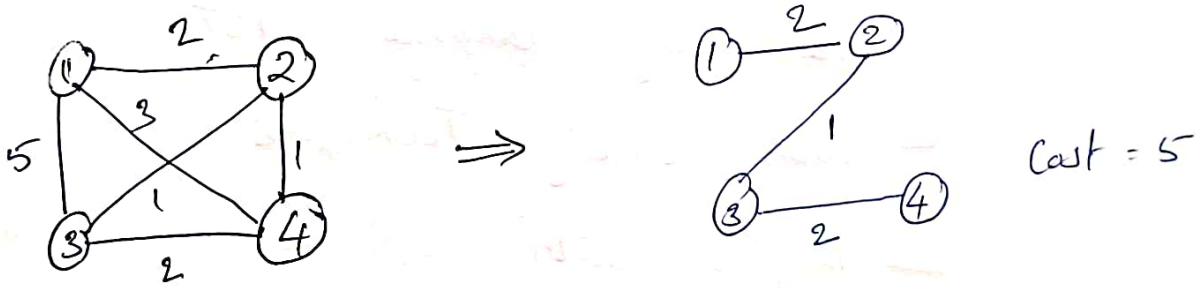
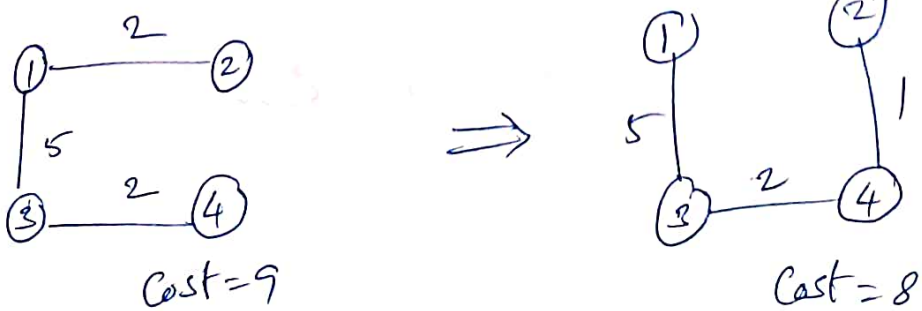
Minimum Spanning Tree

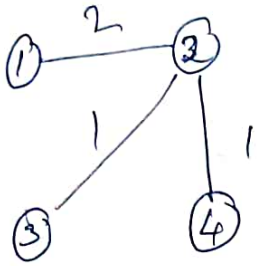
→ A spanning tree of a connected graph is connected acyclic graph that contains all the vertices of a subgraph

- All vertices
- Connected
- No cycles

→ Remove edges if it forms cycle







Cost = 4

2 types algorithm to find MST

① Prim's Algorithm

② kruskal Algorithm

Prim's Algorithm:

→ one way: Compute MST.

→ uses Greedy Techniques

→ begins with set 'u' initialized to '1'

→ choose one edge at a time

→ each state it finds the shortest edge (u, v) → will not in MST.

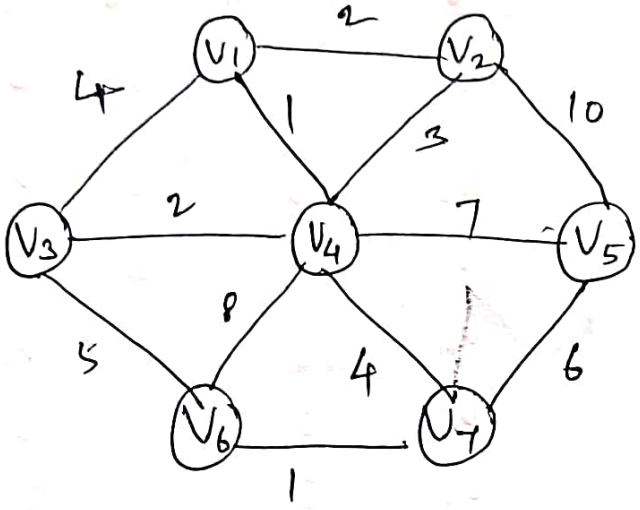
↳ cost is low among all edges.

