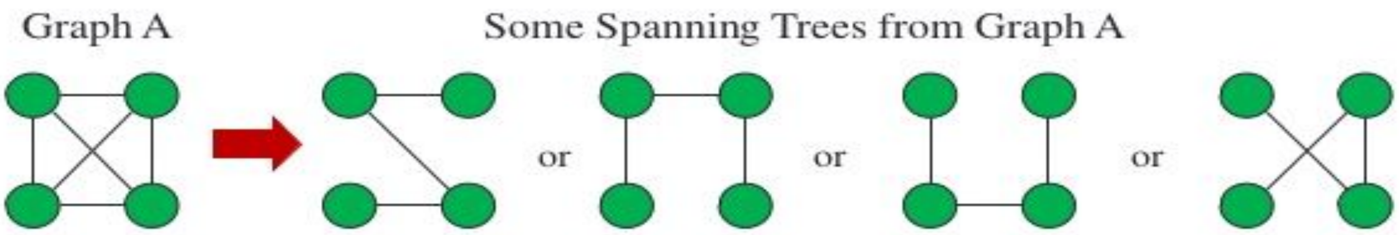
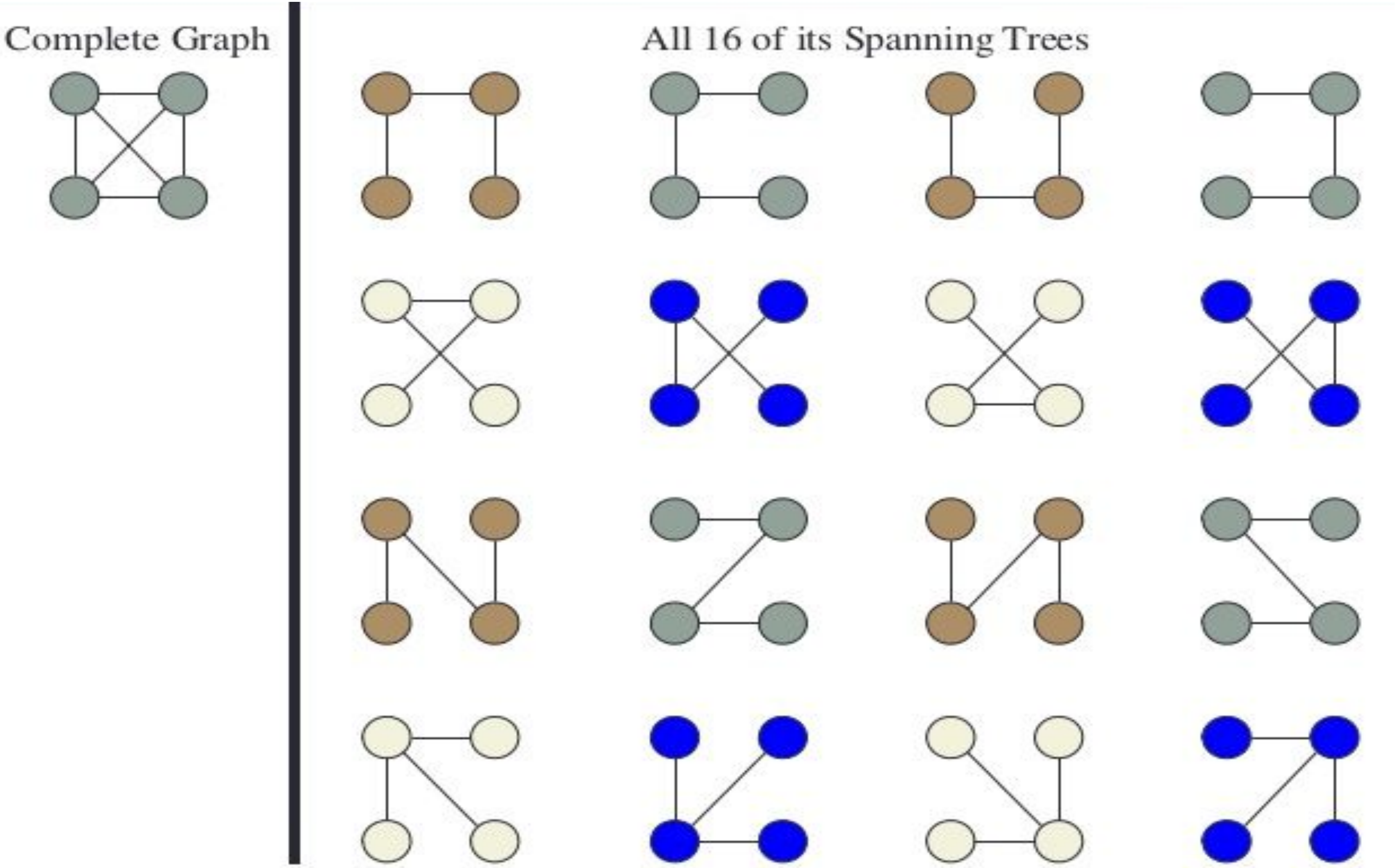
Spanning Trees

