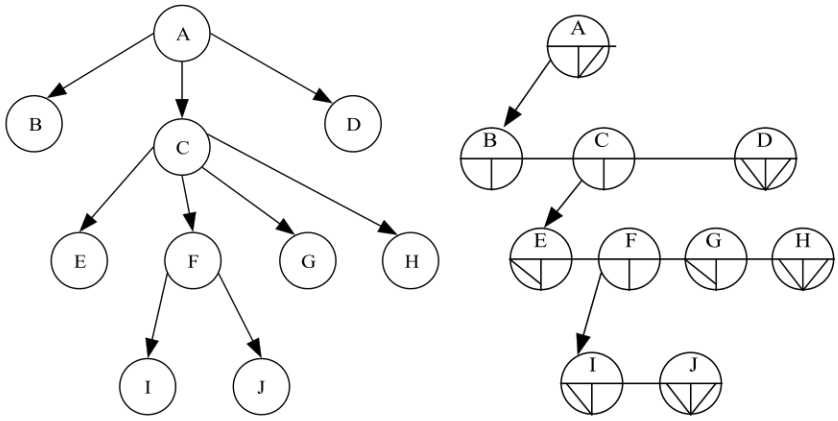


Figura 4. Exemplu de arbore multicăi si arborele binar corespunzator

Uneori această reprezentare este îmbunătățită prin adăugarea în fiecare nod al arborelui a unui pointer care să refere nodul părinte.

2.2. Parcurgerea (traversarea) arborilor

Pentru a determina dacă un arbore dat conține un nod cu o informație căutată trebuie efectuată o operație de căutare în structura

arborelui. Deci trebuie stabilit un procedeu prin care să poată fi vizitat, parcurs orice nod al arborelui până la nodul frunză.
Există două procedee de căutare:

1. **Căutarea transversală** care constă în vizitarea mai întâi a tulpinii, apoi a tuturor fiilor tulpinii, apoi a fiilor fiilor ş.a.m.d. până la frunze.
2. **Căutarea în adâncim**e care constă în: se încearcă să se ajungă pornind de la rădăcină cât mai repede posibil până la prima frunză. La acest tip de căutare, în funcție de ordinea de parcurgere sau de vizitare a nodurilor există trei moduri distincte de parcurgere:
   - **traversarea (parcurgerea) în pre-ordine -** mai este numită şi parcurgerea RSD(Rădăcină, Stânga, Dreapta) şi constă în parcurgerea arborelui în următoarea ordine:
     rădăcină,
     subarborele din stânga,
     subarborele din dreapta.

   Obs: ne referim la arbori binari.

   - **traversarea (parcurgerea) în post-ordine -** mai este numită şi parcurgerea SDR(Stânga, Dreapta, Rădăcină) şi constă în parcurgerea arborelui în următoarea ordine:
     subarborele din stânga,
     subarborele din dreapta,
     rădăcină.
   - **traversarea (parcurgerea) în in-ordine sau traversarea simetrică -** mai este numită şi parcurgerea SRD (Stânga, Rădăcină, Dreapta) şi constă în parcurgerea:
     subarborelui din stânga dinspre frunze,
     rădăcină,
     subarborele din dreapta.

Exemplificare pe caz concret a modurilor de parcurgere a arborilor prezentate mai sus

Considerăm un exemplu de arbore binar reprezentat cu celule alocate dinamic:

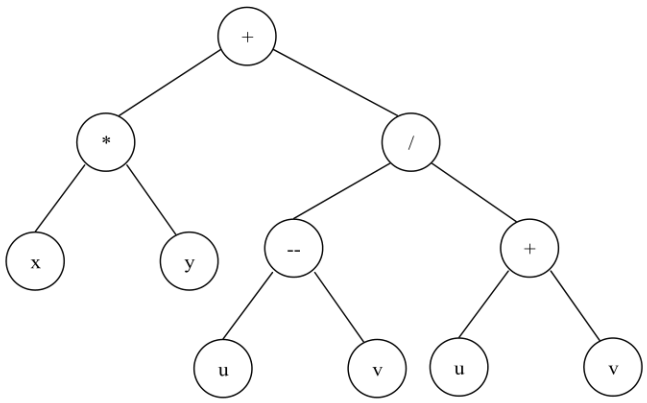


Figura 5 Arbore binar

Parcurgerile menționate anterior vor vizita nodurile arborelui în următoarea ordine:

