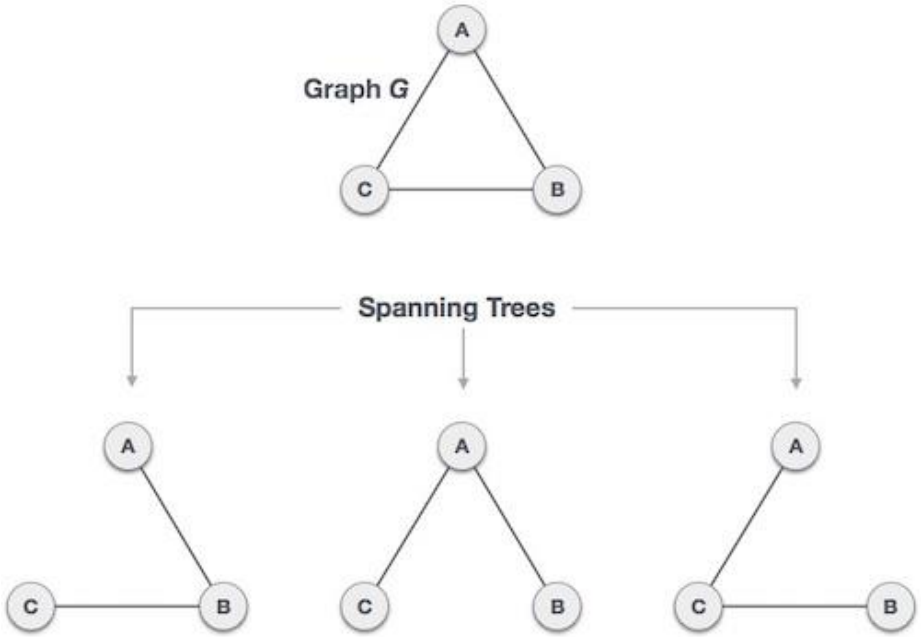
Minimum Spanning Tree

SWE2016-44

Definitions

1. **Spanning Tree**
   - **Given a <u>connected</u> and <u>undirected</u> graph, a <u>*spanning tree*</u> of that graph is a subgraph that is a tree connects all the vertices together.**



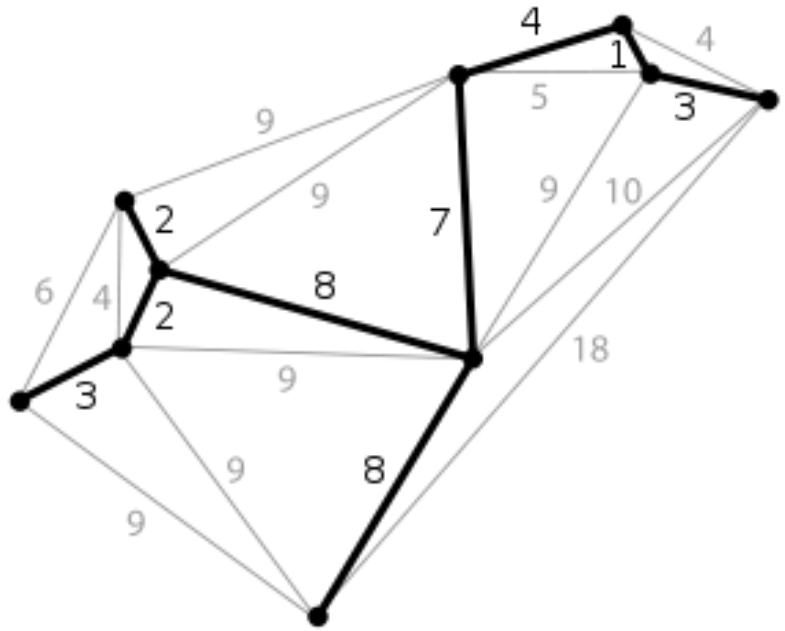
Definitions

1. **Spanning Tree**
   - **Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree connects all the vertices together.**

   - **Properties**
     - **The spanning tree does not have any cycle**
     - **Spanning tree has n-1 edges, where n is the number of vertices**
     - **From a complete graph, by removing maximum e - n + 1 edges, we can construct a spanning tree.**

Definitions

2. **Minimum Spanning Tree (MST)**
   - **The spanning tree of the graph whose sum of weights of edges is minimum.**
     - ***A graph may have more than 1 minimum spanning tree.***



Definitions

2. **Minimum Spanning Tree (MST)**
   - **The spanning tree of the graph whose sum of weights of edges is minimum.**
     - ***A graph may have more than 1 minimum spanning tree.***
   - **Two most important MST**
     - **Kruskal's Algorithm**
     - **Prim's Algorithm**

