Tema: „Analiza prelucrării structurilor de date arborescente ”

Sarcina și obiectivele:

- de studiat și însușit materialul teoretic pentru evidențierea esențialului prelucrării structurilor de date cu **arbori** în elaborarea modelelor soluției, analizând exemplele din text;
- de analizat principiile și modurile de prelucrare, principalele operații cu structuri de date arborescente, dotând cu comentariile de vigoare
- exemplele din 1.3. (Aplicații de *arbori binari* în C), 2. și exemplele 2.7 de derulat, obținând rezultatele de testare, verificare și expuse în raport.
- să se rezolve problemele din compartimentul "3. PROBLEME PROPUSE spre rezolvare" pentru care să se elaboreze scenariile succinte de soluționare, algoritmii funcțiilor principale, organigramele SD și programele cu **arbori și fișiere** (declarări și procesări).
- analizați, comentați, afișați organigramele și toate componentele.
- să se analizeze și să se evidențieze tehnica modelării și programării eficiente pentru diverse compartimente ale diferitor situații cu diverse argumentări și modele de structuri abstracte, incluzând fișiere cu teste de verificare, explicații de rigoare și vizualizări.
- În raport să fie expuse toate activitățile și dificultățile întâlnite pe parcursul efectuării ll.

TEORIE:

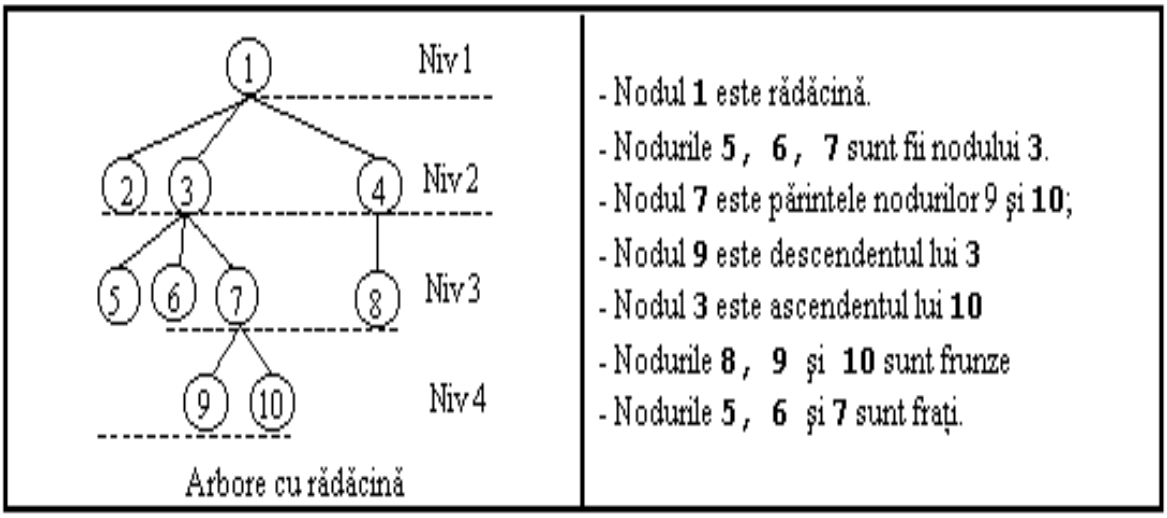
Pentru multe aplicații, timpul de acces la informație cuprinsă într-o listă este foarte mare. De asemenea, pentru modelarea diferitor fenomene sau obiecte de natură este necesară folosirea structurilor ierarhice.

Prin descrierea ierarhică să înțelege descompunerea unei entități în mai multe subentități cu o structură analogică (ex. Structura administrativă, arbore genealogic, planificarea meciurilor, evaluarea unor expresii de calcul etc.)

Un arbore reprezintă o structură de date ce modelează o ierarhie de elemente.
Astfel, fiecare element, numit nod, poate deține un număr de unul sau mai mulți descendenți, iar în acest caz nodul este numit părinte al nodurilor descendente.

Fiecare nod poate avea un singur nod părinte. Un nod fără descendenți este un nod terminal, sau nod frunză.

În schimb, există un singur nod fără părinte, iar acesta este întotdeauna rădăcina arborelui.



Se numește **Arbore cu rădăcină** = graf neorientat conex fără cicluri in care unul din noduri este desemnat ca rădăcină. Nodurile pot fi așezate pe niveluri începând cu rădăcină care este plasata pe nivelul **1**.

Rădăcina = Nod special care generează așezarea unui arbore pe niveluri; Aceasta operație se efectuează in funcție de lungimea lanțurilor prin care celelalte noduri sunt legate de rădăcină.

Descendent = într-un arbore cu rădăcină nodul **y** este descendentul nodului **x** daca este situat pe un nivel mai mare decât nivelul lui **x** si exista un lanț care le unește si nu trece prin rădăcină.

Descendent direct / fiu = într-un arbore cu rădăcină nodul **y** este fiul (descendentul direct) nodului **x** daca este situat pe nivelul imediat următor nivelului lui **x** si exista muchie intre **x** si **y**.