1. căutare transversală (în lățime) **+ * / x y -- + u v u v**
2. căutarea în pre-ordine (RSD) **+ * x y / -- u v + u v**
3. căutarea în post-ordine (SDR) **x y * u v -- u v + / +**
4. căutarea în in-ordine sau simetrică (SRD):
   x * y + u -- v / u + v

3. Probleme rezolvate

Se vor edita şi apoi executa programele descrise în continuare.

1. Reprezentarea unui arbore cu tablouri paralele. Calculăm numărul de frunze ale arborelui reprezentat.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#define N 3 //nr. noduri
void main()
{
 clrscr();
 int aInfo[N];//informatia din noduri
 int aFst[N];//aFst[i] este nr. nodului fiu stang,
//al nodului i
```

```
 int aFdr[N];//aFdr[i] este nr. nodului fiu
//dreapta, al nodului i
 int i;
 int nFrunze=0;//nr. de frunze
 //Nodul i este nod frunza, daca aFst[i]=aFdr[i]=-1
 //Citirea arborelui:
 for(i=0;i<N;i++){
  printf("nodul %d:
info=",i);scanf("%d",&aInfo[i]);
  printf("nodul %d: nr. nod fiu
stanga=",i);scanf("%d",&aFst[i]);
  printf("nodul %d: nr. nod fiu
dreapta=",i);scanf("%d",&aFdr[i]);
 }
 //calcul nr. noduri frunza:
 for(i=0;i<N;i++)
   if((aFst[i]==-1)&&(aFdr[i]==-1))nFrunze++;
 printf("Sunt %d noduri frunza.",nFrunze);
 getch();
}
```

2. Reprezentarea unui arbore cu legaturi fiu-tata. Calculăm şi afişăm numărul de noduri frunza prezente in arbore.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
int estePrezent(int nr, int a[], int dim);
#define N 3 //nr. noduri
void main()
{
 int aInfo[N];//informatia din noduri
 int aTata[N];//aTata[i] este nr. nodului care este
//tatal nodului i
//prin definitie, tatal radacinii are valoarea -1
 int i;
 int nFrunze=0;//nr. de frunze
//Nodul i este nod frunza, daca el nu apare in
//vectorul aTata
 clrscr();
 //Citirea arborelui:
 for(i=0;i<N;i++){
  printf("nodul %d: info=",i);
  scanf("%d",&aInfo[i]);
```

```
  printf("nodul %d: are ca tata pe nodul
",i);scanf("%d",&aTata[i]);
 }
 //calcul nr. noduri frunza, si afisarea lor:
 //parcurgem multimea nodurilor: {0,1,...,N-1}
 for(i=0;i<N;i++)
   if(!estePrezent(i,aTata,N)){
     nFrunze++;
     printf("\nNodul %d cu info %d, este
frunza.",i,aInfo[i]);
   }
 printf("\nSunt %d noduri frunza.",nFrunze);
 getch();
}

int estePrezent(int nr, int a[], int dim)
{
 //cautare liniara a lui nr, in vectorul a:
 int este=0;
 int i;
 for(i=0;i<dim;i++)
   if(nr==a[i]){
     este=1;
     break;
   }
 return este;
}
```

3. Reprezentarea unui arbore cu liste. Prezentăm în cele ce urmează o funcție care construieşte un arbore binar, preluând informații de la tastatură. Informația caracteristică fiecărui nod poate fi citită cu o rutină specifică. În implementarea propusă, tipul informației din nodurile arborelui este int. Afişarea arborelui o vom face în ***inordine*** cu indentare. Algoritmul de inserare a unui nou nod în arbore este următorul:

- dacă elementul ce se doreşte a fi inserat în arbore este primul atunci va deveni nod rădăcină.
- altfel testăm dacă elementul pe care dorim să-l inserăm este mai mic sau mai mare decât cele deja existente:
  - dacă este mai mic decât rădăcina atunci va merge în partea stângă a ei. Daca în partea stângă a radacinii deja mai exista un nod atunci se repetă testul de mai sus (dacă este mai mare decât fiul stâng al rădăcinii

atunci va merge ca fiu dreapta al acestuia din urmă) până când găseşte un loc liber.

