



PROIECTAREA ALGORITMILOR

Lect. univ. dr. Adrian Runceanu

Curs 9

Arbori

Conținutul cursului

9.1. Definitii. Reprezentari ale arborilor

9.2. Grafuri ponderate. Arborele partial de cost minim

9.2.1. Algoritmul lui Kruskal

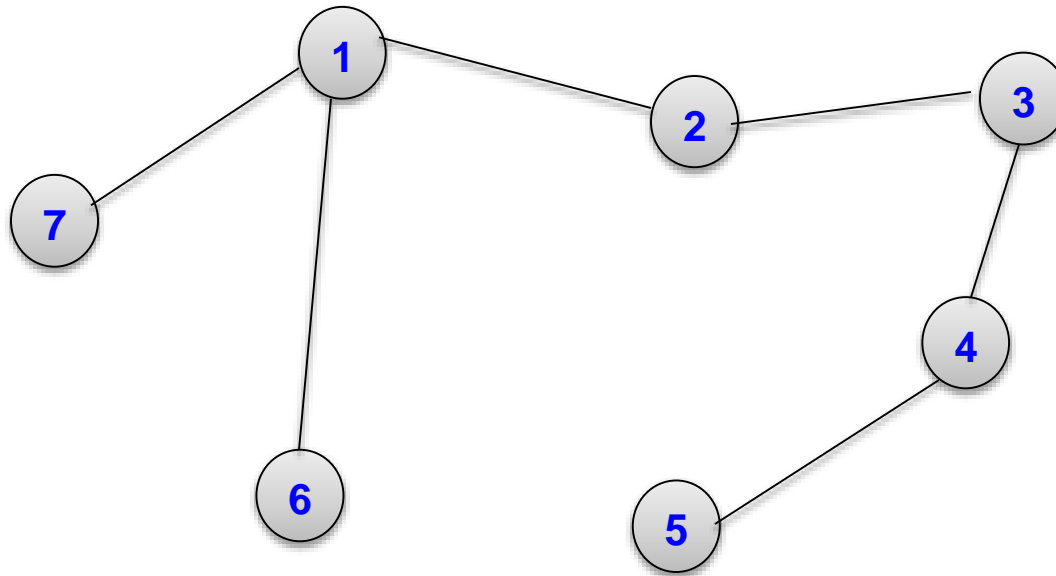
9.2.2. Algoritmul lui Prim

**9.3. Arbori binari. Definitii. Reprezentare.
Metode de parcurgere**

Definitie

Se numeste **arbore** un graf conex si fara cicluri.

Exemplu:



Daca eliminam o muchie, graful isi pierde proprietatea de conexitate, iar daca adaugam o muchie, apare un ciclu.

9.1. Definitii. Reprezentari ale arborilor

Teoremă. Fie G un graf cu $n \geq 1$ vârfuri. Următoarele afirmații sunt echivalente:

- *G este un arbore*
- *G are $n-1$ muchii și nu conține cicluri*
- *G are $n-1$ muchii și este conex*
- *oricare două vârfuri din G sunt unite printr-un unic drum*
- *G nu conține cicluri și adăugarea unei noi muchii produce un unic ciclu elementar*
- *G este conex, dar devine neconex prin ștergerea oricărei muchii*

9.1. Definitii. Reprezentari ale arborilor

Definitie

Fie G un graf. Un graf partial H al sau care in plus este si arbore se numeste **arbore partial**.

Observatie:

Arborele poate fi considerat un graf orientat, stabilind pe fiecare muchie sensul de la nivelul superior către nivelul inferior.

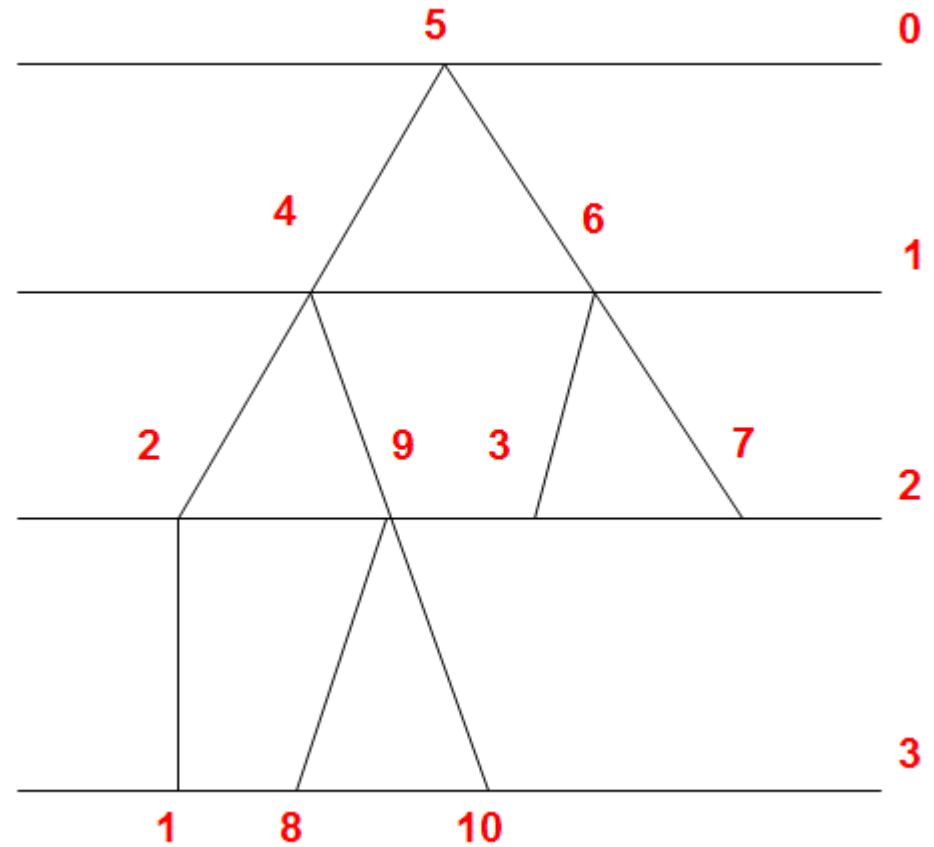
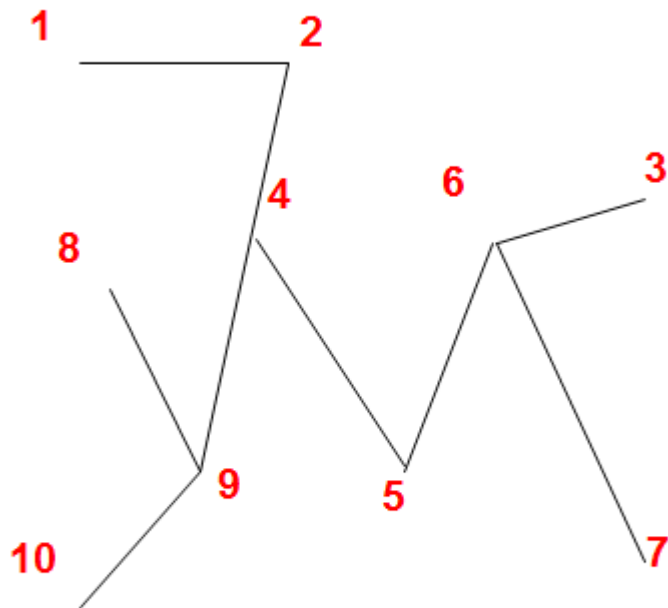
- În foarte multe probleme referitoare la arbori este pus în evidență un vârf al său, numit *rădăcină*.
- Alegerea unui vârf drept rădăcină are două consecințe:

Arborele poate fi așezat pe niveluri astfel:

- rădăcina este așezată pe nivelul 0
- pe fiecare nivel i sunt plasate vârfurile pentru care lungimea drumurilor care le leagă de rădăcină este i
- se trasează muchiile arborelui

Această așezare pe niveluri face mai intuitivă noțiunea de arbore, cu precizarea că în informatică *"arborii cresc în jos"*.

9.1. Definitii. Reprezentari ale arborilor



9.1. Definitii. Reprezentari ale arborilor

Definitie

Daca intr-un arbore eliminam radacina, atunci obtinem **subarbori**.

Definitie

Intr-un arbore, succesorul unui nod se mai numeste si “**fiul**” sau “**urmasul**” sau.

9.1. Definitii. Reprezentari ale arborilor

Definitie

Daca un nod are unul sau mai multi fii, atunci el se numeste “**tatal**” sau “**parintele**” acestora.