A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.

A graph may have many spanning trees.



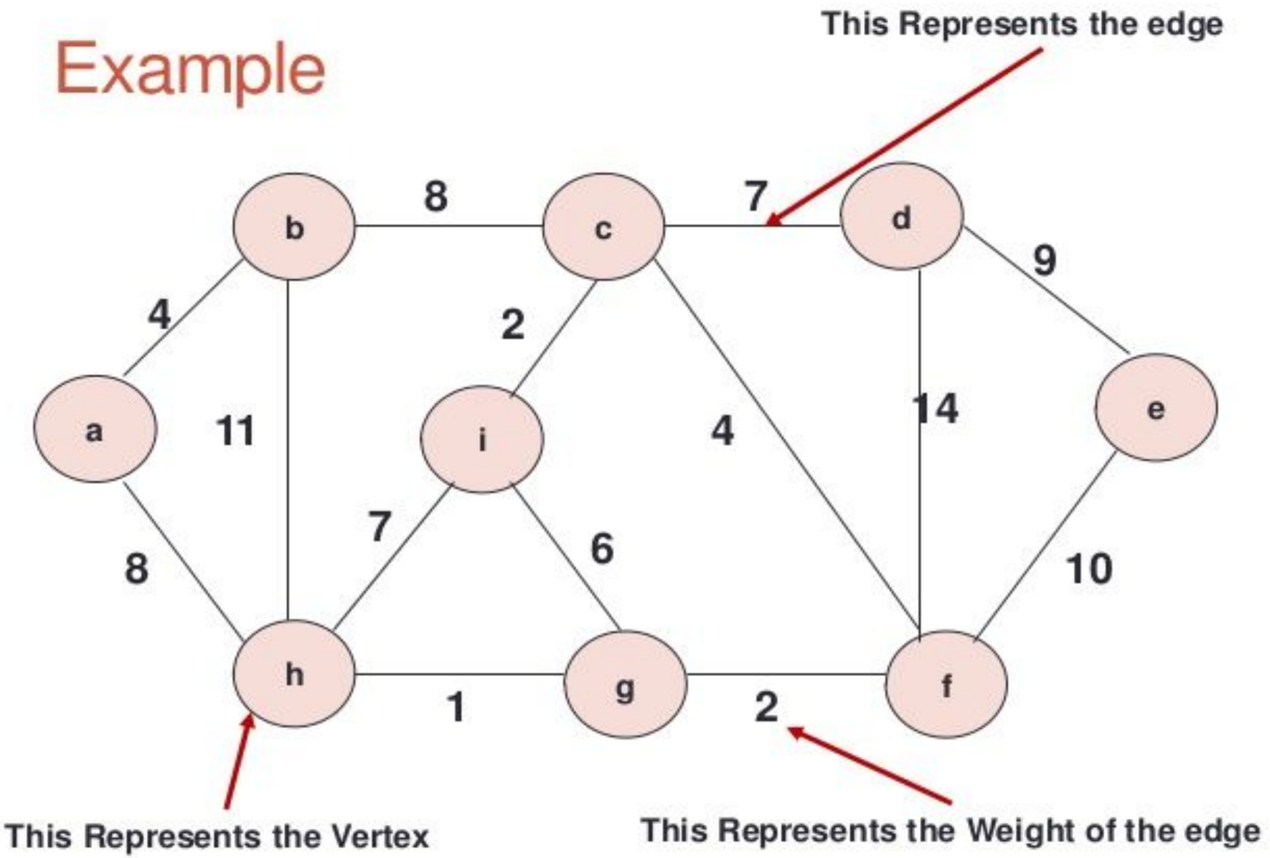

Complete Graph