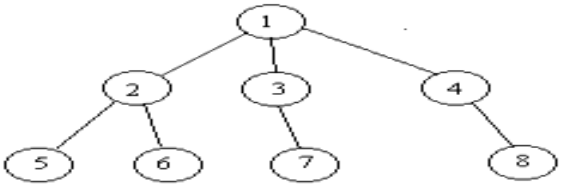
Ascendent = într-un arbore cu rădăcină nodul **x** este ascendentul nodului **y** daca este situat pe un nivel mai mic decât nivelul lui **y** si exista un lanț care le unește si nu trece prin rădăcină.

Ascendent direct / părinte = într-un arbore cu rădăcină nodul x este părintele (ascendentul direct) nodului y daca este situat pe nivelul imediat superior (cu număr de ordine mai mic) nivelului lui y si exista muchie intre x si y.

Frați = într-un arbore cu rădăcină nodul **x** este fratele nodului **y** daca au același părinte.

Frunza = într-un arbore cu rădăcină nodul x este frunza daca nu are nici un descendent direct

Exemplu de arbore oarecare:



- **Parcurgerea în adâncime a acestui arbore: 1 2 5 6 3 7 4 8.**
- **Parcurgerea în lățime: 1 2 3 4 5 6 7 8.**

ARBORI BINARI

Un arbore binar este un caz special de arbore, în care fiecare nod poate avea maxim doi descendenți:

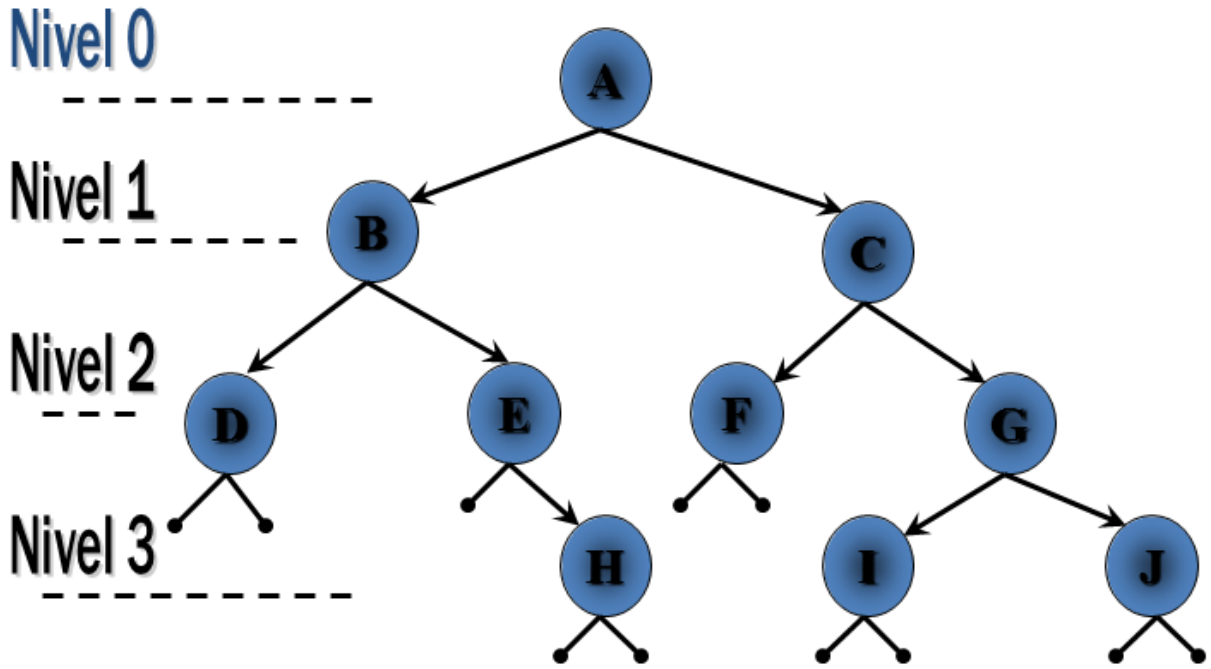
- ***nodul stâng***
- ***nodul drept.***

Structura de arbore binar poate fi utilizată pentru a reprezenta în mod convenabil o mulțime de elemente, în care elementele se regăsesc după o cheie unică.

- **Vom defini arborii binari ca un set finit *T* de unul sau mai multe noduri, astfel încât:**
  - **Există un nod cu destinație specială numit *tulpină (rădăcină)* arborelui;**
  - **Celelalte noduri sunt repartizate în două seturi disjuncte și fiecare din aceste seturi la rândul lui este un arbore binar.**

- **Observații:**
  - **Cei doi arbori subordonați de rădăcină poartă denumirea de *subarbore stâng* și *subarbore drept*;**
  - **Această definiție este recursivă, acest lucru fiind de mare folos în continuare;**
  - **Dacă un nod nu subordonează arbori, îl vom numi *nod terminal*, în caz contrar îl vom numi *nod neterminal*.**

ARBORE BINAR (exemplu)



SUPLIMENTAR !

Am implementat in limbajul C un arbore binar . In acest arbore se poate folosi căutarea caracteristica arborilor de tip binar (elementele mai mic sunt in stânga , mai mari in dreapta).

