4.4 MINIMUM SPANNING TREE

- A Spanning tree of an undirected graph, G is a tree formed from graph edges that connects all vertices of G.
- A Minimum Spanning tree of an undirected graph, G is a tree formed from graph edges that connects all vertices of G at lowest cost.
- A minimum spanning tree exists if and only if G is connected. The number of edges in the minimum spanning tree is |V| -1.
- The minimum spanning tree is a tree because it is acyclic, it is spanning because it covers every vertex, and it is minimum because it covers with minimum cost.
- The minimum spanning tree can be created using two algorithms, that is prim's algorithm and kruskal's algorithm.

4.4.1 PRIM'S ALGORITHM

In this method, minimum spanning tree is constructed in successive stages. In each stage, one node is picked as a root and an edge is added and thus an associated vertex is added to the tree.

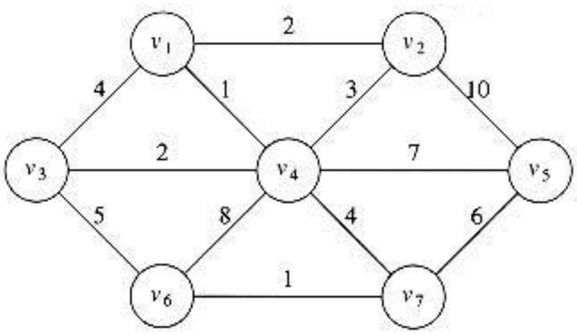
The Strategy

1. One node is picked as a root node (u) from the given connected graph.
2. At each stage choose a new vertex v from u, by considering an edge (u,v) with minimum cost among all edges from u, where u is already in the tree ad v is not in the tree.
3. The prims algorithm table is constructed with three parameters. They are
   - known – known vertex i.e., processed vertex is indicated by 1. Unknown vertex is indicated by zero.
   - dv - Weight of the shortest edge connecting v to the known vertex.

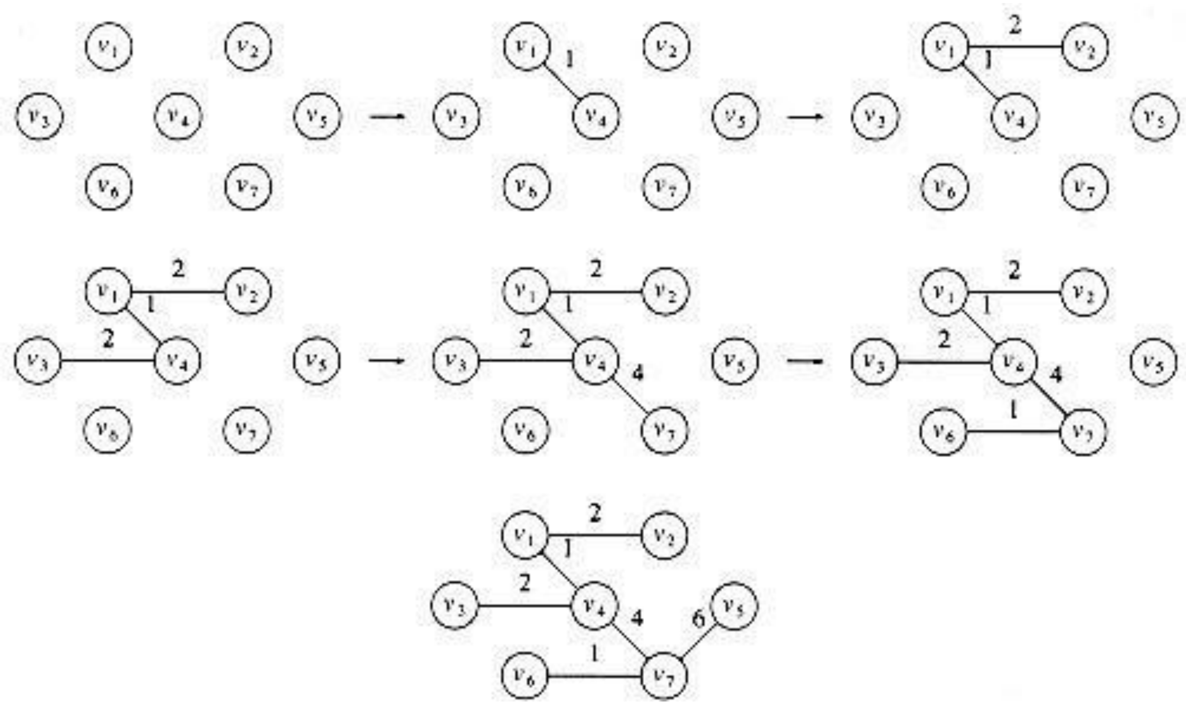
- pv - It contains last vertex to cause a change in dv.