All 16 of its Spanning Trees

Minimum Spanning Tree

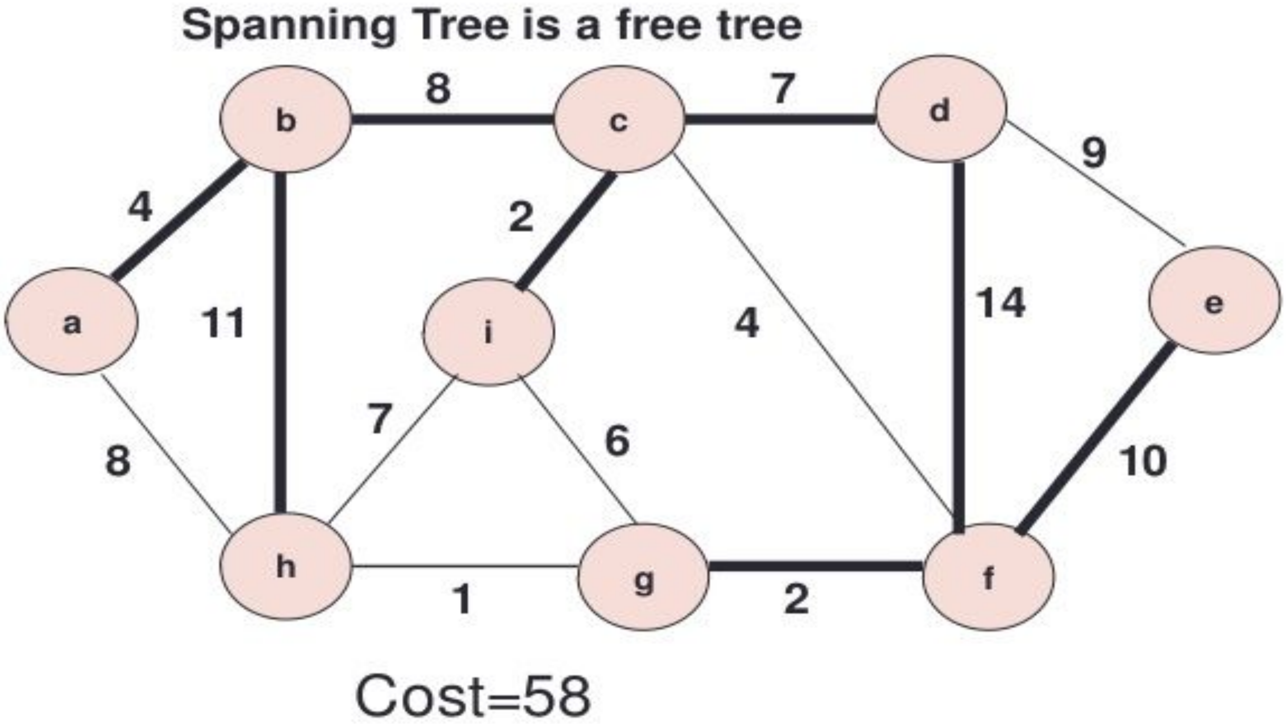
A minimum spanning tree or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