Definitie

Daca un nod are mai multi fii, acestia se numesc “**frati**” intre ei.

9.1. Definitii. Reprezentari ale arborilor

Definitie

Se numeste **inaltimea arborelui** nivelul maxim al nodurilor unui arbore.

Definitie

Se numeste **gradul** unui nod numarul fiilor nodului respectiv.

Definitie

Se numeste **nod terminal(frunza)**, un nod de grad zero, iar un nod de grad diferit de zero se numeste **nod intern**.

9.1. Definitii. Reprezentari ale arborilor

Definitie

Se numeste **gradul arborelui**, gradul maxim al nodurilor unui arbore.

Propozitie:

Orice arbore $H=(X,V)$ cu $n \geq 2$ varfuri contine cel putin doua varfuri terminale.

Propozitie:

Orice arbore cu n varfuri are $n-1$ muchii.

9.1. Definitii. Reprezentari ale arborilor

Aplicatie:

Se citeste un graf neorientat prin matricea de adiacenta. Se cere sa se verifice daca **graful reprezinta un arbore**.

Indicatie:

Aplicam teorema prezentata mai inainte.
(de exemplu: *verificam daca graful are $n-1$ muchii și este conex*)

Codul sursa:

```
#include<iostream.h>
int viz[30],n,i,j,k,u,v,p,a[20][20],c[30],m=0;
int main(void)
{
    cout<<"Dati numarul de varfuri n = ";    cin>>n;
    for(i=1; i<=n-1; i++)
        for(j=i+1; j<=n; j++)
        {
            cout<<"a["<<i<<","<<j<<"]= ";
            cin>>a[i][j];
            a[j][i] = a[i][j];
            m+=a[i][j];                // m = nr. de muchii
        }
    cout<<"Dati varful de plecare ";    cin>>i;
    for(j=1; j<=n; j++) viz[j]=0;
```

```
// parcurgem in latime graful neorientat
```

```
c[1]=i;  
p=1;  
u=1;  
viz[i]=1;  
while(p<=u)  
{  
    v=c[p];  
    for(k=1; k<=n; k++)  
    {  
        if( (a[v][k]==1) && (viz[k]==0) )  
        {  
            u++;  
            c[u]=k;  
            viz[k]=1;  
        }  
    }  
    p++;  
}
```

Dacă un graf este conex, atunci putem să vizităm (parcurgem) toate vârfurile sale pornind de la un vârf dat.

```
int este_arbore=1;
for(i=1;i<=n;i++)
    if(viz[i]==0) este_arbore=0;
if(m==n-1 && este_arbore==1)
    cout<<"Graful dat este ARBORE!\n";
else
    cout<<"Graful dat NU este ARBORE!\n";
}
```


Conținutul cursului

9.1. Definitii. Reprezentari ale arborilor

9.2. Grafuri ponderate. Arborele partial de cost minim

9.2.1. Algoritmul lui Kruskal

9.2.2. Algoritmul lui Prim

**9.3. Arbori binari. Definitii. Reprezentare.
Metode de parcurgere**

9.2. Grafuri ponderate

Pentru rezolvarea problemelor practice cu ajutorul grafurilor, muchiilor(arcelor) li se asociaza **ponderi**:

- greutate
- costuri
- valori, etc

Astfel de grafuri se numesc **grafuri ponderate**.

9.2. Grafuri ponderate

Exemple de aplicatii:

1. Harta traseelor aeriene ale unei zone, unde **arcele** reprezinta rute de zbor, iar *ponderile reprezinta distante sau preturi*.
2. Circuite electrice, unde **arcele** reprezinta legaturi, iar *ponderile pot fi lungimi sau costuri*.
3. Intr-o activitate de planificare in executie a task-urilor, ponderea poate reprezenta fie timpul, fie costul executiei unui task, fie timpul de asteptare pana la lansarea in executie a task-ului.

9.2. Grafuri ponderate

Astfel se pot rezolva probleme legate de minimizare a costurilor:

- 1) Gasirea drumului minim cu costul cel mai redus care conecteaza toate nodurile grafului.*
- 2) Gasirea drumului cu costul cel mai redus care leaga doua puncte date.*

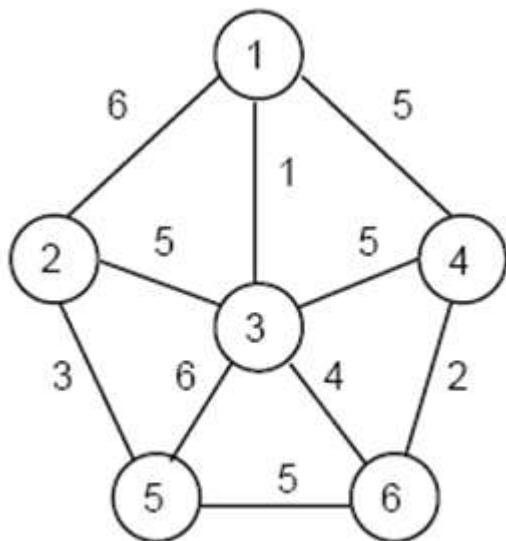
Prima problema care modeleaza circuitele electrice, se numeste **problema arborelui de cost minim**.

A doua problema care modeleaza harti de trasee(aeriane, feroviare, turistice) se numeste **problema drumului minim**.

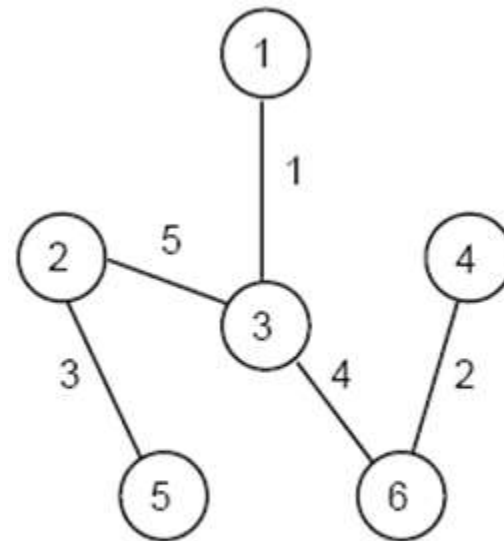
9.2. Grafuri ponderate

Exemplu de reprezentare a unui

(a) **graf ponderat** (b) si a unui **arbore de cost minim**



(a)



(b)

9.2. Grafuri ponderate. Arborele partial de cost minim

Arborele partial de cost minim se poate determina cu ajutorul a trei algoritmi:

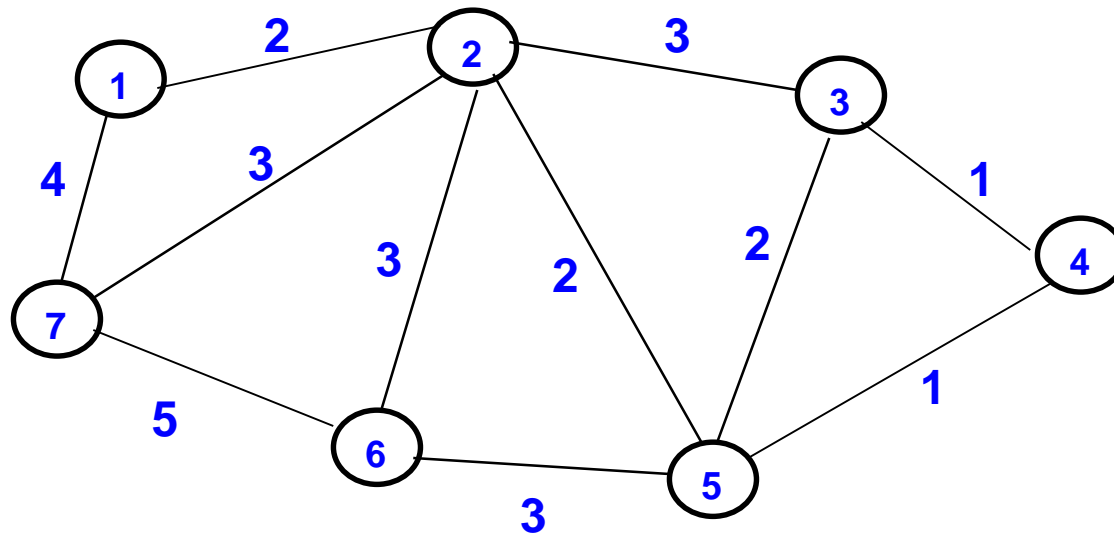
- 1) Algoritmul lui Kruskal
- 2) Algoritmul lui Prim
- 3) Metoda cautarii “bazate pe prioritate”

Definitie

Fie $G=(X,U)$ un graf conex. Pentru graful partial $H=(X,V)$ al lui G , **costul grafului** reprezinta suma costurilor muchiilor sale, adica:

$$c(H) = c(u_1) + c(u_2) + \dots + c(u_m).$$

Exemplu: Fie graful G cu muchiile avand costurile urmatoare:



Pentru $H=(X,V)$, cu $V=\{[1,2], [3,5], [4,3], [6,7]\}$

$$c(H) = c([1,2]) + c([3,5]) + c([4,3]) + c([6,7]) = 2 + 2 + 1 + 5 = 10$$

9.2. Grafuri ponderate. Arborele partial de cost minim

Problema:

Sa se determine un graf partial H al lui G care sa fie conex si sa aiba costul minim.