4. After selecting the vertex v, the update rule is applied for each unknown w adjacent to v. The rule is dw = min (dw , Cw,v).

Example:



Prim's Algorithm after each stage



Steps

i. v1 is selected as initial node and construct initial configuration of the table.

v	Known	dv	pv
V1	0	0	0
V2	0	∞	0

V3	0	∞	0
V4	0	∞	0
V5	0	∞	0
V6	0	∞	0
V7	0	∞	0

ii. v1 is declared as known vertex. Then its adjacent vertices v2, v3, v4 are updated.

T[v2].dist = min(T[v2].dist, Cv1,v2) = min (∞ ,2) = 2

T[v3].dist = min(T[v3].dist, Cv1,v3) = min (∞ ,4) = 4

T[v4].dist = min(T[v4].dist, Cv1,v4) = min (∞ ,1) = 1

v	**Known**	**dv**	**pv**
V1	1	0	0
V2	0	2	V1
V3	0	4	V1
V4	0	1	V1
V5	0	∞	0
V6	0	∞	0
V7	0	∞	0

iii. Among all adjacent vertices V2, V3, V4. V1 -> V4 distance is small. So V4 is selected and declared as known vertex. Its adjacent vertices distance are updated.

- V1 is not examined because it is known vertex.
- No change in V2 , because it has dv = 2 and the edge cost from V4 -> V2 = 3.

    T[v3].dist = min(T[v3].dist, Cv4,v3) = min (4 ,2) = 2

    T[v5].dist = min(T[v5].dist, Cv4,v5) = min (∞ ,7) = 7

T[v6].dist = min(T[v6].dist, Cv4,v6) = min (∞ ,8) = 8

T[v7].dist = min(T[v7].dist, Cv4,v7) = min (∞ ,4) = 4

v	**Known**	**dv**	**pv**
V1	1	0	0
V2	0	2	V1
V3	0	2	V4
V4	1	1	V1
V5	0	7	V4
V6	0	8	V4
V7	0	4	V4
V7	0	4	V4

iv. Among all either we can select v2, or v3 whose dv = 2, smallest among v5, v6 and v7.

- v2 is declared as known vertex.
- Its adjacent vertices are v1, v4 and v5. v1, v4 are known vertex, no change in their dv value.

T[v5].dist = min(T[v5].dist, Cv2,v5) = min (7 ,10) = 7

v	**Known**	**dv**	**pv**
V1	1	0	0
V2	1	2	V1
V3	0	2	V4
V4	1	1	V1
V5	0	7	V4
V6	0	8	V4
V7	0	4	V4

v. Among all vertices v3's dv value is lower so v3 is selected. v3‟s adjacent vertices are v1, v4 and v6. No changes in v1 and v4.

T[v6].dist = min(T[v6].dist, Cv3,v6) = min (8 ,5) = 5

v	Known	dv	pv
V1	1	0	0
V2	1	2	V1
V3	1	2	V4
V4	1	1	V1
V5	0	7	V4
V6	0	5	V3
V7	0	4	V4

vi. Among v5, v6, v7, v7‟s dv value is lesser, so v7 is selected. Its adjacent vertices are v4, v4, and v6. No change in v4.