Applications : Computer Networks

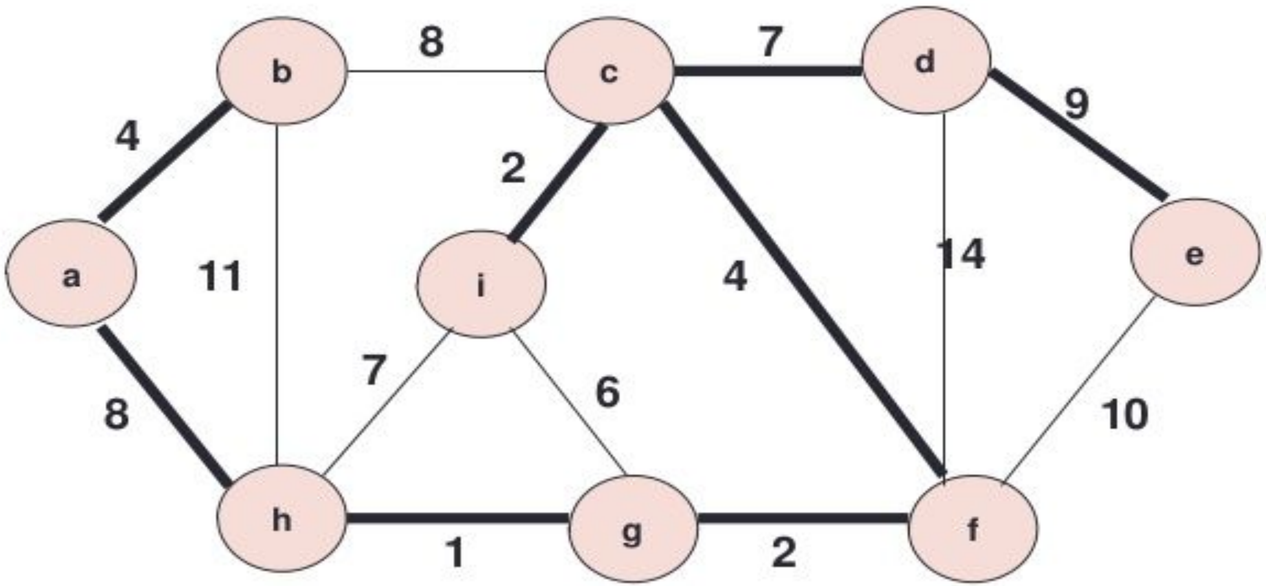
Example


This Represents the edge
b
c
d
a
i
e
h
g
f
8
7
9
4
2
11
4
14
7
6
8
10
1
2
This Represents the Vertex
This Represents the Weight of the edge

Example


Spanning Tree is a free tree
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2
Cost=58

Example



Cost=37

Algorithms for Obtaining the Minimum Spanning Tree

- Kruskal's Algorithm

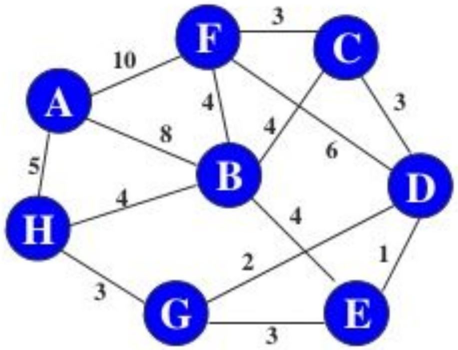
- Prim's Algorithm

Kruskal’s Algorithm

Step 1: Sort the edges according to their weight.

Step 2 : Select the first |V|-1 edges such that they do not make circuit/loop with previously selected edges (If the edges make a loop or not is checked using union and find functions of dsu).

Walk-Through



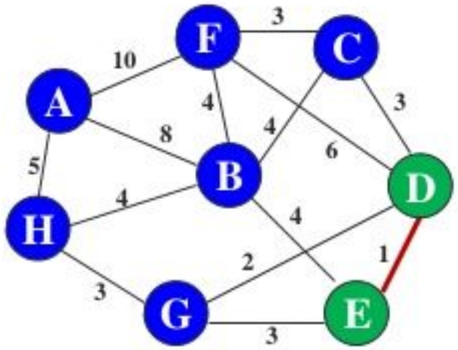
Sort the edges by increasing edge weight

edge	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through

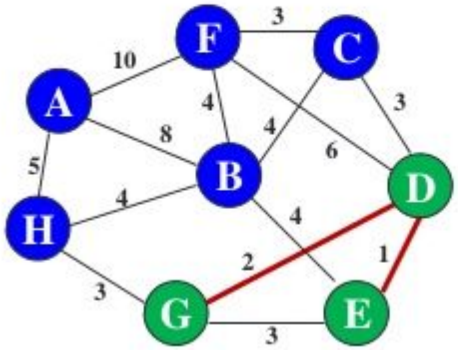
Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through



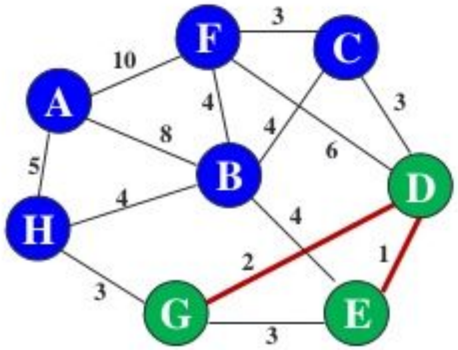
Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through

