Marian Vlad

ARBORI

CRAIOVA
Martie, 2019

ARBORI

Fie G un graf orientat. G este un arbore cu radacina r, daca exista in G un varf r din care oricare alt varf poate fi ajuns printr-un drum unic.

Definitia este valabila si pentru cazul unui graf neorientat, alegerea unei radacini fiind insa in acest caz arbitrara: orice arbore este un arbore cu radacina, iar radacina poate fi fixata in oricare varf al sau. Aceasta, deoarece dintr-un varf oarecare se poate ajunge in oricare alt varf printr-un drum unic.

Cand nu va fi pericol de confuzie, vom folosi termenul “arbore”, in loc de termenul corect “arbore cu radacina”. Cel mai intuitiv este sa reprezentam un arbore cu radacina, ca pe un arbore propriu-zis. In Figura 3.1, vom spune ca beta este tatal lui delta si fiul lui alpha, ca beta si gamma sunt frati, ca delta este un descendent al lui alpha, iar alpha este un ascendent al lui delta. Un varf terminal este un varf fara descendenti. Varfurile care nu sunt terminale sunt neterminale. De multe ori, vom considera ca exista o ordonare a descendentilor aceluiasi parinte: beta este situat la stanga lui gamma, adica beta este fratele mai varstnic al lui gamma.

Orice varf al unui arbore cu radacina este radacina unui subarbore constand din varful respectiv si toti descendentii sai. O multime de arbori disjuncti formeaza o padure.

Intr-un arbore cu radacina vom adopta urmatoarele notatii. Adancimea unui varf este lungimea drumului dintre radacina si acest varf; inaltimea unui varf este lungimea celui mai lung drum dintre acest varf si un varf terminal; inaltimea arborelui este inaltimea radacinii; nivelul unui varf este inaltimea arborelui, minus adancimea acestui varf.

Reprezentarea unui arbore cu radacina se poate face prin adrese, ca si in cazul listelor inlantuite. Fiecare varf va fi memorat in trei locatii diferite, reprezentand informatia propriu-zisa a varfului (valoarea varfului), adresa celui mai varstnic fiu si adresa urmatorului frate. Pastrand analogia cu listele inlantuite, daca se cunoaste de la inceput numarul maxim de varfuri, atunci implementarea arborilor cu radacina se poate face prin tablouri paralele

Au fost studiate diferite tipuri de arbori binari, adică arbori pentru care e-gradul fiecărui nod este mai mic sau egal cu 2. Arborii care au e-gradul mai mare sau egal cu 2 se numesc arbori multicăi.

Dacă se doreşte să se prezinte descendenţa unei persoane din punct de vedere al strămoşilor, i se asociază persoanei doi părinţi, obţinându-se un arbore binar.

Dacă problema este abordată din punct de vedere al urmaşilor, atunci o familie poate avea mai mulţi copii, rezultând noduri cu mai multe ramuri.

Se consideră problema construirii şi explorării informaţiei conţinute în arbori de mari dimensiuni; se consideră şi operaţiile executate unor astfel de arbori.

Să notăm că astfel de arbori sunt păstraţi pe suporturi auxiliare; atunci nodurile arborelui sunt memorate pe un suport auxiliar şi sunt transferate pe rând sau pe grupe în memoria centrală.

Structurile dinamice sunt cele utilizate eficient pentru implementarea unor astfel de arbori. În acest caz pointerii nodurilor nu vor mai indica adrese de memorie.

Utilizând un arbore cu 10^6 noduri, vor fi necesare aproximativ $\log_2 10^6$ paşi pentru căutarea unor elemente.

Deoarece fiecare pas necesită un acces la memoria auxiliară rezultă necesitatea unei organizări care să reducă numărul de accese.

Este ştiut faptul că după realizarea accesului la un anumit element al memoriei auxiliare este uşor accesibil fiecare element al arborelui din zona respectivă. Acest lucru sugerează că un arbore poate fi divizat în subarbori ce pot fi reprezentaţi ca unităţi la care accesul se realizează deodată. Subarborii în care sunt divizaţi arborii de mari dimensiuni şi care au proprietatea de mai sus se numesc pagini.

Deci o pagină poate fi privită ca un nod de la care pot pleca oricâte ramuri; în acest mod arborele binar poate fi transformat într-un arbore multicăi. Dacă *p* este numărul de pagini, *n* numărul de noduri din arborele binar original, procesul de căutare a unui element va necesita $\log_2 p$ accese la memoria auxiliară, adică un număr mai mic de accese decât $\log_2 n$.

