Arbore parțial de cost minim

1. **Algoritmul lui Kruskal – număr mic de muchii**
2. **Algoritmul lui Prim – număr mare de muchii**

Ambii algoritmi utilizează meotda Greedy

1. **Algoritmul lui Kruskal – număr mic de muchii**

Datele grafului ponderat sunt memorate în vector de muchii.

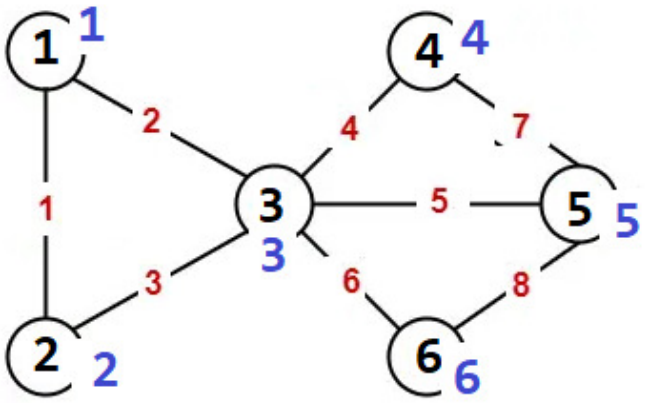
Se consideră inițial că fiecare nod face parte dintr-un APM (t[i] este i)

Muchiile sunt sortate crescator după cost.

Se alege prima muchie. Costul ei este adăugat la costul total al APM.

Se parcurge vectorul de muchii sortat pînă în momentul în care am ales n-1 muchii.

O muchie parcursă este aleasă să participe la crearea APM dacă are extremitățile plasate în arbori separați (distincți) . Pentru fiecare muchie plasată în APM se actualizează etichetele vectorului de tați.



```
ifstream f("kruskal.in");
ofstream g("kruskal.out");
struct muchie
{
    int i,j,c;
}x[5000];

int n , m , t[105],v[5000];

int main()
{
    f>> n >> m;
    for(int i = 0 ; i < m ; ++i)
      f>>x[i].i>>x[i].j>>x[i].c;

    for(int i = 0 ; i < m - 1; i ++)
      for(int j = i + 1 ; j < m ; ++j)
         if(x[i].c> x[j].c)
            swap(x[i],x[j]);// sortare muchii

    for(int i =1 ; i <= n ; ++i) t[i] = i;
    int S = x[0].c; v[0]=1;
    t[x[0].i]=t[x[0].j];// am ales prima muchie
    for(int i = 1 ; i < m ;i++)
    {
        int k1=x[i].i,k2=x[i].j;// muchia i are extremitatile k1 si k2
        if(t[k1] != t[k2]) // daca au tati diferiti, sunt in APM partiali diferiti
        {
          v[i] = 1; // marcare muchia i ca a fost plasata in APM
          S += x[i].c; // actualizare costul APM cu noua muchie
          int a= t[k1];
          int b= t[k2];
          for(int j =1 ; j <= n ; ++j) // toate nodurile ce se afla in APM b sunt trecute in APM a
                if(t[j] == b)
                    t[j] = a;
        }
    }
    g << S << ' ';
    for(int i = 0 ; i < m ; ++i)
       if(v[i] == 1)
          g<<x[i].i<<" "<<x[i].j<<" ";
    return 0;
}
```

Fig. 1 cod sursă algoritmul lui Kruskal

**Obs. Îmbunatățire cod astfel încât să se incheie după alegerea a n-1 muchii.*

1. **Algoritmul lui Prim – număr mare de muchii**

Datele grafului sunt stocate în matricea ponderilor (0, infinit sau cost).

Putem plasa pentru început nodul 1 în APM (în final în APM oricum vom avea toate nodurile. Deci este irelevant cu ce nod începem construcția APM).

Căutăm cel mai apropiat nod p de nodul 1 din APM. Acesta este plasat în APM

Cautăm cel mai apropiat nod p de unul din cele 2 noduri existente în APM. Este plasat în APM

Căutăm cel mai apropiat nod p de unul din cele 3 noduri din APM etc.