     ※ **Both are greedy algorithms.**

Kruskal's Algorithm

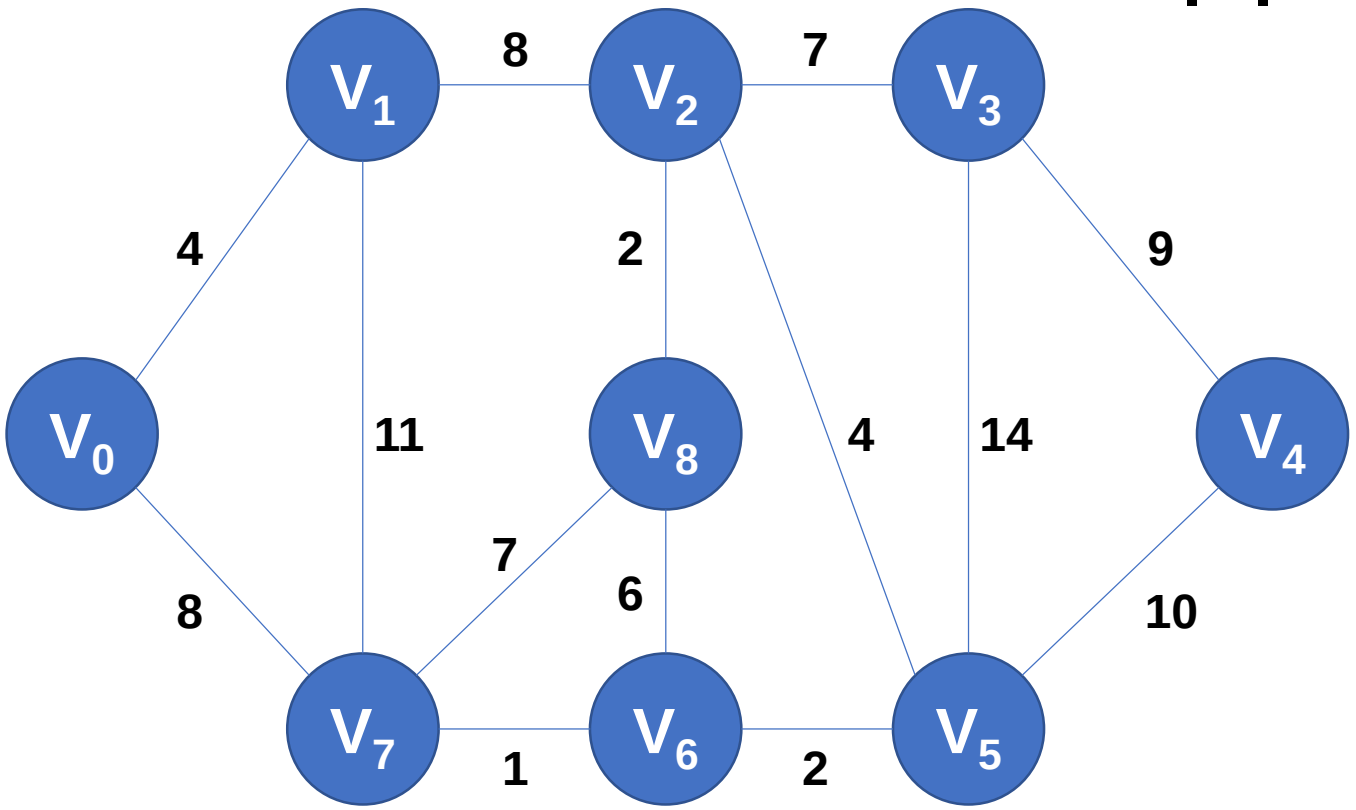
Kruskal's Algorithm

1. **Sort all the edges in non-decreasing order of their weight.**

2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**

3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

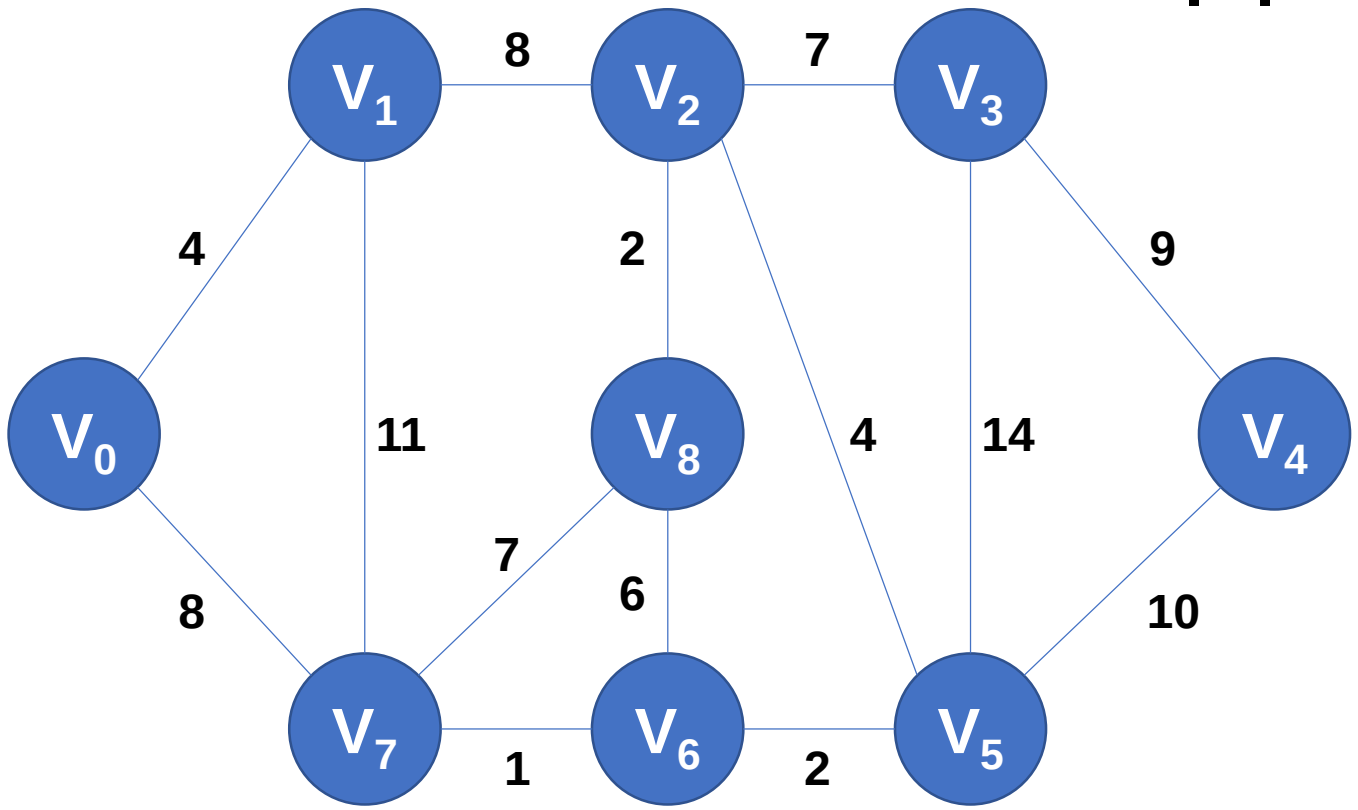
Examples



1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Examples

|V|=9

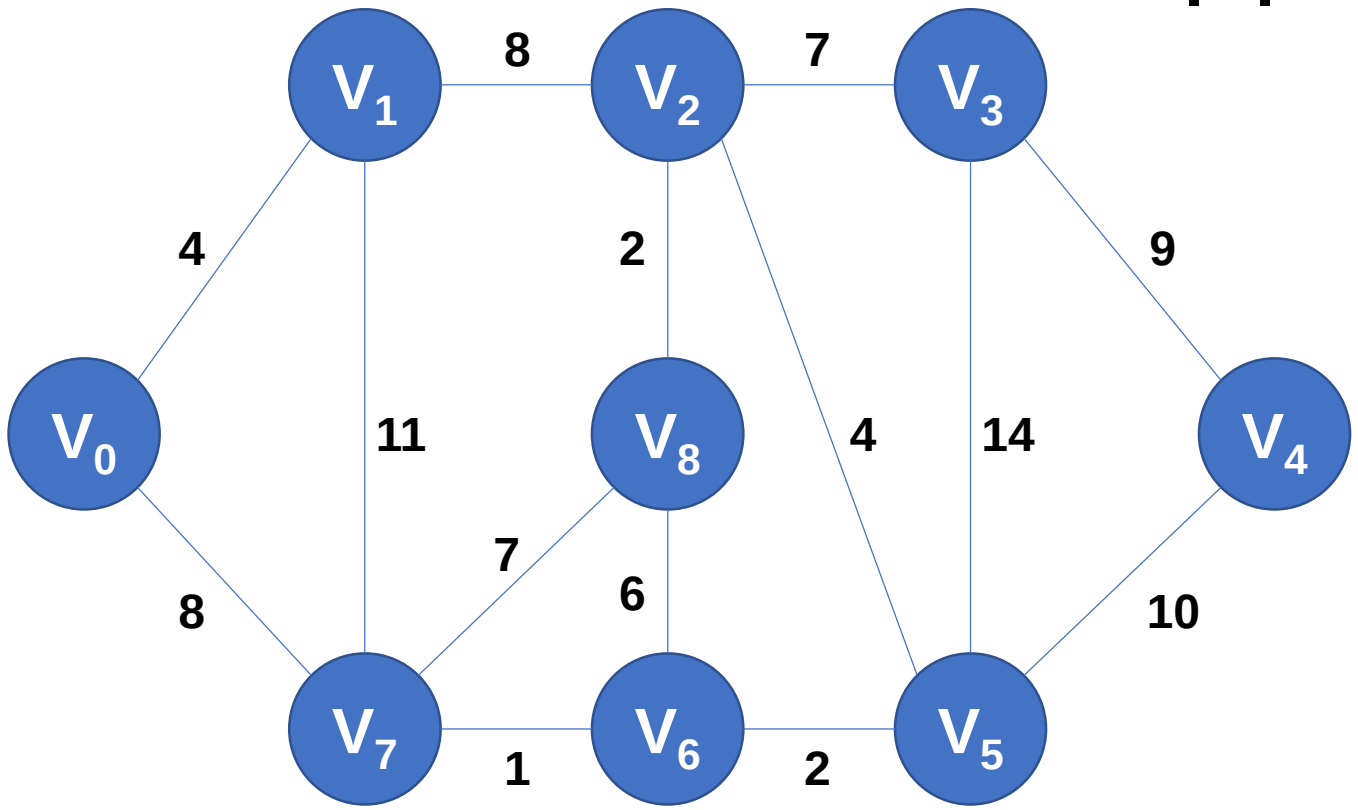


1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9

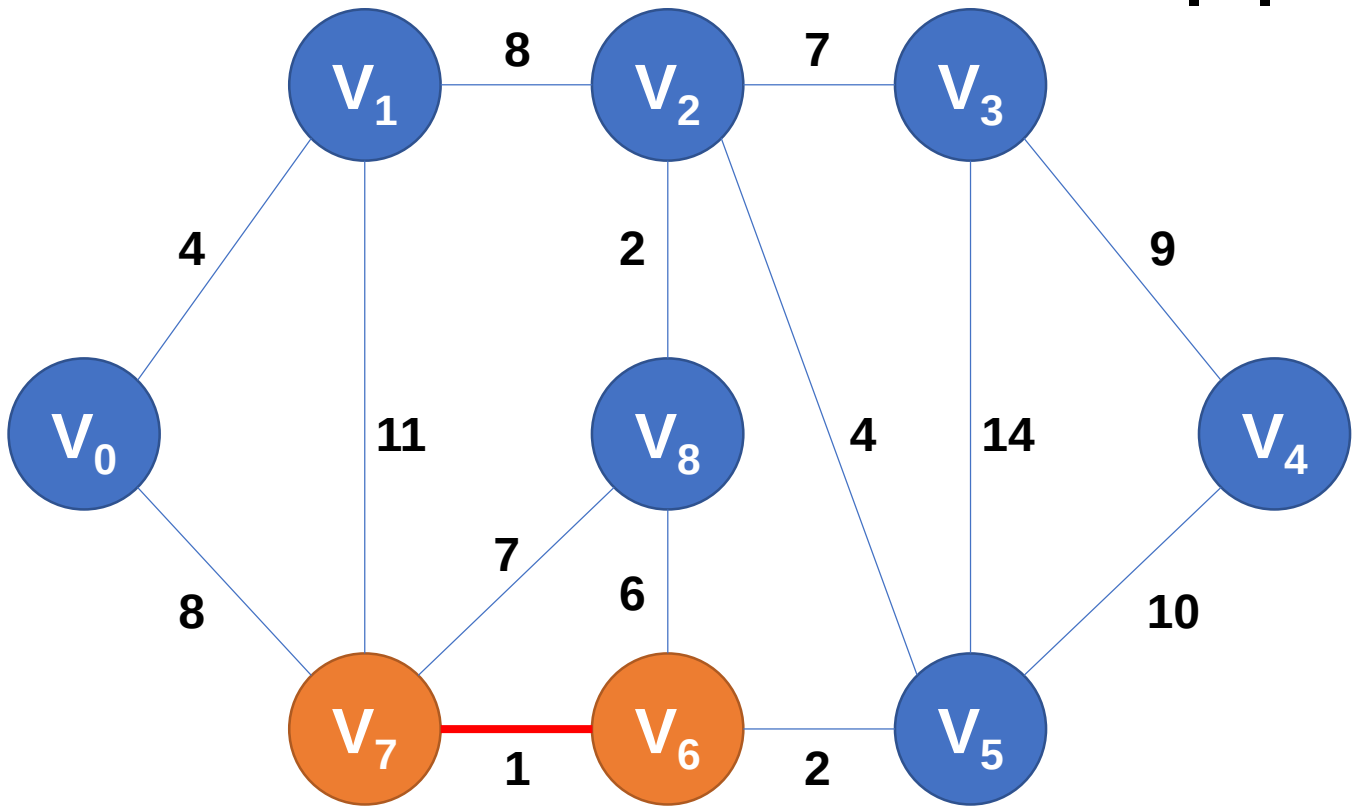


1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



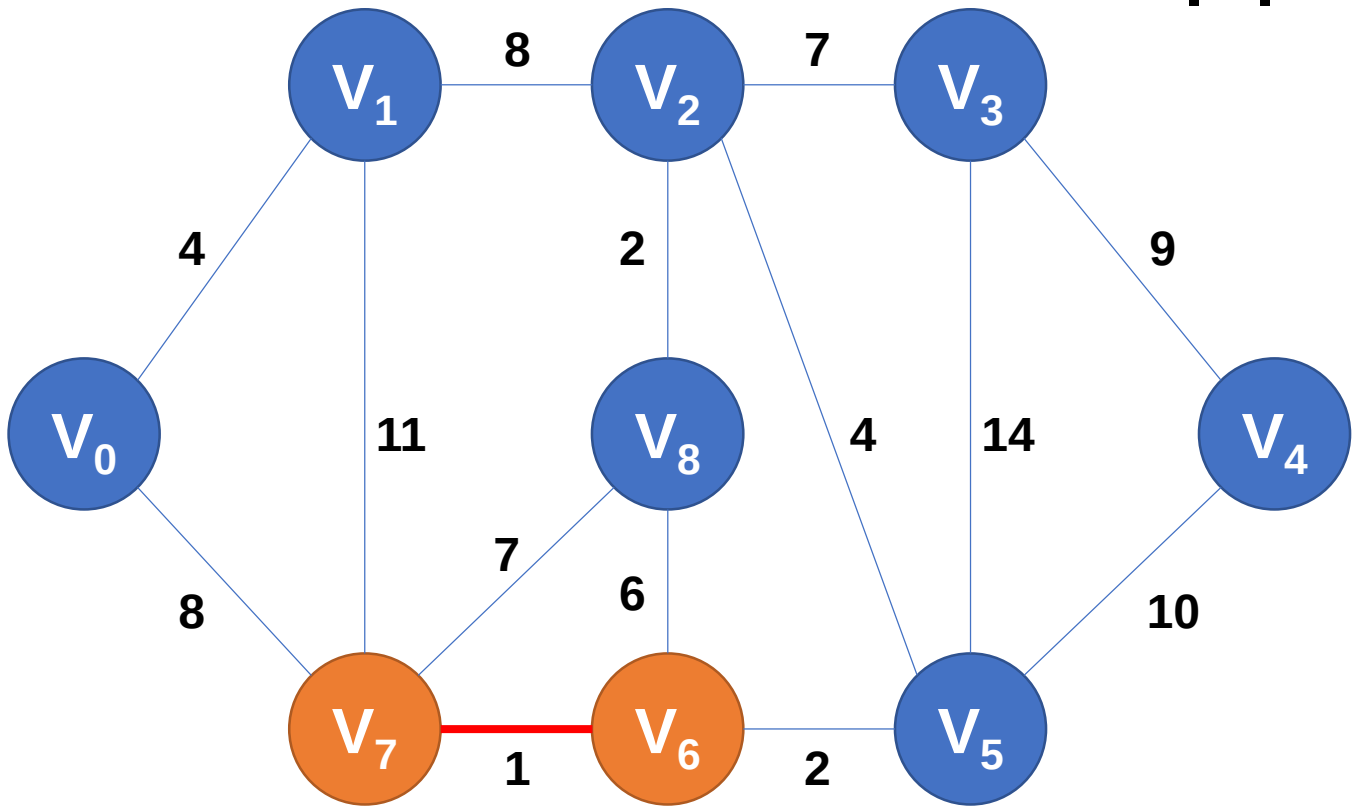
# of Edges in MST = 1

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



