Design and Analysis of Algorithm

Lecture- 12:
Greedy Algorithm

Contents

Algorithm

```
Algorithm Kruskal(E, cost, n, t)
// E is the set of edges in G. G has n vertices. cost[u, v] is the
// cost of edge (u, v). t is the set of edges in the minimum-cost
// spanning tree. The final cost is returned.
{
    Construct a heap out of the edge costs using Heapify;
    for i := 1 to n do parent[i] := -1;
    // Each vertex is in a different set.
    i := 0; mincost := 0.0;
    while ((i < n - 1) and (heap not empty)) do
    {
        Delete a minimum cost edge (u, v) from the heap
        and reheapify using Adjust;
        j := Find(u); k := Find(v);
        if (j ≠ k) then
        {
            i := i + 1;
            t[i, 1] := u; t[i, 2] := v;
            mincost := mincost + cost[u, v];
            Union(j, k);
        }
    }
    if (i ≠ n - 1) then write ("No spanning tree");
    else return mincost;
}
```

Complexity

Removing an element from Heap $O(\log n)$

Removing n elements from heap $(O(n \log n))$

$$O((E \log V))$$

Heap

A heap is a binary tree that can be of two types:

Min heap:

- The element of each node is greater than or equal to the element at its parent. The minimum value element is at the root.

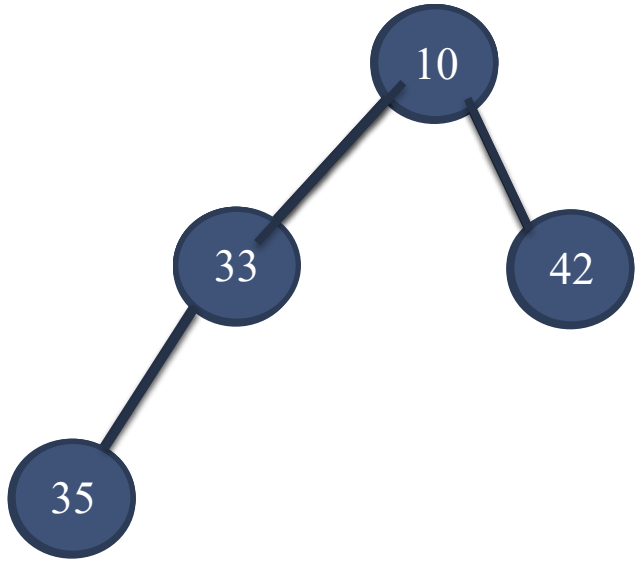
Max heap:

- The element of each node is less than the element at its parent. The maximum value element is at the root.

Heap Construction

Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.

35, 33, 42, 10



Kruskal's Algorithm

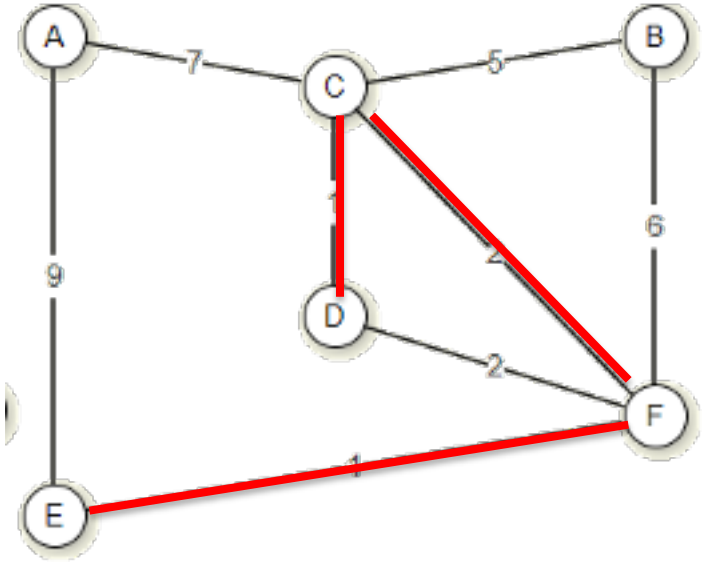
Kruskal's algorithm is a simple, greedy algorithm.

First partition the set of vertices into | V | disjoint sets, each consisting of one vertex.

Then process the edges in order of weight.

An edge is added to the MCST, and two disjoint sets combined, if the edge connects two vertices in different disjoint sets.

This process is repeated until only one disjoint set remains.