↓

will be in MST

→ $d_w = \text{Min}(d_w, c_{w,v})$

Eg:



① Choose an arbitrary start vertex

② keep including connected edges

Initialize Config

V	known	d_v	P_v
V_1	0	0	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0
V_6	0	∞	0
V_7	0	∞	0

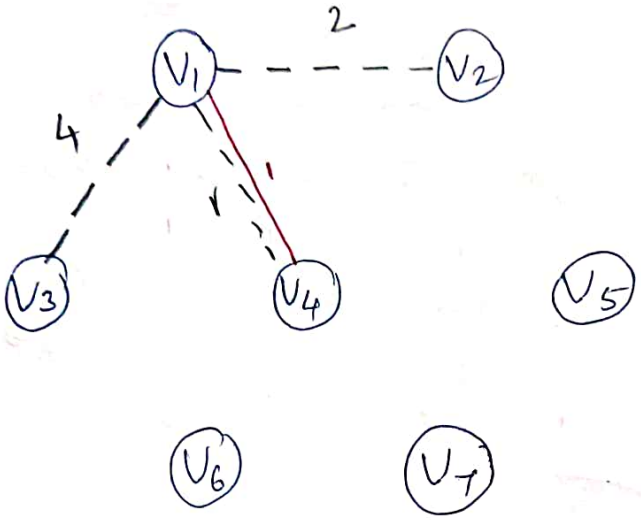
* Consider V_1 as source vertex, and find adjacent vertex of V_1

$V_1 \rightarrow V_2, V_3, V_4$

$V_1 \xrightarrow{2} V_2$

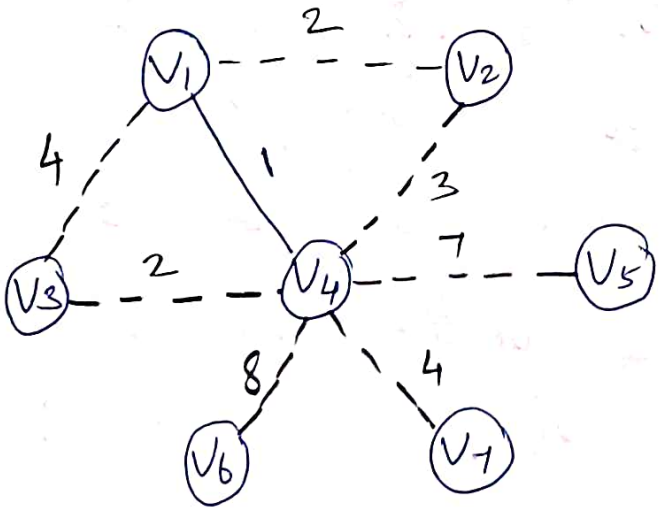
$V_1 \xrightarrow{4} V_3$

$V_1 \xrightarrow{1} V_4$ - minimum distance so select edges



V	Known	d_V	P_V
V_1	1	0	0
V_2	0	2	V_1
V_3	0	4	V_1
V_4	0	1	V_1
V_5	0	∞	0
V_6	0	∞	0
V_7	0	∞	0

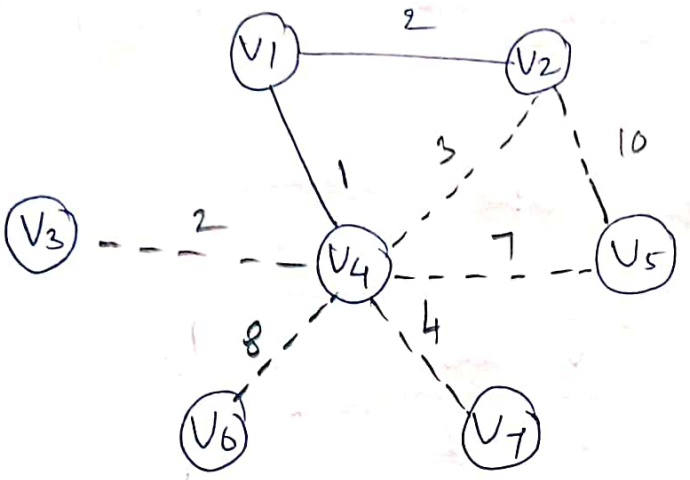
→ In this table minimum d_V is 1 i.e V_4 it is unknown vertex, choose V_4 is known vertex find adjacent vertex $V_4 \rightarrow V_2, V_3, V_5, V_6, V_7$



V	Known	d_V	P_V
V_1	1	0	0
V_2	0	2	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	8	V_4
V_7	0	4	V_4

→ Next minimum d_V is V_2, V_3 [unknown vertex], choose V_2 as known vertex

$V_2 \rightarrow V_1, V_4, V_5$ [V_1, V_4 already known vertex].