Pentru descompunerea în pagini a unui arbore binar trebuie avute în vedere următoarele aspecte:

a) modul de grupare a cheilor într-un arbore multicăi;

b) modul de plasare a elementelor corespunzătoare diverselor chei;

c) tehnica de inserare sau eliminare a unei chei;

d) modul de aranjare a cheilor în cadrul unui nod.

Dintre toate modurile de organizare a arborilor multicăi cel mai eficient este arborele 3-2, care reprezintă o variantă de arbore echilibrat; un nod al unui astfel de arbore poate avea cel mult 3 descendenţi direcţi.

Arbori partiali de cost minim

Fie $G = <X, V>$ un graf neorientat conex, unde *X* este multimea varfurilor si *U* este multimea muchiilor.Un arbore este un asemenea graf ce nu are cicluri. Fiecare muchie are un *cost* pozitiv (sau o *lungime* pozitiva). Pentru a gasi un arbore se pune problema sa gasim o submultime *A* inclusa in *U*, astfel incat toate varfurile din *X* sa ramina conectate atunci cand sunt folosite doar muchii din *A*.Numim arbore partial de cost minim acel arbore ce are multimea varfurilor X si a muchiilor A iar suma lungimilor muchiilor din *A* este minima.Cautam deci o submultime *A* de cost total minim care sa lege printr-un drum oricare doua noduri din X. Aceasta problema se mai numeste si *problema conectarii oraselor cu cost minim*, avand numeroase aplicatii.

Problema conectarii oraselor de cost minim:Se dau n orase precum si costul conectarii anumitor perechi de orase.Se cere sa se eleaga acele muchii care asigura existenta unui drum intre oricare doua orase astfel incat costul total sa fie minim.

Graful partial $<X, A>$ este un arbore si este *numit arborele partial de cost minim* al grafului *G* (*minimal spanning tree*). Un graf poate avea mai multi arbori partiali de cost minim si acest lucru se poate verifica pe un exemplu.Vom prezenta doi algoritmi greedy care determina arborele partial de cost minim al unui graf. In terminologia metodei greedy, vom spune ca o multime de muchii este o *solutie*, daca constituie un arbore partial al grafului *G*, si este *fezabila*, daca nu contine cicluri. O multime fezabila de muchii este *promitatoare*, daca poate fi completata pentru a forma solutia optima. O muchie *atinge* o multime data de varfuri, daca exact un capat al muchiei este in multime. Urmatoarea proprietate va fi folosita pentru a demonstra corectitudinea celor doi algoritmi.

Multimea initiala a candidatilor este *V*. Cei doi algoritmi greedy aleg muchiile una cate una intr-o anumita ordine, aceasta ordine fiind specifica fiecarui algoritm.

Algoritmul lui Kruskal

Arborele partial de cost minim poate fi construit muchie cu muchie, dupa urmatoarea metoda a lui Kruskal (1956): se alege intai muchia de cost minim, iar apoi se adauga repetat muchia de cost minim nealeasa anterior si care nu formeaza cu precedentele un ciclu. Alegem astfel X–1 muchii. Este usor de dedus ca obtinem in final un arbore. Este insa acesta chiar arborele partial de cost minim cautat?

Inainte de a raspunde la intrebare, sa consideram, de exemplu, graful din Figura 6.1.a. Ordonam crescator (in functie de cost) muchiile grafului: {1, 2}, {2, 3}, {4, 5}, {6, 7}, {1, 4}, {2, 5}, {4, 7}, {3, 5}, {2, 4}, {3, 6}, {5, 7}, {5, 6} si apoi aplicam algoritmul. Structura componentelor conexe este ilustrata, pentru fiecare pas, in Tabelul 1.

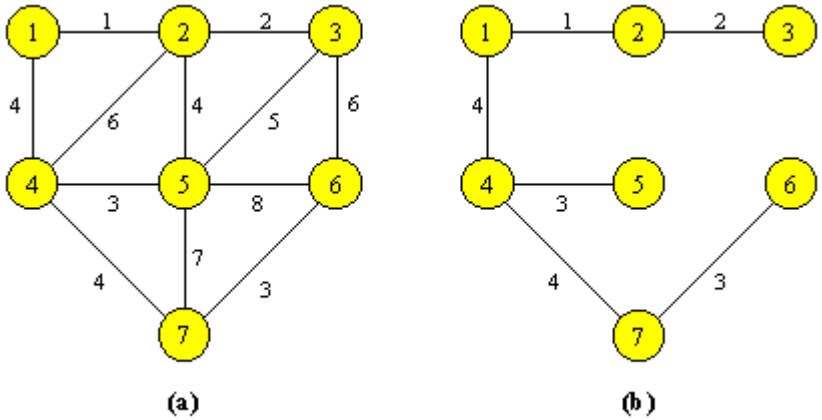


Figura 1 Un graf si arborele sau partial de cost minim.