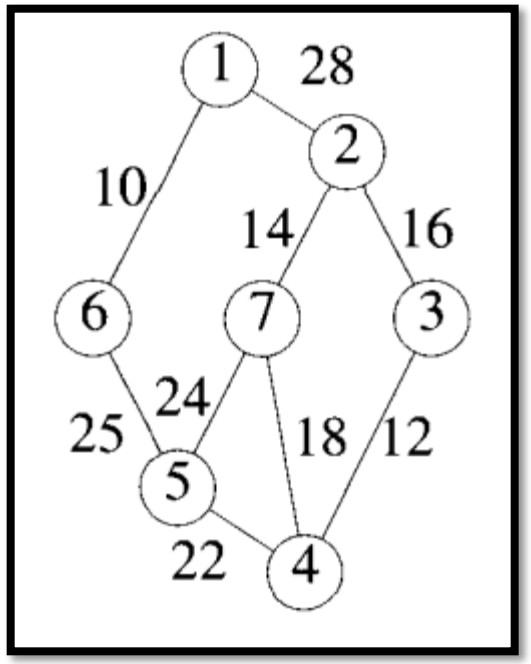


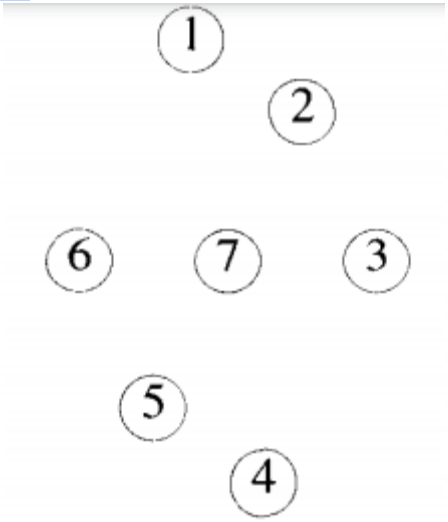
(A,C)	7
(A,E)	9
(C,B)	5
(C,D)	1
(C,F)	2
(D,F)	2
(E,F)	1
(F,B)	6