T[v5].dist = min(T[v5].dist, Cv7,v5)

= min (7,6) = 6

T[v6].dist = min(T[v6].dist, Cv7,v6)

= min (5,1) = 1

v	Known	dv	pv
V1	1	0	0
V2	1	2	V1
V3	1	2	V4
V4	1	1	V1
V5	0	6	V7
V6	0	1	V7
V7	1	4	V4

vii. Among v5 and v6, v6 is declared as known vertex. v6‟s adjacent vertices are v3, v4, and v7, no change in dv value, all are known vertices.

v	Known	dv	pv
V1	1	0	0
V2	1	2	V1
V3	1	2	V4
V4	1	1	V1
V5	0	6	V7
V6	1	1	V7
V7	1	4	V4

The minimum cost of spanning tree is 16.

viii. Finally, v5 is declared as known vertex. Its adjacent vertices are v2, v4, and v7, no change in dv value, all are known vertices.

v	Known	dv	pv
V1	1	0	0
V2	1	2	V1
V3	1	2	V4
V4	1	1	V1
V5	1	6	V7
V6	1	1	V7
V7	1	4	V4

The minimum cost of spanning tree is 16.

Algorithm Analysis

The running time is O(|V|2) in case of adjacency list and O(|E| log |V|) in case of binary heap.

4.4.2 KRUSKAL'S ALGORITHM

A second greedy strategy is repeatedly to select the edges in order of smallest weight and accept an edge if it does not cause a cycle. Steps:

1. Initially, there are |V| single-node trees. Adding an edge merges two trees into one.
2. When the algorithm terminates, there is only one tree, and this is the minimum spanning tree.
3. The algorithm terminates when enough edges are accepted.

The strategy

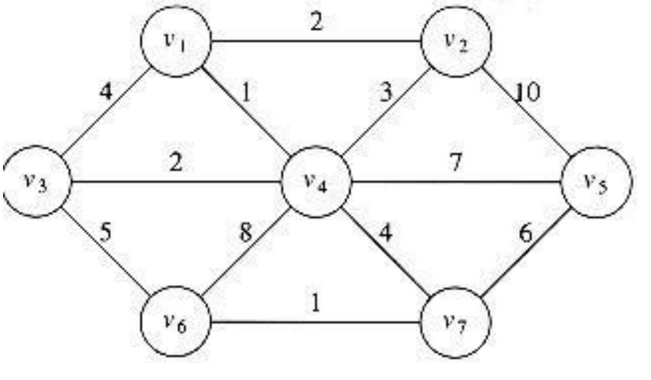
i. The edges are built into a minheap structure and each vertex is considered as a sigle node tree.

ii. The deletemin operation is used to find the minimum cost edge (u,v).

iii. The vertices u and v are searched in the spanning tree set S and if the returned sets are not same then (u,v) is added to the set s with the constraint that adding (u,v) will not create a cycle in spanning tree set S.

iv. Repeat step (ii) and (iii) until a spanning tree is constructed with |V| - 1 edges.

Example



i. Initially all the vertices are single node trees.

ii. Select the smallest edge v1 to v4, both the nodes are different sets, it does not form cycle.

iii. Select the next smallest edge v6 to v7. These two vertices are different sets; it does not form a cycle, so it is included in the MST.

iv. Select the next smallest edge v1 to v2. These two vertices are different sets; it does not form a cycle, so it is included in the MST.

v. Select the next smallest edge v3 to v4. These two vertices are different sets; it does not form a cycle, so it is included in the MST.

vi. Select the next smallest edge v2 to v4 both v2 and v4 are same set, it forms cycle so v2 – v4 edge is rejected.

vii. Select the next smallest edge v1 to v3, it forms cycle so v1 – v3 edge is rejected.

viii. Select the next smallest edge v4 to v7, it does not form a cycle so it is included in the tree.

ix. Select the next smallest edge v3 to v6, it forms a cycle so v3 – v6 edge is rejected.

x. Select the next smallest edge v5 to v7, it does not form a cycle so it is included in the tree.