Select first |V|–1 edges which do not generate a cycle



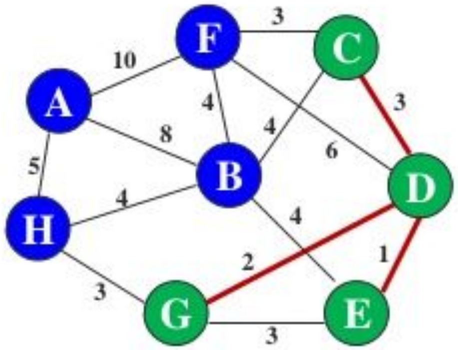
edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (E,G) would create a cycle

Walk-Through

Select first |V|–1 edges which do not generate a cycle

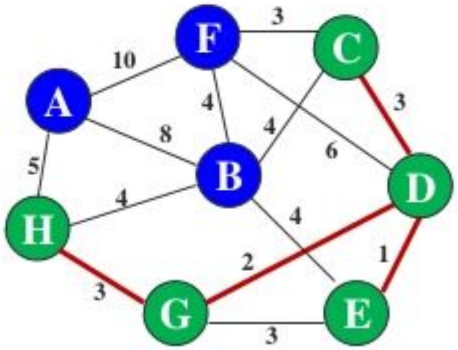


edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through

Select first |V|–1 edges which do not generate a cycle

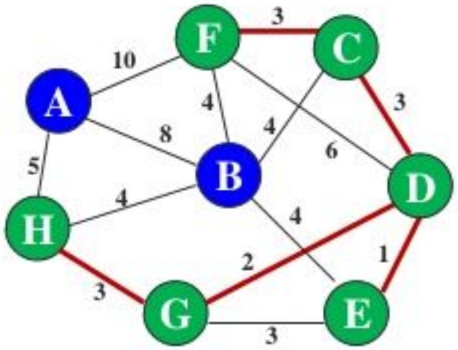


edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through

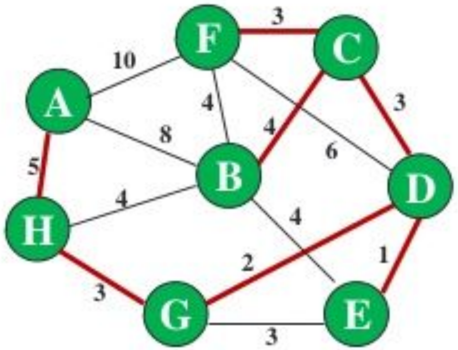
Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Walk-Through

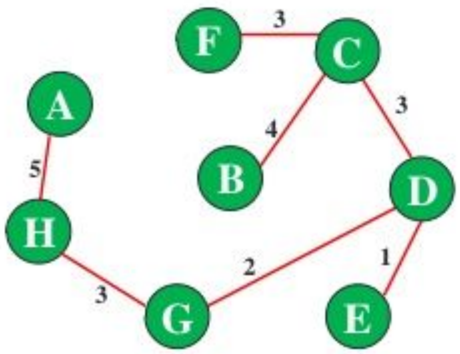


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	χ
(A,B)	8	χ
(A,F)	10	χ

Kruskal's Algorithm



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	χ
(A,B)	8	χ
(A,F)	10	χ

Done!

Total Cost $= \Sigma\, d_v = 21$

Code:

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>

using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
    for(int i = 0;i < MAX;++i)
        id[i] = i;
}