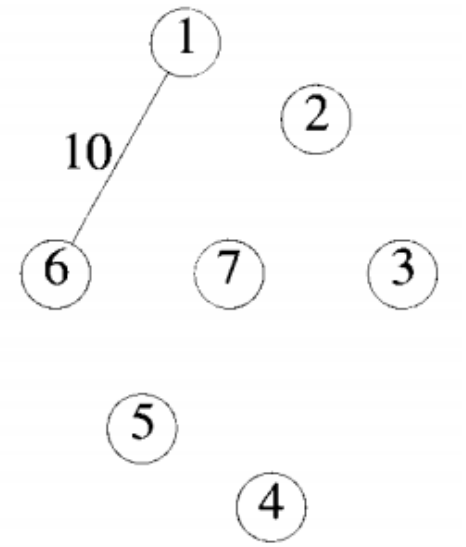
/	/	/	/	/	/
A	B	C	D	E	F
0	1	2	3	4	5

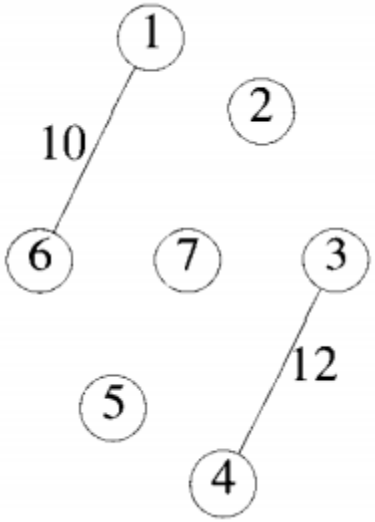
A	B	CD	E	F	
A	B	CD	EF		
A	B	CDEF			

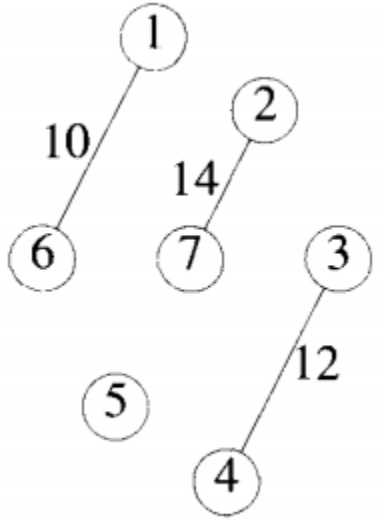
Example

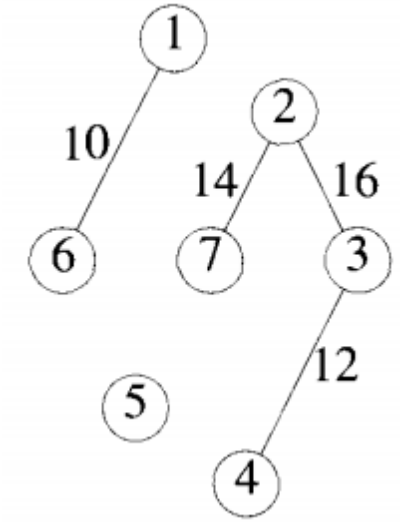

1
28
2
10
14
16
6
7
3
24
25
18
12
5
22
4

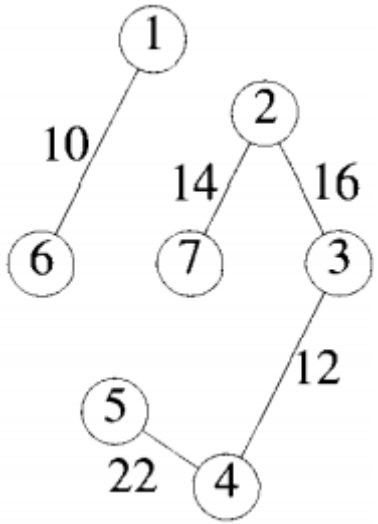

1
2
6
7
3
5
4

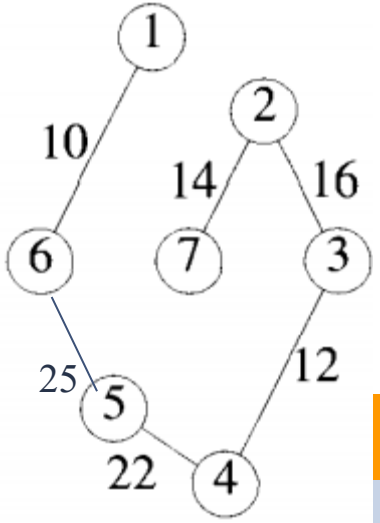

1
10
2
6
7
3
5
4


1
10
2
6
7
3
12
5
4

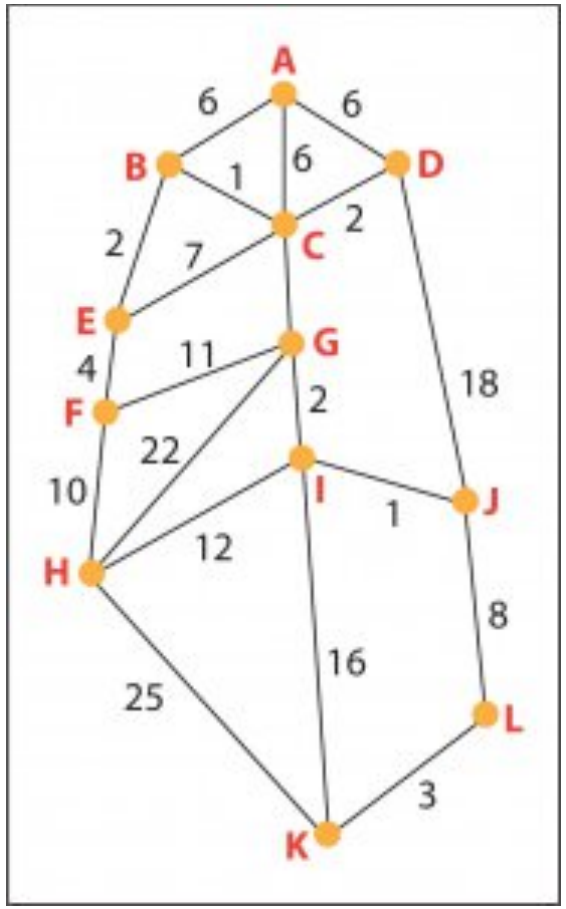

1
10
2
14
6
7
3
12
5
4

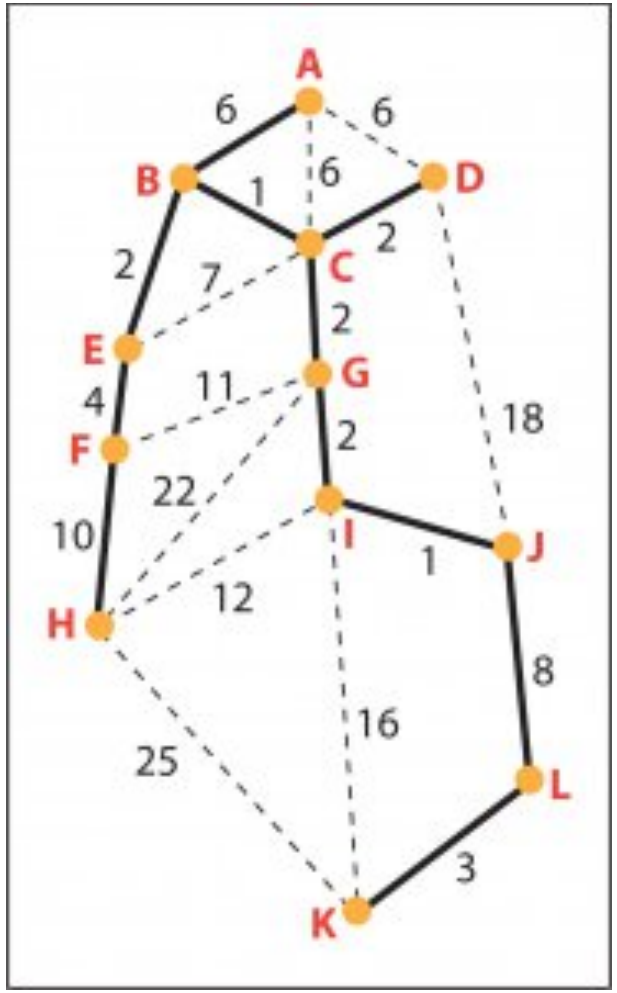

1
10
2
14
16
6
7