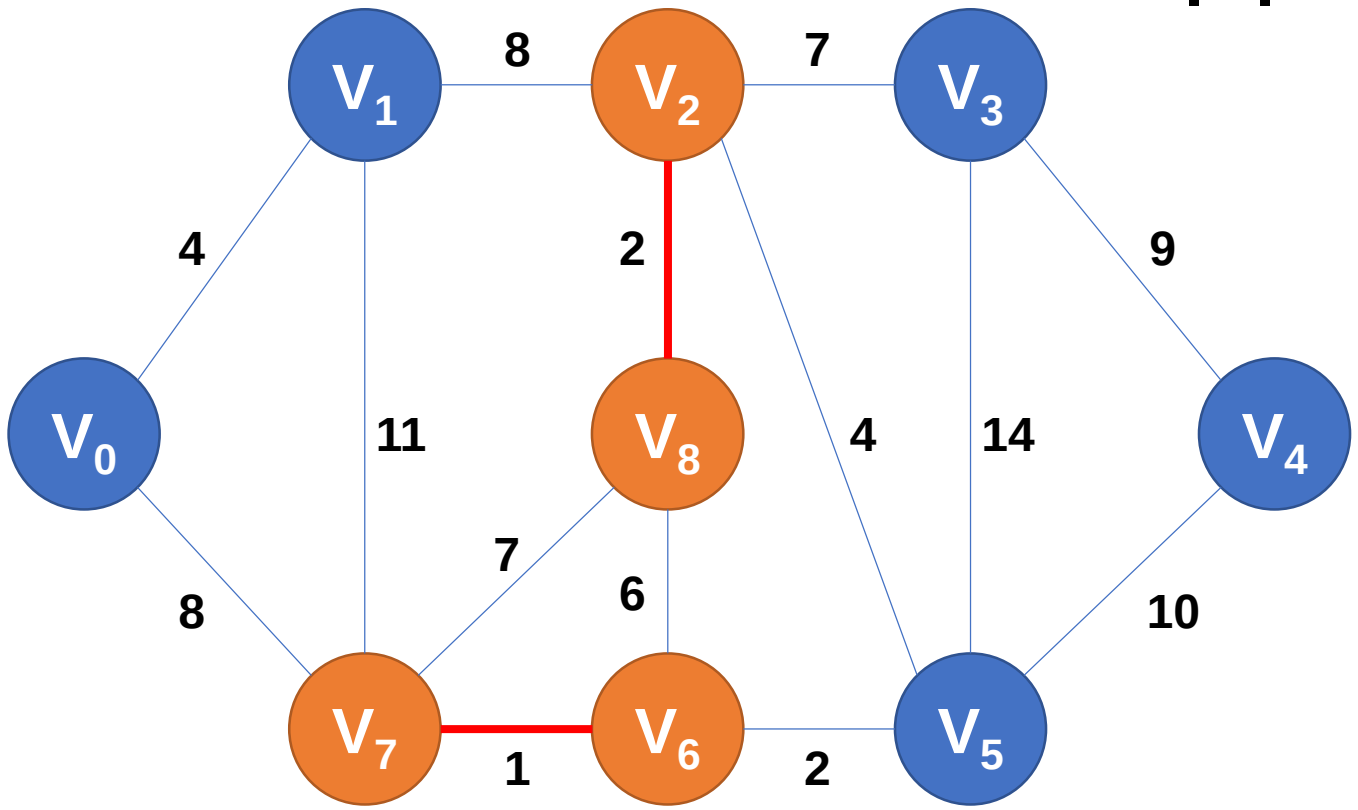
# of Edges in MST = 1

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



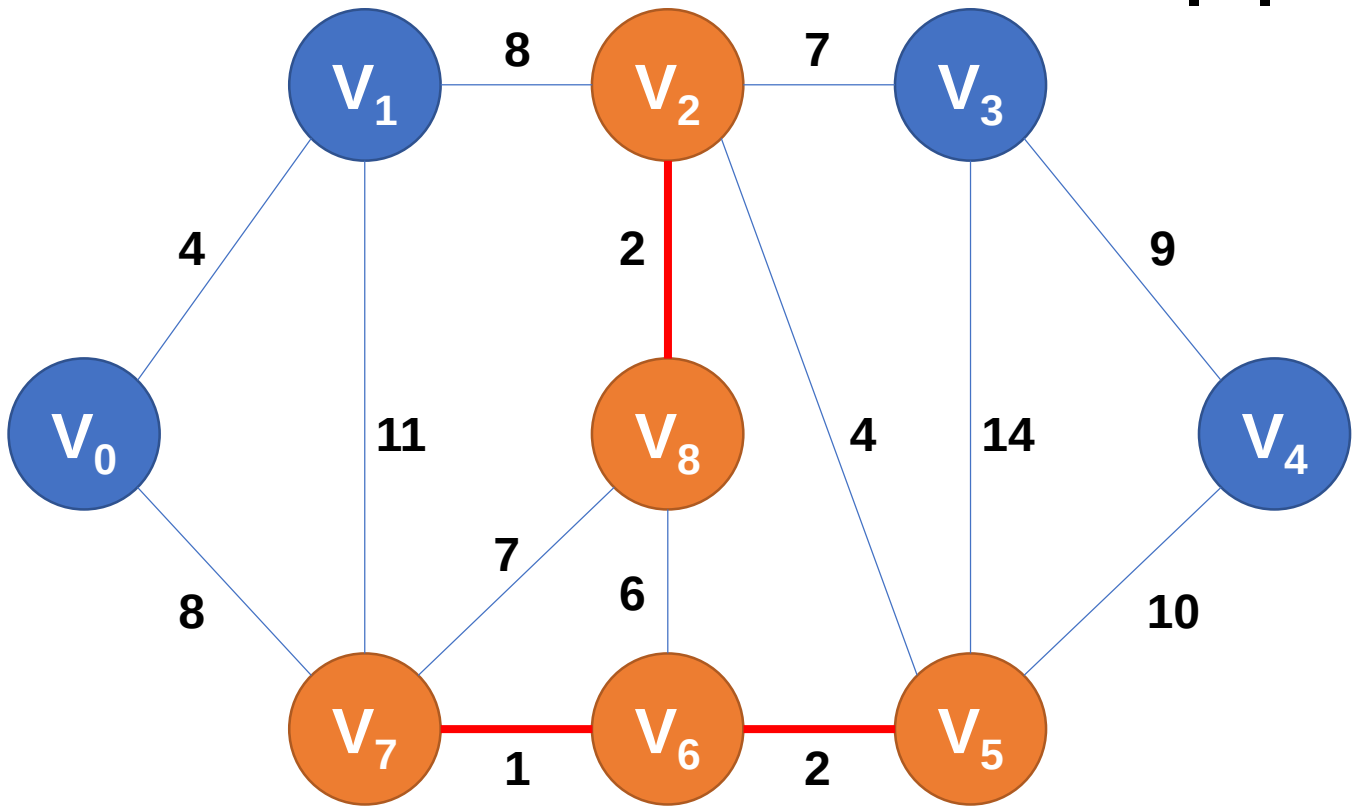
# of Edges in MST = 2

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



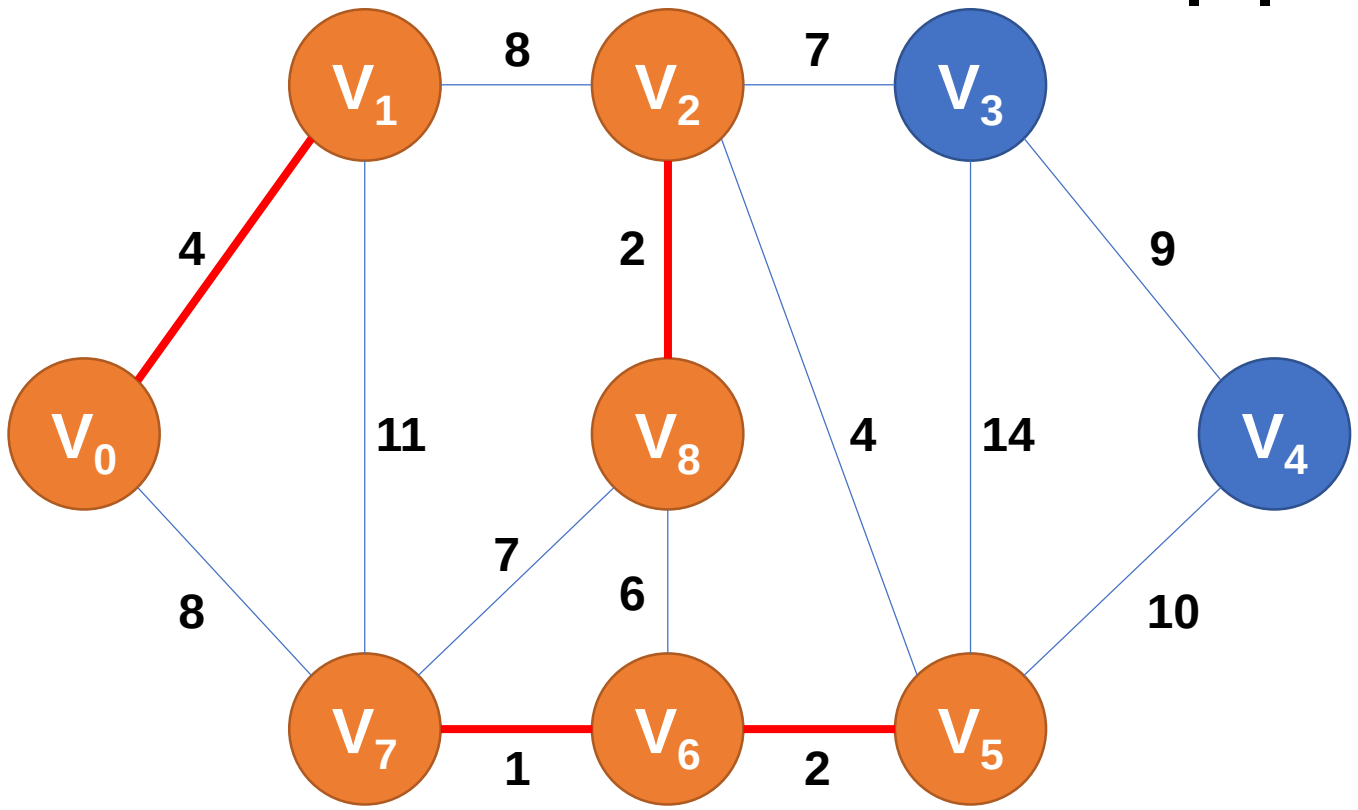
# of Edges in MST = 3

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



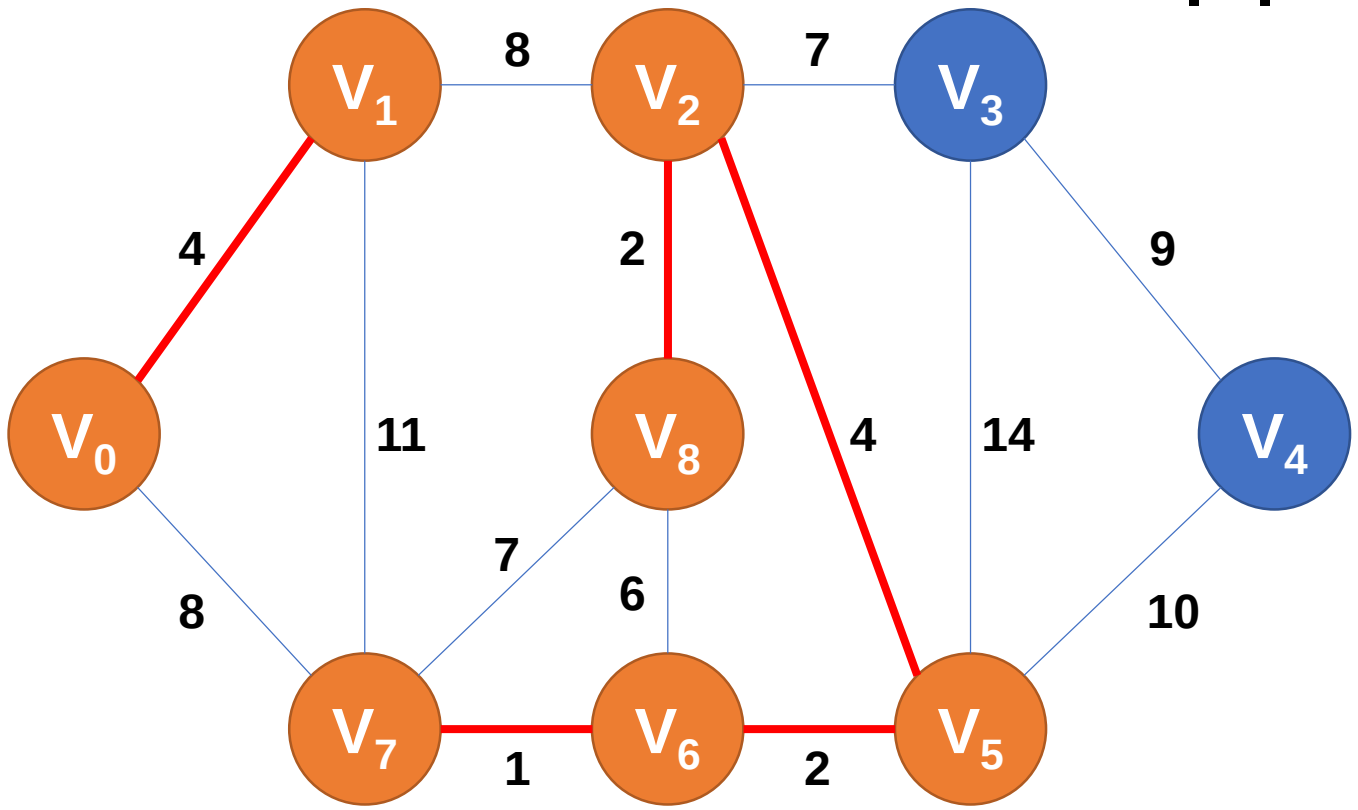
# of Edges in MST = 4

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



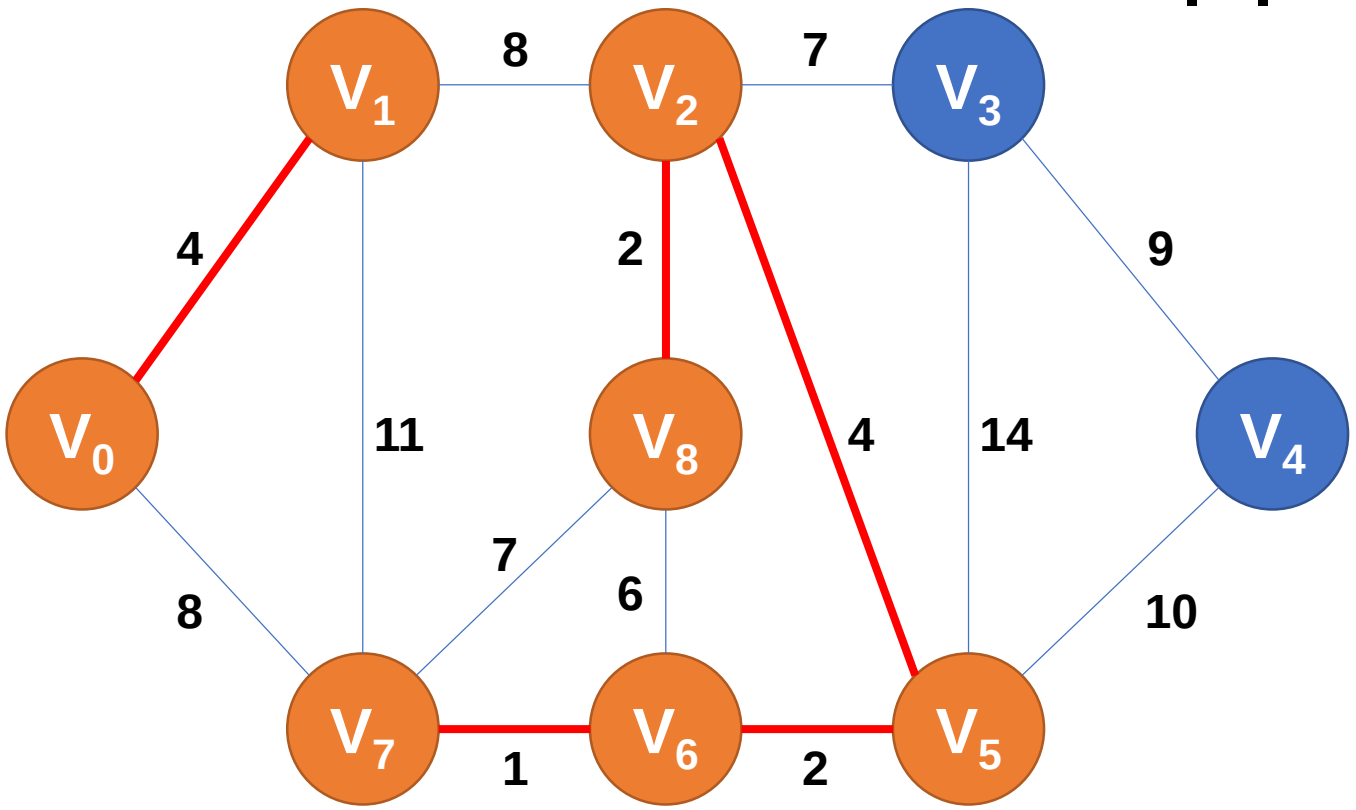
# of Edges in MST = 5

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	v_8,v_6	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	6	7	7	8	8	9	10	11	14