Pasul	**Muchia considerata**	**Componentele conexe ale subgrafului** <X, A>
Initializare	—	{1}, {2}, {3}, {4}, {5}, {6}, {7}
1	{1, 2}	{1, 2}, {3}, {4}, {5}, {6}, {7}
2	{2, 3}	{1, 2, 3}, {4}, {5}, {6}, {7}
3	{4, 5}	{1, 2, 3}, {4, 5}, {6}, {7}
4	{6, 7}	{1, 2, 3}, {4, 5}, {6, 7}
5	{1, 4}	{1, 2, 3, 4, 5}, {6, 7}
6	{2, 5}	respinsa (formeaza ciclu)
7	{4, 7}	{1, 2, 3, 4, 5, 6, 7}

Tabelul 1 Algoritmul lui Kruskal aplicat grafului din Figura1a.

Multimea A este initial vida si se completeaza pe parcurs cu muchii acceptate (care nu formeaza un ciclu cu muchiile deja existente in A). In final, multimea A va contine muchiile {1, 2}, {2, 3}, {4, 5}, {6, 7}, {1, 4}, {4, 7}. La fiecare pas, graful partial <X, A> formeaza o padure de componente conexe, obtinuta din padurea precedenta unind doua componente. Fiecare componenta conexa este la randul ei un arbore partial de cost minim pentru varfurile pe care le conecteaza. Initial, fiecare varf formeaza o componenta conexa. La sfarsit, vom avea o singura componenta conexa, care este arborele partial de cost minim cautat (Figura 1b).Ceea ce am observat in acest caz particular este valabil si pentru cazul general.

In vectorul V vom sorta in ordine crescatoare numarul muchiilor in ordine crescatoare in functie de costul fiecareia.In vectorul X vom retine pentru fiecare nod numarul componenetei din care face parte acesta si care se schimba o data ce adaugam o noua muchie.Modificarea acestuia se

face in functie de apartenenta uneia dintre extremitati la un arbore cu mai mult de un nod.In multimea B se retin numerele de ordine ale muchiilor ce apartin arborelui de cost minim.

```
procedure sortare;      {se face sortarea intr-un vector v a muchiilor in ordine
var i,k,min:integer;      crescatoare a costurilor muchiilor}
    c:vector;
begin
  c:=a;k:=0;
  repeat
    min:=maxint;
    for i:=1 to m do
    if (c[i].cost<min) and (c[i].cost<>0) then
    begin
      min:=c[i].cost;
      j:=i;
    end;
    inc(k);
    v[k]:=j;
    c[j].cost:=0;
  until k=m;
end;
procedure kruskal;
var i,k,j:integer;
begin
  k:=0;
 for i:=1 to m do
 begin
     if (x[a[v[i]].vf1]=0) and (x[a[v[i]].vf2]=0) then  {ambele extremitati
        begin                                    formeaza un arbore cu un singur nod}
           b:=b+[v[i]];
           inc(k);
           x[a[v[i]].vf1]:=k;
           x[a[v[i]].vf2]:=k;
        end
     else
     if x[a[v[i]].vf1]<>x[a[v[i]].vf2] then   {se verifica daca extremitatile
     begin                                    muchiei sunt din arbori diferiti}
        b:=b+[v[i]];
       if (x[a[v[i]].vf1]<>0) and (x[a[v[i]].vf2]<>0) then
       begin
         for j:=1 to n do
         if (x[j]=x[a[v[i]].vf1]) and (a[v[i]].vf1<>j) then
          x[j]:=x[a[v[i]].vf2];
          x[a[v[i]].vf1]:=x[a[v[i]].vf2];
       end;
       if (x[a[v[i]].vf1]=0) then
           x[a[v[i]].vf1]:=x[a[v[i]].vf2];
       if (x[a[v[i]].vf2]=0) then
           x[a[v[i]].vf2]:=x[a[v[i]].vf1];
    end;
 suma:=suma+a[v[i]].cost;
 end;
end;
begin {main}
clrscr;
   citire;
   for i:=1 to m do
   v[i]:=0;
   sortare;
   writeln('Vectorul sortat in ordinea muchiilor este:');
   for i:=1 to m do
   write(v[i],' ');
   writeln;
```

```
   for i:=1 to n do
   x[i]:=0;
   b:=[];
   kruskal;
   writeln('Muchiile arborelui sunt:');
   suma:=0;
   for i:=1 to m do
   if i in b then
   begin
      writeln('Muchia ',i,':',a[i].vf1,' ',a[i].vf2,' de cost ',a[i].cost);
      suma:=suma+a[i].cost;
   end;
   write('Costul arborelui este:',suma);
readkey;
end.
```