Notam **arborele partial de cost minim** cu **APM**.

Aceasta problema este cunoscuta sub numele de ***problema conectarii cu cost minim a oraselor***.

Propozitie:

Pentru un graf G conex, exista un graf partial H conex si de cost minim, care in plus este si arbore.

Conținutul cursului

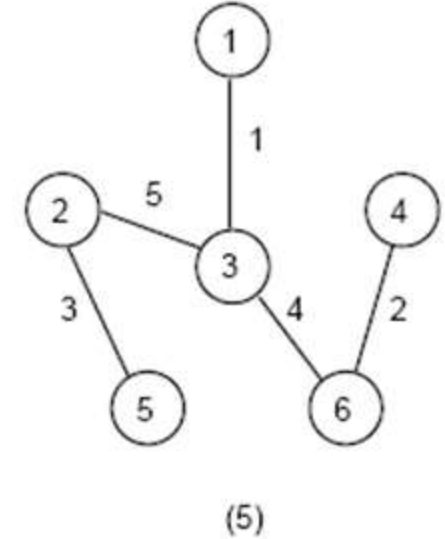
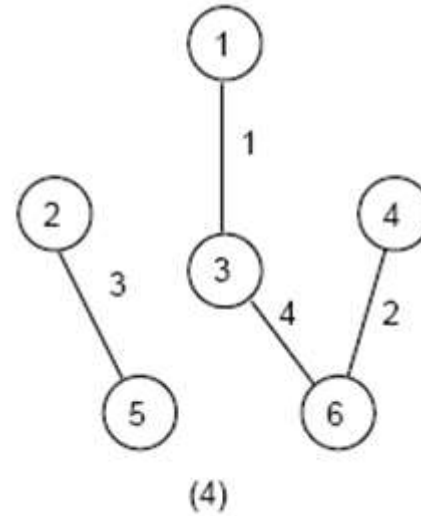
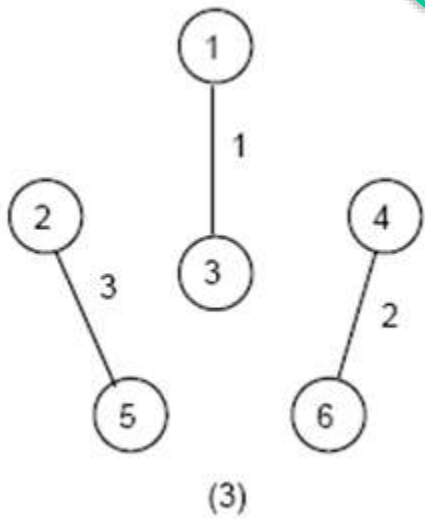
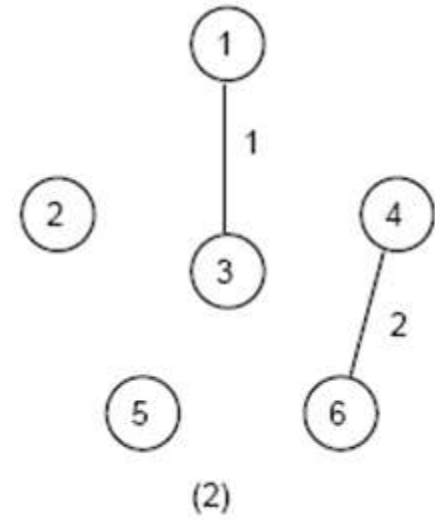
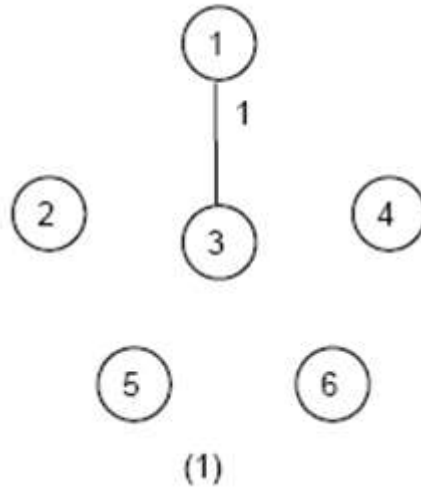
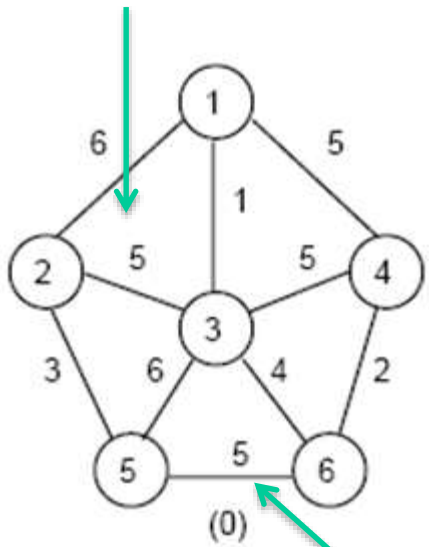
- 9.1. Definitii. Reprezentari ale arborilor**
- 9.2. Grafuri ponderate. Arborele partial de cost minim**
 - 9.2.1. Algoritmul lui Kruskal**
 - 9.2.2. Algoritmul lui Prim**
- 9.3. Arbori binari. Definitii. Reprezentare. Metode de parcurgere**

1) Algoritmul lui Kruskal

- Se pleaca initial de la n arbori distincti H_1, H_2, \dots, H_n , unde $i=1, \dots, n$.
- La pasul k , unde $k=1, \dots, n-2$, avem $n-k$ arbori distincti H_1, H_2, \dots, H_{n-k} , cu $H_i=(X_i, U_i)$, astfel incat $X_1 \cup X_2 \cup \dots \cup X_{n-k}$.
- Prin unificare a doi arbori existenti obtinem $n-k-1$ arbori disjuncti.

Alegerea celor doi arbori se poate face astfel:

- Dintre toate muchiile nealese inca, se selecteaza aceea de cost minim care are doua extremitati in doua multimi diferite X_i si X_j (pentru ca sa nu formam un ciclu).
- Prin adaugarea acestei muchii u , din arborii H_i si H_j se va forma un nou arbore $H'=(X', U')$, $X'=X_i \cup X_j$, si $U'=U_i \cup U_j \cup \{u\}$.
- Dupa $n-2$ pasi obtinem un singur arbore.



```
#include<iostream.h>
struct muchie
{
    int x,y;
    float cost;
} b[30],f;
int n,m,i,k,ct,l[30],j;
int main()
{
    cout<<"nr. de varfuri";    cin>>n;
    cout<<"nr. de muchii";    cin>>m;
```

```
for(i=1;i<=m;i++)
{
    cout<<"muchia " <<i<<"( x,";
    cin>>b[i].x;
    cout<<"y) = ";
    cin>>b[i].y;
    cout<<"costul muchiei = ";
    cin>>b[i].cost;
}
for(i=1;i<=n;i++)    l[i]=i;
```

```
// ordonarea muchiilor dupa cost
for(i=1;i<m;i++)
    for(j=i+1;j<=m;j++)
        if(b[i].cost>b[j].cost)
            {
                f=b[i];
                b[i]=b[j];
                b[j]=f;
            }
k=0; // nr. de muchii selectate
ct=0; // costul total
i=1;
```

```
while(k<n-1)
{
    if(l[b[i].x]!=l[b[i].y])
    {
        k++;
        ct+=b[i].cost;
        for(j=1;j<=n;j++)
            if(l[j]==l[b[i].x])
                l[j]=l[b[i].y];
        cout<<b[i].x<<" " <<b[i].y<<endl;
    }
    i++;
}
cout<<"Costul total = " <<ct<<endl;
}
```

```
E:\Universitate\2009-2010\Semestrul
muchia 8< x,2
y) = 7
costul muchiei = 3
muchia 9< x,2
y) = 6
costul muchiei = 3
muchia 10< x,2
y) = 5
costul muchiei = 2
muchia 11< x,3
y) = 5
costul muchiei = 2
3 4
4 5
1 2
2 5
2 7
2 6
12

Terminated with return code 0
Press any key to continue ...
```