- dacă este mai mare decât rădăcina atunci va merge în partea dreaptă a ei. Daca în partea dreaptă a radacinii deja mai exista un nod atunci se repetă testul de mai sus (dacă este mai mic decât fiul drept al rădăcinii atunci va merge ca fiu stânga al acestuia din urmă) până când găseşte un loc liber.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
//un nod al unui arbore binar
typedef struct tnod{
        int id;//cheia
        tnod* pSt;
        tnod* pDr;}TNOD;
//var. Globala - pointer spre radacina:
TNOD* pRad;
//prototipuri:
void init();
void insert(int x);
void afisare(TNOD* pNodCrt, int nivel);

void main()
{
init();
insert(63);  insert(80);  insert(27);  insert(70);
insert(51);  insert(92);  insert(13);  insert(33);
insert(82);  insert(58);  insert(26);  insert(60);
insert(57);
clrscr();
afisare(pRad,0);
getch();
}

void init()
{
 pRad=NULL;
}
void afisare(TNOD* pNodCrt, int nivel)
//inordine cu indentare:
{
 int i;
```

```
if(pNodCrt !=NULL){
   afisare(pNodCrt->pDr,nivel+1);
   //radacina:
   for(i=0;i<nivel;i++)printf("    ");
   printf("%d\n",pNodCrt->id);
   afisare(pNodCrt->pSt,nivel+1);
 }
}
```