Funcțiile programului :

1) Crearea arborelui binar de tip BST (binary search tree)
2) Parcurgere : nivele , preordine , inordine , postordine
3) Determinarea adâncimii arborelui.
4) Ștergerea unui nod.
5)Afișarea oricărui Nivel x;
6)Căutarea oricărui element x;
7) Găsirea elementului minim si maxim;

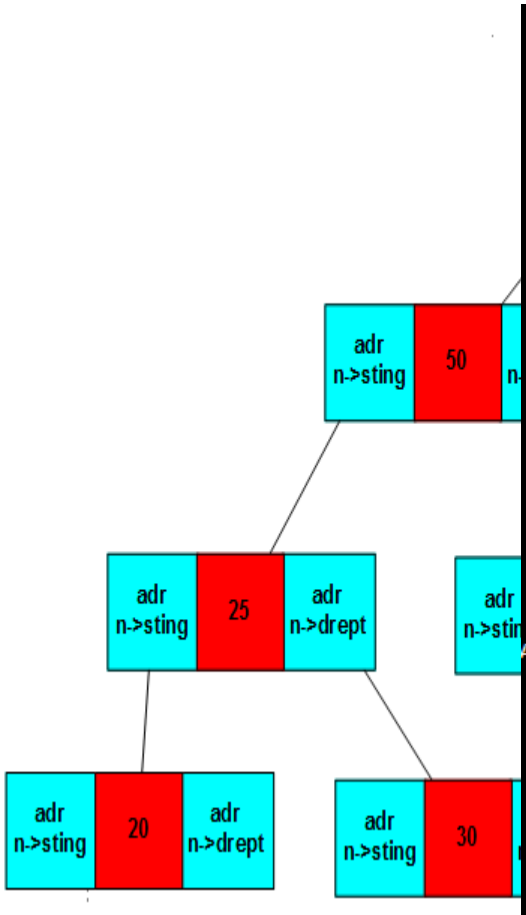
LISTING-UL PROGRAMULUI

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

struct NOD  //nodurile din arbore
{
 int data;
 struct NOD *sting;
 struct NOD *drept;

};

struct NODC  // crearea cozii
{
 struct NOD* tree;
 struct NODC* next;

}*nodc,*cap,*coada;


 int i=0,n=0,j=0;
 NOD *Adauga(NOD* nod,int val);  // funcția de adăugare
 void nivele(NOD*); //afișează nodurile pe nivele
 void adauga(NOD * nod); //funcție de adăugare pentru coada
 int cautare(NOD* nod,int val); // căutarea valorii
 NOD *stergere(NOD* nod,int val); //ștergerea din arbore
 NOD *minim(NOD*); //găsirea minimului pentru ștergere
 void sterge(); //ștergere din coada
 void preordine(NOD*); //parcurgere preordine
 void inordine(NOD*); //parcurgere inordine
 void postordine(NOD*); //parcurgere postordine
 int nrnivel(NOD*);  //afișarea numărului de nivele
 void afisnivel(NOD*); //afișarea oricărui nivel

int main()
{

 int val=0;
 struct NOD *root=NULL;

 root=(NOD*)malloc(sizeof(NOD));

 root->sting=root->drept=NULL;
 printf("\r\n Numărul de elemente :");
 scanf("%d",&n); n=n-1;
 printf("\r\n Introdu valoarea rădăcinii : ");
 scanf("%d",&root->data);

 for(i=0;i<n;i++)  //adăugăm noduri in arbore
 {
 printf("\r\n Introdu valoarea : ");
 scanf("%d",&val);
 Adauga(root,val);
 }


 printf("\r\nAdincimea arborelui : %d  ",nrnivel(root));

 printf("\r\n\t\t\t PARCURGERI \r\n\n");
 printf("\r\n\n NIVELE : ");
 cap=coada=NULL;
 nivele(root);
 printf("\r\n Preordine : ");
 preordine(root);
  printf("\r\n\n Inordine: ");
 inordine(root);
 printf("\r\n\n Postordine ");
 postordine(root);
 printf("\r\n\n Introdu nivelul care dorești sa afișezi :");
 scanf("%d",&i);
 afisnivel(root);

 printf("\r\nIntrodu valoarea care dorești sa ștergi : ");
 scanf("%d",&val);
 stergere(root,val);
 printf("\r\n\t\t\t PARCURGERI \r\n\n");
 printf("\r\n Preordine : ");
 preordine(root);
 printf("\r\n\n Inordine: ");
 inordine(root);
 printf("\r\n\n Postordine ");
 postordine(root);
while(1)  //căutarea elementului
 {
  printf("\r\n Enter the value you want to find : ");
  scanf("%d",&val);
  if(cautare(root,val)==1) printf("\r\n Valoarea este");
  else printf("\r\n Valoarea nu este ");

 }

}

NOD* Adauga(NOD* nod,int val)
{

 if(nod==NULL)
 {
```

```
 nod=(NOD*)malloc(sizeof(NOD));
 nod->data=val;
 nod->drept=nod->sting=NULL;
 }
 else if ( val<=nod ->data ) //adăugăm in stânga
 {
  nod->sting=Adauga(nod->sting,val);
 }
 else
 {
  nod->drept=Adauga(nod->drept,val); //adăugăm in dreapta
 }
 return nod;

