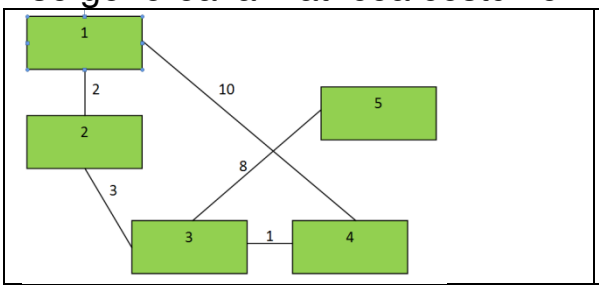
Algoritmul lui Roy-Floyd

Algoritmul Roy-Floyd este folosit in diverse domeni des intalnite, de la controlul avioanelor de pe un anumit aeroport pana la jocuri informatice, rolul principal fiind acela de gasire a drumului de cost minim intre un obiect principal si o tinta anume.

Fie **G=(V, E)** un graf neorientat, unde V are n elemente (n noduri) si E are m elemente (m muchii) memorat prin matricea ponderilor. Se cere ca pentru doua noduri x,y citite sa se determine lungimea minima a lantului de la x la y.

Algoritmul:
-se genereaza matricea costurilor:



0	2	∞	10	∞
2	0	3	∞	∞
∞	3	0	1	8
10	∞	1	0	∞
∞	∞	8	∞	0

Unde ∞ reprezinta plus infinit, drum ce nu exista.

Initial matricea costurilor pentru nodurile 1 si 4 va retine 10. Se observa ca lantul 1,2,3,4 determina o suma a costurilor mai mica: 2+3+1=6. Lungime minima a lantului de la 1 la 4 este 6.
-se incearca pentru oricare pereche de noduri i,j sa se obtina “drumuri” mai scurte prin noduri intermediare k (k∈1…n).

Acest lucru se determina comparand “lungimea” lantului a[i,j] cu lungimea lantului care trece prin k si daca:

a[i,j] > a[i,k]+a[k,j] atunci se atribuie: a[i,j] ← a[i,k]+a[k,j]

Nodurile ale caror costuri vor fi modificate sunt urmatoareale intrucat intre ele exista costuri mai mici.

Nod initial	Nod final	Cost initial	Cost final	Calea
1	**3**	∞	5	1-2-3

1	4	10	6	1-2-3-4
1	5	∞	13	1-2-3-5
2	4	∞	4	2-3-4
2	5	∞	11	2-3-5
4	5	∞	9	4-3-5
Astfel matricea costurilor va arata ca in partea dreapta.			**0** 2 5 6 13 2 **0** 3 4 11 5 3 **0** 1 8 6 4 1 **0** 9 13 11 8 9 **0**	

Astfel generarea matricii drumurilor optime se realizeaza cu urmatoarea secventa:

```
roy_floyd ()
  i, j, k
pentru k→1,n executa
  pentru i→1,n executa
    pentru j→1,n executa
      daca (A[i][k]<>INF si A[k][j]<>INF) atunci
        daca A[i][j]>A[i][k]+A[k][j] atunci
          A[i][j] ←A[i][k] + A[k][j]
```

In continuare, dupa determinarea matricii lanturilor optime, pentru doua noduri citite x, y se cere sa se reconstituie un lant optim de la x la y (pot fi mai multe solutii).

Solutie:

-se determina daca exista un lant de la x la y (ar putea sa nu existe un astfel de lant daca graful nu este conex):

cand a[x,y] $\neq\infty$

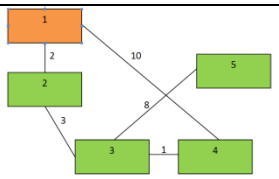
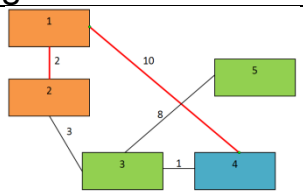
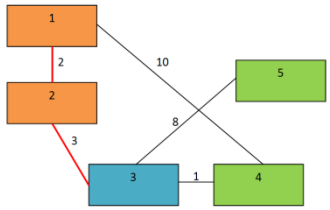
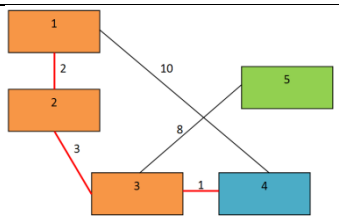
-se descompune “drumul” de la x la y prin ca atunci cand:
a[x][y]=a[x][k]+a[k][y];