Examples

|V|=9



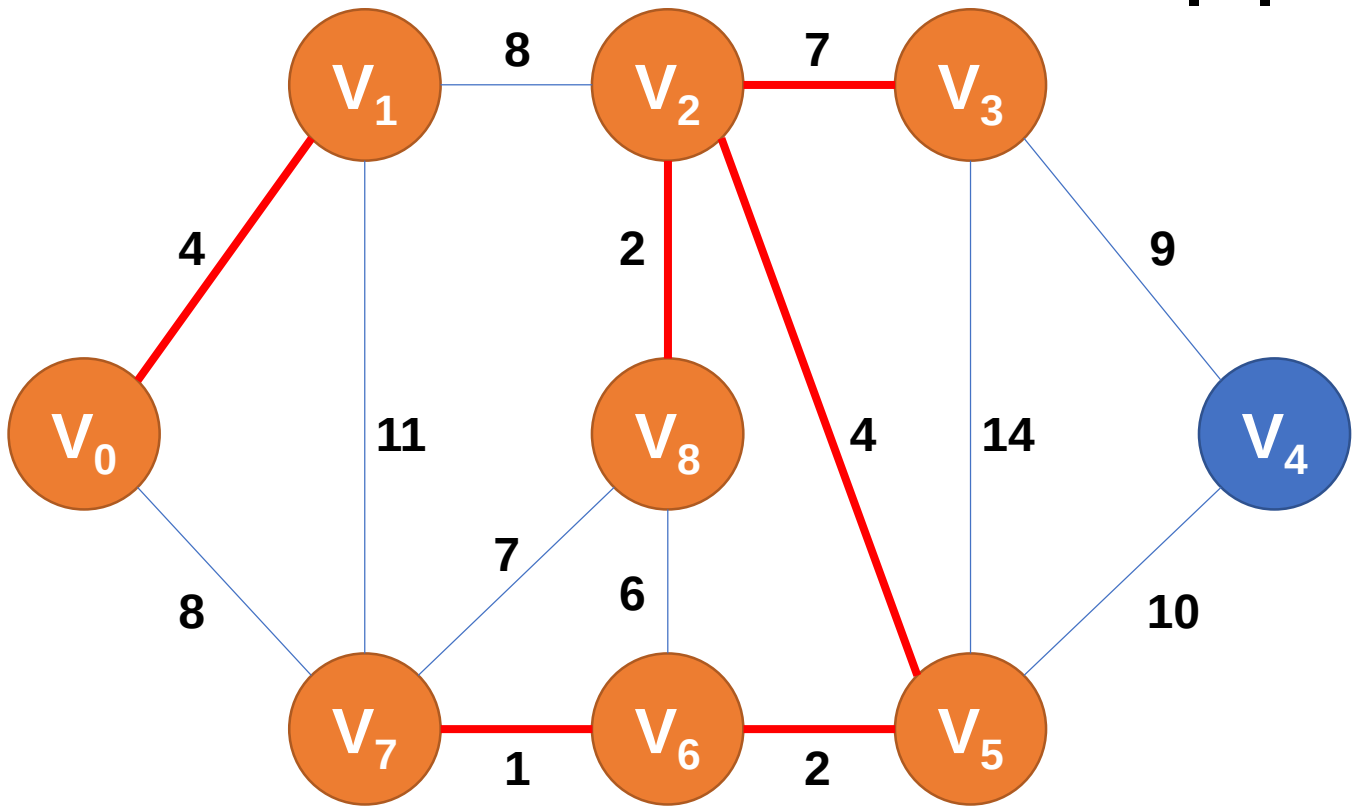
# of Edges in MST = 5

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	7	8	8	9	10	11	14

Examples

|V|=9



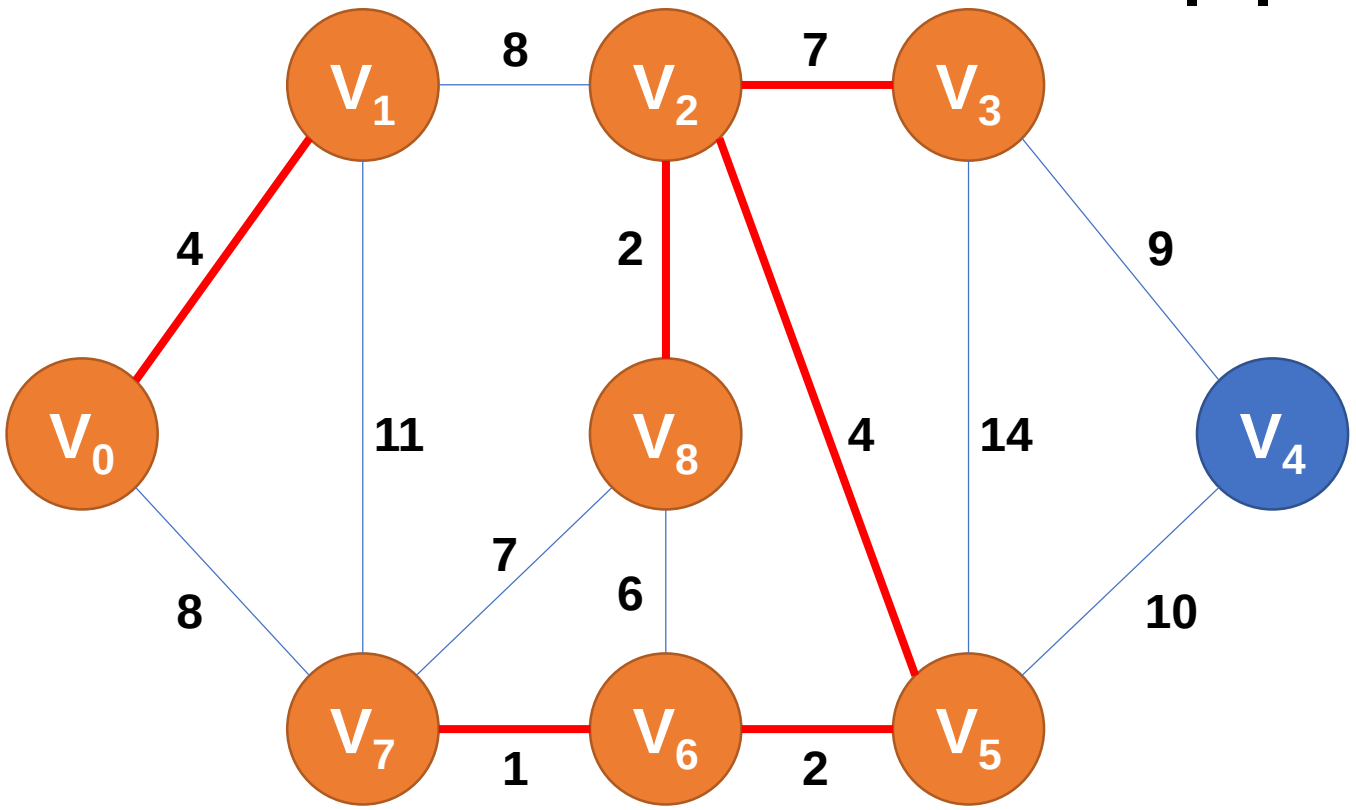
# of Edges in MST = 6

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	v_7,v_8	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	7	8	8	9	10	11	14

Examples

|V|=9



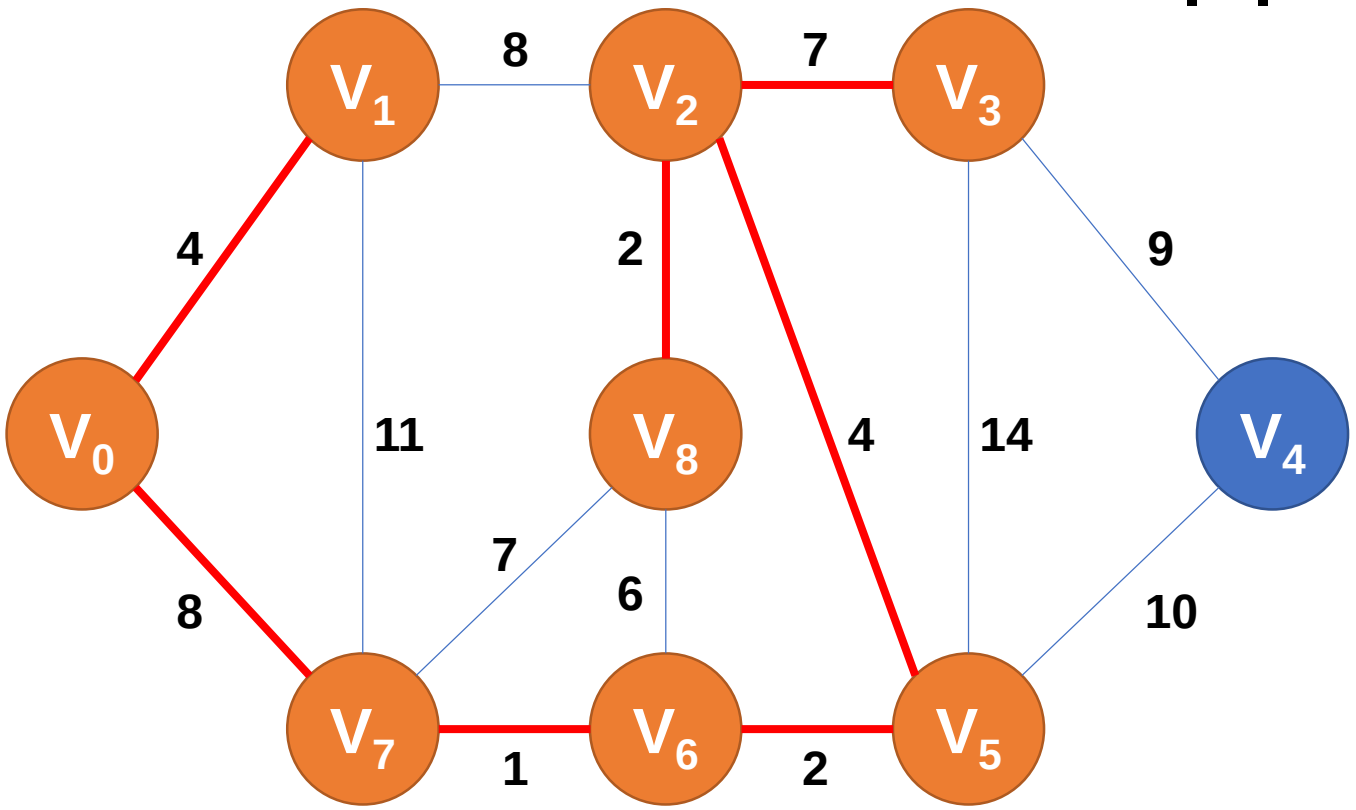
# of Edges in MST = 6

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	8	9	10	11	14

Examples

|V|=9



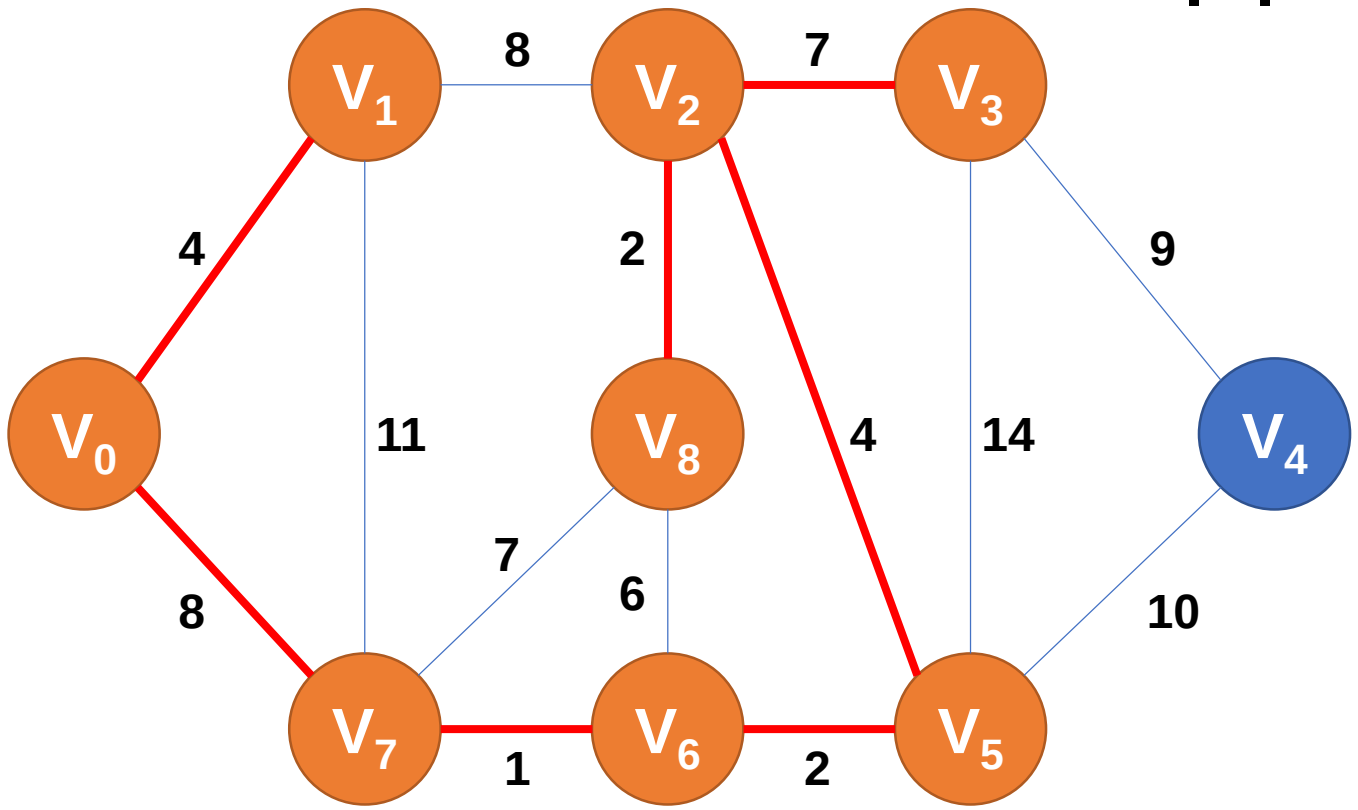
# of Edges in MST = 7

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	v_1,v_2	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	8	9	10	11	14

Examples

|V|=9



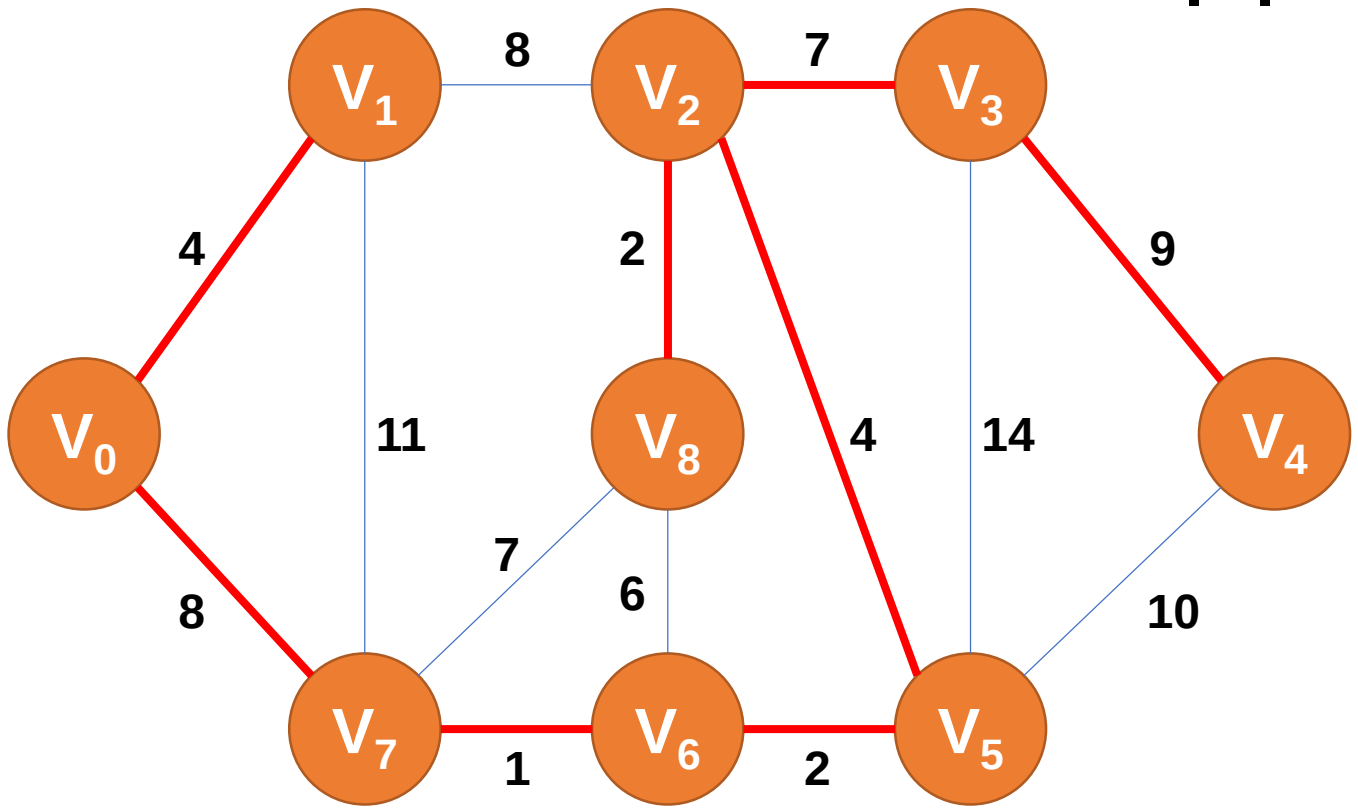
# of Edges in MST = 7

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	~~v_1,v_2~~	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

|V|=9



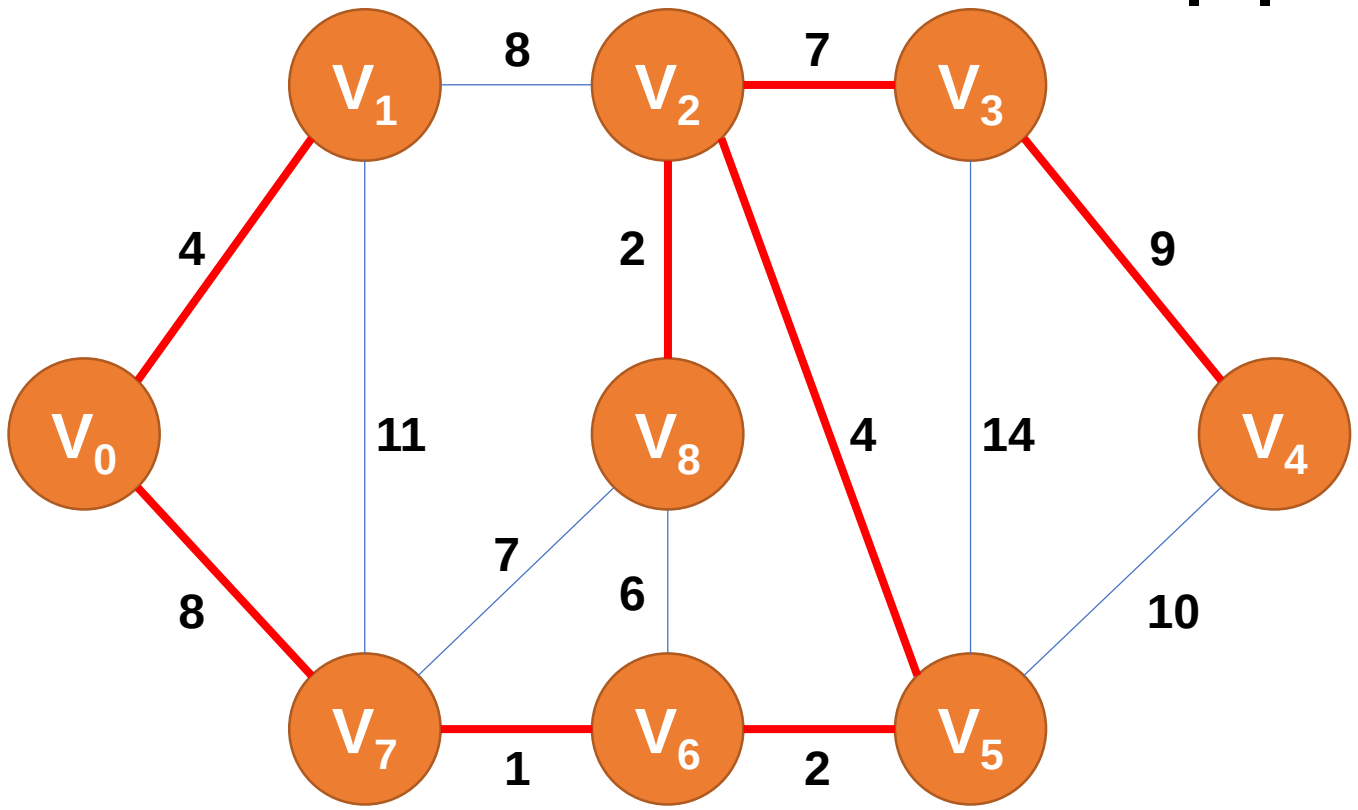
# of Edges in MST = 8

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	~~v_1,v_2~~	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