Algoritmul lui Prim

Cel de-al doilea algoritm greedy pentru determinarea arborelui partial de cost minim al unui graf se datoreaza lui Prim (1957). In acest algoritm, la fiecare pas, multimea *A* de muchii alese impreuna cu multimea *X* a varfurilor pe care le conecteaza formeaza un arbore partial de cost minim pentru subgraful <*X*, *A*> al lui *G*. Initial, multimea *W* a varfurilor acestui arbore contine un singur varf oarecare din *X*, care va fi radacina, iar multimea *A* a muchiilor este vida. La fiecare pas, se alege o muchie de cost minim, care se adauga la arborele precedent, dand nastere unui nou arbore partial de cost minim (deci, exact una dintre extremitatile acestei muchii este un varf in arborele precedent). Arborele partial de cost minim creste "natural", cu cate o ramura, pina cand va atinge toate varfurile din *X*, adica pina cand *W* = *X*. Functionarea algoritmului, pentru exemplul din Figura 6.1a, este ilustrata in Tabelul 2. La sfarsit, *A* va contine aceleasi muchii ca si in cazul algoritmului lui Kruskal.

Pasul	**Muchia considerata**	***W***
Initializare	—	{1}
1	{2, 1}	{1, 2}
2	{3, 2}	{1, 2, 3}
3	{4, 1}	{1, 2, 3, 4}
4	{5, 4}	{1, 2, 3, 4, 5}
5	{7, 4}	{1, 2, 3, 4, 5, 6}
6	{6, 7}	{1, 2, 3, 4, 5, 6, 7}

Tabelul 6.2 Algoritmul lui Prim aplicat grafului din Figura 6.1a.

Descrierea algoritmului este data in continuare.

Pentru a obtine o implementare simpla, presupunem ca: varfurile din *X* sunt numerotate de la 1 la *n*, *X* = {1, 2, ..., *n*}; matricea simetrica *C* da costul fiecarei muchii, cu *C*[*i*, *j*] = maxint, daca muchia {*i*, *j*} nu exista. Folosim doi vectori,vectorul parintilor T[i] si un vector s[i].

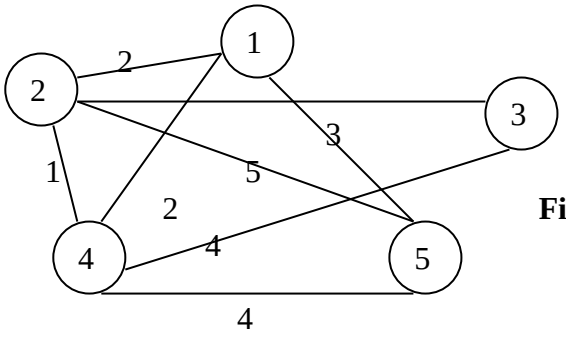
Vectorul T este vectorul tata in care ,pentru fiecare nod i din X,T[i] este egal cu parintele lui i. Vectorul S este definit astfel:

S[i]= 0 daca i apartine arborelui partial construit pana atunci

K daca :- i nu apartine arborelui partial deja construit

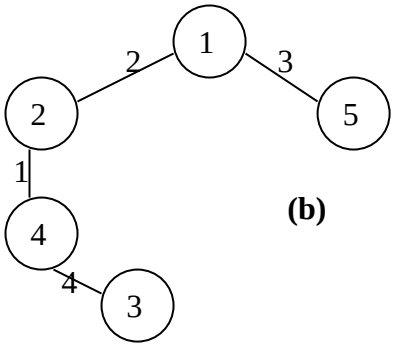
-muchia de cost minim care uneste i cu un nod din graful deja construit este [i,k] cu k neapartinand arborelui partial

Initial vectorul tata este 0 peste tot iar vectorul S este definit astfel:S[v]=0 si S[i]=v pentru i<>v,unde v este varful arborelui.Se alege apoi muchia de cost minim (i,j) care are numai o extremitate in arborele partial construit adica S[i]=0 iar S[j]<>0.Se reactualizeaza cei doi vectori:vectorul S pentru j adica S[j]=0 iar vectorul tata T[j]=S[j].Se reia cautarea muchiei de cost minim daca nu au fost alese n-1 muchii.