}
int cautare(NOD* nod,int val)
{
if(nod==NULL) return 0;
else if(nod->data==val) return 1;
else if(val<nod->data ) return cautare(nod->sting,val);
else return cautare(nod->drept,val);

}
NOD* stergere(NOD* nod,int val)
{
if(nod==NULL) return nod;
else if( val < nod->data) nod->sting=stergere(nod->sting,val);
else if(val > nod->data) nod->drept=stergere(nod->drept,val);
else
{
 if(nod->sting == NULL && nod->drept==NULL)
 {
  free (nod); // eliberam nodul din memorie
  nod=NULL;
 }

 else if(nod->sting==NULL)
 {
  struct NOD* temp=nod;
  nod=nod->drept;
  free(temp);


 }
 else if(nod->drept==NULL)
 {
  struct NOD* temp=nod;
  nod=nod->sting;
  free(temp); // eliberam nodul din memorie

 }
 else
 {
  struct NOD* temp=minim(nod->drept);
  nod->data=temp->data;
  nod->drept=stergere(nod->drept,temp->data);

 }

}

 return nod;


}

void nivele(NOD* nod)
{
if(nod==NULL) return;
adauga(nod);
while(cap!=NULL)
{
 NODC* current = cap; // introducem rădăcină ca cap
 printf("%d ",current->tree->data);
 if(current->tree->sting!=NULL) adauga(current->tree-
>sting);
 if(current->tree->drept!=NULL) adauga(current->tree-
>drept);
 sterge();
}

void adauga(NOD* nod) //funcția adaugă elemente in coada
{
  nodc=(NODC*)malloc(sizeof(NODC));
  nodc->tree=nod;
  nodc->next=NULL;

if(cap==NULL)
  {
  coada=cap=nodc;
  return;
 }
 else if(cap==coada)
 {
  coada=nodc;
  cap->next=nodc;
  return;

 }
 else
  {
   coada->next=nodc;
   coada=nodc;
   return;
  }


}

void sterge() //ștergere din coada
{

 if(cap==coada)
 {
  nodc=cap;
  cap=NULL;
  coada=NULL;
  free(nodc); // eliberarea zonei de memorie

 }
 else
 {
  nodc=cap;
  cap=nodc->next;
  free(nodc);

 }

}

NOD* minim(NOD* nod)
{
 NOD* min=nod;

 while(nod!=NULL)
 {
  if(nod->data < min->data) min=nod;
  nod=nod->drept;

 }
 return min;
}

void preordine(NOD* nod)
{
 if(nod==NULL) return;
 printf("%d ",nod->data);
 preordine(nod->sting);
 preordine(nod->drept);

}

void inordine(NOD* nod)
{
 if(nod==NULL) return;
```

```
 inordine(nod->sting);
 printf("%d ",nod->data);
 inordine(nod->drept);

}

void postordine(NOD* nod)
{
 if(nod==NULL) return;

 postordine(nod->sting);
 postordine(nod->drept);
 printf("%d ",nod->data);

}

int nrnivel(NOD* nod)
{
 if(nod==NULL)
  return -1;
 return max(nrnivel(nod->sting), nrnivel(nod->drept) ) +1;


}


void afisnivel(NOD* nod)
{
 if(nod==NULL) return;
 if(i==j) printf("%d ",nod->data);
 j++; // trecerea la nivel mai mare
 afisnivel(nod->sting);
 afisnivel(nod->drept);
 j--; //reîntoarcerea la nivel
}
 j--;


}
```

ORGANIGRAMA




```
Numarul de elemente :11
Introdu valoarea radacinei : 100
Introdu valoarea : 50
Introdu valoarea : 200
Introdu valoarea : 25
Introdu valoarea : 75
Introdu valoarea : 20
Introdu valoarea : 30
Introdu valoarea : 150
Introdu valoarea : 250
Introdu valoarea : 230
Introdu valoarea : 260
Adincimea arborelui : 3
                    PARCURGERI