```
void insert(int x)
{
//Gasim nodul parinte al nodului de inserat. Acesta
//se va insera fie ca fiu stanga, fie ca fiu
//dreapta.
 TNOD* pNodNou=new TNOD;
 pNodNou->id=x;
 pNodNou->pSt=NULL;
 pNodNou->pDr=NULL;
 if(pRad==NULL){pRad=pNodNou; return;}
 //unde il inseram pe acest nod nou?
 TNOD* pParinte;
 //plecam de la radacina:
 TNOD* pNodCrt=pRad;
 for(;;){
   pParinte=pNodCrt;
   //deplasare stanga sau dreapta,  in arbore?
   if(x<pNodCrt->id){//deplasare in stanga:
        pNodCrt=pNodCrt->pSt;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu stang.
      pParinte->pSt=pNodNou;
      return;}
   }else{//deplasare dreapta:
        pNodCrt=pNodCrt->pDr;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu drept.
      pParinte->pDr=pNodNou;
      return;}
   }
  }//for
}
```

4. Reprezentarea unui arbore cu liste. Construirea şi afişarea arborelui se face similar ca la exemplul precedent. În plus căutăm şi

afişăm dacă am găsit în nodurile arborelui un anumit element introdus de la tastatură. Căutarea elementului o facem în două moduri:

1. Creem o funcție în care într-un for infinit comparăm informația din fiecare nod al arborelui cu elementul de căutat. În momentul în care elementul a fost găsit funcția returnează 1, în caz contrar, după parcurgerea tuturor nodurilor, returnează 0.
2. Creem o funcție în care căutarea elementului dorit se face ***recursiv.*** Funcția returnează NULL în cazul în care nu există şi nodul respectiv în cazul în care elementul a fost găsit.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>

//un nod al unui arbore binar
typedef struct tnod{
        int id;//cheia
        tnod* pSt;
        tnod* pDr;}TNOD;

//var. globala - pointer spre radacina:
TNOD* pRad;

//prototipuri:
void init();
void insert(int x);
int cautare(int x);
TNOD* find(int x, TNOD* pNodCrt);
void afisare(TNOD* pNodCrt, int nivel);

void main()
{
init();
insert(63);  insert(80);  insert(27);  insert(70);
insert(51);  insert(92);  insert(13);  insert(33);
insert(82);  insert(58);  insert(26);  insert(60);
insert(57);
clrscr();
afisare(pRad,0);
getch();
//elementul pe care dorim sa-l cautam in nodurile
arborelui
```

```
 printf("\nx=");
 int x;
 scanf("%d",&x);
 if(cautare(x)==1)printf("DA\n");
 else printf("NU\n");
 TNOD *p=find(x,pRad);
 //if(find(x,pRad)==NULL)printf("nu\n");
 // else printf("da\n");
 printf("\n%d,  fiu stanga=%d, fiu dreapta=%d,",
p->id,p->pSt->id, p->pDr->id);
getch();
}

void init()
{
 pRad=NULL;
}

void afisare(TNOD* pNodCrt, int nivel)
//inordine cu indentare:
{
 int i;
 if(pNodCrt !=NULL){
   afisare(pNodCrt->pDr,nivel+1);
   //radacina:
   for(i=0;i<nivel;i++)printf("    ");
   printf("%d\n",pNodCrt->id);
   afisare(pNodCrt->pSt,nivel+1);
 }
}

void insert(int x)
{
//Gasim  nodul  parinte  al  nodului  de  inserat.
//Acesta se va insera fie ca fiu stanga, fie ca fiu
//dreapta.
 TNOD* pNodNou=new TNOD;
 pNodNou->id=x;
 pNodNou->pSt=NULL;
 pNodNou->pDr=NULL;
 if(pRad==NULL){pRad=pNodNou; return;}
 //unde-l inseram pe acest nod nou?
 TNOD* pParinte;
 //plecam de la radacina:
```

```
 TNOD* pNodCrt=pRad;
 for(;;){
   pParinte=pNodCrt;
   //deplasare stanga sau dreapta,  in arbore?
   if(x<pNodCrt->id){//deplasare in stanga:
        pNodCrt=pNodCrt->pSt;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu stang.
      pParinte->pSt=pNodNou;
      return;}
   }else{//deplasare dreapta:
        pNodCrt=pNodCrt->pDr;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu drept.
      pParinte->pDr=pNodNou;
      return;}
   }
  }//for
}
```

```
int cautare(int x)
//in arbore binar de cautare
{
 TNOD* pNodCrt;
 pNodCrt=pRad;
 for(;;){
   if(pNodCrt==NULL)return 0;//nu este
   //analizam in nodul curent, cele 3 posibilitati:
  if(x<pNodCrt->id)pNodCrt=pNodCrt->pSt;//cautam
//mai departe in stanga
  else if(x==pNodCrt->id)return 1;//este prezent
  else pNodCrt=pNodCrt->pDr;//cautam mai departe in
//dreapta
 }
}
```

```
TNOD* find(int x, TNOD* pNodCrt)
{
 if(pNodCrt==NULL)return NULL;//nu exista x
 if(x<pNodCrt->id)return find(x,pNodCrt->pSt);
 else if(x>pNodCrt->id)return find(x,pNodCrt->pDr);
 else return pNodCrt;
}
```

5. Reprezentarea unui arbore cu liste. Construirea şi afişarea arborelui se face similar ca la exemplul anterior. În plus prezentăm traversarea arborelui în **in-ordine (SRD), pre-ordine (RSD) şi post-ordine (SDR)**.

Observație: *Traversarea in in-ordine a unui arbore binar de cautare (traversare SRD) va conduce la vizitarea nodurilor, in ordinea crescatoare a valorilor cheilor acestora. Astfel, daca vrem sa obtinem un vector sortat care sa cuprinda toate informatiile dintr-un arbore, vom traversa arborele in in-ordine. Cel mai simplu mod de a traversa un arbore este utilizand recursivitatea. Functia inordine() are ca parametru un pointer spre nodul curent, pNodCrt. Initial, la primul apel, acesta va fi pointerul spre radacina arborelui. In functie se fac trei operatii:*

1. *apel recursiv al aceleiasi functii pentru a traversa subaroborele stang;*
2. *se afiseaza informatia din nodul curent;*
3. *apel recursiv pentru a traversa subarborele drept.*