Pentru fiecare nod p anterior determinat trebuie actualizat vectorul distanțelor nodurilor i ce nu au fost încă plasate în APM

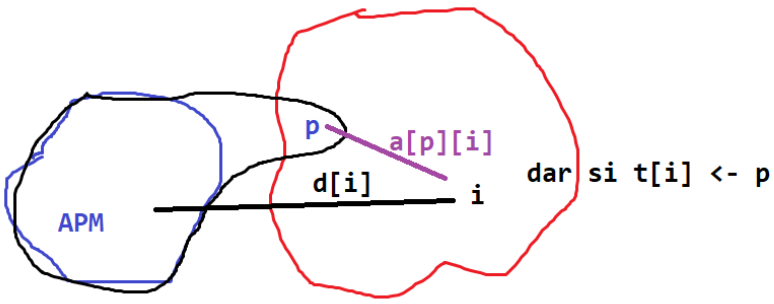
Var. I

```
using namespace std;
ifstream f("prim.in");
ofstream g("prim.out");
int n , a[105][105], v[105], t[105];
int main()
{
    int i , j , c , m,x,y,S=0;
    f>> n >> m;
    for(i =1 ; i <= n ; ++i)
        for(j = 1 ; j <= n ; ++j)
           if(i!=j) a[i][j] = INF;
    while( m --)
    {
        f >> x >> y >> c;
        a[x][y] = a[y][x] = c;
    }
    for(i =2 ; i <= n ; i ++ )
        t[i] = 1;
    v[1] = 1;     t[1] = 0;
    for(int k = 2 ; k <= n ; ++k)
    {
        int p = -1,q, dist=INF;
        for(i = 1 ; i <= n ; ++i)// i este in APM
         if(v[i]==1)
           for(j=1; j<=n; j++) // j nu este in APM
              if(v[j]==0 && a[i][j]<dist )
                 { dist=a[i][j]; p=j;q=i;}

        if(p !=-1)
        {
            v[p] = 1; S+=dist;
            t[p] = q;
        }
    }
    g << S << endl;
    for(i = 1 ; i <= n ; ++i)
        g << t[i] << " ";
    return 0;
}
```

Fig 2. Cod sursa algoritmul lui Prim (Var. I)

Var. II



d[i] reprezintă distanța minima de la nodul i la un alt nod oarecare aflat deja in APM.
Prin plasarea lui p in APM incercam sa actualizam aceasta distanta d[i].

```cpp
#include <iostream>
#include <fstream>
#define INF 1000000000
using namespace std;
ifstream f("prim.in");
ofstream g("prim.out");
int n , a[105][105], v[105], d[105], t[105];
int main()
{
    int i , j , c , m,x,y;
    f>> n >> m;
    for(i =1 ; i <= n ; ++i)
        for(j = 1 ; j <= n ; ++j)
           if(i!=j) a[i][j] = INF;// matricea costurilor. are valorile 0,INF sau costul c
    while( m --)
    {
        f >> x >> y >> c;
        a[x][y] = a[y][x] = c;
    }
    // plecare din nodul 1
    for(i =2 ; i <= n ; i ++ )
    {
        d[i] = a[1][i]; // dăii retine cea mai scurta distanța intre nodul i si nodul 1 (plasat in APM
        t[i] = 1; // se considera ca ascendentul oricarui nod este 1
    }
    v[1] = 1; // marcare, este plasat in APM
    t[1] = 0; // 1 nu are ascendent
    d[1] = 0; // distanta de la 1 la 1 este 0
    d[0] = INF;
    for(int k = 1 ; k < n ; ++k)
    {
        int p = 0; // cautam un nod i nevizitat si aflat la distanta cea mai mica de un nod din APM
        for(i = 1 ; i <= n ; ++i)
            if(v[i] == 0 && d[i] < d[p])
                p = i;// il retinem in p
        if(p > 0)// daca exista un nod p ce indeplineste cerinta
        {
            v[p] = 1; // este plasat in APM
            for(i = 1; i <= n ; ++i) // actualizam toate nodurile neplasate in APM
                if(v[i] == 0 && d[i] > a[p][i])//distanta de la i la APM este mai mare decit distanta
                  { d[i] = a[p][i];// de la nodul anterior ales p la el, actualizam distanta
                   t[i] = p; }// tatal lui i este p
        }
    }
    int S = 0;
    for(i = 1 ; i <= n ; ++i)
        S += d[i];  // costul total al APM
    g << S << endl;
    for(i = 1 ; i <= n ; ++i)
        g << t[i] << " ";
    return 0;
}
```

C++

Fig. 3 cod sursa algoritmul lui Prim (Var . II)