NIVELE : 100 50 200 25 75 150 250 20 30 230 260
Preordine : 100 50 25 20 30 75 200 150 250 230 260
Inordine: 20 25 30 50 75 100 150 200 230 250 260
Postordine 20 30 25 75 50 150 230 260 250 200 100
Introdu nivelul care doresti sa afisezi :
```

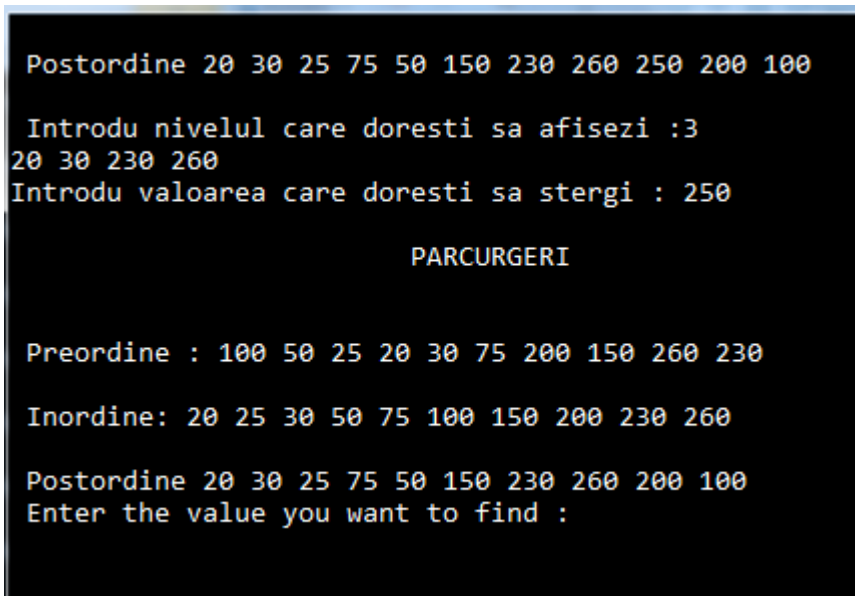
1) Crearea arborelui binar de tip BST (binary search tree)

2) Parcurgere : nivele , preordine , inordine , postordine

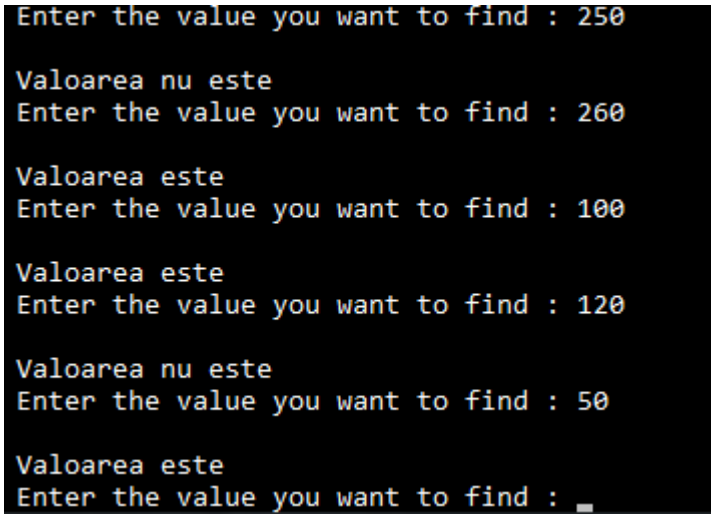
3) Determinarea adâncimii arborelui.

4) Afișarea oricărui Nivel x;

5) Ștergerea unui nod.



6)Căutarea oricărui element x;



7) Găsirea elementului minim si maxim;

```
int minim(NOD* nod)
{
 int min=nod->data;

 while(nod!=NULL)
 {
  if (nod->data < min) min=nod->data;
  nod=nod->sting;

 }
 return min;

}

int maxim(NOD* nod){
 int max=nod->data;

 while(nod!=NULL)
 {
  if (nod->data >max) max=nod->data;
  nod=nod->drept;



