## 3.4 MINIMUM SPANNING TREE: PRIM'S ALGORITHM

- 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**.

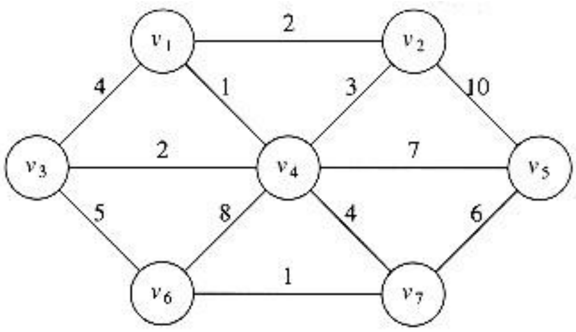
### PRIM'S ALGORITHM

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.

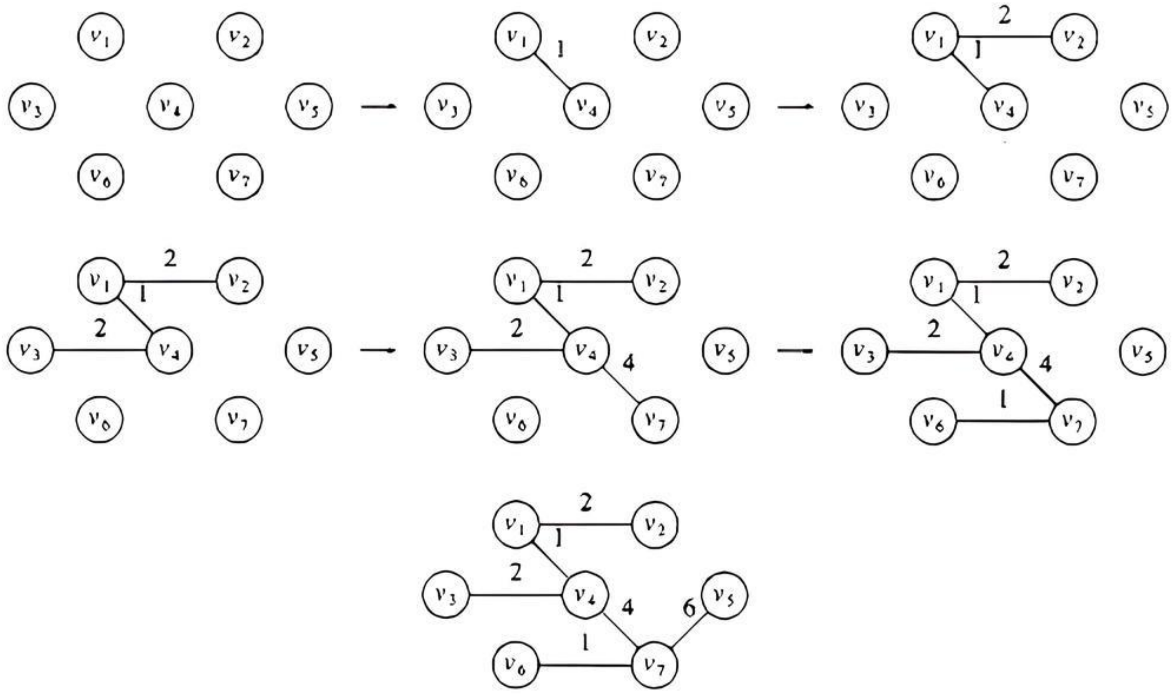
### The Strategy

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 |

**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.