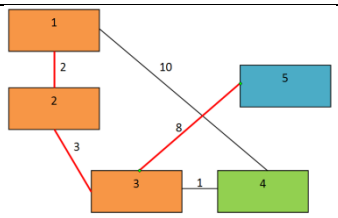
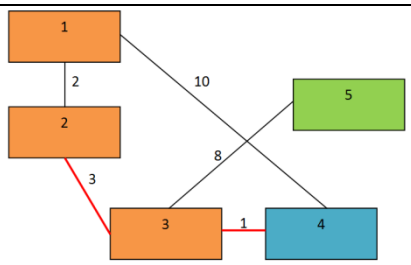
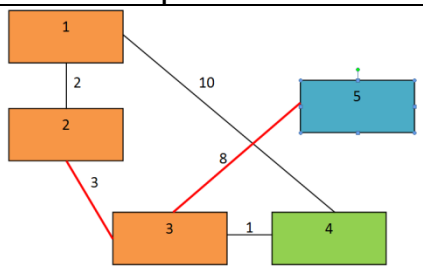
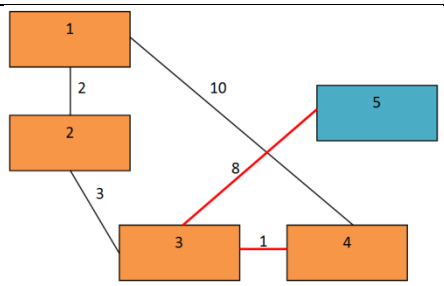
- - pentru un astfel de algoritm se utilizeaza strategia Divide et Impera

Astfel pentru drumul minim de la 1 la 4 se va afisa valoarea 6 si drumul 1,2,3,4.

```
descompun_drum
    ok=0
    k=1
    cat_timp k<=n && ok<>0 executa
     daca i<>k && j<>k atunci
      daca a[i][j] = a[i][k]+a[k][j] atunci
            descompun_drum(i,k)
               descompun_drum(k,j)
            ok←1
       k←k+1
        daca ok<>0
        scrie j
```

Modificarile in matricea costurilor se fac dupa cum urmeaza prin algoritmul Roy-Floyd.

0 2 ∞ 10 ∞ 2 0 3 ∞ ∞ ∞ 3 0 1 8 10 ∞ 1 0 ∞ ∞ ∞ 8 ∞ 0	 Initial matricea este completata cu costurile initiale.∞ indica un drum maxim care inca nu exista sau nu va exista daca graful nu este conex. Se incepe cu nodul 1 ca si vizitat.
0 2 ∞ 10 ∞ 2 0 3 **12** ∞ ∞ 3 0 1 8 10 **12** 1 0 ∞ ∞ ∞ 8 ∞ 0	 Cum nici un nod in afara de de 1 nu a fost vizitat se calculeaza drumul minim dintre nodul urmator in matrice, adica 2, si celelalte noduri doar prin noduri vizitate. Astfel intre nodul 2 si 4 avem lungimea de 2+10 adica drumul 2,1,4. Se observa ca inspre nodul 5 nu exista momentan drum intrucat nu s-a vizitat nodul 3 care este intermediar intre ele.
0 2 **5** 10 ∞ 2 0 3 12 ∞ **5** 3 0 1 8 10 12 1 0 ∞ ∞ ∞ 8 ∞ 0	 Acum ca nodul 2 a fost vizitat se poate calcula un drum de la nodul 1 la nodul 3 prin nodul 2 adica 2+3=5. De la 1 la 5 inca nu exista drum. Nodul 3 nefiind vizitat.
0 2 5 **6** ∞ 2 0 3 12 ∞ 5 3 0 1 8 **6** 12 1 0 ∞ ∞ ∞ 8 ∞ 0	 Nodul 3 a fost vizitat acum se poate calcula un drum de cost mai mic prin nodurile 2 si 3 adica 2+3+1=6 care este mai mic decat drumul direct de valoare 10. Asfel in matrice se plaseaza valoare 6 ca drum intre nodurile 1 si 4