3
12
5
4


1
10
2
14
16
6
7
3
12
5
22
4


1
10
2
14
16
6
7
3
25
12
5
22
4

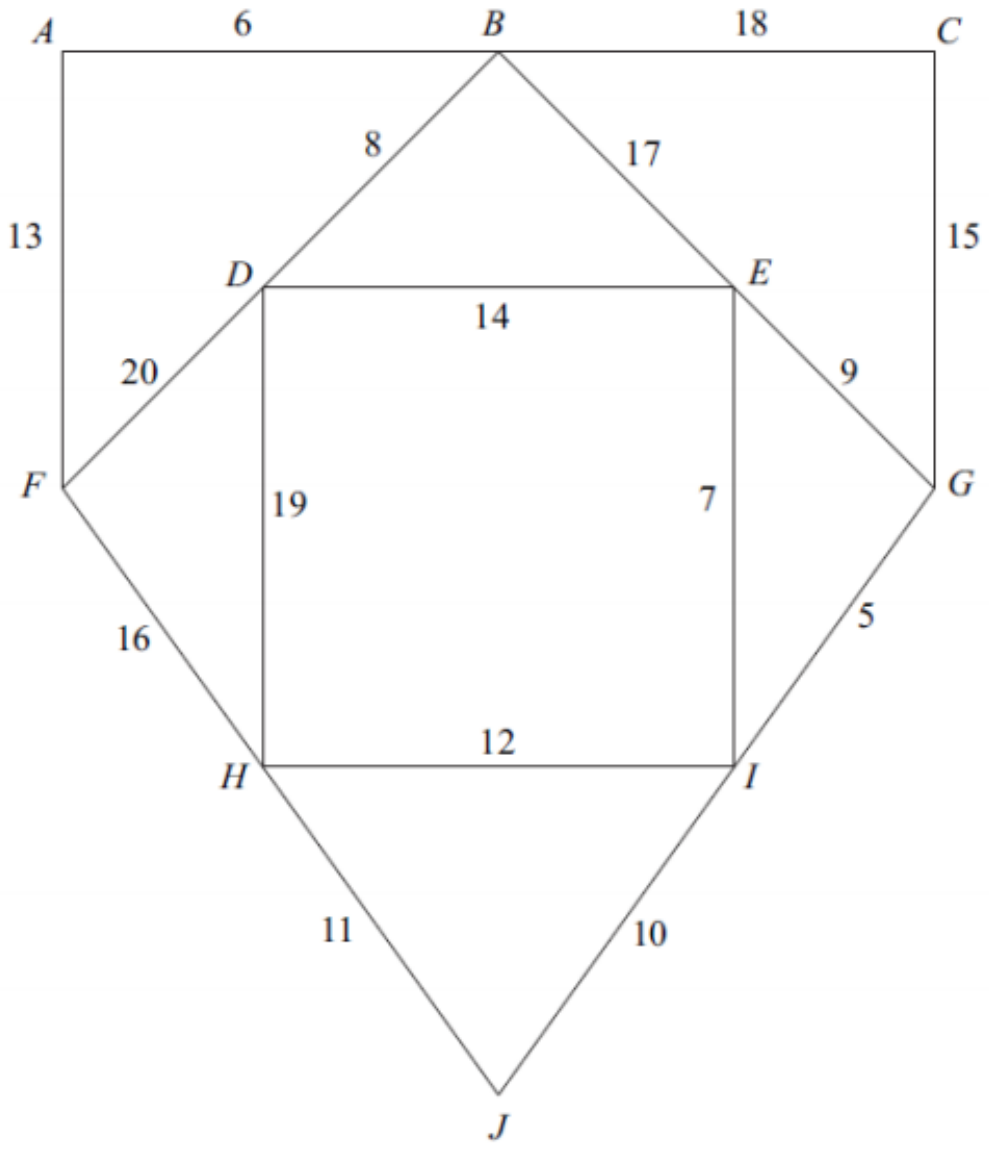
Question



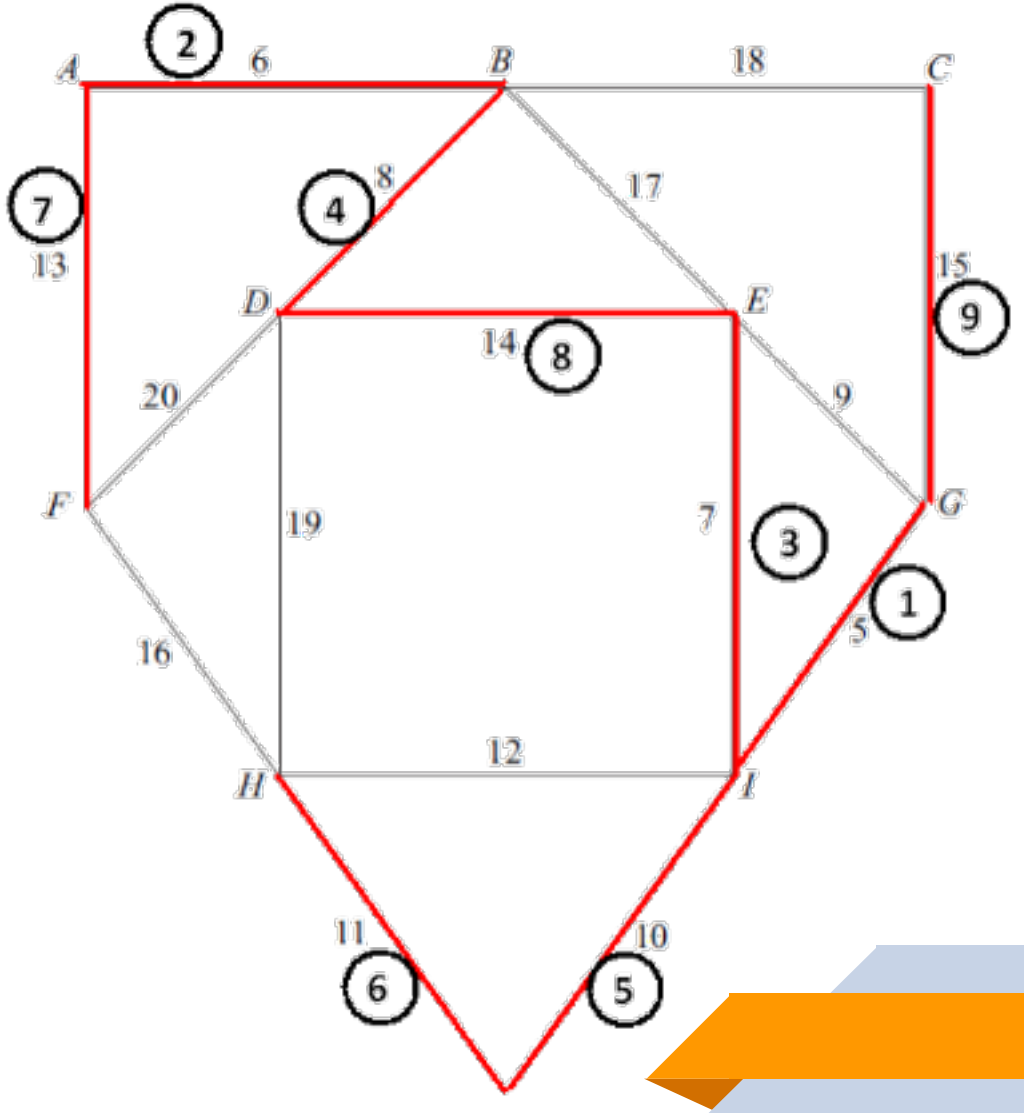


Total Cost=1+1+2 +2+2+2+3+4 +6+8+10

Question



What is the cost and sequence of Minimum spanning tree using Kruskal's algorithm.



Theorem

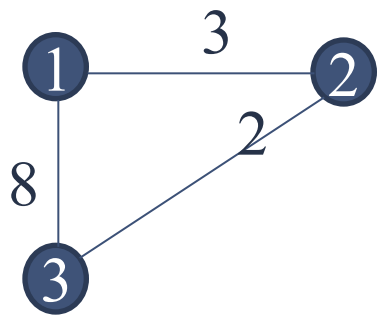
Theorem 1: Kruskal's algorithm generates a minimum-cost spanning tree for every connected undirected graph G.

Note: In the middle of the Kruskal algorithm sometime we get disconnected graph. But for Prim's algorithm we always get connected graph

Spanning Tree

A connected sub-graph H of a given graph G is said to be spanning tree if and only if

- H should contain all vertices of G.
- H should contain $n-1$ edges if G contains n vertices.

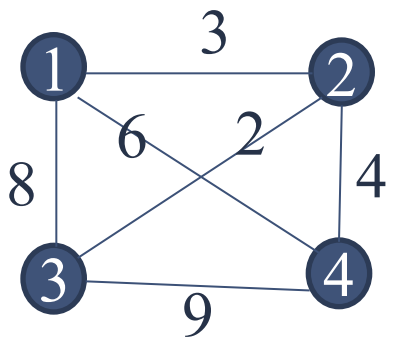


Feasible Solutions	Cost
1-> (2, 3)	11
2-> (1,3)	5
3-> (1,2)	10

Spanning Tree

A connected sub-graph H of a given graph G is said to be spanning tree if and only if

- H should contain all vertices of G.
- H should contain $n-1$ edges if G contains n vertices.