int root(int x)
{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}

void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}
```

```
long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {
        // Selecting edges one by one in increasing order from the beginning
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        // Check if the selected edge is creating a cycle or not
        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    initialize();
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }
    // Sort the edges in the ascending order
    sort(p, p + edges);
    minimumCost = kruskal(p);
    cout << minimumCost << endl;
    return 0;
}
```

Time Complexity:

Complexity:

- Time complexity: $O(m\log(n))$ where m is the number of edges and n is the number of nodes.

Ques:

There are N homes in a village, we have to facilitate water supply in each of them. We can either build a well in a home or connect it with pipe to some different home already having water supply. More formally, we can either build a new well in the home or connect it with a pipeline to some different home which either has it’s own well or further gets water supply from a different home and so on. There is some cost associated with both building a new well and laying down a new pipeline. We have to supply water in all homes and minimise the total cost.

Floyd Warshall Algorithm

Floyd Warshall Algorithm

- Dynamic programming approach.
- All pair shortest path algorithm.
- Works for graph with negative weight.

Application

- Transitive closure of graph.

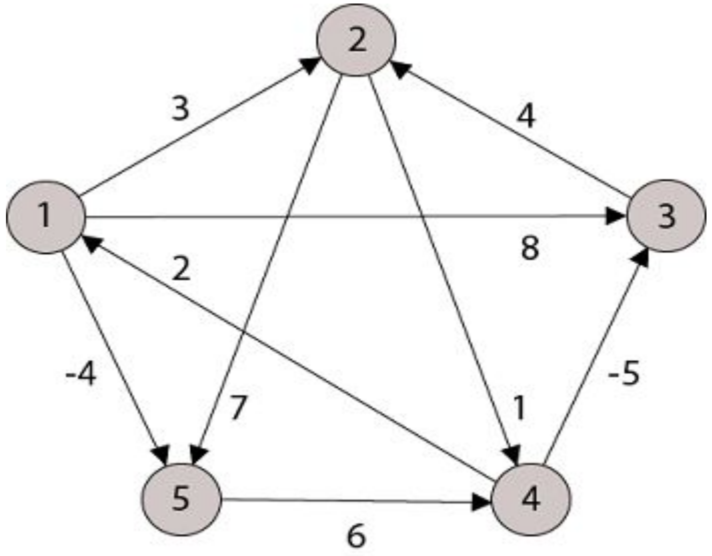
Floyd Warshall Algorithm

Pseudo Code

- n ← rows [W].
- D^0 ← W
- for k ← 1 to n
  - do for i ← 1 to n
    - do for j ← 1 to n
      - do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
- return $D^{(n)}$

Floyd Warshall Algorithm

Example :-

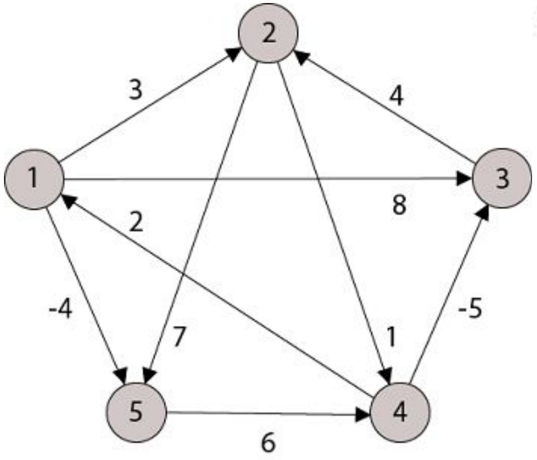


$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Floyd Warshall Algorithm

Step 0

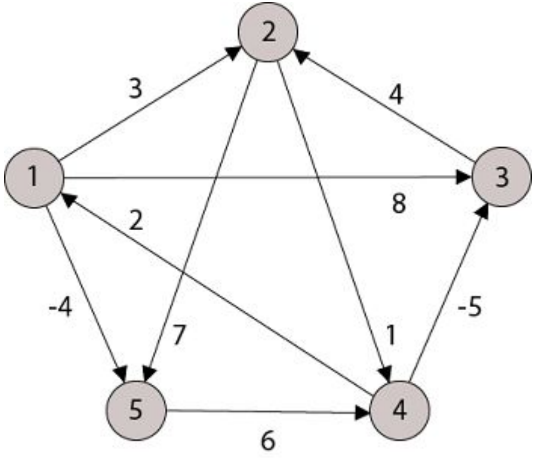