Conținutul cursului

- 9.1. Definitii. Reprezentari ale arborilor**
- 9.2. Grafuri ponderate. Arborele partial de cost minim**
 - 9.2.1. Algoritmul lui Kruskal**
 - 9.2.2. Algoritmul lui Prim**
- 9.3. Arbori binari. Definitii. Reprezentare. Metode de parcurgere**

2) Algoritmul lui Prim

- Graful este dat prin **matricea costurilor**
- $$C_{ij} = \begin{cases} \text{costul muchiei } [i,j], \text{ daca } [i,j] \in U \\ 0, \text{ in caz contrar} \end{cases}$$
- Se porneste initial, ca si in algoritmul lui Kruskal, cu n arbori distincti, si la fiecare pas k vom avea un arbore cu $k+1$ varfuri si $n-k-1$ arbori cu cate un singur varf fiecare
 - *La pasul k se alege muchia de cost minim care are o extremitate in arborele cu mai multe varfuri iar cealalta intr-unul din ceilalti arbori.*
 - Initial se alege muchia de cost minim.

Implementarea algoritmului lui Prim:

```
# include <iostream.h>
# include <stdio.h>
typedef struct min
    {
        int lin,col;
    }min;
int n,a[50][50],m[50][50];

void afisare(int a[50][50],int n)
{
    int i,j;
    for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++) cout<<" "<<a[i][j];
            cout<<"\n";
        }
}
```

```

int in(int v[50],int n,int x)
{
    for(int i=1;i<=n;i++)
        if (v[i]==x) return 1;
    return 0;
}
min minim(int v[50],int k)
{
    min min0;
    int i,j;
    int min1=32000;
    for(i=1;i<=k;i++)
        for(j=1;j<=n;j++)
            if (m[v[i]][j]<min1 && m[v[i]][j]!=0 && in(v,k,j)==0)
                { min0.lin=v[i];
                  min0.col=j;
                  min1=m[v[i]][j];
                }
    return min0;
}

```

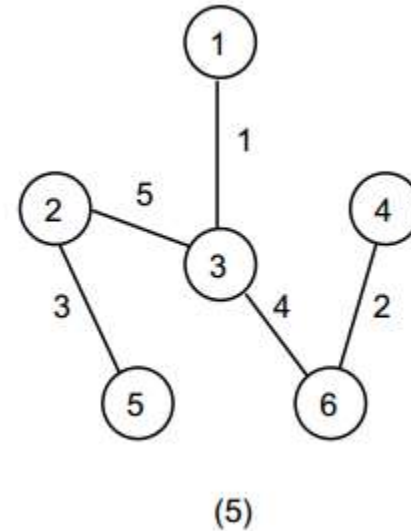
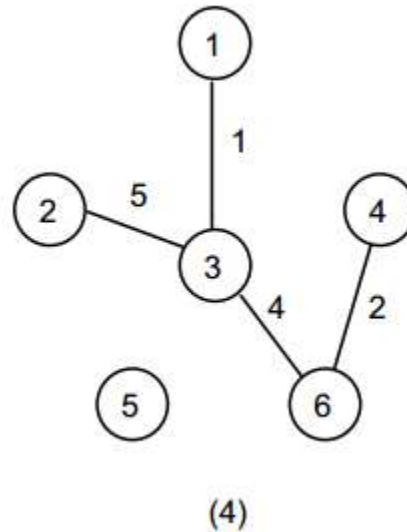
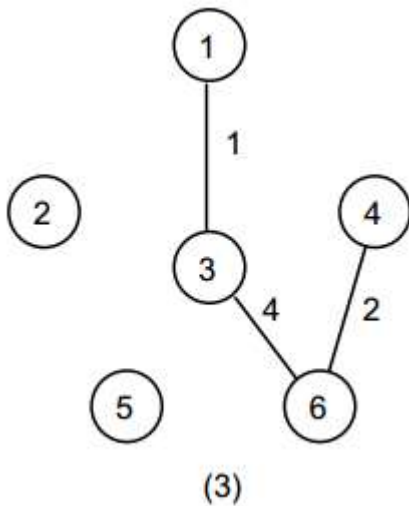
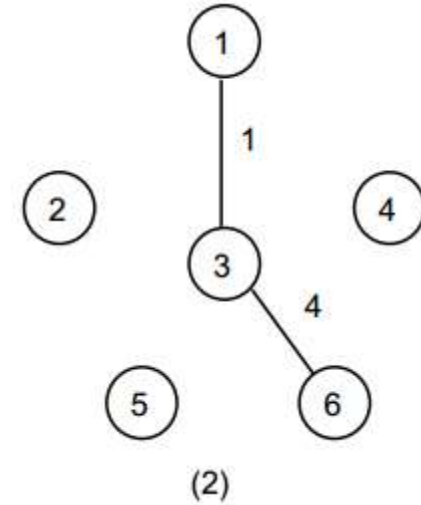
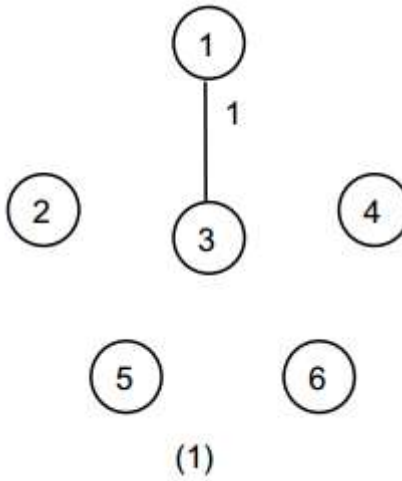
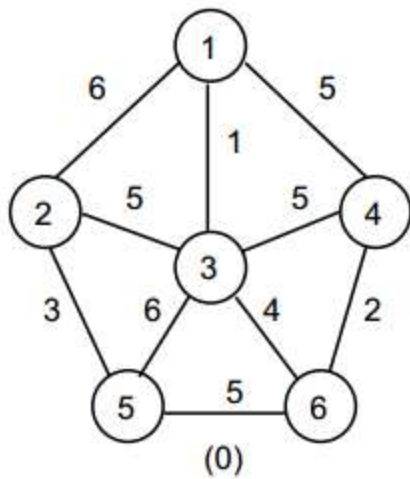
Verificam daca muchia (v[i], j):

- este cea mai mica
- nu a fost selectata
- si are o extremitate in arborele cu cele mai multe varfuri si cea de a doua extremitate intr-unul din ceilalti arbori

```
int main(void)
{
    int t[50][50],vec[50],i,j,v,k,x,y;
    cout<<"Citirea matricei de adiacenta a grafului
    \n";
    cout<<"Dati numarul de varfuri n = "; cin>>n;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];
    cout<<"\n graful are "<<n<<" noduri, matricea
    sa de adiacenta fiind:\n";
    afisare(a,n);
}
```

```
for(i=1;i<=n;i++) m[i][i]=0;
for(i=1;i<n;i++)
    for(j=i+1;j<=n;j++)
        if (a[i][j]==1)
            {
                cout<<"\n costul muchiei ("<<i<<","<<j<<")=";
                cin>>m[i][j];
                m[j][i]=m[i][j];
            }
        else
            {
                m[i][j]=1000;
                m[j][i]=1000;
            }
cout<<"\n matricea costurilor este :\n";
afisare(m,n);
```

```
cout<<"introduceti un nod v (1<=v<="<<n<<")";
cin>>v;
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++) t[i][j]=0;
k=1;
vec[k]=v;
for(i=1;i<n;i++)
{
    x=minim(vec,k).lin;
    y=minim(vec,k).col;
    t[x][y]=m[x][y];
    t[y][x]=t[x][y];
    m[x][y]=1000;
    m[y][x]=1000;
    k++;
    vec[k]=y;
}
cout<<"arborele de cost minim rezultat este:\n";
afisare(t,n);
}
```



Construcția unui arbore de acoperire minim al unui graf pe baza
algoritmului lui Prim