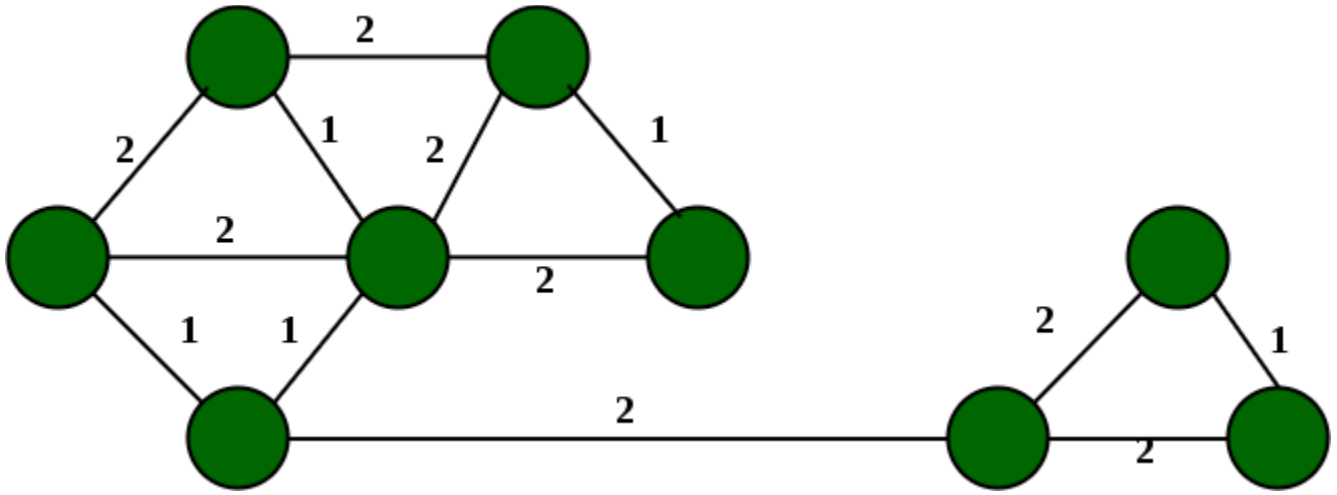
Feasible Solutions
1,2,4,3
1,2,3,4
1,3,2,4
1,3,4,2
1,4,2,3
1,4,3,2
2,1,3,4
2,3,1,4
2,1,4,3
3,1,4,2
3,2,1,4
3,1,2,4
1 –> (2,3,4)
2–> (1,3,4)
3 –> (1,2,4)
4 –> (1,3,2)

Question

What is the no. of spanning trees in a complete graph with n vertices.

a) n
b) n^2
c) n^n
d) n^{n-2}

The number of distinct minimum spanning trees for the weighted graph is



a) 2
b) 6
c) 10
d) 7

Graph

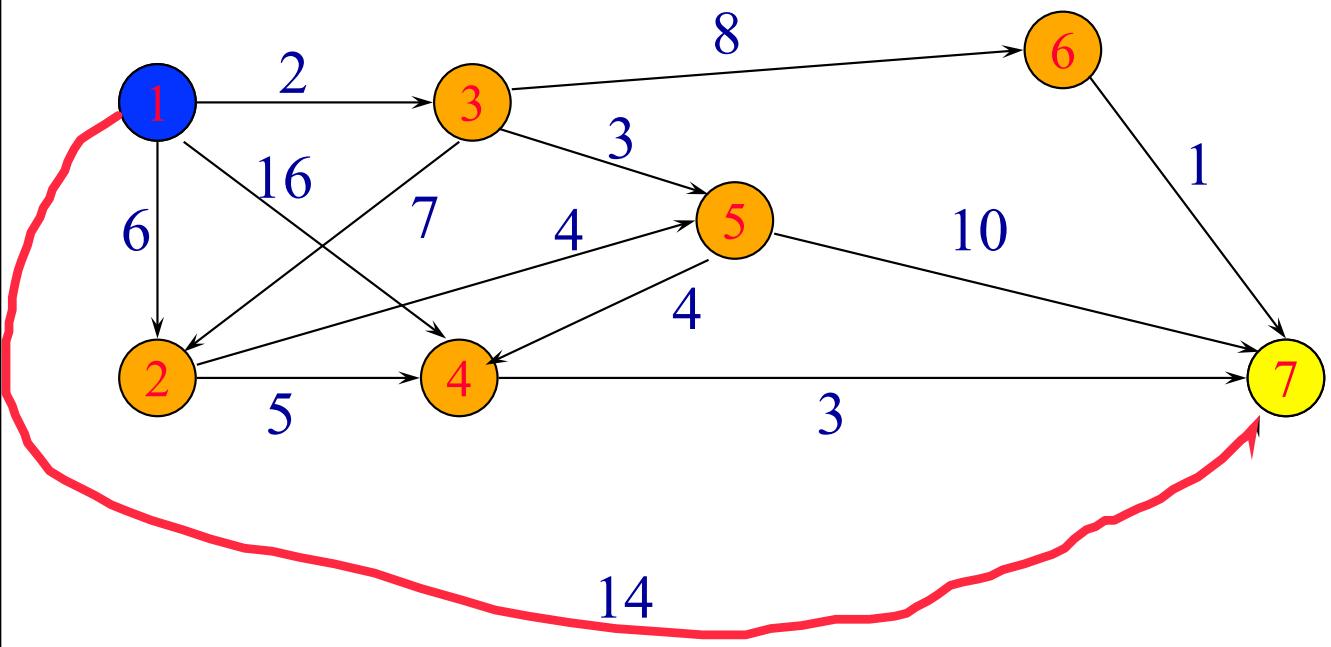
Graphs can be used to represent the connecting structure,

e.g. road of a state or country with vertices representing cities and edges representing sections of roads. The edges can then be assigned weights which may be either the distance between the two cities or the cost of moving from one city to another

Suppose you are to attend a party and for that your friend has invited you to the famous restaurant of city. Now you are interested in following answers

- If there is a path from your home to restaurant?
- If more than one path available ?
- Which is the shortest path?