(a)

Figura 2



(b)

Aplicand acest algoritm pentru graful din Figura 2.a,se vor urma pasii:
-n=5,a matricea de cost si varful
-se alege varful,de exemplu v=1 iar vectorii sunt:

S

0	1	1	1	1

T

0	0	0	0	0

-si ia i=2,n si se alege muchia de cost minim determinata de a[i,S[i]],(in acest caz se alege j=2).
-se reactualizeaza vectorii T si S;T[j]=S[j](T[2]=1) si S comparandu-se valoarea muchiei [i,S[i]] cu cea a muchiei [i,j] si daca este mai mica se modifica S(S[i]=j) unde S[i]<>0 si j este ultimul varf introdus.Cei doi vectori vor fi:

S

0	0	2	2	1

T

0	1	0	0	0

-se cauta din nou muchia de cost minim si se repeta faza precedunta pana se aleg n-1 muchii(in cazul Figurii 2.a, 4 muchii).Vectorii vor suferii urmatoarele transformari:

1. S

0	0	4	0	1

T

0	1	0	2	0

. 2. S

0	0	4	0	0

T

0	1	0	2	1

3. S

0	0	0	0	0

T

0	1	4	2	1

-in final vectorul S va fi zero iar vectorul T va fi vectorul tata a arboreluipartial de cost minim.Costul arborelui(Figura 2.b) va fi 10.

Se cere sa se dea varful dar aceste poate fi luat intotdeauna 1 daca se cauta sa se afle numai costul arborelui deoarece,muchiile nefiind orientate, se obtine intotdeauna acelasi arbore.Cel ce difera este insa vectorul tata in funtie de varful de pornire dar acesta poate fi refacut dupa vectorul tata al grafului ce are varful 1.

Algoritmul lui Prim de determinare al arborelui partial de cost minim este:

```
procedure prim;
var i,j,min:integer;
begin
   for i:=1 to n do
   s[i]:=v;
   s[v]:=0;
   for i:=1 to n do
   t[i]:=0;
   cost:=0;
   for k:=1 to n-1 do
   begin
      min:=maxint;
      for i:=1 to n do
      if (s[i]<>0) then
        if (a[s[i],i]<min) and (a[s[i],i]<>0) then
        begin
          min:=a[s[i],i];
          j:=i;
        end;
```

```
        t[j]:=s[j];
        cost:=cost+a[j,s[j]];
        s[j]:=0;
      for i:=1 to n do
      if  (s[i]<>0) then
      if (a[i,s[i]]=0) or (a[i,s[i]]>a[i,j]) then
      if a[i,j]<>0 then
      s[i]:=j;
   end;
end;
```

Bucla principala se executa de n–1 ori si buclele **for** din interior necesita un timp in $O(n)$. Algoritmul *Prim* necesita, deci, un timp in $O(n^2)$. Am vazut ca timpul pentru algoritmul lui Kruskal este in $O(m \log n)$. Pentru un graf *dens* (adica, cu foarte multe muchii), se deduce ca m se apropie de $n(n–1)/2$. In acest caz, algoritmul *Kruskal* necesita un timp in $O(n^2 \log n)$ si algoritmul *Prim* este probabil mai bun. Pentru un graf *rar* (adica, cu un numar foarte mic de muchii), m se apropie de n si algoritmul *Kruskal* necesita un timp in $O(n \log n)$, fiind probabil mai eficient decat algoritmul *Prim*.

Probleme propuse:

Problema 1. Se dau n orase. Se cunoaste distanta dintre oricare doua orase.Un distribuitor de carte cauta sa-si faca un depozit in unul dintre aceste orase.Se cere sa se gaseasca traseul optim de la depozit catre celelalte orase astfel incat distanta totala pe care o va parcurge pentru a distribui in toate celelalte n-1 orase sa fie minima.sa se precizeze care ar fi orasul in care sa se afle depoitul pentru ca toate celelalte orase sa fie usor accesibile{din acel centru de depozitare sa se poata pleca sper cat mai multe alte orase}.
Date se citesc dintr-un fisier astfel:
-pe prima linie numarul de orase
-pe urmatoarele linii traseele existente astfel:orasul din care pleaca,orasul in care ajunge si distanta in km a drumului.