Sursa programului:

```
#include <stdio.h>
#include <conio.h>

//un nod al unui arbore binar
typedef struct tnod{
        int id;//cheia
        tnod* pSt;
        tnod* pDr;}TNOD;

//var. globala - pointer spre radacina:
TNOD* pRad;

//prototipuri:
void init();
void insert(int x);
void inordine(TNOD* pNodCrt);
void preordine(TNOD* pNodCrt);
void postordine(TNOD* pNodCrt);
void afisare(TNOD* pNodCrt, int nivel);

void main()
{
 init();
```

```
insert(63);  insert(80);  insert(27);  insert(70);
insert(51);  insert(92);  insert(13);  insert(33);
insert(82);  insert(58);  insert(26);  insert(60);
insert(57);
clrscr();
afisare(pRad,0);
getch();
printf("Traversare in inordine (SRD):\n");
inordine(pRad);
getch();
printf("Traversare in preordine (RSD):\n");
preordine(pRad);
getch();
printf("Traversare in postordine (SDR):\n");
postordine(pRad);
getch();
}

void init()
{
 pRad=NULL;
}

void afisare(TNOD* pNodCrt, int nivel)
//inordine cu indentare:
{
 int i;
 if(pNodCrt !=NULL){
   afisare(pNodCrt->pDr,nivel+1);
   //radacina:
   for(i=0;i<nivel;i++)printf("    ");
   printf("%d\n",pNodCrt->id);
   afisare(pNodCrt->pSt,nivel+1);
 }
}

void insert(int x)
{
//Gasim nodul parinte al nodului de inserat. Acesta
//se va insera fie ca fiu stanga,  fie ca fiu
//dreapta.
  TNOD* pNodNou=new TNOD;
 pNodNou->id=x;
 pNodNou->pSt=NULL;
```

```
 pNodNou->pDr=NULL;
 if(pRad==NULL){pRad=pNodNou; return;}
 //unde-l inseram pe acest nod nou?
 TNOD* pParinte;
 //plecam de la radacina:
 TNOD* pNodCrt=pRad;
 for(;;){
   pParinte=pNodCrt;
   //deplasare stanga sau dreapta,  in arbore?
   if(x<pNodCrt->id){//deplasare in stanga:
        pNodCrt=pNodCrt->pSt;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu stang.
      pParinte->pSt=pNodNou;
      return;}
   }else{//deplasare dreapta:
        pNodCrt=pNodCrt->pDr;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu drept.
      pParinte->pDr=pNodNou;
      return;}
   }
  }//for
}
```

```
void inordine(TNOD* pNodCrt)
{
 //SRD
 if(pNodCrt==NULL)return;
 inordine(pNodCrt->pSt);
 printf("%d\n",pNodCrt->id);
 inordine(pNodCrt->pDr);
}

void preordine(TNOD* pNodCrt)
{
 //RSD
 if(pNodCrt==NULL)return;
 printf("%d\n",pNodCrt->id);
 preordine(pNodCrt->pSt);
 preordine(pNodCrt->pDr);
}
```

```
void postordine(TNOD* pNodCrt)
```

```
{
 //SDR
 if(pNodCrt==NULL)return;
 postordine(pNodCrt->pSt);
 postordine(pNodCrt->pDr);
 printf("%d\n",pNodCrt->id);
}
```

6. Reprezentarea unui arbore cu liste. Construirea şi afişarea arborelui se face similar ca la exemplul anterior. În plus prezentăm funcţiile care permit afişarea nerecursivă şi ştergerea unui arbore binar de căutare .