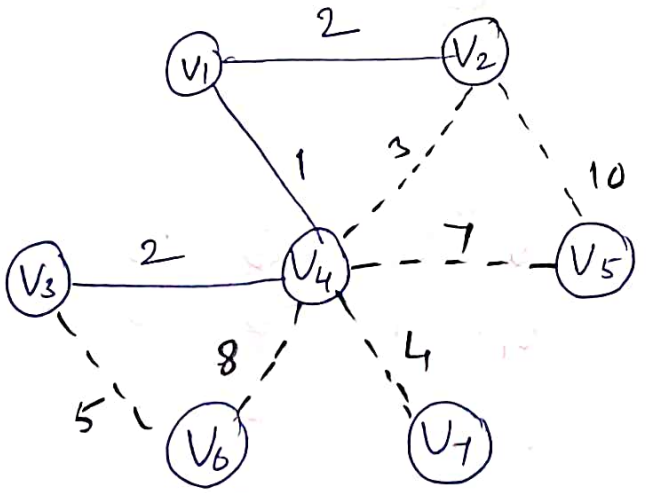
V	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	8	V_4
V_7	0	4	V_4

V_3 as known

adjacent of $V_3 \rightarrow V_1, V_4, V_6$ (already known vertex)

$V_4 - V_6 = 8$

$V_3 \rightarrow V_6 = 5 \rightarrow$ Minimum distance, update table entry



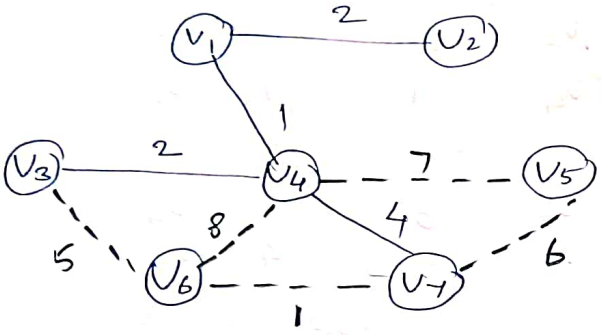
V	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	5	V_4
V_7	0	4	V_4

V_7 having minimum distance, so consider V_7 as known vertex

V_7 adjacent vertex is = V_6, V_4, V_5 (Already known.)

$V_4 - V_5$ is 7
$V_5 - V_7$ is 6 } → Min update table

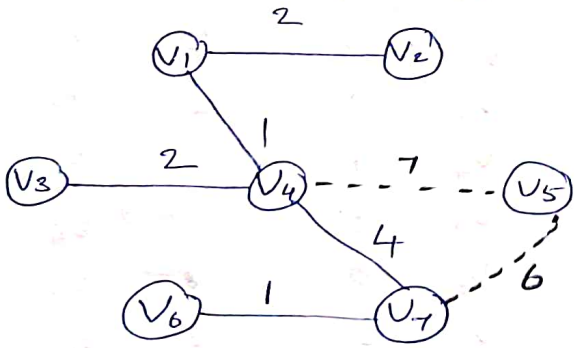
$V_6 - V_3 = 5$
$V_6 - V_7 = 1$ → Min update table



v	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	6	V_7
V_6	0	1	V_7
V_7	1	4	V_4

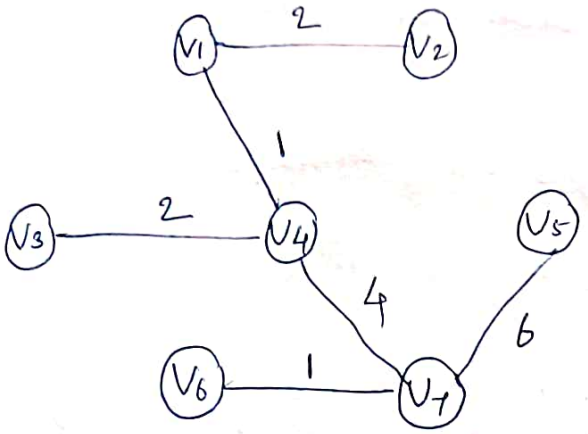
→ V_6 as known vertex.

V_6 adjacent – V_3, V_4, V_7 (known vertex)



V	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	6	V_7
V_6	1	1	V_7
V_7	1	4	V_4

V_5 as known vertex



V	known	dv	Pv
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	1	6	V_7
V_6	1	1	V_7
V_7	1	4	V_4

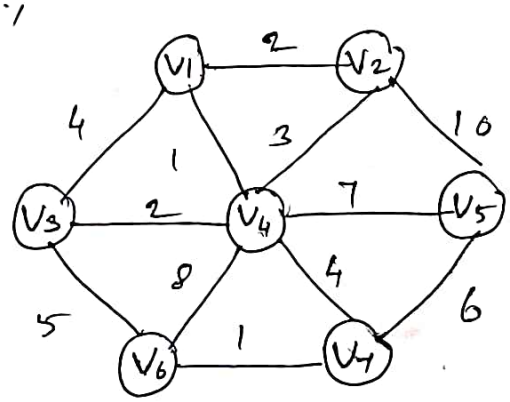
Routine:

```
Void Prims(Table T)
{ int i;
  Vertex v,w;
  for (i=0; i<numvertex; i++)
   {
     T[i].known = false;
     T[i].Dist = Infinity;
     T[i].path = 0;
   }
  for(;;)
  {
    let v be the start vertex
    with smallest distance
     T[v].dist=0;
     T[v].known = True;
     for each w adjacent to v
      if (!T[w].known)
       { T[w].Dist = Min(T[w].Dist, Cvw);
         T[w].Path = v;
       }
  }
```