|V|=9

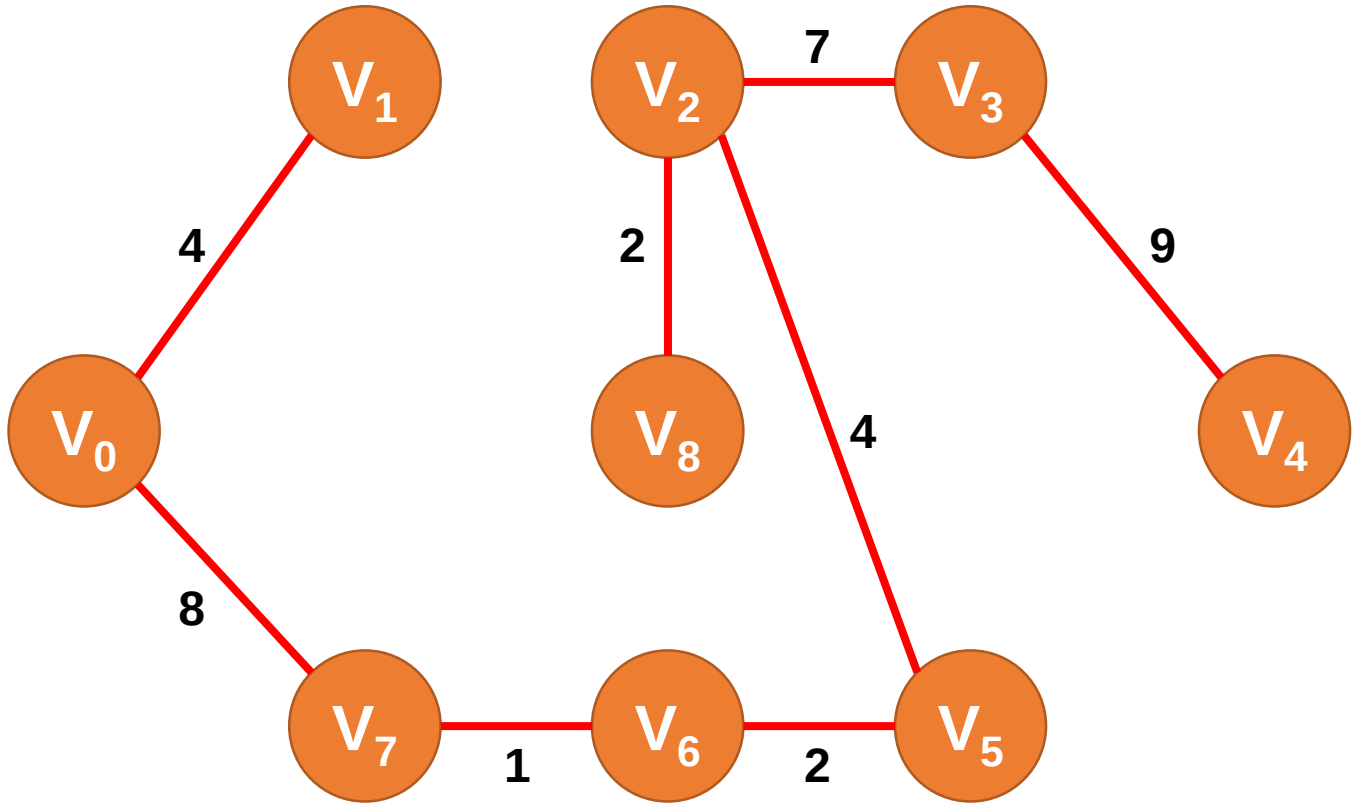


Terminate!!!

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	~~v_1,v_2~~	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples



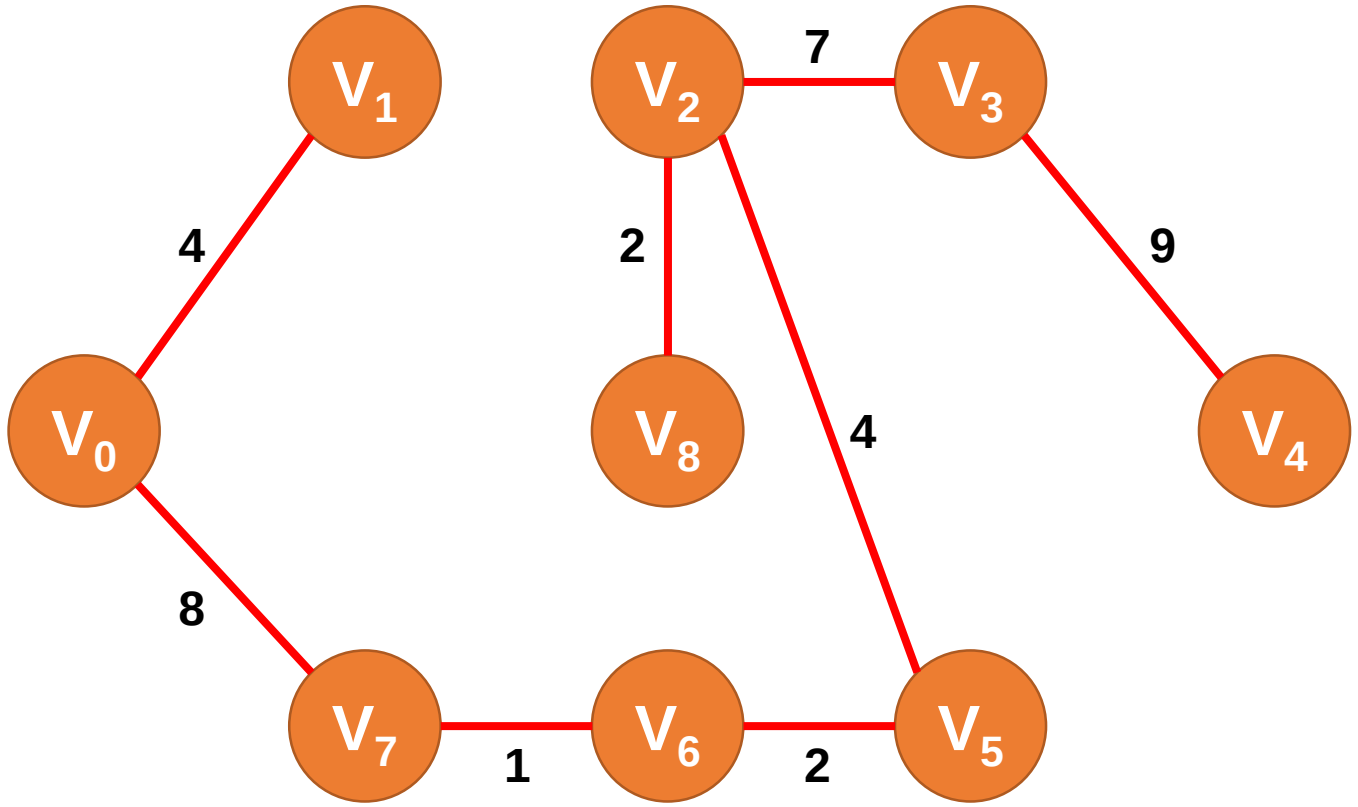
Now we have our MST!

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	~~v_1,v_2~~	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Examples

Weights of MST = 37



Now we have our MST!

1. **Sort all the edges in non-decreasing order of their weight.**
2. **Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.**
3. **Repeat step#2 until there are (V-1) edges in the spanning tree.**

v_7,v_6	v_8,v_2	v_6,v_5	v_0,v_1	v_2,v_5	~~v_8,v_6~~	v_2,v_3	~~v_7,v_8~~	v_0,v_7	~~v_1,v_2~~	v_3,v_4	v_5,v_4	v_1,v_7	v_3,v_5
1	2	2	4	4	~~6~~	7	~~7~~	8	~~8~~	9	10	11	14

Pseudocode

```
MST-KRUSKAL(G, w)
1  A = ∅                                                          O(1)
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w     O(E log E)
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)                               O(E log V)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

Overall Complexity = O(E Log E + E Log V)

→ V-1 ≤ E ≤ $(V^2-V)/2$ → O(Log E) ≈ O(Log V)

→ Therefore, time complexity is O(E Log E) or O(E Log V)

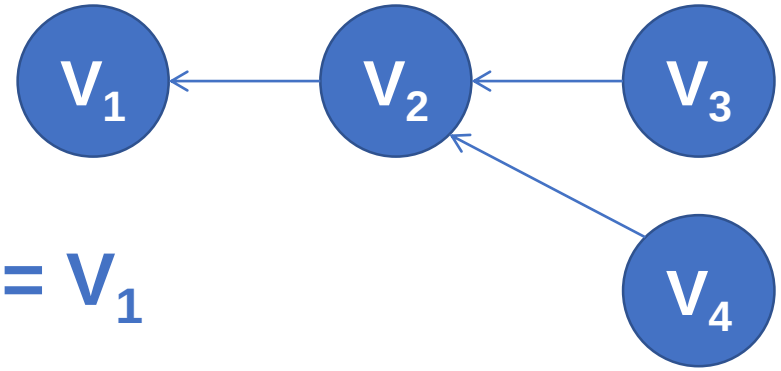
Pseudocode

MAKE-SET: Create a node whose parent is itself, a root

→ MAKE-SET(v): O(1)

FIND-SET(i): Follow the pointer from i to the root of its tree

→ FIND-SET(u): O(h), where h is the height of the tree.

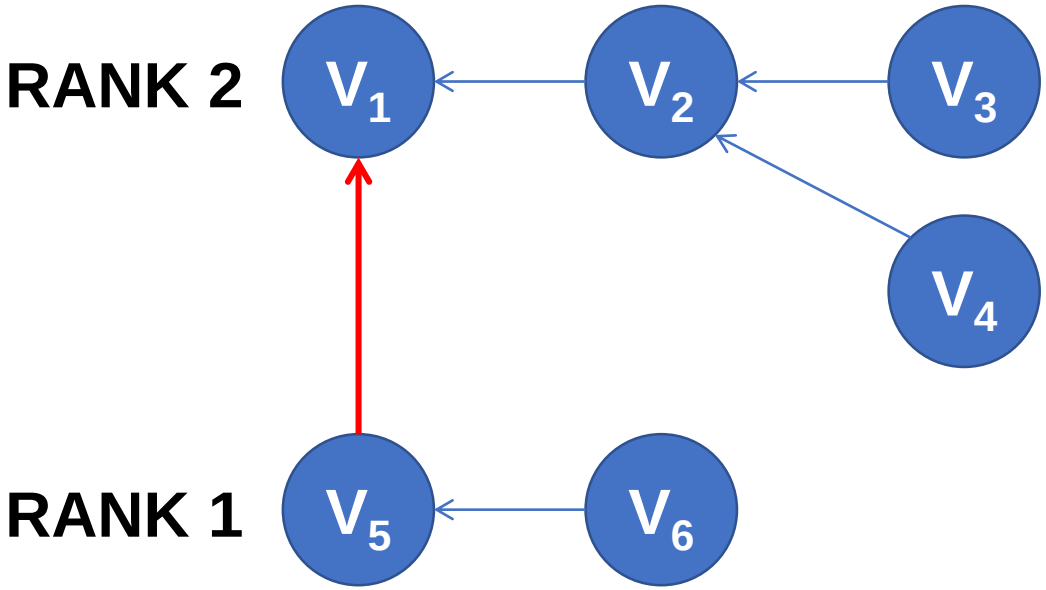


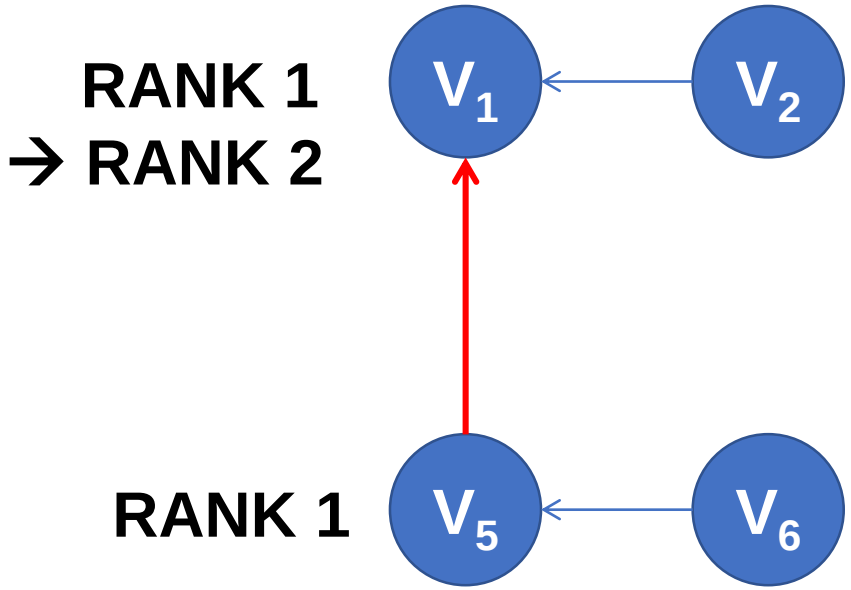
FIND-SET(V_3) = V_1

Pseudocode

UNION(x,y): If rank of x > rank of y, make y point to x. If rank of x = rank of y, make y point to x and increase rank of x.

→ UNION(x,y): O(1)





Implementation (1)