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include<iostream.h>

//un nod al unui arbore binar
typedef struct tnod{
        int id;//cheia
        tnod* pSt;
        tnod* pDr;}TNOD;

//var. globala - pointer spre radacina:
TNOD* pRad;

//prototipuri:
void init();
void insert(int x);
void afisare(TNOD* pNodCrt, int nivel);
int sterge(int key);//nu pot s-i zic delete !
TNOD* getInlocuitor(TNOD* pNodCrt);
void initStiva(int& iV);
void push(TNOD* S[],int& iV, TNOD* x);
TNOD* pop(TNOD* S[], int& iV);
int esteVida(int iV);
void afisareArbore();

void main()
{
 init();
insert(15);   insert(3);   insert(16);   insert(20);
insert(18);  insert(23);
clrscr();
```

```
 afisare(pRad,0);
 afisareArbore();
 getch();
 int optiune;//aleg momentul in care nu mai vreau
sa sterg
 for(;;){
 cout<<endl<<"1 - stergere"<<endl;
 cout<<"2 - iesire"<<endl;
 cin>>optiune;
 switch(optiune){
        case 1:
                printf("\ncheia nod de sters = ");
                int x;
                scanf("%d",&x);
                sterge(x);
                printf("\n\n\n");
                afisare(pRad,0);
                afisareArbore();
                getch();
                break;
        case 2:return;
     }//switch
 }//for(;;)
}

void init()
{
 pRad=NULL;
}

void afisare(TNOD* pNodCrt, int nivel)
//inordine cu indentare:
{
 int i;
 if(pNodCrt !=NULL){
   afisare(pNodCrt->pDr,nivel+1);
   //inf. din radacina subarborelui :
   for(i=0;i<nivel;i++)printf("    ");
   printf("%d\n",pNodCrt->id);
   afisare(pNodCrt->pSt,nivel+1);
 }
}

void insert(int x)
```

```
{
//Gasim nodul parinte al nodului de inserat. Acesta
//se va insrea fie ca fiu stanga, fie ca fiu
//dreapta.
 TNOD* pNodNou=new TNOD;
 pNodNou->id=x;
 pNodNou->pSt=NULL;
 pNodNou->pDr=NULL;
 if(pRad==NULL){pRad=pNodNou; return;}
 //unde-l inseram pe acest nod nou?
 TNOD* pParinte;
 //plecam de la radacina:
 TNOD* pNodCrt=pRad;
 for(;;){
   pParinte=pNodCrt;
   //deplasare stanga sau dreapta,  in arbore?
   if(x<pNodCrt->id){//deplasare in stanga:
        pNodCrt=pNodCrt->pSt;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu stang.
      pParinte->pSt=pNodNou;
      return;}
   }else{//deplasare dreapta:
        pNodCrt=pNodCrt->pDr;
        if(pNodCrt==NULL){
      //nodul parinte, nu mai are nici un fiu drept.
      pParinte->pDr=pNodNou;
      return;}
   }
  }//for
}
```

```
int sterge(int key)
//stergere nod cu cheia key, in arbore binar de
cautare
{
 if(pRad==NULL){cout<<endl<<"Arborele    nu    este
construit - nu are nici un nod";
                    return 0;}
 //cauta nodul:
 TNOD* pNodCrt=pRad;
  //Pt  stergere,  avem  nevoie  sa  stim  si  adresa
//nodului parinte.
 TNOD* pNodParinte;
```

```
#define STANGA 1
 #define DREAPTA 0
 int esteFiu;//semafor
 for(;;){
  //l-a gasit?
  if(pNodCrt->id==key)break;
  //nu este nodul curent trece la fiul lui coresp.:
  pNodParinte=pNodCrt;
  if(key<pNodCrt->id){
       pNodCrt=pNodCrt->pSt;
       esteFiu=STANGA;}
  else if(key>pNodCrt->id){
       pNodCrt=pNodCrt->pDr;
       esteFiu=DREAPTA;}
  if(pNodCrt==NULL) return 0;//nu a gasit cheia in
tot arborele
 }
 //A gasit pNodCrt ce are cheia. Este fiu stanga
//sau dreapta dupa valoarea  semaforului esteFiu.
 //Are fii ?
 if((pNodCrt->pSt==NULL)&&(pNodCrt->pDr==NULL)){
   //nu are nici un fiu. Este nod terminal.
   //Il elimin, direct:
   //Daca e chiar radacina, arborele devine vid:
   if(pNodCrt==pRad)pRad=NULL;
   else if(esteFiu==STANGA)pNodParinte->pSt=NULL;
   else //este fiu dreapta
        pNodParinte->pDr=NULL;
   delete pNodCrt; //eliberam memoria dinamica
   return 1;//iesire din functia sterge
 }//end cazul este nod terminal
 //Cazul  nodul  de  sters  nu  are  fiu  stang:
(inlocuiesc cu subarborele drept):
 if(pNodCrt->pSt==NULL){
   //daca pNodCrt este radacina:
   if(pNodCrt==pRad)pRad=pNodCrt->pDr;        //noua
//radacina
   else if(esteFiu==STANGA)//cel care-l sterg este
//fiu stanga:
        pNodParinte->pSt=pNodCrt->pDr;//inlocuire
//cu subarbore drept
   else if(esteFiu==DREAPTA)
```