(0) initializare

Nod	1	2	3	4	5	6
Apropiat	-	1	1	1	1	1
CostMin	∞	6	1	5	∞	∞

$U=\{1\}$
 $N-U=\{2,3,4,5,6\}$

(1) se selecteaza nodul 3

Nod	1	2	3	4	5	6
Apropiat	-	3	-	1	3	3
CostMin	∞	5	∞	5	6	4

$U=\{1,3\}$
 $N-U=\{2,4,5,6\}$

(2) se selecteaza nodul 6

Nod	1	2	3	4	5	6
Apropiat	-	3	-	6	3	-
CostMin	∞	5	∞	2	6	∞

$U=\{1,3,6\}$
 $N-U=\{2,4,5\}$

(3) se selecteaza nodul 4

Nod	1	2	3	4	5	6
Apropiat	-	3	-	-	3	-
CostMin	∞	5	∞	∞	6	∞

$U=\{1,3,6,4\}$
 $N-U=\{2,5\}$

(4) se selecteaza nodul 2

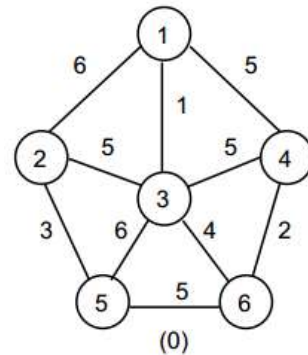
Nod	1	2	3	4	5	6
Apropiat	-	-	-	-	2	-
CostMin	∞	∞	∞	∞	3	∞

$U=\{1,3,6,4,2\}$
 $N-U=\{5\}$

(5) se selecteaza nodul 5

Nod	1	2	3	4	5	6
Apropiat	-	-	-	-	-	-
CostMin	∞	∞	∞	∞	∞	∞

$U=\{1,3,6,4,2,5\}$
 $N-U=\{\}$



Exemplu de aplicare al algoritmului lui PRIM

Aplicatie:

O companie construiește o rețea de șosele.

Se propun 3 soluții care urmăresc următoarele aspecte:

1. minimizarea costului lucrărilor
2. minimizarea costului de acces din capitală la oricare din localitățile județului
3. o variantă de compromis

Propuneri de solutionare:

- 3 metode de determinare al arborelui de acoperire minim
- 1 metodă de determinare a drumurilor minime
- 1 metodă de compromis – arborele de acoperire minim cu rădăcina în nodul preferențial

Conținutul cursului

- 9.1. Definitii. Reprezentari ale arborilor**
- 9.2. Grafuri ponderate. Arborele partial de cost minim**
 - 9.2.1. Algoritmul lui Kruskal**
 - 9.2.2. Algoritmul lui Prim**
- 9.3. Arbori binari. Definitii. Reprezentare. Metode de parcurgere**

9.3. Arbori binari

Definitie

Un **arbore binar** este *un arbore în care orice vârf are cel mult doi descendenți*, cu precizarea că se face distincție între descendentul stâng și cel drept.

Primele probleme care se pun pentru arborii binari sunt:

1. modul de reprezentare
2. metode de parcurgerea a lor

9.3. Arbori binari

Forma standard de reprezentare a unui arbore binar constă în:

- a preciza *rădăcina arborelui = (notatie) răd*
- a preciza pentru fiecare vârf i *tripletul*:
 - $st(i)$ = descendentul stâng
 - $dr(i)$ = descendentul drept
 - $info(i)$ = informația atașată vârfului

Trebuie stabilită o convenție pentru lipsa unuia sau a ambilor descendenți, ca de exemplu specificarea lor prin valoarea 0.

9.3. Arbori binari

Tipuri de **reprezentari statice**:

1) Cu ajutorul a doi vectori:

Vectorul **st** – care contine descendenti stangi ai unui nod

Vectorul **dr** - care contine descendenti drepti ai unui nod

9.3. Arbori binari

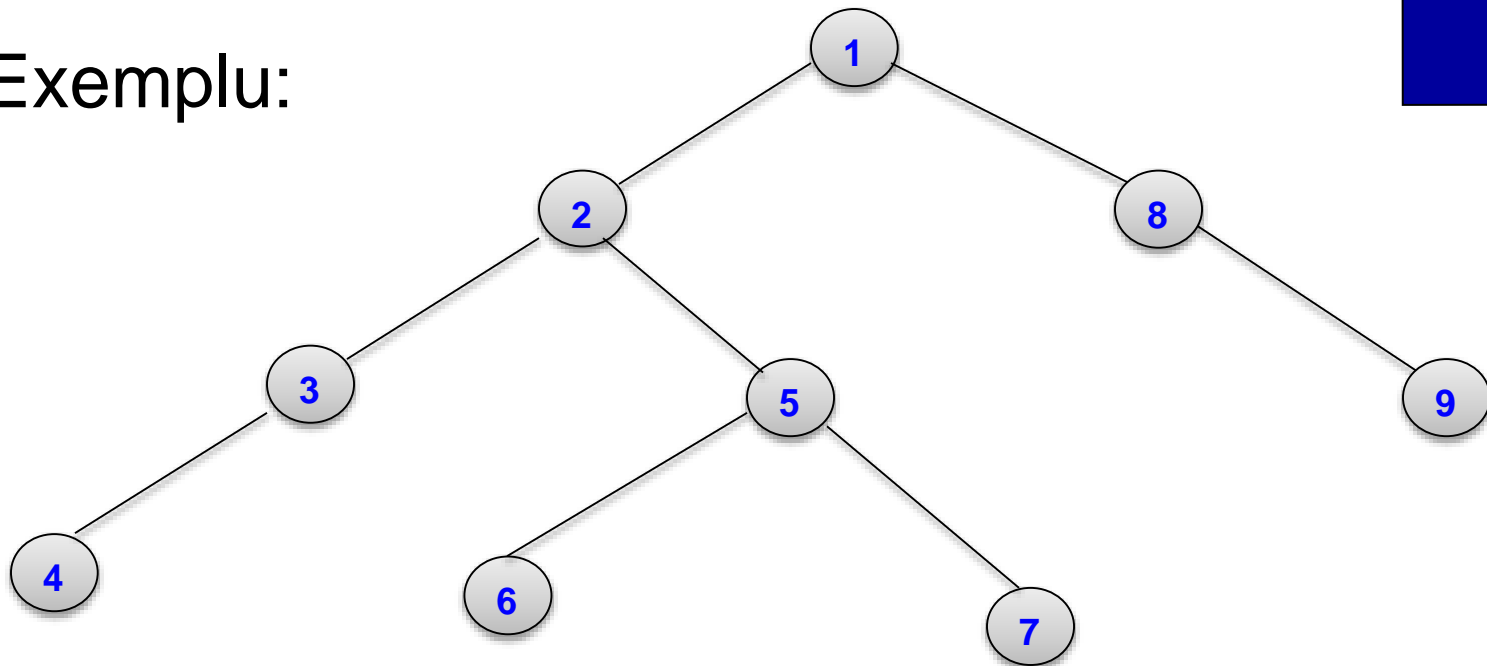
Tipuri de **reprezentari statice**:

2) Cu ajutorul altor doi vectori:

Vectorul **tata** care retine valorile care sunt descendenti ai fiecarui nod in parte