 }

return max;}
```

```
Introdu numarul de elemente :11
Introdu radacina : 100
Introdu valoarea : 50
Introdu valoarea : 200
Introdu valoarea : 25
Introdu valoarea : 75
Introdu valoarea : 150
Introdu valoarea : 250
Introdu valoarea : 20
Introdu valoarea : 30
Introdu valoarea : 230
Introdu valoarea : 260
Minim: 20
Maxim: 260
```

Arbore binar este un arbore ordonat, deoarece in fiecare nod subarborele sting precede subarborele drept.

Un arbore binar ordonat se bucură de următoarea proprietate:
dacă n este un nod oarecare al arborelui, având cheia c, atunci:

- **toate nodurile din subarborele stâng a lui n au cheile mai mici sau egale cu c**
- **toate nodurile din subarborele drept al lui n au chei mai mari sau egale cu c.**

Este simplu de observat că un arbore are înălțimea minimă dacă fiecare nivel al său conține numărul maxim de noduri, cu excepția posibilă a ultimului nivel.
• Deoarece numărul maxim de noduri al nivelului i este 2^{i-1}, rezultă că înălțimea minimă a unui arbore binar cu n noduri este:

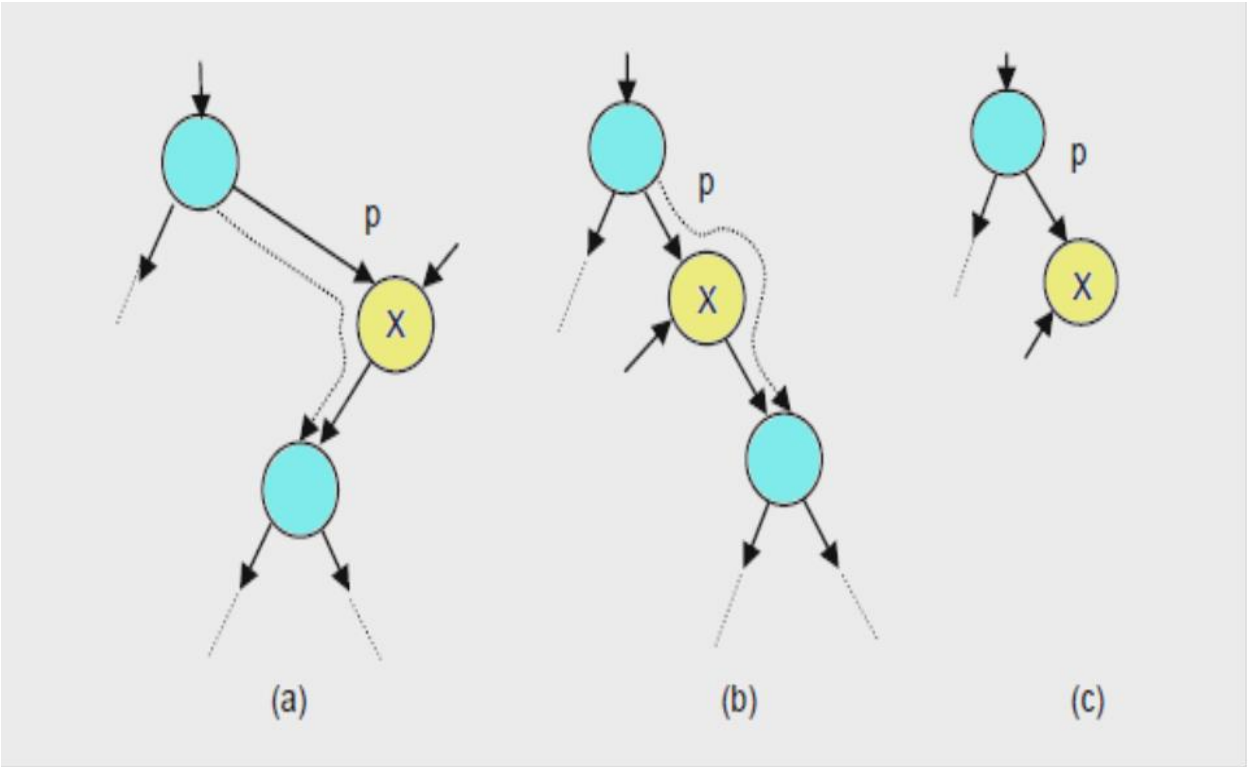
$$h_{\min} = \lceil \log_2 (n+1) \rceil$$

• Prin aceasta se justifică și afirmația că o căutare într-un arbore binar ordonat necesită aproximativ $\log^2 n$ comparații de chei

Această afirmație este valabilă în ipoteza că nodurile s-au organizat într-o structură de arbore binar ordonat de înălțime minimă.

Dacă această condiție nu este satisfăcută, eficiența procesului de căutare poate fi mult redusă, în cazul cel mai defavorabil arborele degenerând într-o structură de listă liniară.
• Aceasta se întâmplă când subarborele drept (stâng) al tuturor nodurilor este vid, caz în care înălțimea arborelui devine egală cu n, iar căutarea nu este mai eficientă decât căutarea într-o listă liniară (O *(n/2)*).

STERGEREA NODULUI :



Nodul ce se suprimă x se înlocuiește cu

- **cel mai mare nod al subarborelui stâng al subarborelui binar care are rădăcina x.**
- **Sau se caută nodul cu cea mai mică cheie a subarborelui drept al subarborelui care-l are pe x drept rădăcină**

TIPURI DE PARCURGERI:

- ***Parcurgerea in inordine***
  (stânga – vârf – dreapta) SVD
- ***Parcurgerea in preordine***
  (vârf - stânga – dreapta) VSD
- ***Parcurgerea in postordine***
  (stânga – dreapta – vârf) SDV

Arbori Binari Optimi

- ***Despre arbori binari optimi putem vorbi atunci când, pentru fiecare dintre cheile unui arbore binar ordonat cunoaștem frecventa cu care aceasta cheie va apărea in cadrul operațiilor ulterioare de căutare***
- ***Aceasta frecventa poate fi văzută si ca "probabilitatea" ca o anumita cheie sa fie subiectul (argumentul) unei operații viitoare de căutare in arborele binar ordonat***
- ***Vorbim despre operații de căutare deoarece, in general, acestea sunt cele care ne interesează atunci când lucram cu arbori binari ordonați***

- ***Pentru o cheie oarecare, timpul de căutare va fi, in cel mai rău caz proporțional cu înălțimea arborelui***

- ***Pentru o cheie care exista in arbore, timpul de căutare este proporțional cu adâncimea nodului care conține cheia (distanta de la acesta la rădăcină)***
- ***Ideal ar fi ca acele chei care sunt căutate mai des sa fie plasate mai aproape de rădăcină arborelui iar acele chei care sunt căutate mai rar sa fie plasate mai departe de rădăcină arborelui***

Sarcina Lucrării :

Fie un arbore de memorat:

1. Sa se determine nodul rădăcină
2. Sa se determine nodurile terminale
3. Sa se afișeze pentru fiecare nod predecesorii si nivelul pe care se găsește
4. Care este adâncimea arborelui?
5. Sa se afișeze nodurile de pe nivelul oarecare x.

Listingul Programului

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct nod
{
 int inf;
 int fii;
 struct nod* nr[100];
};

int j=-1,adincimea=0,nrniv,key,cap=0;
struct nod* coada[50]; //coada pentru nivele
struct nod * adăugare(void); //adaugă nod
void adincime(struct nod *); //parcurgere adâncime
void largime(struct nod *);  // parcurgere lărgime
void termin(struct nod *p); // afișarea nodurilor terminale
void nivel(struct nod *p);  //afișarea numărului de nivele
void predecesori(struct nod *p); //afișarea predecesorilor

int main()
{
        struct nod *root;



        root = adăugare();
        printf("\r\nParcurgere in adâncime:");
        adâncime(root);
        printf("\r\n\nParcurgere in lățime:");
        largime(root);
        printf("\r\n\nAdincimea este %d :",adâncimea);
        printf("\r\n\nNodurile terminale : ");
        termin(root);
        printf("\r\n Introdu nivelul care dorești sa afișezi : ");
        j=-1; // j=-1 deoarece in funcție se incrementează
        scanf("%d",&nrniv);
        printf("\r\n Valorile de pe nivelul %d : ",nrniv);
        nivel(root);
        printf("\r\n Introdu nodul la care dorești sa afli
predecesorii : ");
        j=0;
        scanf("%d",&key);
        printf("\r\n Predecesorii sunt : ");
        predecesori(root);
        return 0;
}
struct nod * adăugare() //crearea nodului
{
                int inf;
                int n, i;
                struct nod *p;
                printf("Valoarea = ");
                scanf("%d", &inf);
        p =(nod*) malloc( sizeof(struct nod) ); //alocare
                p->inf = inf; //introducem info
                printf("Numărul de fii lui %d : " , p->inf);
                scanf("%d", &n); //introducem nr de fii
                p->fii = n;
                for(i = 0; i < n; i++) //înlănțuim fiii
                p->nr[i] = adăugare(); //apel recursiv
                return p;
}
void adâncime(struct nod *p)
{

int i;
                if( p != NULL )
{
j++; //nivelul s-a mărit
                printf("%d ", p->inf);
                for( i = 0; i < p->fii; i++)
        adâncime( p->nr[i] ); //apel recursiv
 if(j>adâncimea) adâncimea=j;//găsirea adâncimii max
        j--; // nivelul s-a scăzut
        }
}
void largime(struct nod *p)
{
                struct nod *coada[100];
                struct nod *t; int front = 0, back = 0;
                int i;
                coada[ front++ ] = p; //capul = rădăcină
```