Edge	Weight	Action
(v1,v4)	1	Accepted
(v6,v7)	1	Accepted
(v1,v2)	2	Accepted
(v3,v4)	2	Accepted
(v2,v4)	3	Rejected
(v1,v3)	4	Rejected
(v4,v7)	4	Accepted
(v3,v6)	5	Rejected

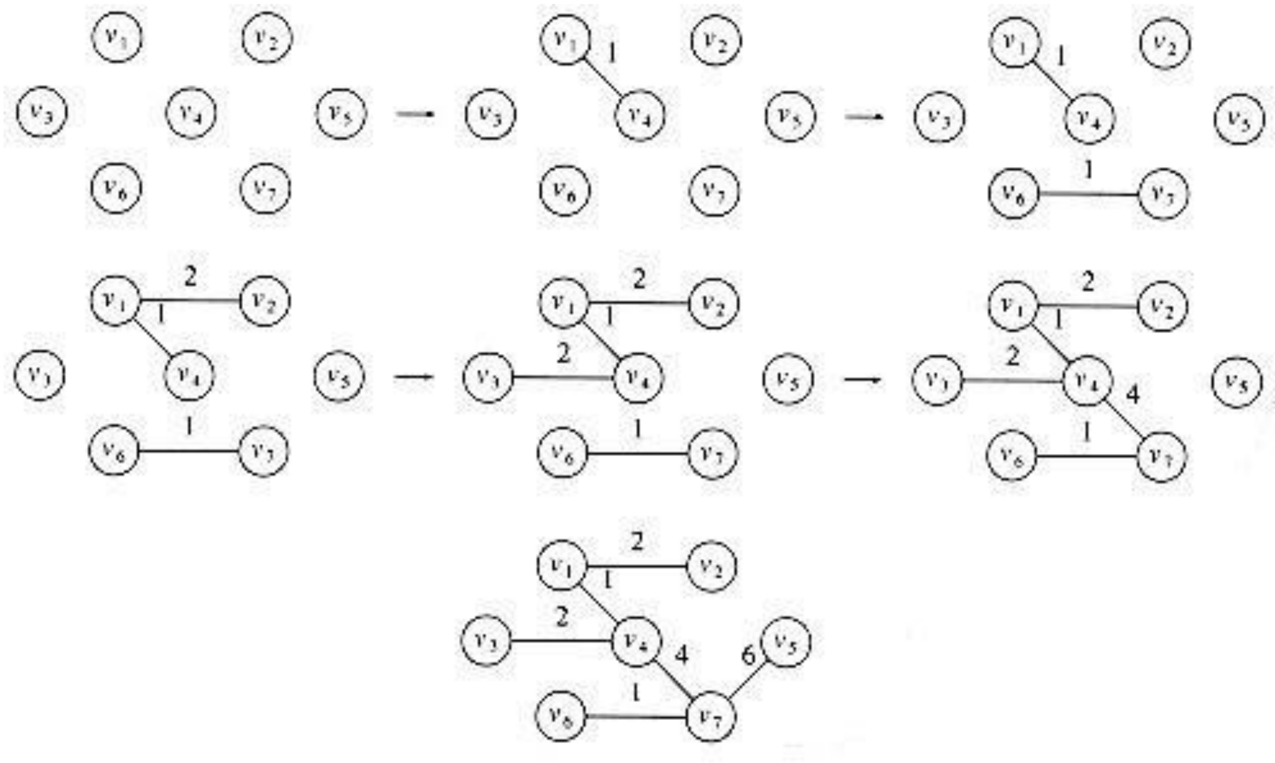
(v5,v7)	6	Accepted
(v3,v6)	5	Rejected
(v5,v7)	6	Accepted

Figure: Action of Kruskal's algorithm on G

All the nodes are included. The cost of minimum spanning tree = 16 (2 + 1+ 2

+ 4

+ 1 + 6).



Routine for kruskals algorithm

```
void kruskal( graph G )
{
    int EdgesAccepte d;
    DisjSet S;
    PriorityQueue H;
    vertex u, v;
    SetType uset, vset;
    Edge e;
   Initialize( S );        // form a single node tree ReadGraphIntoHeapArray( G, H );
```

```
        BuildHeap( H );
        EdgesAccepted = 0;
        while( EdgesAccepted < NumVertex-1 )
        {
                e = DeleteMin( H );        // Selection of minimum edge
                uset = Find( u, S );
                vset = Find( v, S );
                if( uset != vset )
                {
                        /* accept the edge */
                        EdgesAccepted++;
                        SetUnion( S, uset, vset);
                }
        }
}
```

- The appropriate data structure is the union/find algorithm
- The worst-case running time of this algorithm is O(|E| log |E|), which is dominated by the heap operations. Notice that since |E| = O(|V|2), this running time is actually O(|E| log |V|).