```
// a structure to represent a weighted edge in graph
class Edge
{
    public:
    int src, dest, weight;
};

// a structure to represent a connected, undirected
// and weighted graph
class Graph
{
    public:
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges.
    // Since the graph is undirected, the edge
    // from src to dest is also edge from dest
    // to src. Both are counted as 1 edge here.
    Edge* edge;
};

// Creates a graph with V vertices and E edges
Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;

    graph->edge = new Edge[E];

    return graph;
}
```

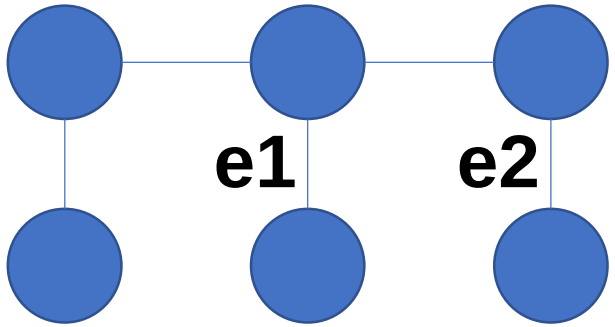
```
// A structure to represent a subset for union-find
class subset
{
    public:
    int parent;
    int rank;
};

// A utility function to find set of an element i
// (uses path compression technique)
int find(subset subsets[], int i)
{
    // find root and make root as parent of i
    // (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high
    // rank tree (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and
    // increment its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

Implementation (2)

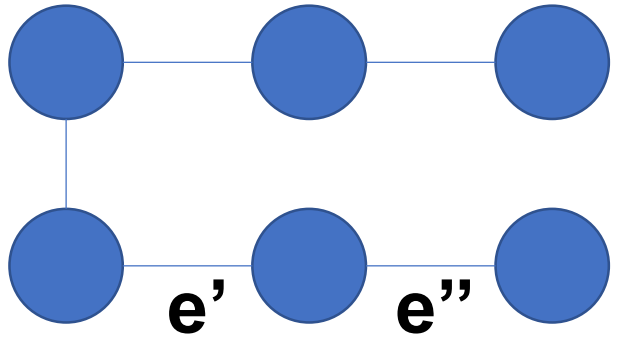
```
// Compare two edges according to their weights.
// Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST(Graph* graph)
{
    int V = graph->V;
    Edge result[V]; // Tnis will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges

    // Step 1: Sort all the edges in non-decreasing
    // order of their weight. If we are not allowed to
    // change the given graph, we can create a copy of
    // array of edges
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    // Allocate memory for creating V ssubsets
    subset *subsets = new subset[( V * sizeof(subset) )];

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Number of edges to be taken is equal to V-1
    while (e < V - 1 && i < graph->E)
    {
        // Step 2: Pick the smallest edge. And increment
        // the index for next iteration
        Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // If including this edge does't cause cycle,
        // include it in result and increment the index
        // of result for next edge
        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
        // Else discard the next_edge
    }

    // print the contents of result[] to display the
    // built MST
    cout<<"Following are the edges in the constructed MST\n";
    for (i = 0; i < e; ++i)
        cout<<result[i].src<<" -- "<<result[i].dest<<" == "<<result[i].weight<<endl;
    return;
}
```

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



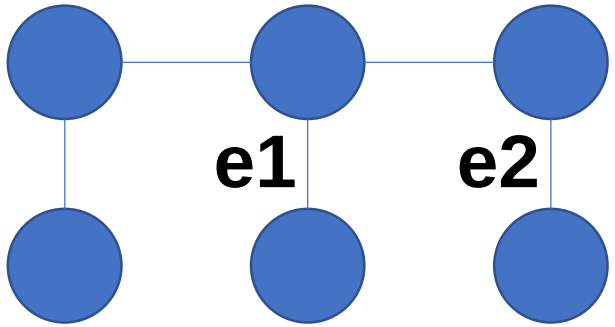
Hypothetical MST: T'



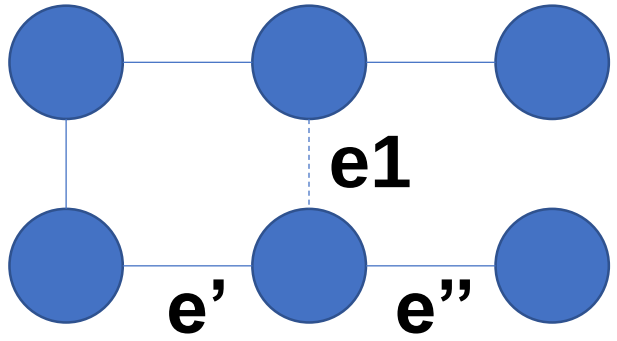
Assume T is not a MST and T' is a MST.
Then, weight(T) > weight(T')

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



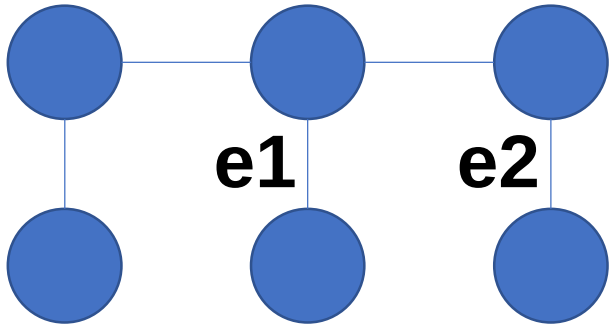
Hypothetical MST: T'



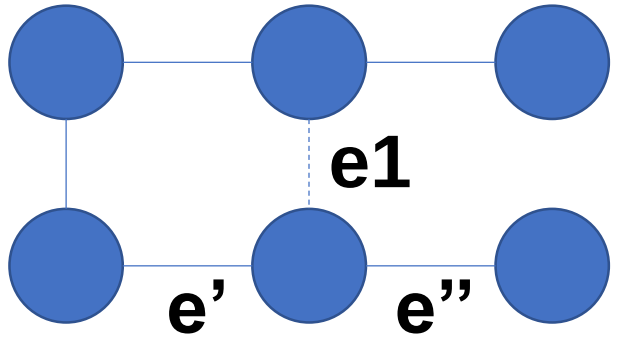
If weight(e1) > weight(e'), we have to select e' instead of e1 for Kruskal's algorithm.
However, since we selected e1, weight(e1) <= weight(e').

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



Hypothetical MST: T'

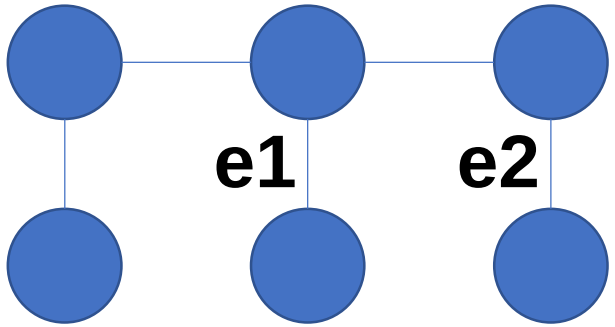


weight(e1) <= weight(e')

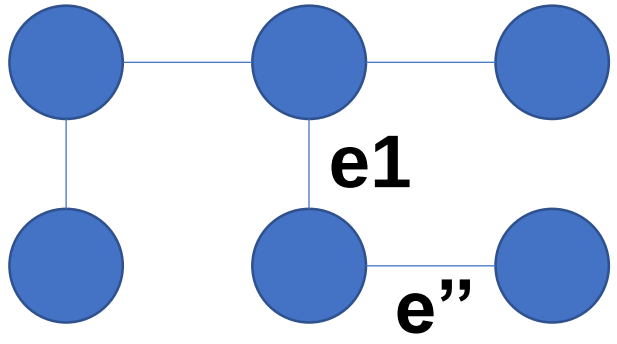
weight(T') – weight(e') + weight(e1) <= weight(T')
→ We should select e1 instead of e' for T'.

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm

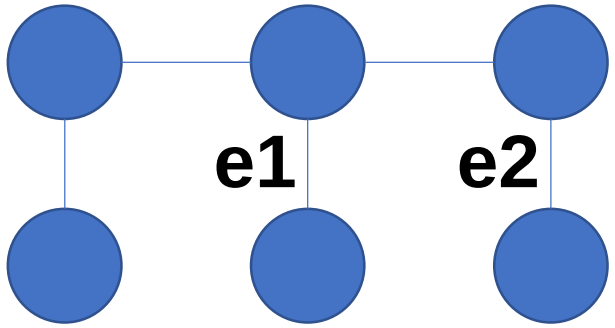


Hypothetical MST: T'

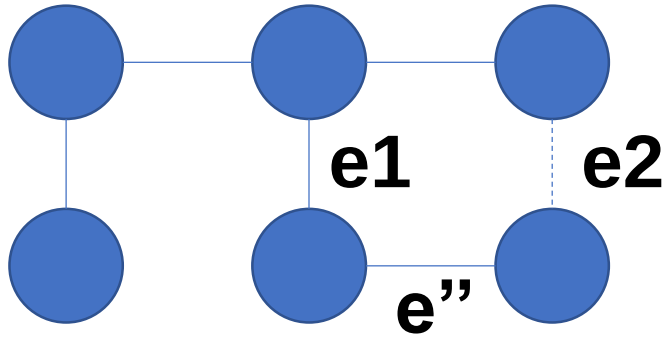


Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



Hypothetical MST: T'

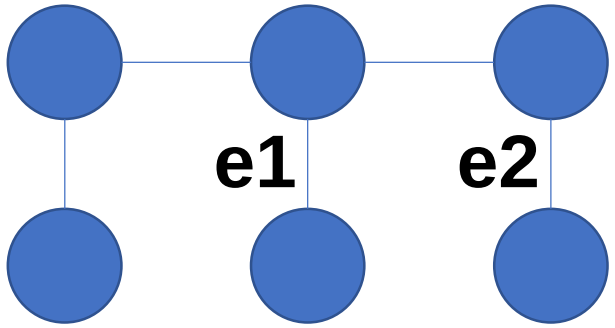


weight(e2) <= weight(e'')

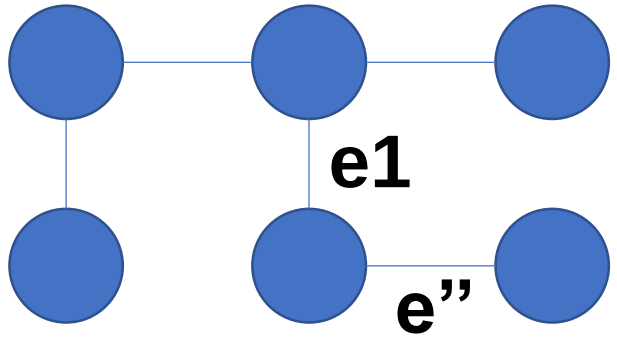
With every edge change and the corresponding Reduction in the edge difference, weight(T') only reduces further and eventually as T' becomes the same as T, both have the same Total weight. weight(T') >= weight(T).

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



Hypothetical MST: T'

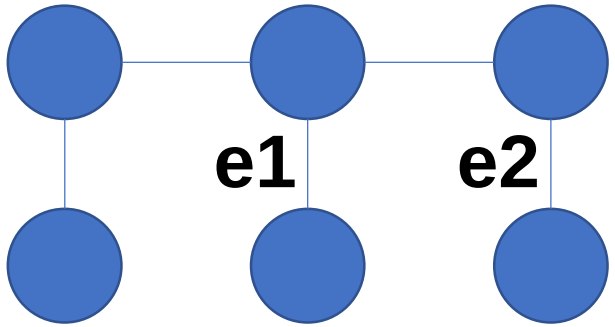


weight(e2) <= weight(e'')

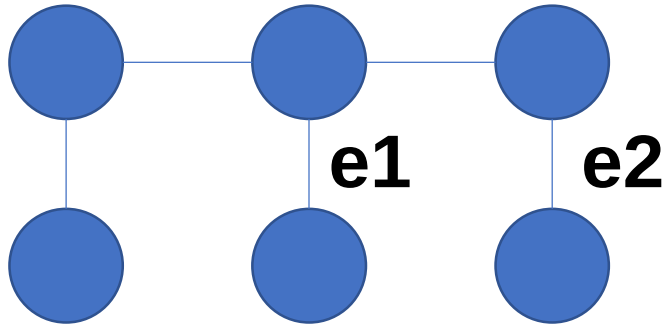
With every edge change and the corresponding Reduction in the edge difference, weight(T') only reduces further and eventually as T' becomes the same as T, both have the same Total weight. weight(T') >= weight(T).

Kruskal's algorithm Proof Correctness

Tree (T): Determined using Kruskal's algorithm



Hypothetical MST: T'



T'=T

Prim’s Algorithm

Prim’s Algorithm (1)

1. Create a set mstSet that keeps track of vertices already included in MST.

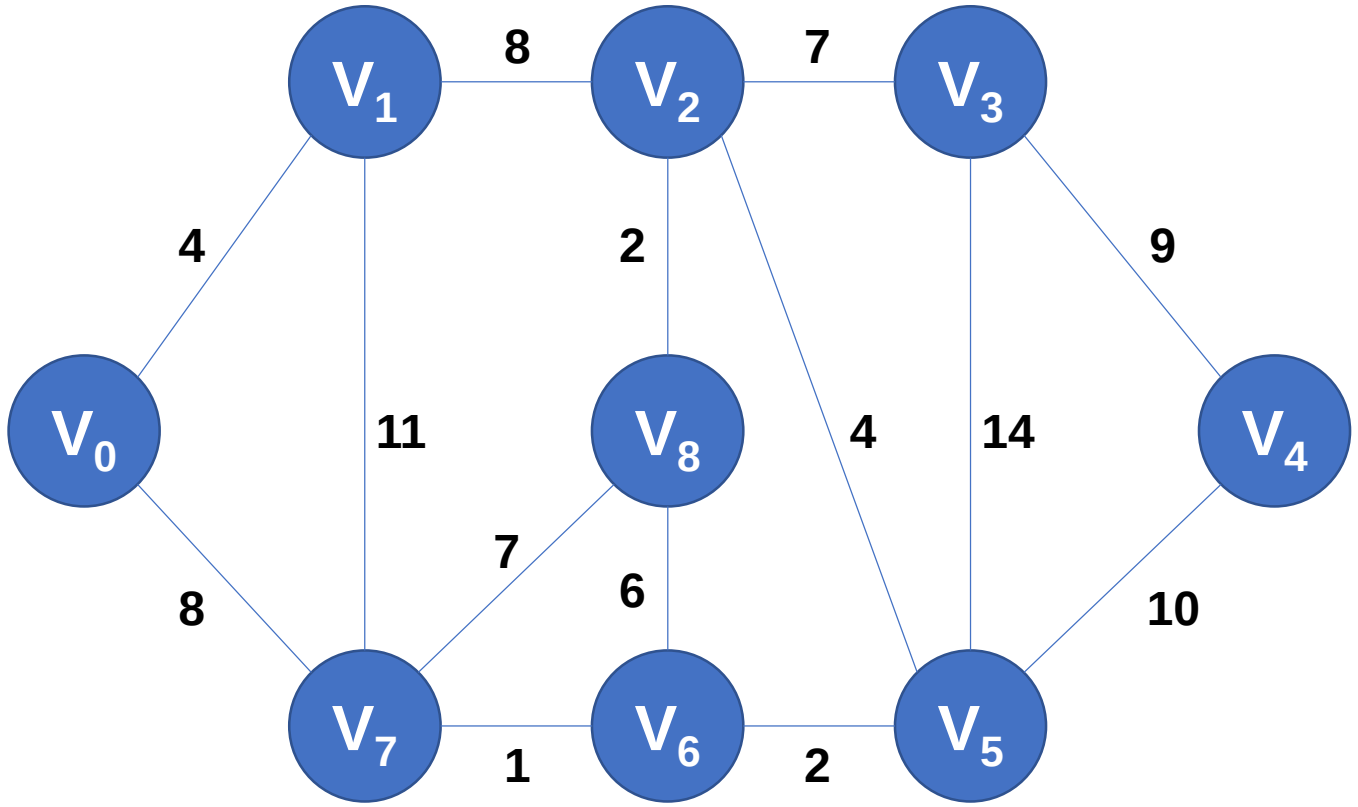
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