{Deoarece vor exista foarte multe trasee algoritmul lui Prim este mai bun.Fiind un graf neorientat se poate pleca cu varful 1.Pentru a afla care ar fi orasul optim vedem inarbore care este nodul cu cei mai multi fii si refacem vectorul tata.}

Rezolvare:

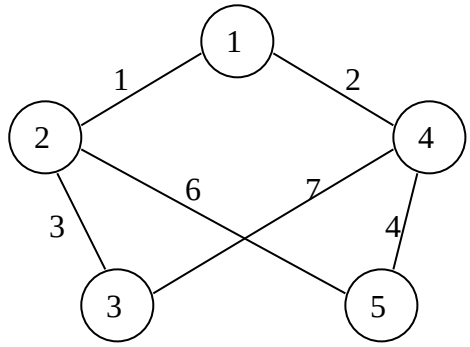
```
program oras_depozit;
uses crt;
type muchie=record
     vf1,vf2,cost:integer;
     end;
type vector=array[1..100] of longint;
     vector1=array[1..100] of muchie;
     matrice=array[1..50,1..50] of longint;
var n,i,j,k,v,cost:integer;
    s,t:vector;
    x:vector1;
    a:matrice;
    f:text;
procedure citire;
var i,j,m:integer;
begin
    assign(f,'depozit.txt');
    reset(f);
    readln(f,n);m:=0;
    while not eof(f) do
    begin
       inc(m);
       read(f,x[m].vf1);
       read(f,x[m].vf2);
       read(f,x[m].cost);
       readln(f);
    end;
    for i:=1 to m do
    begin
        a[x[i].vf1,x[i].vf2]:=x[i].cost;
        a[x[i].vf2,x[i].vf1]:=x[i].cost;
    end;
    writeln('Matricea costurilor este:');
    for i:=1 to n do
    begin
      for j:=1 to n do
      write(a[i,j],' ');
      writeln;
    end;
end;
```

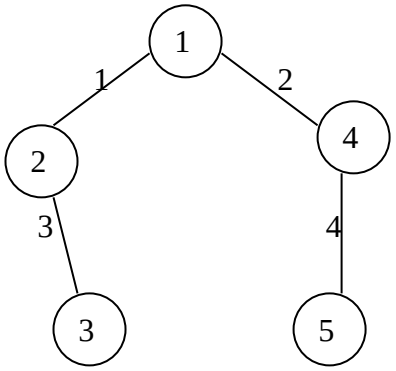
```
procedure prim;
var i,j,min:integer;
begin
   for i:=1 to n do
   s[i]:=v;
   s[v]:=0;
   for i:=1 to n do
   t[i]:=0;
   cost:=0;
   for k:=1 to n-1 do
   begin
      min:=maxint;
      for i:=1 to n do
      if (s[i]<>0) then
        if (a[s[i],i]<min) and (a[s[i],i]<>0) then
        begin
          min:=a[s[i],i];
          j:=i;
        end;
        t[j]:=s[j];
        cost:=cost+a[j,s[j]];
        s[j]:=0;
      for i:=1 to n do
      if  (s[i]<>0) then
      if (a[i,s[i]]=0) or (a[i,s[i]]>a[i,j]) then
      if a[i,j]<>0 then
      s[i]:=j;
   end;
end;
function fii(x:integer):integer;
var k:integer;
begin
  k:=0;
  for i:=1 to n do
  if t[i]=x then
  inc(k);
  fii:=k;
end;
procedure tata(v:integer);
var i:integer;
begin
  for i:=1 to n do
  if t[v]=i then
  begin
    t[i]:=v;
    t[v]:=0;
  end
end;
procedure oras;
var max,i,j:integer;
begin
   max:=0;
   for i:=1 to n do
   if fii(i)>max then
   max:=fii(i);
   writeln('Orasele optime sunt:');
   for i:=1 to n do
   if fii(i)=max then
   begin
      write(i,' ');
      tata(i);
      write('Vectorul tata este:');
      for j:=1 to n do write(t[j],' ');
      writeln;
```

```
   end;
end;
begin {main}
clrscr;
   citire;
   writeln('Dati vf de pornire!');readln(v);
   prim;
   writeln('Costul arborelui este:',cost);
   oras;
readkey;
end.
```

Problema 2. Se da un graf neorientat.Sa se creeze un arbore partial de cost minim care sa poata fi memorat apoi sub forma unei liste.