0 2 5 6 **13** 2 0 3 12 ∞ 5 3 0 1 8 6 12 1 0 ∞ **13** ∞ 8 ∞ 0	 Drumul de la nodul 1 la nodul 5 se poate calcula prin nodurile 2 si 3 si are valoarea 2+3+8=13
0 2 5 6 13 2 0 3 **4** ∞ 5 3 0 1 8 6 **4** 1 0 ∞ 13 ∞ 8 ∞ 0	 Din nodul 2 la nodul 4 se gaseste un nou drum de valoarea mai mica prin nodul intermediar 3 care este 3+1=4 < 12.
0 2 5 6 13 2 0 3 4 **11** 5 3 0 1 8 6 4 1 0 ∞ 13 **11** 8 ∞ 0	 Intre nodul 2 si 5 se poate calcula acum un drum prin nodul intermediar 3 cu valoarea 3+8=11.
0 2 5 6 13 2 0 3 4 11 5 3 0 1 8 6 4 1 0 **9** 13 11 8 **9** 0	 Intre 4 si 5 exista drum prin nodul intermediar 3 de valoarea 1+8=9.
0 2 5 6 13 2 **0** 3 4 11 5 3 **0** 1 8 6 4 1 **0** 9 13 11 8 9 **0**	Nu mai exista drumuri mai mici, deci matricea drumurilor optime este completa. Se observa ca este simetrica fata de diagonala principala.

Afisarea drumului de la un nod la altul se face cu ajutorui functiei descompun_drum ce se bazeaza pe metoda divide_et_impera .

De exemplu pentru un drum de la nodul 1 la nodul 4 avem initial cale directa 1-4 cu costul 10 dar tot odata mai exista o cale prin nodurile intermediare cu costul mai mic de valoare 6.

Pentru evidențierea costurilor tuturor arcelor unui graf cu **n** noduri se poate defini o matrice **a**, cu **n** linii ***n** coloane.există două forme ale acestei matrici:

Forma a): Fiecare element **a[i,j]** poate fi:

-**c**, dacă există un arc de cost **c>0** între nodurile **i** şi **j**;

-0, dacă **i=j;**

-+∞, dacă nu există arc între nodurile **i** şi **j**.

Forma b): Este absolut similară, cu singura deosebire că în loc de +∞ avem -∞.

Forma a)se foloseşte pentru determinarea drumurilor de cost minim între două noduri, iar forma b) este utilizată în aflarea drumurilor de cost maxim.

Dacă dorim să citim matricea costurilor, evident că nu putem introduce de la tastatură “+∞”! În loc de “+∞” vom da un numar de la tastatură foarte mare.

Problema determinării drumului minim/ maxim între două noduri face obiectul algoritmului următor.

Observatii!

- Drumurile minime între toate nodurile se regăsesc la finele algoritmului tot în matricea costurilor, care a suferit **n** trasformări, pentru **k=1,2,…,n.**
- Unele elemente pot fi +∞, iar pentru simularea lui +∞ am spus că se introduce un număr întreg foarte mare. Prin adunări repetate a două numere întregi foarte mari putem ajunge la un rezultat care depăşeşte cea mai mare valoare posibilă de tipul **integer**. De aceea, recomandăm ca elementele matricei costurilor să fie de tipul **longint.**
- În cazul în care problema cerea pentru fiecare pereche de noduri **(i, j)** costul drumului maxim, modificările necesare ar fi minore:

-se foloseşte *forma b)* a matricei costurilor;
-condiția testată în linia **if** devine "**a[i, k]+a[k, j]<a[i, j]**"