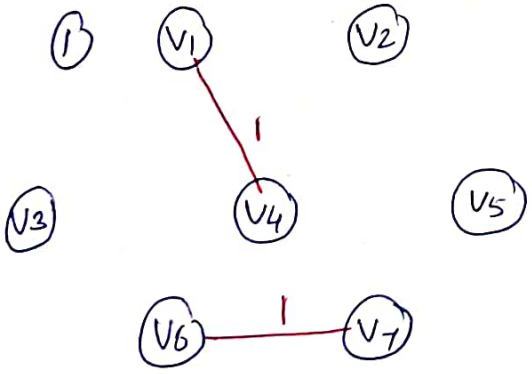
Kruskal Algorithm:

→ Compute MST
→ select edges in the order of smallest weight
→ decesnot form cycle
→ Terminate if all edges are accepted
→ Main tain forest collection of trees.
→ Two operation
① find — returns the root of the tree
② union — Merge two trees, make root ptr.

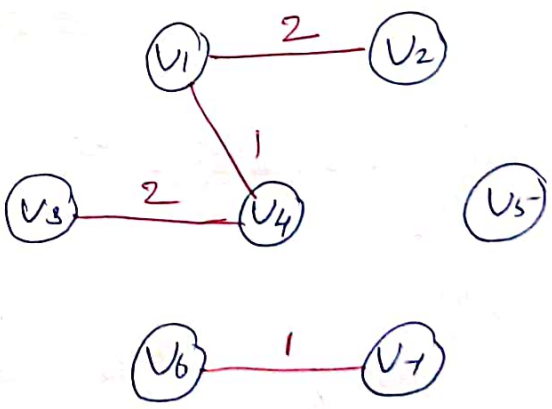
Eg:1



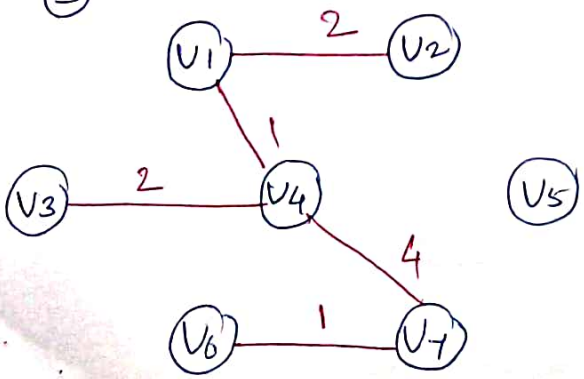
(1)



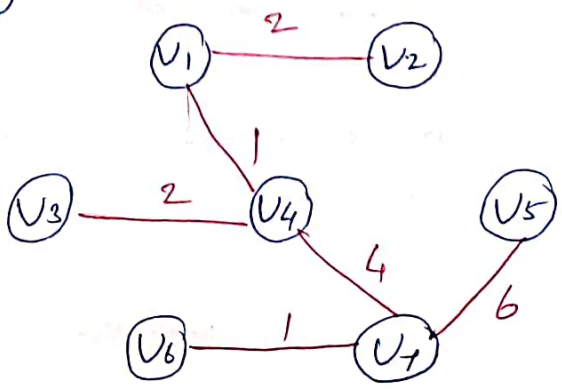
(2)



(3)



(4)



Edge	weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_4, v_3)	2	Accepted
(v_4, v_2)	3	Rejected
(v_4, v_7)	4	Accepted
(v_1, v_2)	4	Rejected
(v_4, v_5)	7	Rejected
(v_4, v_5)	6	Accepted
(v_2, v_5)	10	Rejected

Routine:

```
void Kruskal (Graph G)
{
    int edgesaccepted;
    Disjset S;
    PriorityQueue H;
    Vertex U, V;
    setType Uset, Vset;
    Edge E;
    Initialize (S);
    Read graph Into Heap array (G, H);
    Build heap (H);
    Edges accepted = 0;
    while Edges Accepted < NumVertex - 1>
    {
```

```
E = DeleteMin (H);
Vset = find (u, s);
Vset = find (v, s);
if (Vset != Vset)
{
    Edges Accepted++;
    setUnion (s, Vset, Vset);
}
}
```