Exemplu:





{Se foloseste algoritmul lui Prim iar pentru fiecare nou nod introdus se verifica daca parintele sau{s[i]} are mai un fiu sau cel mult unul in cazul in care este chiar varful.Daca sunt indeplinite aceste conditii se adauga varful la arbore ,daca nu se aplica algoritmul lui prim pentru o noua matrice A1.Matricea A1 se obtine din A punand costul mechiei care era minima in A egala cu 0.Se repeta procesul pana s-au introdus n-1 varfuri}

Rezolvare:

```
program arbore_lista;
uses crt;
type muchie=record
     vf1,vf2,cost:integer;
     end;
type vector=array[1..50] of longint;
     vector1=array[1..100] of muchie;
     matrice=array[1..50,1..50] of longint;
var n,i,j,k,v,cost,y,z,m:integer;
    s,t,s1,t1:vector;
    x:vector1;
    a,a1:matrice;
    f:text;
procedure citire;
var i,j,m:integer;
begin
    assign(f,'depozit.txt');
    reset(f);
    readln(f,n);m:=0;
    while not eof(f) do
    begin
       inc(m);
       read(f,x[m].vf1);
       read(f,x[m].vf2);
```

```
       read(f,x[m].cost);
       readln(f);
    end;
    for i:=1 to m do
    begin
        a[x[i].vf1,x[i].vf2]:=x[i].cost;
        a[x[i].vf2,x[i].vf1]:=x[i].cost;
    end;
    writeln('Matricea costurilor este:');
    for i:=1 to n do
    begin
      for j:=1 to n do
      write(a[i,j],' ');
      writeln;
    end;
end;
function fii(y:integer):integer;
var k,j:integer;
begin
  k:=0;
  for j:=1 to n do
  if t[j]=y then
  inc(k);
  fii:=k;
end;
procedure prim(a:matrice);
var i,j,min:integer;
begin
      min:=maxint;
      for i:=1 to n do
      if (s[i]<>0) then
        if (a[s[i],i]<min) and (a[s[i],i]<>0) then
        begin
          min:=a[s[i],i];
          j:=i;
        end;
    if (((s[j]<>v) and (fii(s[j])=0)) or ((s[j]=v) and (fii(s[j])<=1))) then
    begin
        t[j]:=s[j];
        cost:=cost+a[j,s[j]];
        s[j]:=0;
        for i:=1 to n do
        if  (s[i]<>0) then
        if (a[i,s[i]]=0) or (a[i,s[i]]>a[i,j]) then
        if a[i,j]<>0 then
        s[i]:=j;
        inc(m);
     end
    else
     begin
       a1:=a;
       a1[s[j],j]:=0;
       prim(a1);
     end;
end;
begin {main}
clrscr;
   citire;
   writeln('Dati vf de pornire!');readln(v);
  m:=0;
   for i:=1 to n do
   s[i]:=v;
   s[v]:=0;
   for i:=1 to n do
```

```
    t[i]:=0;
    cost:=0;
    repeat prim(a);
   until m=n-1;
    write('Vectorul tata este: ');
    for i:=1 to n do
      write(t[i],' ');
      writeln;
    writeln('Costul arborelui este:',cost);
readkey;
end.
```

Problema 3. Se da un graf orientat si se cere sa se afle daca exista un arbore partial de cost minim.Dar o arborescenta de cost minim?Daca exista sa se afle care este este varful acesteia.

