Varianta ce prevede algoritmul Kruskal VS Prim

```
#include <iostream>
using namespace std;

typedef struct
{
   int x,y,cost;  //Vectorul este format din x-nodul1, y-nodul2 si
costul de pe muchia lor
} muchie;

int viz[30], n, m, c;
muchie v[30], aux;

//cititm datele problemei sub forma: nod1 -> nod2 – cost

void citire()
{
    int i;
    cin>>n;
    cin>>m;
    for(i=1; i<=m; i++)
        cin>>v[i].x>>v[i].y>>v[i].cost;

}

// Partea „Gready” din algoritm

void sortare()
{
    int k=0, i;
    while(k==0)
    {
        k=1;
        for(i=1;i<=m-1;i++)
        {
            if(v[i].cost>v[i+1].cost)
            {
                aux=v[i+1];
                v[i+1]=v[i];
                v[i]=aux;
                k=0;
            }
        }
    }
}
```

```
void kruskal()
{
    int i,j,k;
    i=1;
    for(k=1; k<=n-1; k++)
    {           // verificam sa nu vizitam noduri deja vizitate
        while(viz[v[i].x]==viz[v[i].y]&&viz[v[i].x]!=0)
            i++;
        //calculam costul total al arborelui de cost minim
        c+=v[i].cost;
        //afisam cele 2 noduri
        cout<<v[i].x<<" "<<v[i].y<<"\n";
        //daca cele 2 noduri nu sunt vizitate atunci le vizitam si le
salvam o valoare din cele 2 //in „viz”
        if(viz[v[i].x]+viz[v[i].y]==0)
            viz[v[i].x]=viz[v[i].y]=v[i].x;
        //daca 1 din cele 2 valori este 0 o completam cu valoarea
celeilalte
        else
        //daca ambele valori fac parte deja dintr-un arbore secundar,
modificam toate valorile
                //lui cu valorile celuilalt; practic legam cei 2
arbori
            if(viz[v[i].x]*viz[v[i].y]==0)
                viz[v[i].x]=viz[v[i].y]=viz[v[i].x]+viz[v[i].y];
            else
            {
                for(j=1; j<=n; j++)
                    if(viz[j]==viz[v[i].x]&&j!=v[i].x)
                        viz[j]=viz[v[i].y];

                viz[v[i].x]=viz[v[i].y];
            }
        i++;
    }
    cout<<c;
}

int main()
{
    citire();
    sortare();
    kruskal();
}
```

# Algoritmul lui Prim

```
#include <cstring>
#include <iostream>
using namespace std;

#define INF 9999999

// numarul de noduri
#define V 5


int G[V][V] = {
  {0, 9, 75, 0, 0},
  {9, 0, 95, 19, 42},
  {75, 95, 0, 51, 66},
  {0, 19, 51, 0, 31},
  {0, 42, 66, 31, 0}};

int main() {
  int no_edge;  // number of edge

  // cream un array pentru a urmari nodurile selectate
  // *cele selectate vor avea valoarea true
  int selected[V];

  // le dam valoare la toate ca false
  memset(selected, false, sizeof(selected));

  // setam numarul liniei/muchiei 0
  no_edge = 0;

  // incepem parcurgerea grafului
  selected[0] = true;

  int x;
  int y;

  cout << "Edge"
     << " : "
     << "Weight";
  cout << endl;
  while (no_edge < V - 1) {
    //pentru fiecare nod in S, cautam nodurile adiacente,
    //calculam distanta dintre noduri, fata de nodul de
plecare/initial.
```

```
    int min = INF;
    x = 0;
    y = 0;

    for (int i = 0; i < V; i++) {
      if (selected[i]) {
        for (int j = 0; j < V; j++) {
          if (!selected[j] && G[i][j]) {
            if (min > G[i][j]) {
              min = G[i][j];
              x = i;
              y = j;
            }
          }
        }
      }
    }
    cout << x << " - " << y << " :  " << G[x][y];
    cout << endl;
    selected[y] = true;
    no_edge++;
  }

  return 0;
}
```

# TEORIE

## Algoritmul Kruskal

Algoritmul funcționează în felul următor:

- creează o pădure *F* (o mulțime de arbori), unde fiecare vârf din graf este un arbore separat
- creează o mulțime *S* care conține toate muchiile din graf
- atât timp cât *S* este nevidă
    - elimină o muchie de cost minim din *S*
    - dacă acea muchie conectează doi arbori distincți, atunci adaugă muchia în pădure, combinând cei doi arbori într-unul singur
    - altfel, ignoră muchia

## Algoritmul Kruskal

Algoritmul incrementează mărimea unui arbore, pornind de la un nod, până când sunt incluse toate nodurile.

- Intrare: Un graf conex ponderat cu nodurile V și muchiile E.
- Initializare: $V_{nou}$ = {x}, unde x este un nod arbitrar (punct de plecare) din V, $E_{nou}$= {}
- repetă până când $V_{nou}$=V:
    - Alege muchia (u,v) din E de cost minim astfel încât u este în $V_{nou}$ și v nu e (dacă există mai multe astfel de muchii, se alege arbitrar)
    - Se adaugă v la $V_{nou}$, (u,v) la $E_{nou}$
- Ieșire: $V_{nou}$ și $E_{nou}$ descriu arborele parțial de cost minim

## Bibliografie Teoretica


Algoritmul lui Kruskal - Wikipedia

Algoritmul lui Prim - Wikipedia