Path between nodes

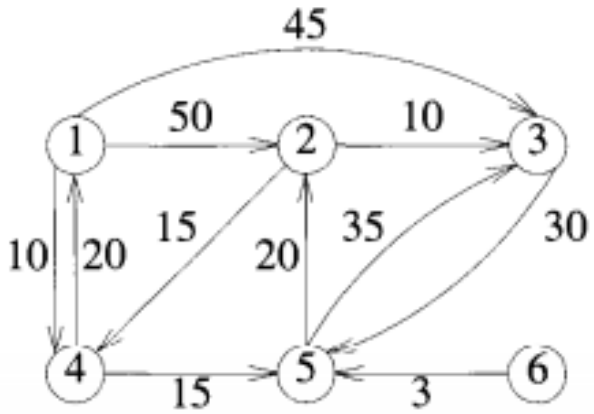


- Consider Source node 1 and destination node 7
- A direct path between Nodes 1 and 7 will cost 14
- A shorter path will cost only 11

Path construction

The greedy way to generate the shortest paths from v_0 to the remaining vertices is to generate the paths in non decreasing order of path length.

- First, a shortest path to the nearest vertex is generated.
- Then a shortest path to the second nearest vertex is generated , and so on



Path	Length
1,4	10
1,4,5	25
1,4,5,2	45
1,3	45

The length of a path is defined to be the sum of the weights of the edges on that path. The starting vertex of the path is referred to as the *source* ,and the last vertex the *destination*

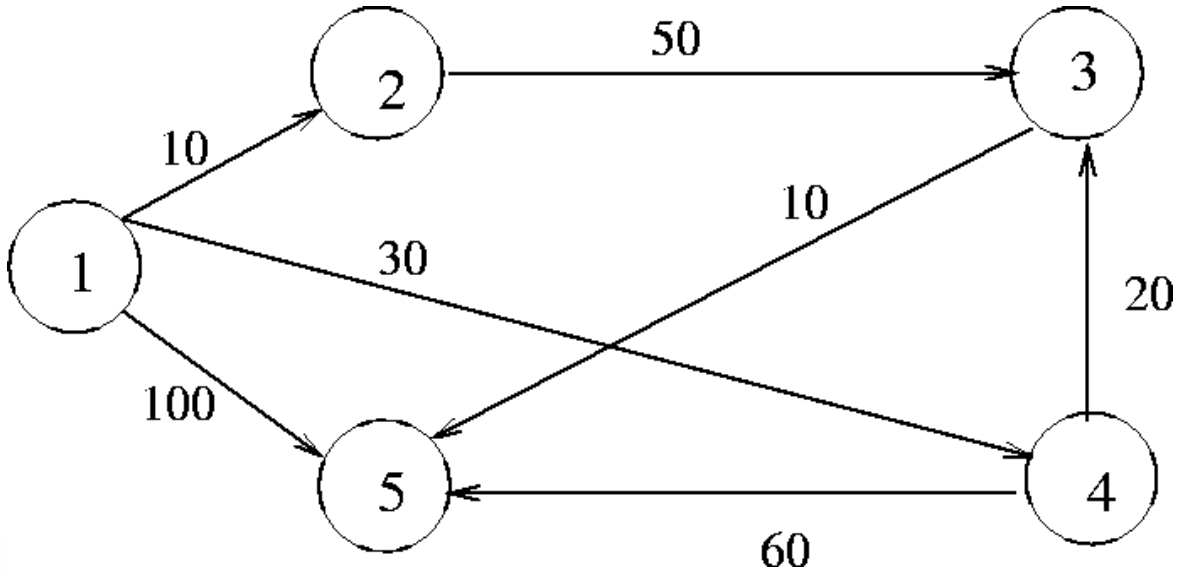
Algorithm

```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1 : n, 1 : n].
{
    for i := 1 to n do
    { // Initialize S.
        S[i] := false; dist[i] := cost[v, i];
    }
    S[v] := true; dist[v] := 0.0; // Put v in S.
    for num := 2 to n − 1 do
    {
        // Determine n − 1 paths from v.
        Choose u from among those vertices not
        in S such that dist[u] is minimum;
        S[u] := true; // Put u in S.
        for (each w adjacent to u with S[w] = false) do
            // Update distances.
            if (dist[w] > dist[u] + cost[u, w])) then
                dist[w] := dist[u] + cost[u, w];
    }
}
```