Cod sursa C/C++

```
#include<stdio.h>
#include<conio.h>
#define  pinf 10000 //pentru plus infinit, o valoare mare care nu exista


int a[20][20],n,m;


void citire_cost()
{   FILE *f;
 int i,j,x,y,c;
 f= fopen("roy_in.txt","rt");
 if(f)
   printf("deschiderea a reusit");
 else
   printf("eroare la deschidere!");

 printf("\n");
 fscanf(f,"%d",&n);
 fscanf(f,"%d\n",&m);
```

```
 //initializare matrice
 for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
      if(i==j)
          a[i][j]=0;
       else
          a[i][j]=pinf;
  for(i=1;i<=m;i++)
     {fscanf(f,"%d %d %d\n",&x,&y,&c);
      a[x][y]=a[y][x]=c;}

  fclose(f);
}

void afisare_mat()
{for(int i=1;i<=n;i++)
   {for(int j=1;j<=n;j++)
          if(a[i][j]==pinf)
             printf("    - ");
          else
            printf("%4d ",a[i][j]);
    printf("\n"); }
}

void RoyFloyd()  // roy floyd
{for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
       for(int j=1;j<=n;j++)
            if(a[i][j]>a[i][k]+a[k][j])
            {   a[i][j]=a[i][k]+a[k][j];

}
}

void descompun_drum(int i,int j) //realizeaza descompunerea portiunii de la
i la j prin k
{int ok=0,k=1;

 while(k<=n&&!ok)
          {
```

```
                if(i!=k&&j!=k)
    if(a[i][j]==a[i][k]+a[k][j])
                  {
           descompun_drum(i,k);
                   descompun_drum(k,j);

                   ok=1;}   //g marcheaza daca se poate realiza
descompunerea
   k++;
     }
  if(!ok)
  {
  printf(" %d ",j); }//cand “drumul” nu mai poate fi descompus afisez
extremitatea finala

}

void scriu_drum(int nod_initial,int nod_final) // functia primeste ca parametri
cele doua noduri pt care se determina optimul
{if(a[nod_initial][nod_final]<pinf)
     {
          printf("lantul de la %d la %d are lungimea %d
",nod_initial,nod_final,a[nod_initial][nod_final]);
      printf("\n un drum optim este: %d ",nod_initial);
      descompun_drum(nod_initial,nod_final); // apeleaza functia care
afiseaza efectiv lantul
     }
 else
     printf("nu exista lant de la %d la %d ",nod_initial,nod_final);

}


void main()
{

int x,y;
citire_cost();
printf("\nmatricea ponderilor \n");
afisare_mat();
```

```
RoyFloyd();
printf("\n matricea drumurilor optime \n");
afisare_mat();
printf("\n Se determina drumul minim intre varfurile x si y \n");
printf("x=");
scanf("%d",&x);
printf("y=");
scanf("%d",&y);
scriu_drum(x,y);
_getch();
}
```

Fisierul Roy_in.txt este de forma

```
5 5 // numarul de noduri si numarul de muchi
1 2 2 // intre nodul 1 si 2 exista muchia de cost 2
2 3 3 // intre nodul 2 si 3 exista muchia de cost 3
3 4 1
3 5 8
4 1 10
```

Probleme propuse!

1. Micul Floyd locuieste intr-un oras mare, in care exista 5 intersectii. Fiecare pereche de intersectii este conectata printr-un drum bidirectional avand o lungime pozitiva data. Micul Floyd este un baiat curios si i-ar placea sa stie care este distanta minima pe care cineva ar trebui sa o parcurga de-a lungul drumurilor existente daca ar vrea sa mearga din intersectia 1 in intersectia 5. Deoarece ii plac foarte mult intersectiile ar vrea de asemenea sa stie, in cazul in care exista mai multe drumuri intre 1 si 5 de aceeasi lungime minima, care este numarul maxim de strazi pe care ar putea sa mearga cineva pentru a obtine aceasta distanta minima.
   El stie ca intre intersecti exista lungimi dupa cum urmeaza :

Intersectie		Lungime
1	2	6
1	4	8
1	5	10
1	3	5
3	4	1
4	5	2

2. Gigel este angajatul unui mare aeroport din România. El se ocupă de relațiile cu publicul. Zilnic sute de clienți îl sună solicitând informații referitoare la cele mai mici costuri între oricare două aeroporturi. Salariul lui Gigel depinde de numărul de clienți corect informați, de aceea Gigel, cum ajunge la serviciu începe să calculeze rute. De curând aeroportul l-a dotat cu un calculator. Ajută-l pe Gigel să obțină un salariu cât mai mare, scriindu-i un algoritm de găsire a drumurilor de cost minim între aeroportul 1 si aeroportul 5.
   El stie ca intre aeroporturi exista distante dupa cum urmeaza :

Aeroport		Lungime
1	2	9

1	4	12
1	5	10
1	3	4
5	2	2
4	5	1

3. Alex lucreaza la o mare firma de IT unde este game-programmer. El are de realizat interfata pentru un joc de strategie unde diferite unitati trebuie sa ajunga in coordinate prestabilite de un jucator. Ajuta-l pe Alex sa realizeze acest lucru printr-un algoritm ce calculeaza drumul minim intre pozitia initiala a unei unitati si pozitia tinta, considerand ca intre aceste 2 poziti exista obstacole sau nu.
   Se considera pozitia initiala ca fiind 1 iar pozitia finala fiind 6.
   Se stie ca intre pozitia initiala si cea finala exista si pozitii intermediare cu distante dupa cum urmeaza :

Pozitii :		Lungime
1	2	9
1	4	12
1	5	10
1	3	4
5	2	2
4	5	1
5	6	4
1	6	15
3	5	3
2	4	10
3	6	2