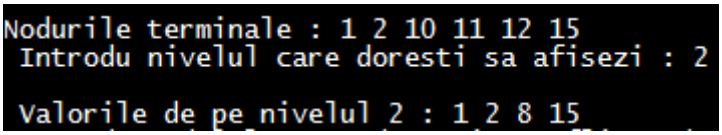
```
                while ( front != back) / capul!=coada
                {
        t = coada[ back ]; back++;//t – variabila temporara
                printf("%d ", t->inf);
                for(i = 0; i < t->fii; i++)
coada[ front++ ] = t->nr[i];
                } //adăugarea fiilor in coada
}
void termin(struct nod *p)
{
int i;
                if( p != NULL )
{

 if(p->fii==0)printf("%d ", p->inf); //daca nu are fii se afișează
                for( i = 0; i < p->fii; i++)
        termin( p->nr[i] );


        }
}

void nivel(struct nod *p)
{

int i;
                if( p != NULL )
{
j++;
 if(j==nrniv)printf("%d ", p->inf);//daca este nivelul x se afis
                for( i = 0; i < p->fii; i++)
                nivel( p->nr[i] );
                j--;
                }


}

void predecesori(struct nod* p)
{
int i,v;
                if( p != NULL )
{
coada[j++]=p; // se adaugă nodul in coada
        if(coada[j-1]->inf==key) //daca s-a găsit elementul cheie
                {
 for(v=0;v<j;v++) printf("%d ",coada[v]->inf); //se afișează coada
                printf("\r\n Nivelul este : %d ",j-1);
                                exit(0);


                }
                for( i = 0; i < p->fii; i++)
        predecesori( p->nr[i] );
        j--;

        }


}
```

Introducerea
Parcurgerea + Adâncimea

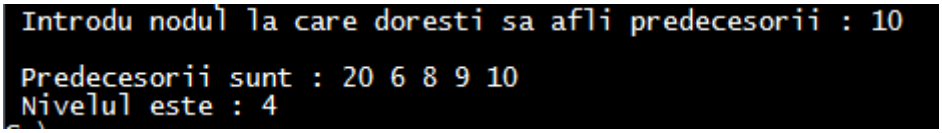
```
C:\c>arbore.exe
Valoarea = 20
Numarul de fii lui 20 : 3
Valoarea = 40
Numarul de fii lui 40 : 2
Valoarea = 1
Numarul de fii lui 1 : 0
Valoarea = 2
Numarul de fii lui 2 : 0
Valoarea = 6
Numarul de fii lui 6 : 1
Valoarea = 8
Numarul de fii lui 8 : 1
Valoarea = 9
Numarul de fii lui 9 : 3
Valoarea = 10
Numarul de fii lui 10 : 0
Valoarea = 11
Numarul de fii lui 11 : 0
Valoarea = 12
Numarul de fii lui 12 : 0
Valoarea = 3
Numarul de fii lui 3 : 1
Valoarea = 15
Numarul de fii lui 15 : 0
```

```
Parcurgere in adincime:20 40 1 2 6 8 9 10 11 12 3 15

Parcurgere in latime:20 40 6 3 1 2 8 15 9 10 11 12

Adincimea este 4 :
```