$$D^{(0)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$\pi^{(0)} = \begin{matrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 3 & \text{NIL} \\ 4 & \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{matrix}$$

Floyd Warshall Algorithm

Step 1

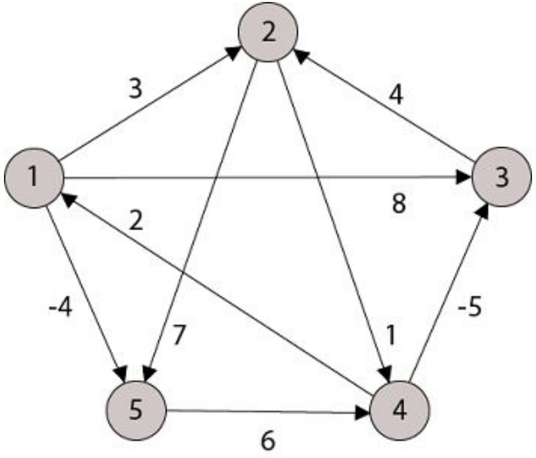


$$D_{ij}^{(1)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$\pi^{(1)} = \begin{matrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 3 & \text{NIL} \\ 4 & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{matrix}$$

Floyd Warshall Algorithm

Step 2

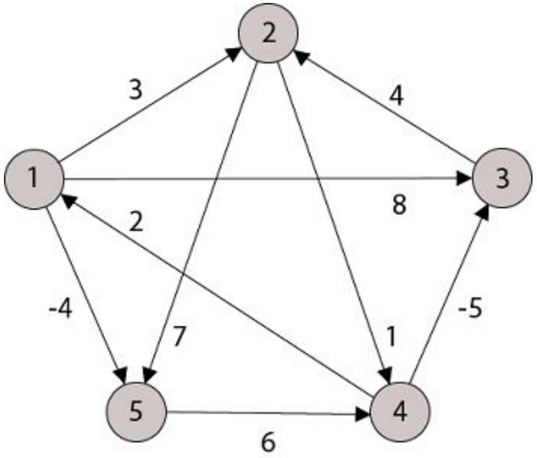


$$D_{ij}^{(2)} = \begin{matrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$\pi^{(2)} = \begin{matrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 3 & 2 \\ 4 & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{matrix}$$

Floyd Warshall Algorithm

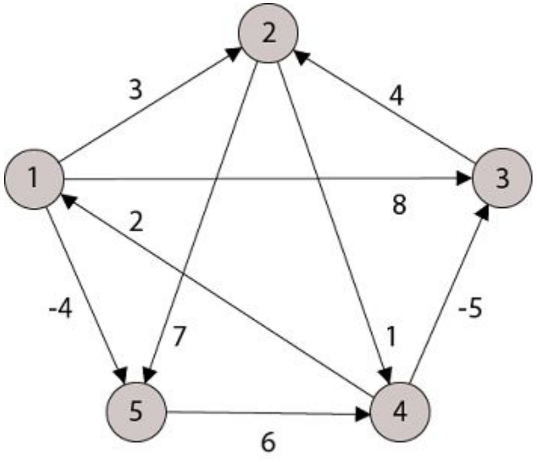
Step 3



$$D_{ij}^{(3)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix} \qquad \pi^{(3)} = \begin{matrix} NIL & 1 & 1 & 3 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 3 & 2 \\ 4 & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{matrix}$$

Floyd Warshall Algorithm

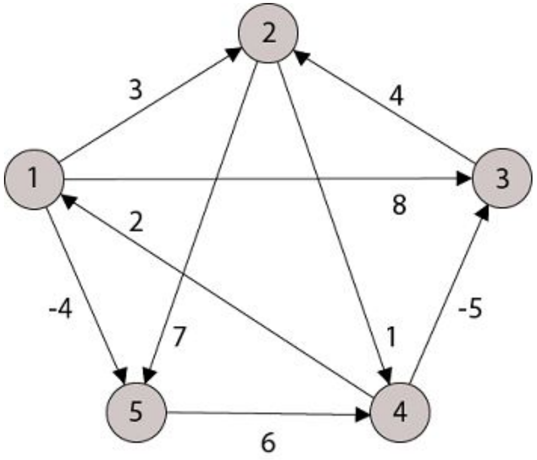
Step 4



$$D_{ij}^{(4)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix} \qquad \pi^{(4)} = \begin{matrix} \text{NIL} & 1 & 1 & 3 & 1 \\ 4 & \text{NIL} & 4 & 2 & 2 \\ 4 & 4 & \text{NIL} & 3 & 4 \\ 4 & 1 & 1 & \text{NIL} & 1 \\ 4 & 4 & 4 & 5 & \text{NIL} \end{matrix}$$

Floyd Warshall Algorithm

Step 5



$$D_{ij}^{(5)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix} \qquad \pi^{(5)} = \begin{matrix} \text{NIL} & 1 & 1 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 4 \\ 4 & 4 & \text{NIL} & 3 & 4 \\ 4 & 1 & 1 & \text{NIL} & 1 \\ 4 & 4 & 4 & 5 & \text{NIL} \end{matrix}$$

Floyd Warshall Algorithm

Complexity

- Time complexity - O(N^3).
- Space complexity - O(N^2)

Floyd Warshall Algorithm

Negative Cycle Detection

If there exists an *i* from {1,..., n}
such that $d_{ii}^{n} < 0$.
then, graph has negative cycle.