```
        pNodParinte->pDr=pNodCrt->pDr;//inlocuire
//cu subarbore drept
   delete pNodCrt; //eliberam memoria dinamica
   return 1;
 }//end cazul nodul de sters nu are fiu stanga
 //Cazul nodul de sters nu are fiu dreapta:
//(inlocuiesc cu subarborele stang):
 if(pNodCrt->pDr==NULL){
   //daca pNodCrt este radacina:
   if(pNodCrt==pRad)pRad=pNodCrt->pSt;        //noua
//radacina
   else if(esteFiu==STANGA)//cel care-l sterg este
//fiu stanga:
        pNodParinte->pSt=pNodCrt->pSt;//inlocuire
//cu subarbore stanga
   else if(esteFiu==DREAPTA)
        pNodParinte->pDr=pNodCrt->pSt;//inlocuire
//cu subarbore stanga
   delete pNodCrt; //eliberam memoria dinamica
   return 1;
 }//end cazul nodul de sters nu are fiu dreapta
 //Cazul complex, nod crt are doi fii.
 //Ma deplasez in subarborele drept si caut nodul
//de cheie minima
 //(Alternativ, se putea: ma deplasez in subarbore
//stang si caut nodul de cheie maxima.)
 TNOD* pNodInlocuitor=getInlocuitor(pNodCrt);
 //Daca nodul de sters este chiar radacina:
 if(pNodCrt==pRad)
   pRad=pNodInlocuitor;
 else if(esteFiu==STANGA)//cand am cautat nodul de
//sters, am stabilit ce fiu este
   //refac legatura de la parintele nodului de
//sters, la inlocuitor:
   pNodParinte->pSt=pNodInlocuitor;
 else //este sigur fiu dreapta:
   pNodParinte->pDr=pNodInlocuitor;
 //subarborele stang al nodului de sters, il atasam
//tot in  stanga (nu avea !), la inlocuitor:
 pNodInlocuitor->pSt=pNodCrt->pSt;
  //stergem din memorie nodul de sters:
 delete pNodCrt;
 return 1;
}
```

```
TNOD* getInlocuitor(TNOD* pNodDeSters)
{
 //se deplaseaza in fiul drept. Il ia pe cel mai
//mic deci va merge pe stanga:
 //nodul de sters este pNodCrt.
 //Trebuie sa retinem si parintele inlocuitorului!
 //Ma deplasez in subarbore drept. Are sigur.
 //Avem ierarhia:
 //pNodParinte
 //pNodCrt
 //pNodUrm
 TNOD* pNodCrt=pNodDeSters->pDr;
 TNOD* pNodParinte=pNodDeSters;
 for(;;){
   //prima data sigur pNodCrt nu este null!
   TNOD* pNodUrm=pNodCrt->pSt;
   if(pNodUrm==NULL)break;
   pNodParinte=pNodCrt;
   pNodCrt=pNodUrm;
 }
 //schimb notatii, pt. claritate:
 TNOD* pNodInlocuitor=pNodCrt;
 //daca inlocuitorul este chiar fiul drept al
//nodului de sters:
 if(pNodInlocuitor==pNodDeSters->pDr)
   return pNodInlocuitor;
 //daca inlocuitorul nu este direct fiul drept al
//nodului de sters:
 //trebuie sa conectez fiul drept al
//inlocuitorului, ca fiu stanga al parintelui
//inlocuitorului:
 pNodParinte->pSt=pNodInlocuitor->pDr;
 //Noul fiu drept al inlocuitorului va fi
//subarborele drept al nodului de sters:
 pNodInlocuitor->pDr=pNodDeSters->pDr;
 return pNodInlocuitor;
}