Prim's Algorithm (2)

3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

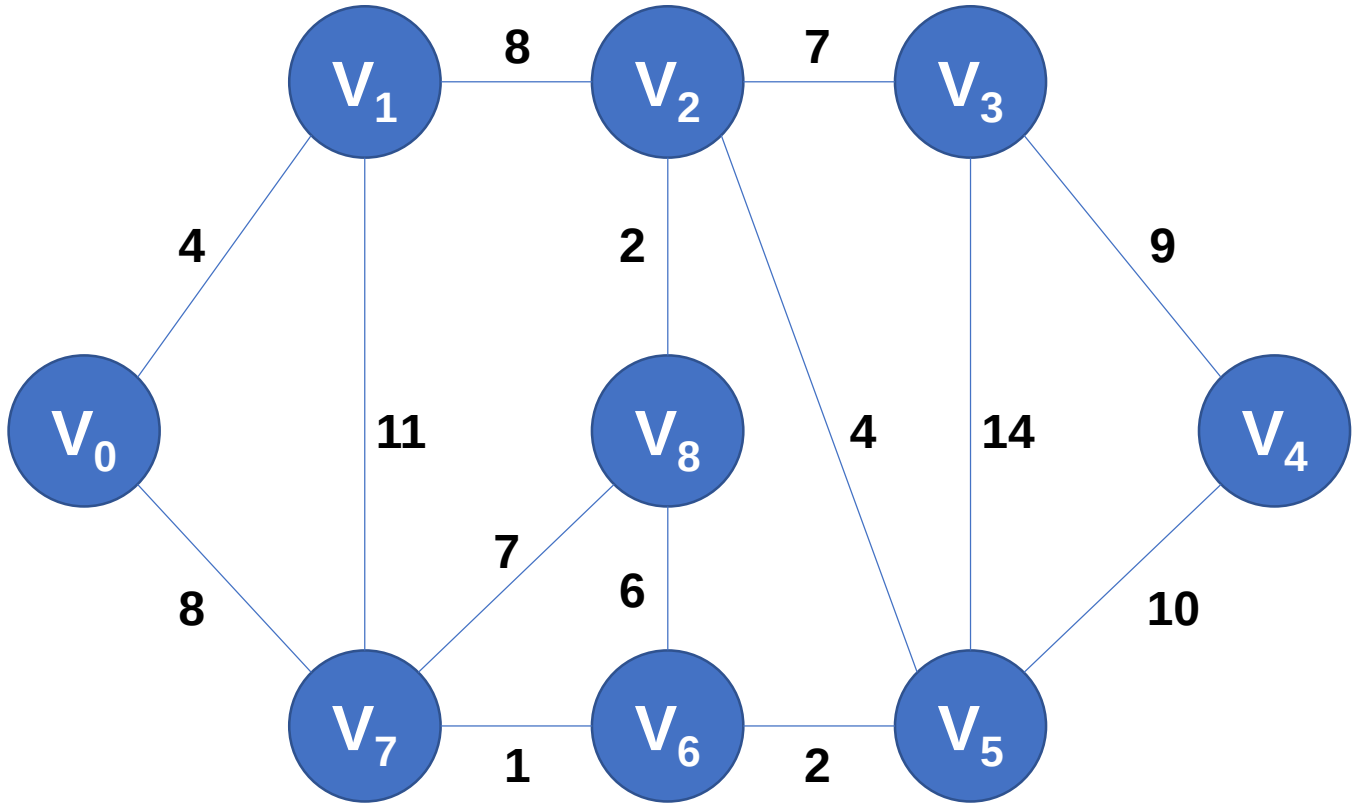
Examples



1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

mstSet={}

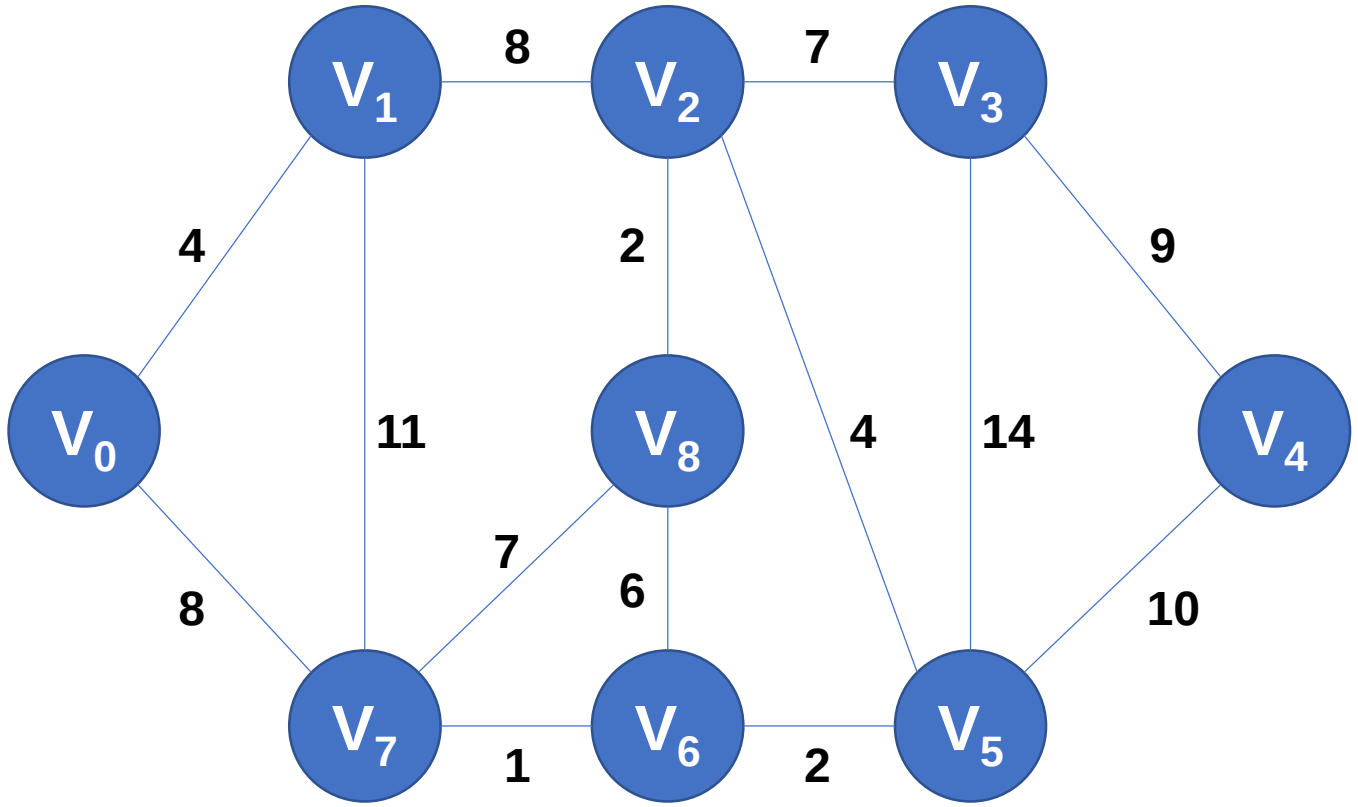
Examples



1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={}

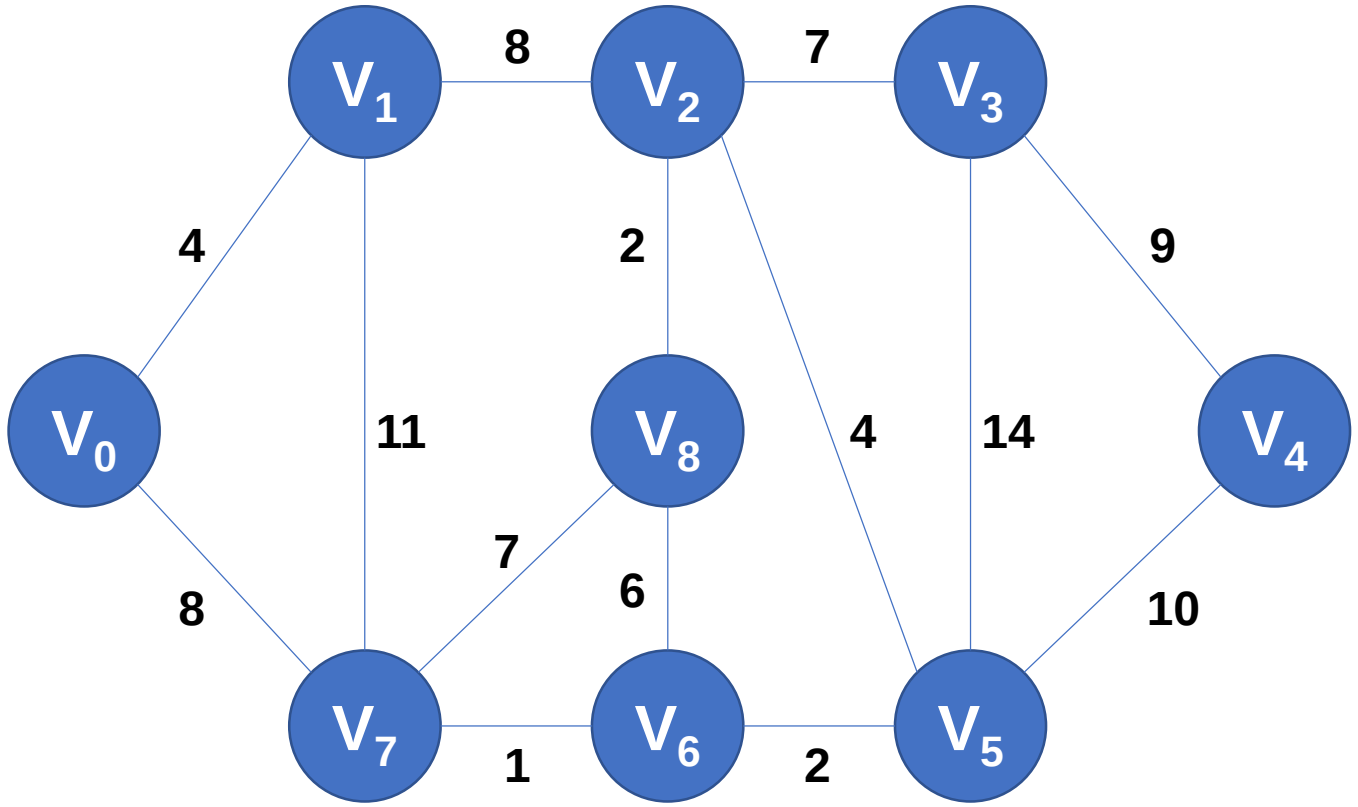


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={}

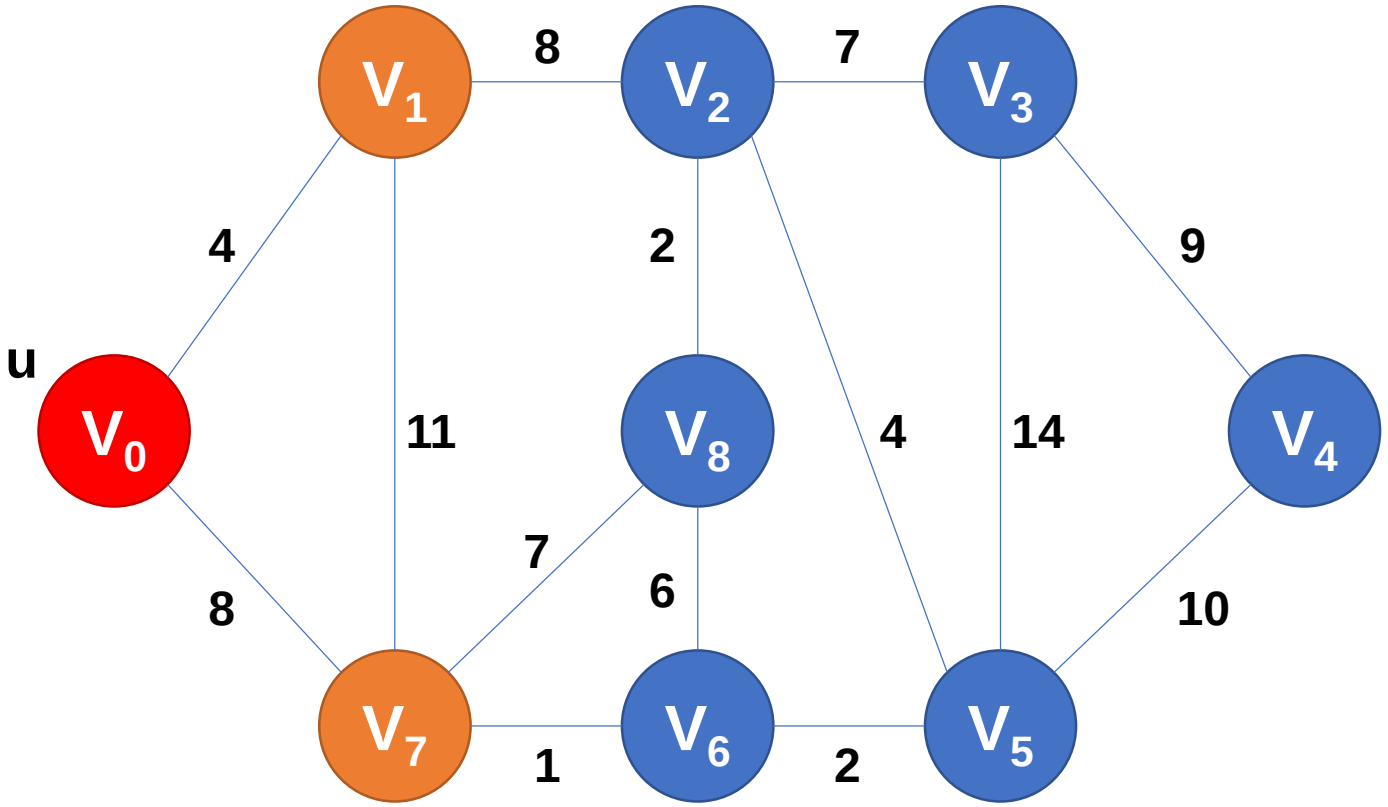