$S = \{\phi\}$

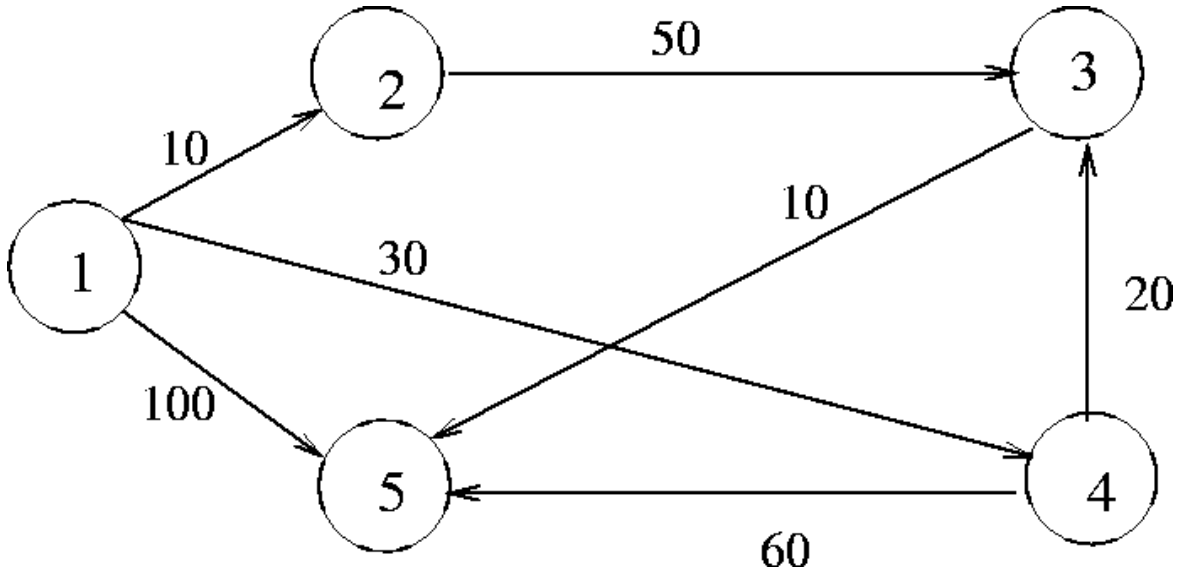
Algorithm

```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1 : n, 1 : n].
{
    for i := 1 to n do
    { // Initialize S.
        S[i] := false; dist[i] := cost[v, i];
    }
    S[v] := true; dist[v] := 0.0; // Put v in S.
    for num := 2 to n − 1 do
    {
        // Determine n − 1 paths from v.
        Choose u from among those vertices not
        in S such that dist[u] is minimum;
        S[u] := true; // Put u in S.
        for (each w adjacent to u with S[w] = false) do
            // Update distances.
            if (dist[w] > dist[u] + cost[u, w])) then
                dist[w] := dist[u] + cost[u, w];
    }
}
```



Initialization: $S = \{1\}$ $D(2) = 10, D(3) = \infty, D(4) = 30, D(5) = 100$

Algorithm



Select $w = 2$, so that $S = \{1, 2\}$

$$D[3] = \min(\infty, D[2] + C[2,3]) = 60$$
$$D[4] = \min(30, D[2] + C[2,4]) = 30$$
$$D[5] = \min(100, D[2] + C[2,5]) = 100$$

Select $w = 4$, so that $S = \{1, 2, 4\}$

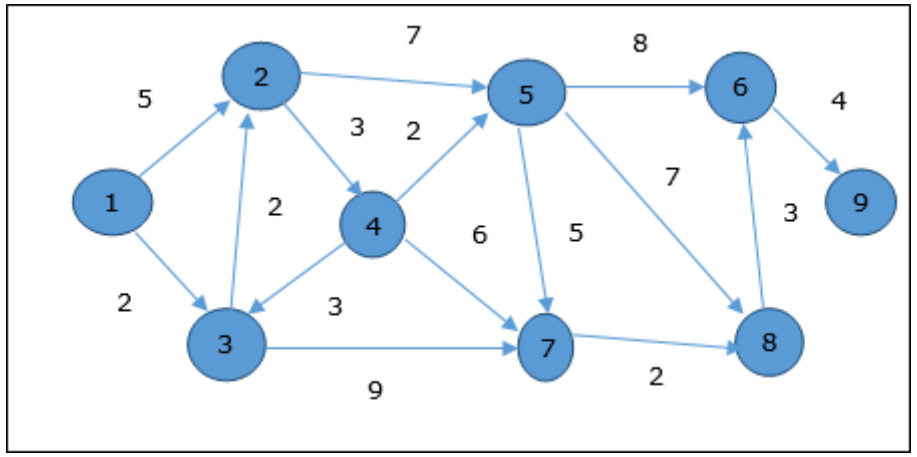
$$D[3] = \min(60, D[4] + C[4,3]) = 50$$
$$D[5] = \min(100, D[4] + C[4,5]) = 90$$

Select $w = 3$, so that $S = \{1, 2, 4, 3\}$

$$D[5] = \min(90, D[3] + C[3,5]) = 60$$

D[2]=10
D[4]=30
D[3]=50
D[5]=60

Example



Node	V_1	V_3	V_2	V_4	V_5	V_7	V_6	V_8
1	0	0	0	0	0	0	0	0
2	5	4	4	4	4	4	4	4
3	2	2	2	2	2	2	2	2
4	∞	∞	7	7	7	7	7	7
5	∞	∞	11	9	9	9	9	9
6	∞	∞	∞	∞	17	17	17	17
7	∞	11	11	11	11	11	11	11
8	∞	∞	∞	∞	16	13	13	13
9	∞	∞	∞	∞	∞	∞	20	20