{Daca exista sau nu o arbore de cost minim se poate spune foarte usor daca verificam conexitatea grafului.Daca graful este tare conex atunci putem spune ca exista un arbore partial de cost minim.Un lant de la x la y va avea costul egal cu costul drumui de la x la y plus costul drumului de la y la x.Pentru a verifica daca exista o arborescenta aplicam unul din cei doi algoritmi.Daca,in cazul lui Prim,exista si un alt nod cu t[i]=0 cu exceptia varfului atunci aceasta nu este arborescenta iar in algoritmul Kruskal,daca multimea B a muchiilor are mai putin de n-1 muchii, atunci nu exista arborescenta.Se aplica algoritmii de n ori pentru fiecare nod drept varf pentru a vedea daca exista o arborescenta de cost minim.Muchiile vor fi orientate iar matricea costurilor nu va fi simetrica.}

```
program arborescenta;        {afiseaza arborescenta in cazul in care exista una}
uses crt;
type muchie=record
     vf1,vf2,cost:integer;
     end;
type vector=array[1..100] of longint;
     vector1=array[1..100] of muchie;
     matrice=array[1..50,1..50] of longint;
var n,i,j,k,v,cost:integer;
    s,t:vector;
    x:vector1;
    a:matrice;
    f:text;
procedure citire;
var i,j,m:integer;
begin
    assign(f,'orient.txt');
    reset(f);
    readln(f,n);m:=0;
    while not eof(f) do
    begin
      inc(m);
      read(f,x[m].vf1);
      read(f,x[m].vf2);
      read(f,x[m].cost);
      readln(f);
    end;
    for i:=1 to m do
       a[x[i].vf1,x[i].vf2]:=x[i].cost;
```

```
    writeln('Matricea costurilor este:');
    for i:=1 to n do
    begin
      for j:=1 to n do
      write(a[i,j],' ');
      writeln;
    end;
end;
procedure prim;
var i,j,min:integer;
begin
   for i:=1 to n do
   s[i]:=v;
   s[v]:=0;
   for i:=1 to n do
   t[i]:=0;
   cost:=0;
   for k:=1 to n-1 do
   begin
      min:=maxint;
      for i:=1 to n do
      if (s[i]<>0) then
        if (a[s[i],i]<min) and (a[s[i],i]<>0) then
        begin
          min:=a[s[i],i];
          j:=i;
        end;
        t[j]:=s[j];
        cost:=cost+a[s[j],j];
        s[j]:=0;
      for i:=1 to n do
      if  (s[i]<>0) then
      if (a[s[i],i]=0) or (a[s[i],i]>a[j,i]) then
      if a[j,i]<>0 then
      s[i]:=j;
   end;
end;
begin {main}
clrscr;
   citire;
   writeln('Dati vf de pornire!');readln(v);
   prim;
   writeln('Vectorul tata este:');
   for i:=1 to n do
   write(t[i],' ');
   writeln('Costul arborelui este:',cost);
readkey;
end.
```

Problema 4. Se da un graf conex.Se cere impartirea acestuia in m arbori partiali de cost minim fiecare cu p varfuri.Sa se afiseze acesti arbori.

{Cel mai favorabil ar fi algoritmul lui Prim.Acesta se aplica de m ori iar for din procedura va fi de la 1 la p-1.Nodurile care au fost incluse intr-un arbore precedent vor fi trecute in vectorii S si T cu valoarea –1 iar matricea va fi zero pe liniile si coloanele respective.}

```
program m_arbori;
uses crt;
type vector=array[1..100] of longint;
     matrice=array[1..50,1..50] of longint;
var n,i,j,k,v,cost,p,m:integer;
    s,t:vector;
    a:matrice;
    f:text;
procedure citire;
var i,j:integer;
begin
    assign(f,'prim.txt');
    reset(f);
    readln(f,n);
    for i:=1 to n do
    begin
      for j:=1 to n do
      read(f,a[i,j]);
      readln(f);
    end;
    writeln('Matricea costurilor este:');
    for i:=1 to n do
    begin
      for j:=1 to n do
      write(a[i,j],' ');
      writeln;
    end;
end;
procedure prim;
var i,j,min,h:integer;
begin
   cost:=0;
   for h:=1 to p-1 do
   begin
      min:=maxint;
      for i:=1 to n do
      if (s[i]>0) then
        if (a[s[i],i]<min) and (a[s[i],i]<>0) then
        begin
          min:=a[s[i],i];
          j:=i;
        end;
        t[j]:=s[j];
        cost:=cost+a[j,s[j]];
        s[j]:=0;
        write(j,' ');
      for i:=1 to n do
      if  (s[i]>0) then
      if (a[i,s[i]]=0) or (a[i,s[i]]>a[i,j]) then
      if a[i,j]<>0 then
      s[i]:=j;
      t[j]:=-1;
      s[j]:=-1;
      for i:=1 to n do
      begin
```

```
         a[i,j]:=0;
         a[j,i]:=0;
      end;
   end;
   write('Costul arborelui este:',cost);
end;
begin {main}
clrscr;
   citire;
   writeln('Dati vf de pornire!');readln(v);
   write('m=');read(m);
   write('p=');read(p);
   for i:=1 to n do
   s[i]:=v;
   s[v]:=0;
   for i:=1 to n do
   t[i]:=0;
   for k:=1 to m-1 do
   begin
      for i:=1 to n do
      begin
        if t[i]=0 then
        begin
           write(i,' ');
           prim;
           for j:=1 to n do
           if t[j]=0 then s[j]:=i;
           s[i]:=-1;writeln;
        end;
        s[v]:=-1;
        t[v]:=-1;
     end;
   end;
readkey;
end.
```