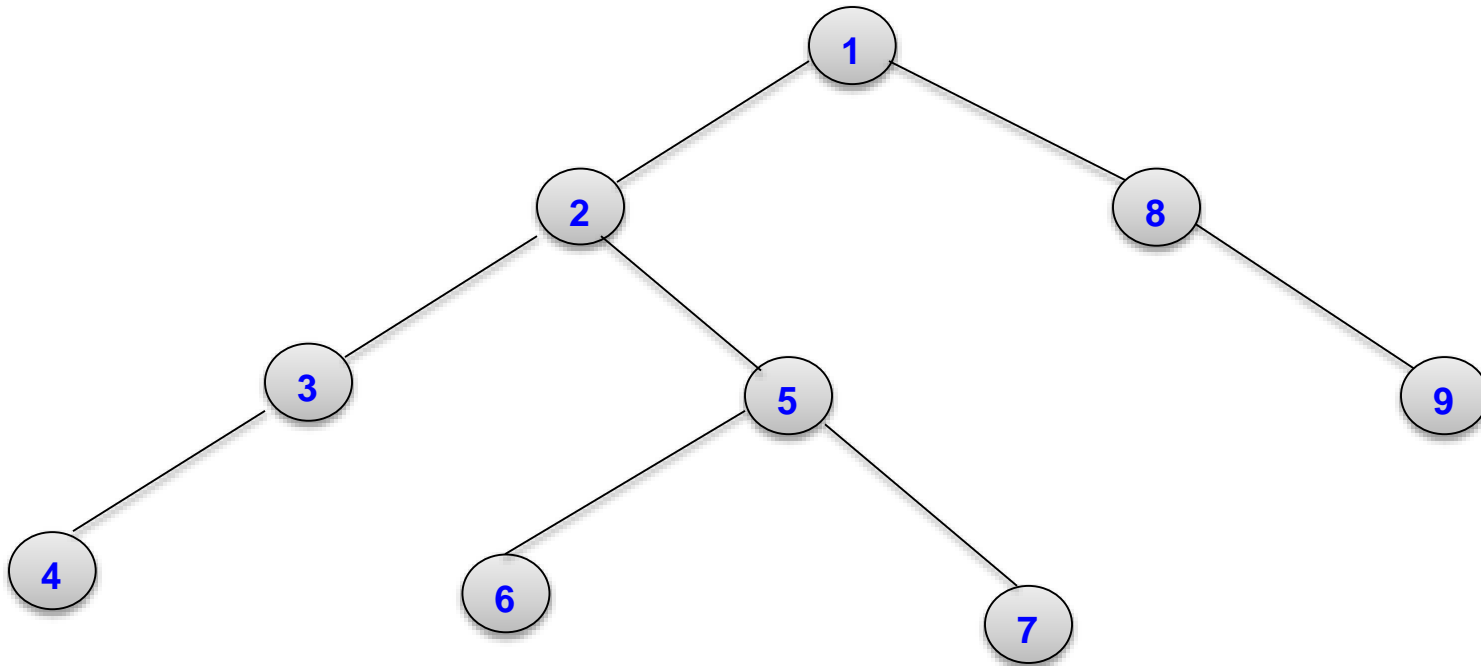
Vectorul **desc** care are valori de -1 sau 1 prin care se specifica daca un nod este descendent stang, respectiv drept.

Exemplu:



Presupunând că informația atașată fiecărui vârf este chiar numărul său de ordine, avem:

- $rad = 1$
- $st = (2,3,4,0,6,0,0,0,0)$
- $dr = (8,5,0,0,7,0,0,9,0)$
- $info = (1,2,3,4,5,6,7,8,9)$



Pentru exemplul de mai sus:

- tata = (0,1,2,3,2,5,5,1,8)
- desc = (0,-1,-1,-1,1,-1,1,1,1)
- info = (1,2,3,4,5,6,7,8,9)

9.3. Arbori binari

Definitie

Numim **nod terminal** sau **frunza**, un nod fara descendenti.

Definitie

Numim **arbore binar complet**, un arbore binar in care fiecare nod care nu este terminal are exact doi descendenti.

Propozitie

Un arbore binar complet care are n noduri terminale, toate situate pe acelasi nivel, are in total $2*n-1$ noduri.

9.3. Arbori binari

Parcurgerea arborilor binari

Problema *parcurgerii unui arbore binar* constă în identificarea unei modalități prin care, plecând din rădăcină și mergând pe muchii, să ajungem în toate vârfurile; în plus, atingerea fiecărui vârf este pusă în evidență o *singură dată*: spunem că *vizităm* vârful respectiv.

Acțiunea întreprinsă la vizitarea unui vârf depinde de problema concretă și poate fi de exemplu tipărirea informației atașate vârfului.

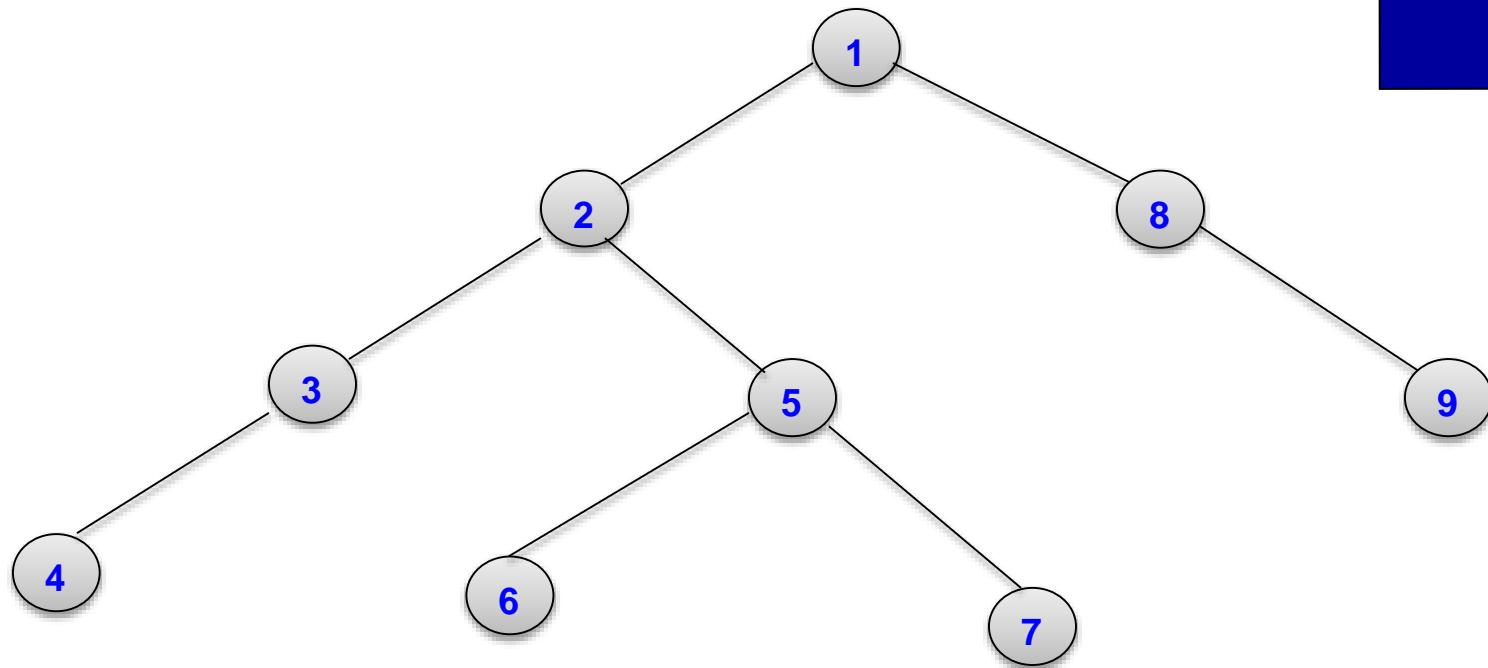
9.3. Arbori binari

Distingem trei modalități standard de parcurgere a unui arbore binar:

1) Parcurgerea în preordine

- Se parcurg recursiv în ordine: *rădăcina*, *subarborele stâng*, *subarborele drept*.
- Concret, se execută apelul `preordine(rad)` pentru procedura:

```
void preordine(int x)
{
    if( x!=0 ){
        cout<<x<<" ";
        preordine(st(x));
        preordine(dr(x));
    }
    return;
}
```



Pentru arborele binar de mai sus acest mod de parcurgere, este precizat figurând îngroșat rădăcinile subarborilor ce trebuie dezvoltați:

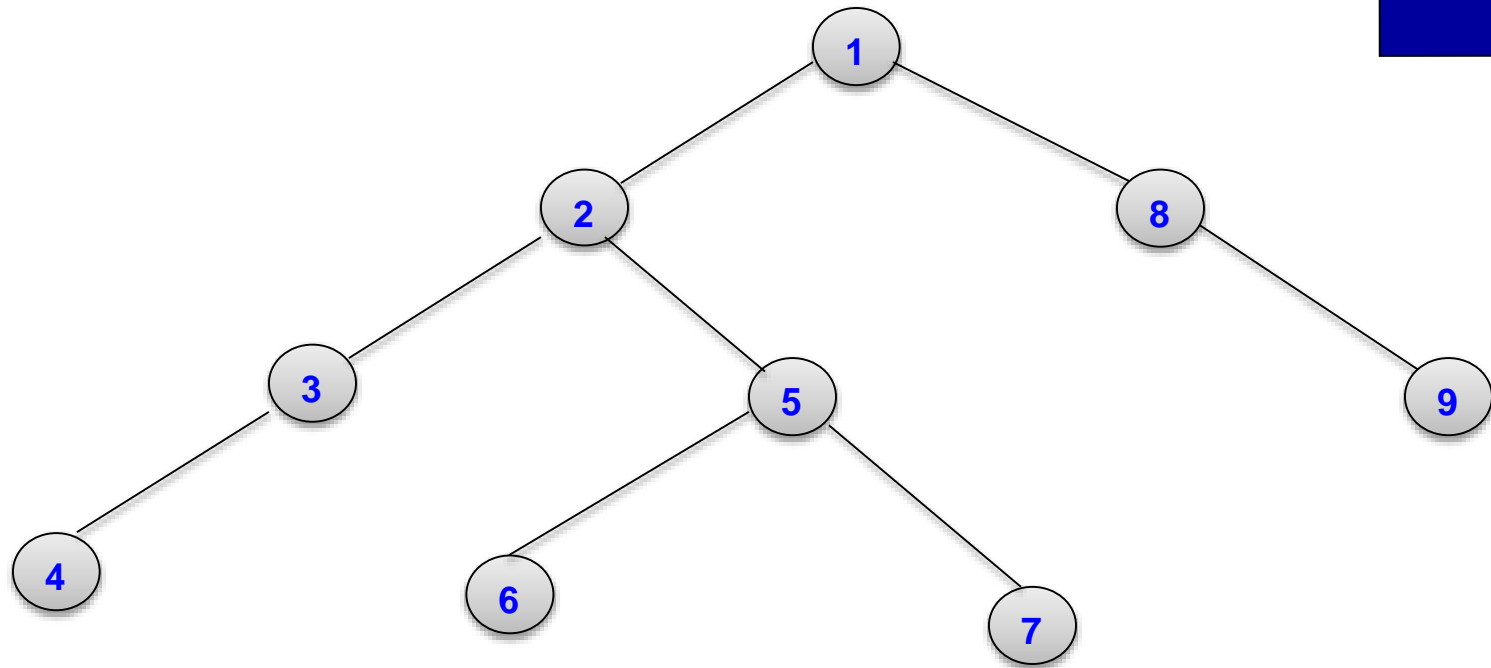
- **1**
- 1, **2**, **8**
- 1, 2, **3**, **5**, 8, 9
- 1, 2, 3, 4, 5, 6, 7, 8, 9

9.3. Arbori binari

2) Parcurgerea în inordine

- Se parcurg recursiv în ordine: *subarborele stâng, rădăcina, subarborele drept.*
- Concret, se execută apelul `inordine(rad)` pentru procedura:

```
void inordine(int x)  
{  
  if( x!=0 )  
  {  
    inordine(st(x));  
    cout<<x<<" ";  
    inordine(dr(x));  
  }  
  return;  
}
```



Modul de parcurgere pentru exemplul de mai sus:

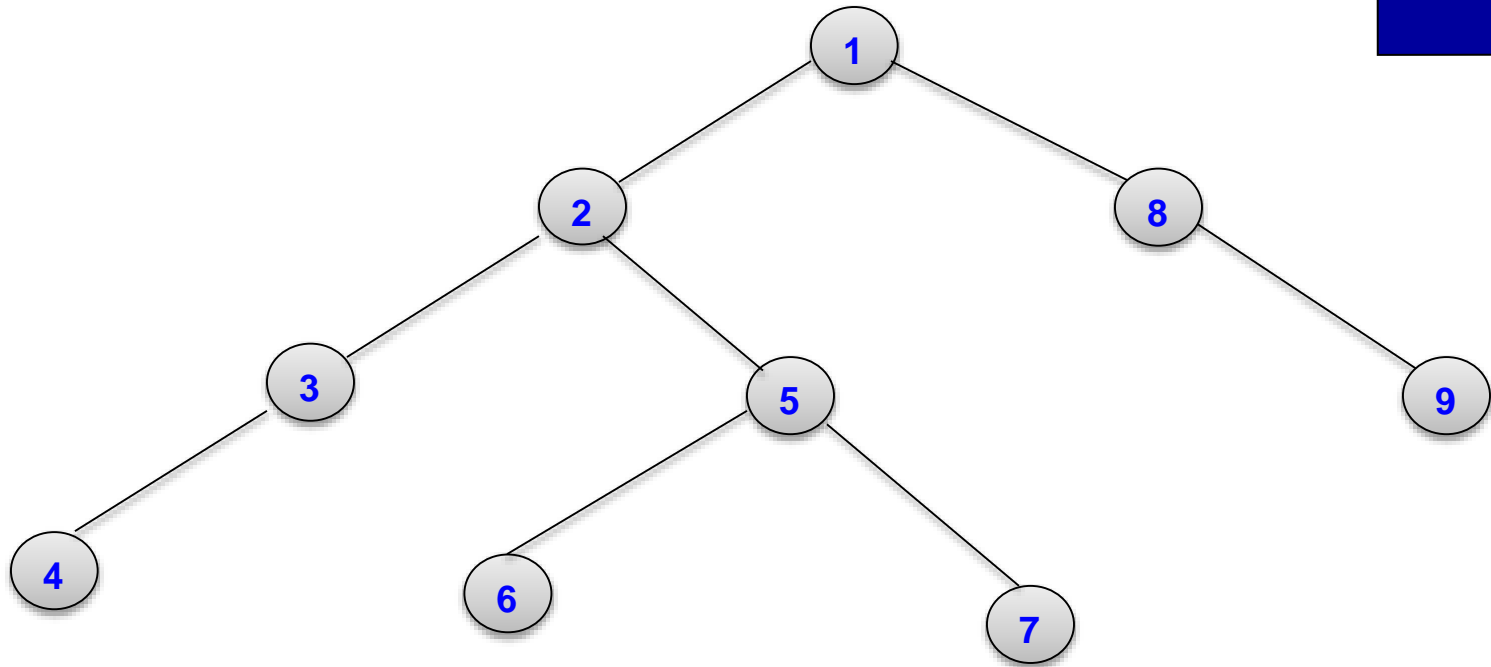
- **1**
- **2**, 1, **8**
- **3**, 2, **5**, 1, 8, 9
- 4, 3, 2, 6, 5, 7, 1, 8, 9

9.3. Arbori binari

3) Parcurgerea în postordine

- Se parcurg recursiv în ordine: *subarborele stâng, subarborele drept, rădăcina*.
- Concret, se execută apelul `postordine(rad)` pentru procedura:

```
void postordine(int x)  
{  
  if( x!=0 )  
  {  
    postordine(st(x));  
    postordine(dr(x));  
    cout<<x<<" ";  
  }  
  return;  
}
```



Modul de parcurgere pentru exemplul de mai sus:

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

9.3. Arbori binari

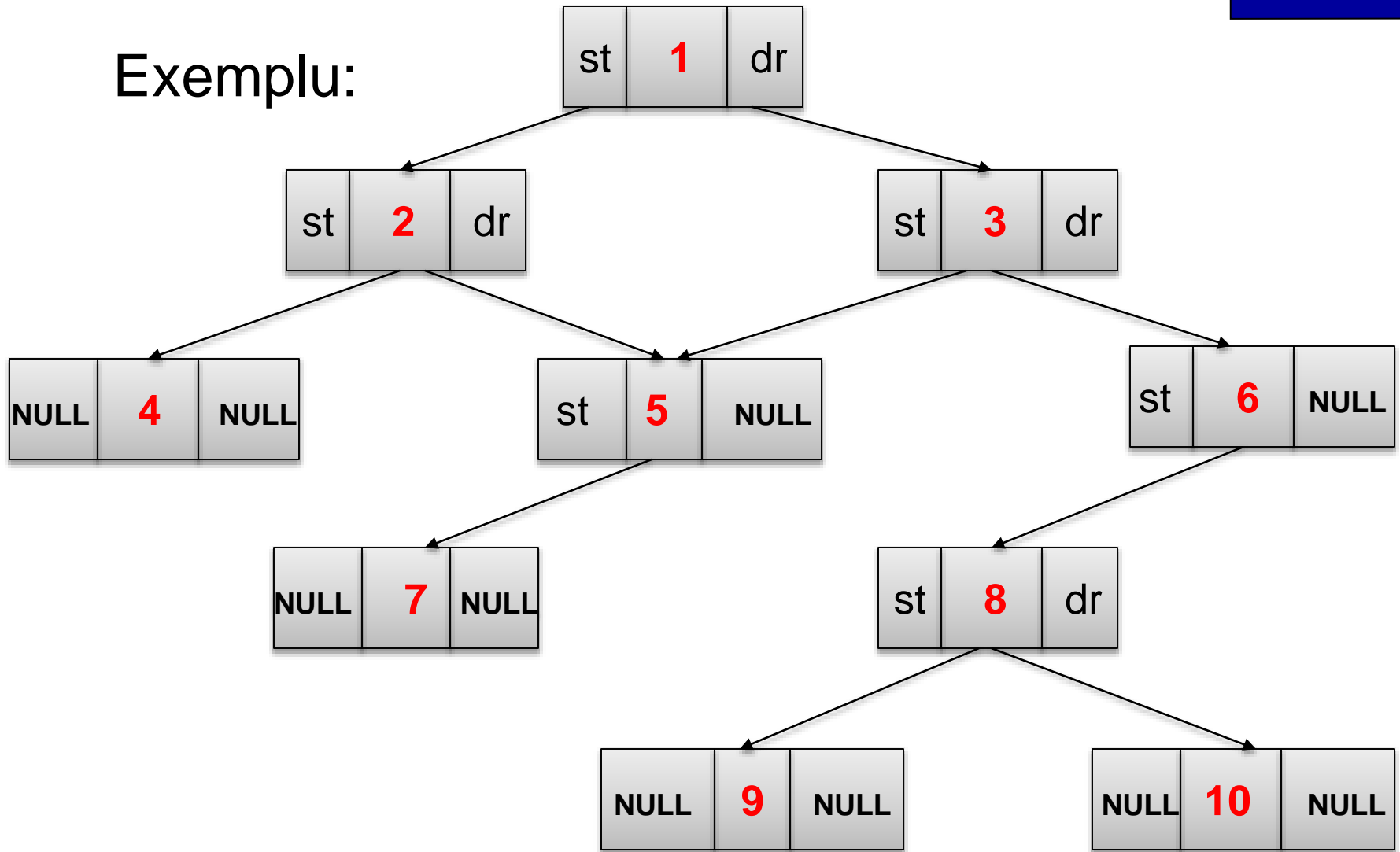
Folosirea **alocării dinamice a memoriei** (listelor dublu înlantuite) pentru a implementa arborii binari: Se poate declara o structura de tip dinamic, astfel:

```
typedef struct arbore  
{  
    int inf;  
    struct arbore *st, *dr;  
}arbore;
```

Pointerul care retine adresa radacinii se poate declara astfel:

```
arbore *rad;
```

Exemplu:



Codul sursa pentru creare a unui arbore binar cu ajutorul alocarii dinamice a memoriei si apoi traversarea arborelui:

```
# include <iostream.h>  
# include <ctype.h>  
typedef struct arbore  
{  
    int inf;  
    struct arbore *st,*dr;  
}arbore;
```

```
arbore *creare(void)
{
    arbore *aux;
    char ch;
    cout<<"\n Introduceti nod NULL? [d/n]";
    cin>>ch;
    ch=toupper(ch);
    if (ch=='N')
    {
        aux = new arbore;
        cout<<"\n informatia="; cin>>aux->inf;
        cout<<"\n urmeaza succesorul stang al nodului cu informatia
"<<aux->inf;
        aux->st=creare();
        cout<<"\n urmeaza succesorul drept al nodului cu informatia
"<<aux->inf;
        aux->dr=creare();
        return aux;
    }
    else return NULL;
}
```

```
void preordine(arbore *a)
{
    if (a!=NULL)
    {
        cout<<" "<<a->inf;
        preordine (a->st);
        preordine (a->dr);
    }
}

void inordine(arbore *a)
{
    if(a!=NULL)
    {
        inordine (a->st);
        cout<<" "<<a->inf;
        inordine (a->dr);
    }
}
```

```
void postordine(arbore *a)
{
    if (a!=NULL)
    {
        postordine (a->st);
        postordine (a->dr);
        cout<<" " <<a->inf;
    }
}
int main(void)
{
    arbore *a;
    a=creare();
    cout<<"\n parcurgerea preordine este ";
    preordine (a);
    cout<<"\n parcurgerea inordine este ";
    inordine (a);
    cout<<"\n parcurgerea postordine este ";
    postordine (a);
}
```



```

informatia=1
urmeaza succesorul stang al nodului cu informatia 1
e nod NULL?[d/n]n

informatia=2
urmeaza succesorul stang al nodului cu informatia 2
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 2
e nod NULL?[d/n]d

urmeaza succesorul drept al nodului cu informatia 1
e nod NULL?[d/n]n

informatia=3
urmeaza succesorul stang al nodului cu informatia 3
e nod NULL?[d/n]n

informatia=5
urmeaza succesorul stang al nodului cu informatia 5
e nod NULL?[d/n]n

informatia=6
urmeaza succesorul stang al nodului cu informatia 6
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 6
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 5
e nod NULL?[d/n]n

informatia=7
urmeaza succesorul stang al nodului cu informatia 7
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 7
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 3
e nod NULL?[d/n]n

informatia=4
urmeaza succesorul stang al nodului cu informatia 4
e nod NULL?[d/n]d
urmeaza succesorul drept al nodului cu informatia 4
e nod NULL?[d/n]d

parcurgerea RSD este 1 2 3 5 6 7 4
parcurgerea SRD este 2 1 6 5 7 3 4
parcurgerea SDR este 2 6 7 5 4 3 1

```

Codul sursa pentru creare a unui arbore binar cu ajutorul vectorilor st si dr si apoi traversarea arborelui:

```
# include <iostream.h>
int st[15],dr[15],n,i,j,rad;
void inordine(int x)
{
    if (st[x]!=0) inordine(st[x]);
    cout<<" "<<x;
    if (dr[x]!=0) inordine(dr[x]);
}
void preordine(int x)
{
    cout<<" "<<x;
    if (st[x]!=0) preordine(st[x]);
    if (dr[x]!=0) preordine(dr[x]);
}
```

```
void postordine(int x)
{
    if (st[x]!=0) postordine(st[x]);
    if (dr[x]!=0) postordine(dr[x]);
    cout<<" "<<x;
}
int main(void)
{
    cout<<"Dati numarul de noduri n = ";cin>>n;
    cout<<"Dati radacina rad = ";cin>>rad;
    for(i=1;i<=n;i++)
    {
        cout<<"st["<<i<<"]="<<st[i];
        cout<<"dr["<<i<<"]="<<dr[i];
    }
    cout<<"\n parcurgerea in inordine este ";
    inordine(rad);
    cout<<"\n parcurgerea in preordine este ";
    preordine(rad);
    cout<<"\n parcurgerea in postordine este ";
    postordine(rad);
}
```

```
c:\E:\Universitate\2009-2010\Semestrul
Dati numarul de noduri n = 7
Dati radacina rad = 1
st[1]=2
dr[1]=3
st[2]=0
dr[2]=0
st[3]=5
dr[3]=4
st[4]=0
dr[4]=0
st[5]=6
dr[5]=7
st[6]=0
dr[6]=0
st[7]=0
dr[7]=0

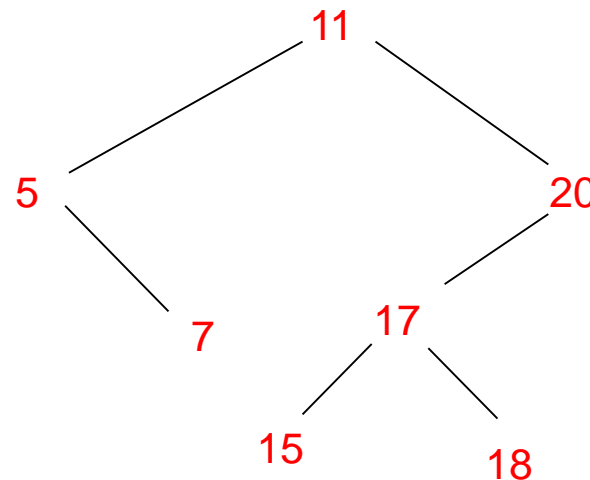
parcurerea in inordine este 2 1 6 5 7 3 4
parcurerea in preordine este 1 2 3 5 6 7 4
parcurerea in postordine este 2 6 7 5 4 3 1

Terminated with return code 0
Press any key to continue ...
```

Arbori de sortare

Definitie

Un **arbore de sortare** este un arbore binar în care pentru orice vârf informația atașată vârfului este mai mare decât informațiile vârfurilor din subarborele stâng și mai mică decât informațiile vârfurilor din subarborele drept.



Observație.

Parcurgerea în inordine a unui arbore de căutare produce informațiile atașate vârfurilor în ordine crescătoare.

Întrebări?