void initStiva(int& iV)
{
 iV=-1;
}
void push(TNOD* S[], int& iV, TNOD* x)
```

```
{
 //nu verific depasire
  iV++;
  S[iV]=x;
}

TNOD* pop(TNOD* S[], int& iV)
{
 //nu verific daca este vida!
 TNOD* elem=S[iV];
 iV--;
 return elem;
}

int esteVida(int iV)
{
 if(iV==-1)return 1;
 else return 0;
}

void afisareArbore()
{
//stiva  de adrese de noduri de pe nivelul curent
(stiva de adrese noduri Tati):
 TNOD* ST[1000];
 int iVT;//index varf in aceasta stiva
//stiva  de  adrese  de  noduri  fii  ai  nodurilor
//nivelului curent:
 TNOD* SF[1000];
 int iVF;//index varf in stiva de fii:
 initStiva(iVT);
 push(ST,iVT,pRad);
 int  nBlanks=32;//nodul  radacina  se  va  afisa  la
//nBlanks blancuri de marginea stanga
 initStiva(iVF);
 for(;;){   int stop=1;
   for(int i=0;i<nBlanks;i++)printf(" ");
   while(!esteVida(iVT)){
       TNOD* pNodNivelCrt=pop(ST,iVT);
       if(pNodNivelCrt!=NULL){
         printf("%d",pNodNivelCrt->id);
         push(SF,iVF,pNodNivelCrt->pSt);
         push(SF,iVF,pNodNivelCrt->pDr);
```

```
         if((pNodNivelCrt-
>pSt!=NULL)||(pNodNivelCrt->pDr!=NULL))stop=0;
//daca cel putin un nod de pe nivelul curent are
//cel putin un fiu,nu ne oprim, vom trece la
//nivelul urmator.
       } //if(pNodNivelCrt!=NULL)
       else{//pNodNivelCrt este null (nu are fiu)
         printf("--");
         push(SF,iVF,NULL);
         push(SF,iVF,NULL);}
      //afisam un nr. de blankuri intre nodul afisat
//si urmatorul nod de afisat de pe acelasi nivel:
      for(int i=0;i<nBlanks*2-2;i++)printf(" ");
   }//while
   if(stop==1)break;//a terminat afisarea
   //urmatorul nivel:
   printf("\n");
   //primul  nod al urmatorului nivel se va afisa
//la o distanta de margine, de:
   nBlanks=nBlanks/2;
   //descarcam stiva
   while(esteVida(iVF)==0)
        push(ST,iVT,pop(SF,iVF));
 }//end for(;;)
}
```

4. Probleme propuse

1. *Se citeşte de la tastatură un arbore multicăi reprezentat prin legături fiu-tată. Să se scrie o funcție care să calculeze numărul de frunze ale acestuia.*
2. *Se citeşte de la tastatură un arbore multicăi reprezentat prin legături fiu-tată. Să se scrie o funcție care să calculeze gradul unui nod oarecare al arborelui. Numărul nodului se va citi de la tastatură.*
3. *Să se scrie o funcție ce returnează valoarea maximului dintr-un arbore binar de căutare.*
4. *Să se scrie o funcție ce returnează numărul de chei pare dintr-un arbore binar de căutare.*
5. *Se citeşte de la tastatură un arbore binar de căutare. Să se scrie funcții care să permită: afişarea recursivă a informațiilor din nodurile arborelui (informația din noduri se va afişa*

identat), parcurgerea in-ordine (SRD), pre-ordine (RSD), post-ordine (SDR), ştergerea unui nod din arbore.

6. *Dându-se un arbore binar să se afişeze dacă este arbore binar de căutare.* **Indicație:** Se parcurge arborele în ordinea SRD şi se testează dacă cheile sunt în ordine crescătoare.