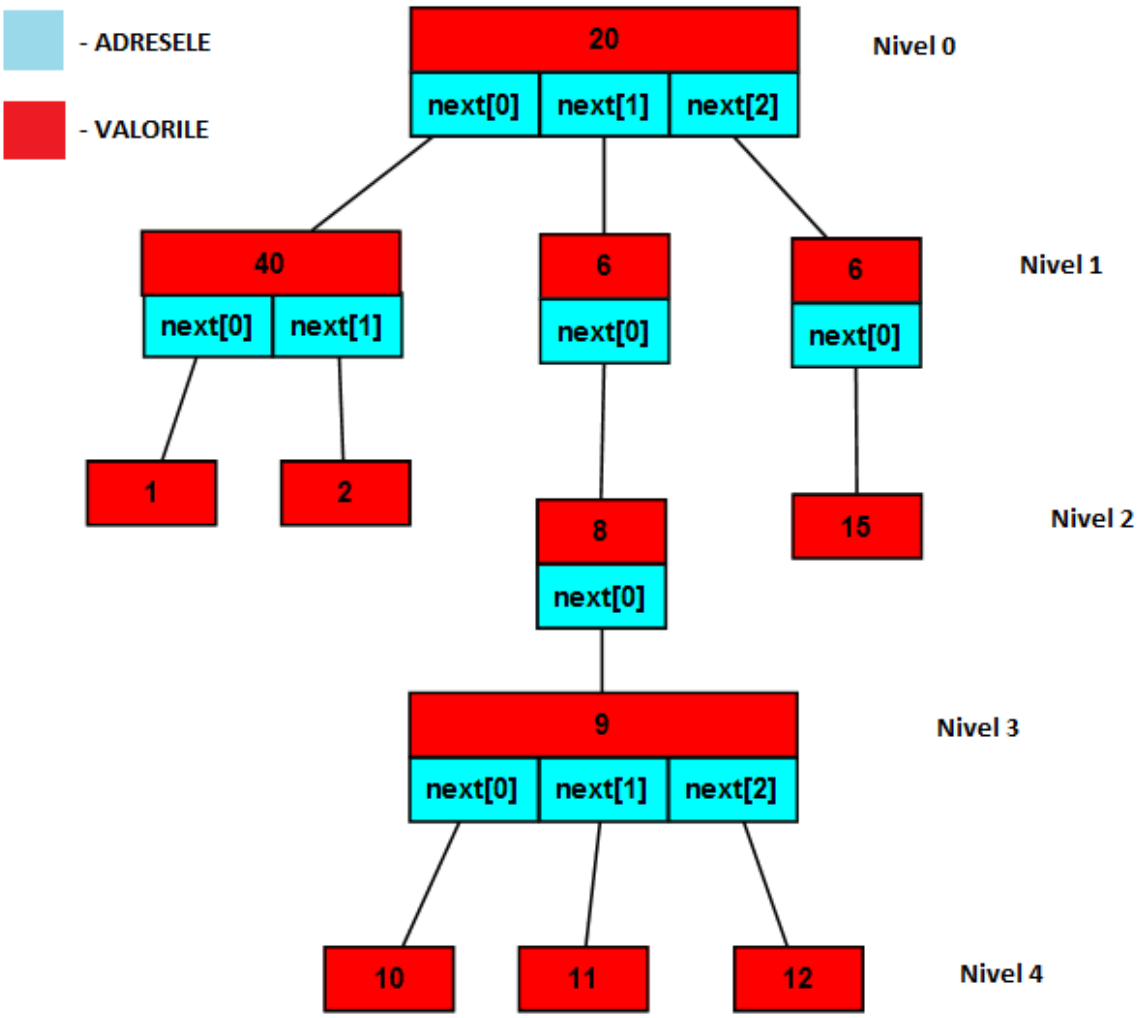
Afișarea nodurilor terminale + valori pe nivelul x



Aflarea predecesorilor



ORGANIGRAMA



Concluzie:

In lucrarea data am studiat arborii. Arbori simpli si binari . Am utilizat cunoștințele din lucrarea trecuta (LISTE) . Totodată am implementat acești arbori in limbajul C . Am observat ca cu arbori binari timpul de căutare a elementelor este considerabil mai mic , deoarece sunt nevoie de mai puține operații . Totodată putem folosi arborii si ca reprezentare a structurilor ierarhice (ca de exemplu arborele genealogic). Cunoștințele date le putem aplica si in domeniul matematicii discrete. In lucrarea data am folosit si „coada'' pentru a putea afișa arborele pe nivele. Am executat programe complexe cu multe funcții care permit prelucrarea arborilor . Este foarte util sa folosim tipul abstract de date Arbori in programe ca de exemplu dicționare , timpul de căutare va scădea considerabil din cauza unui nr de căutări mai mic.