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0}



u

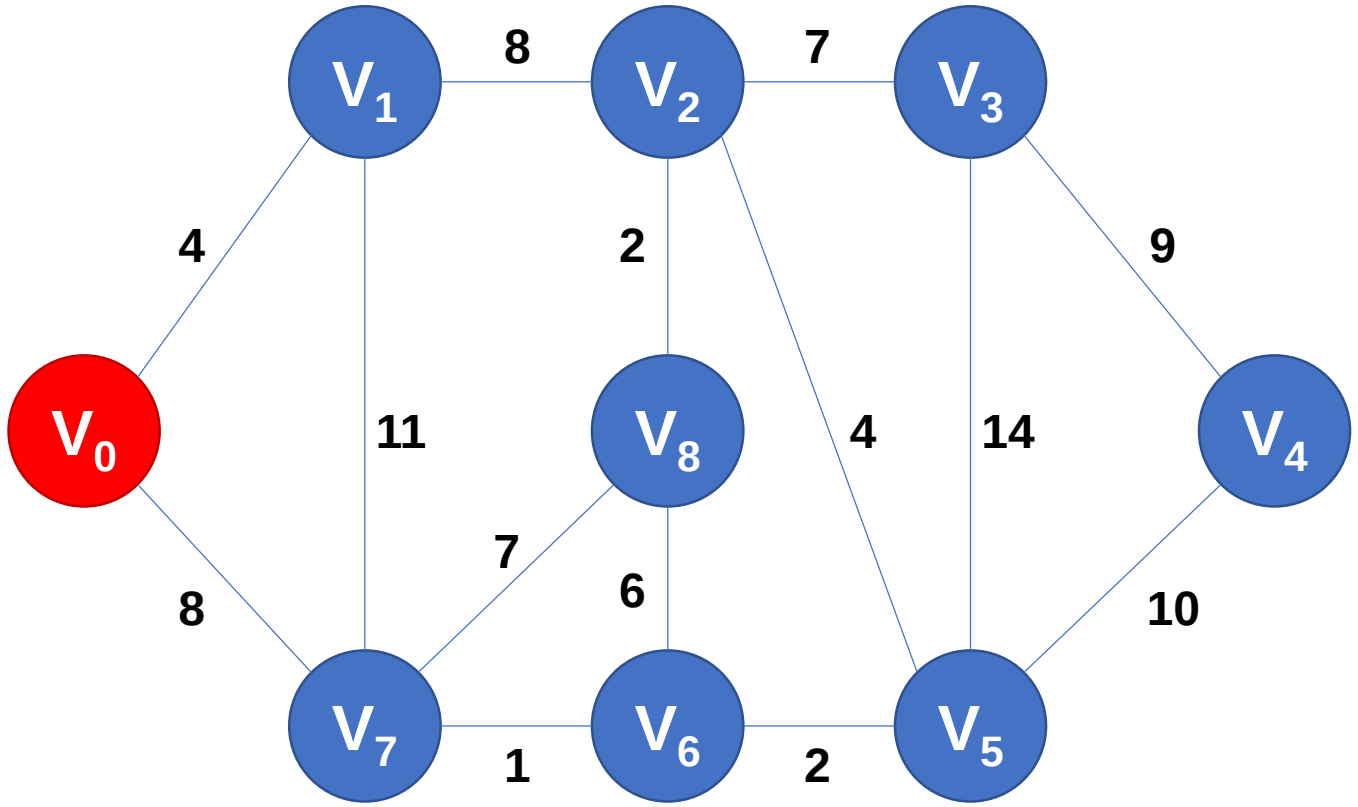
If w(u,v)<v.key, v.key=w(u,v)

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0}

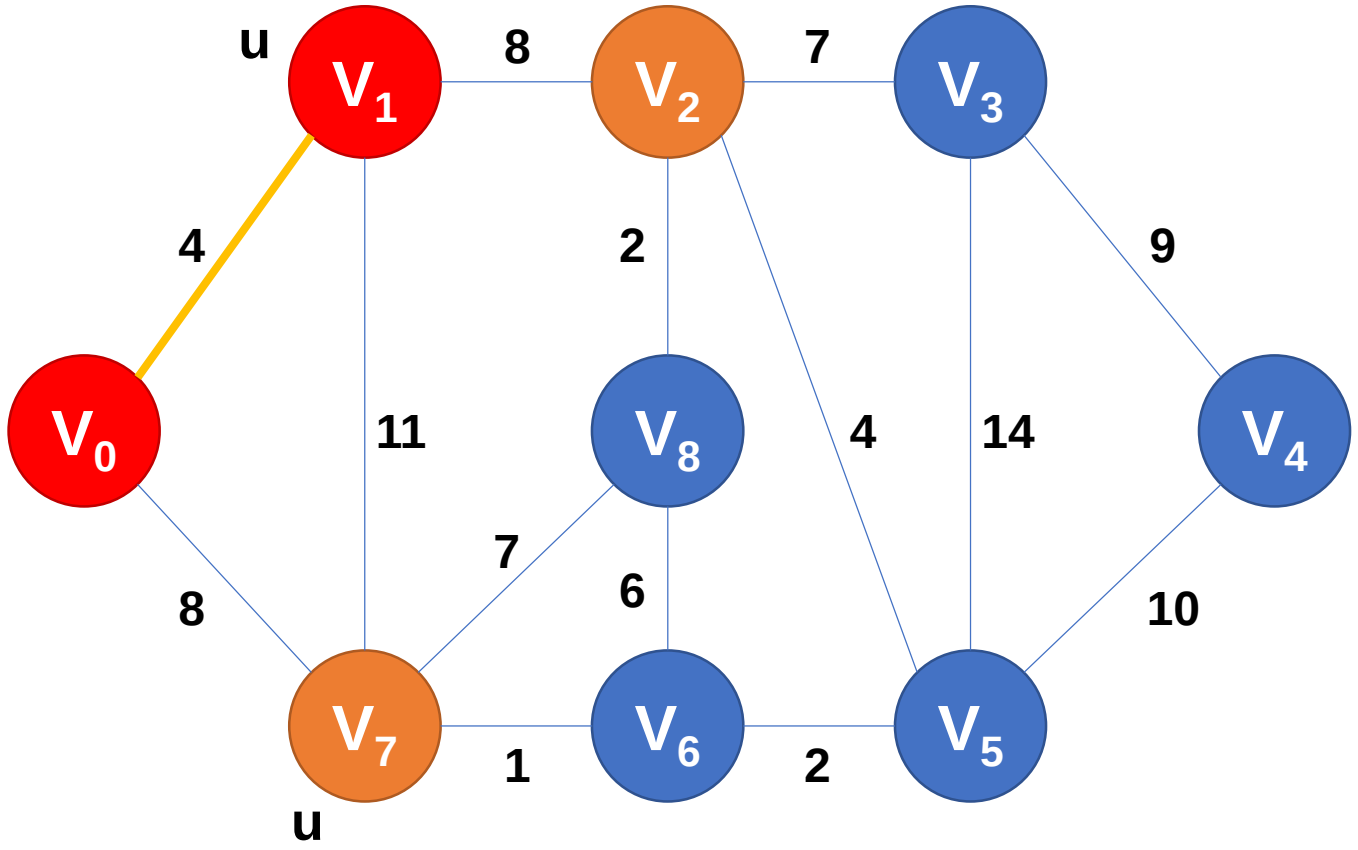


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1}

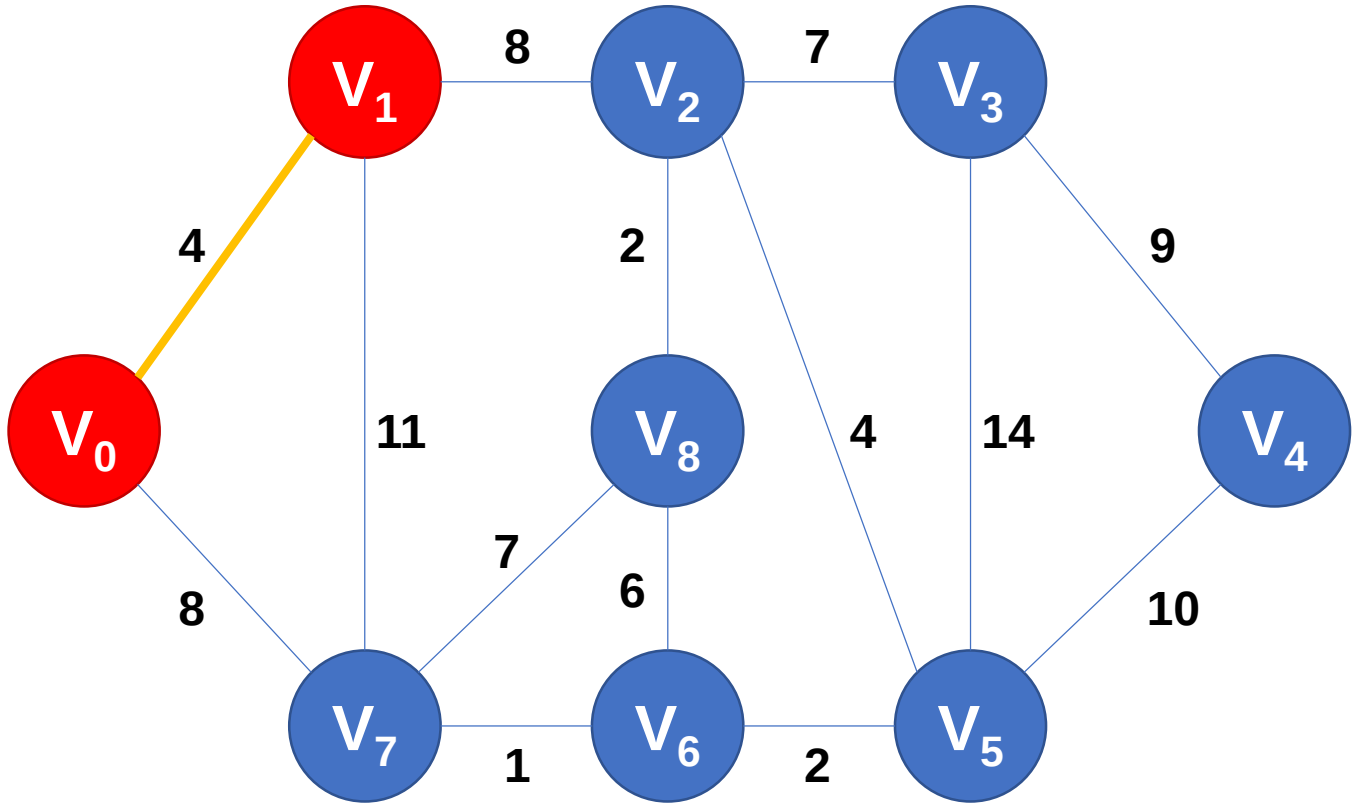


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1}

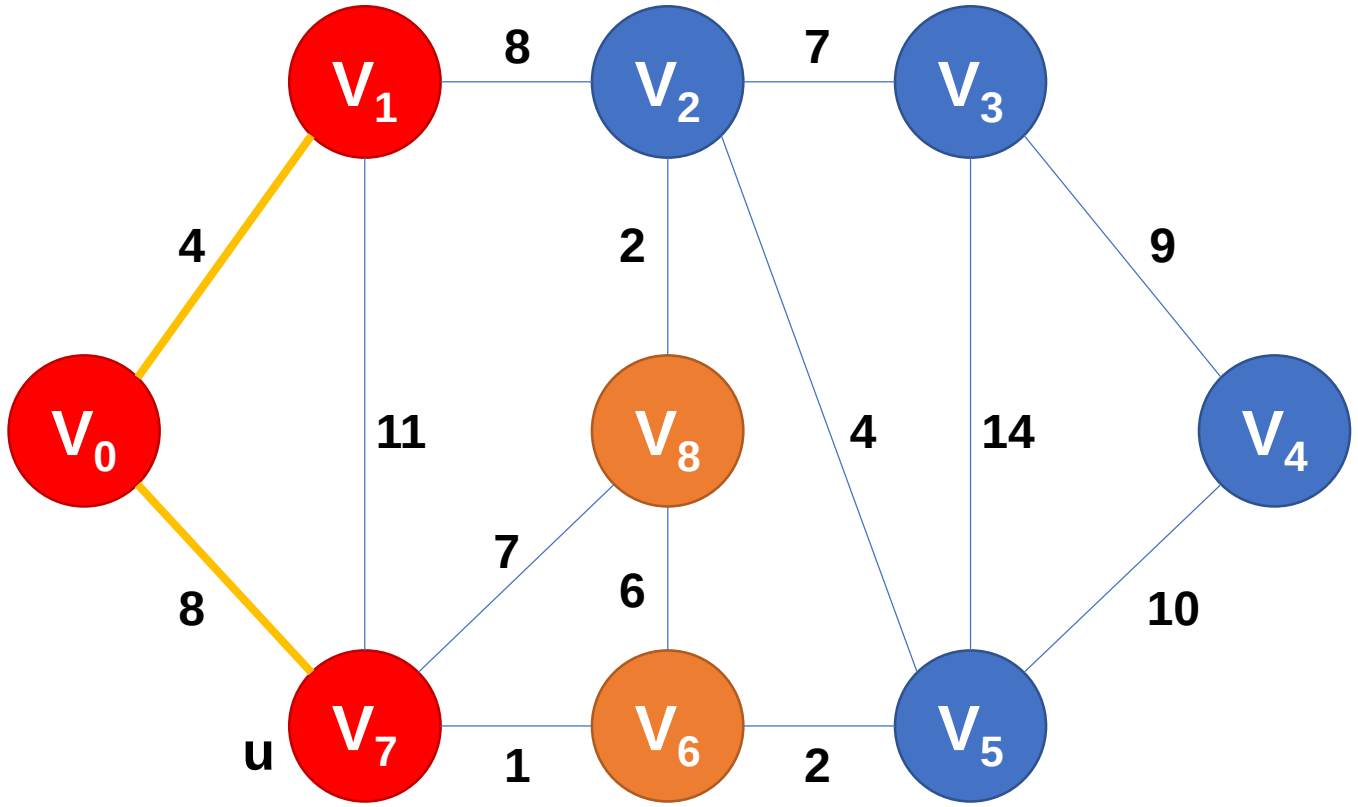


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1, v_7}



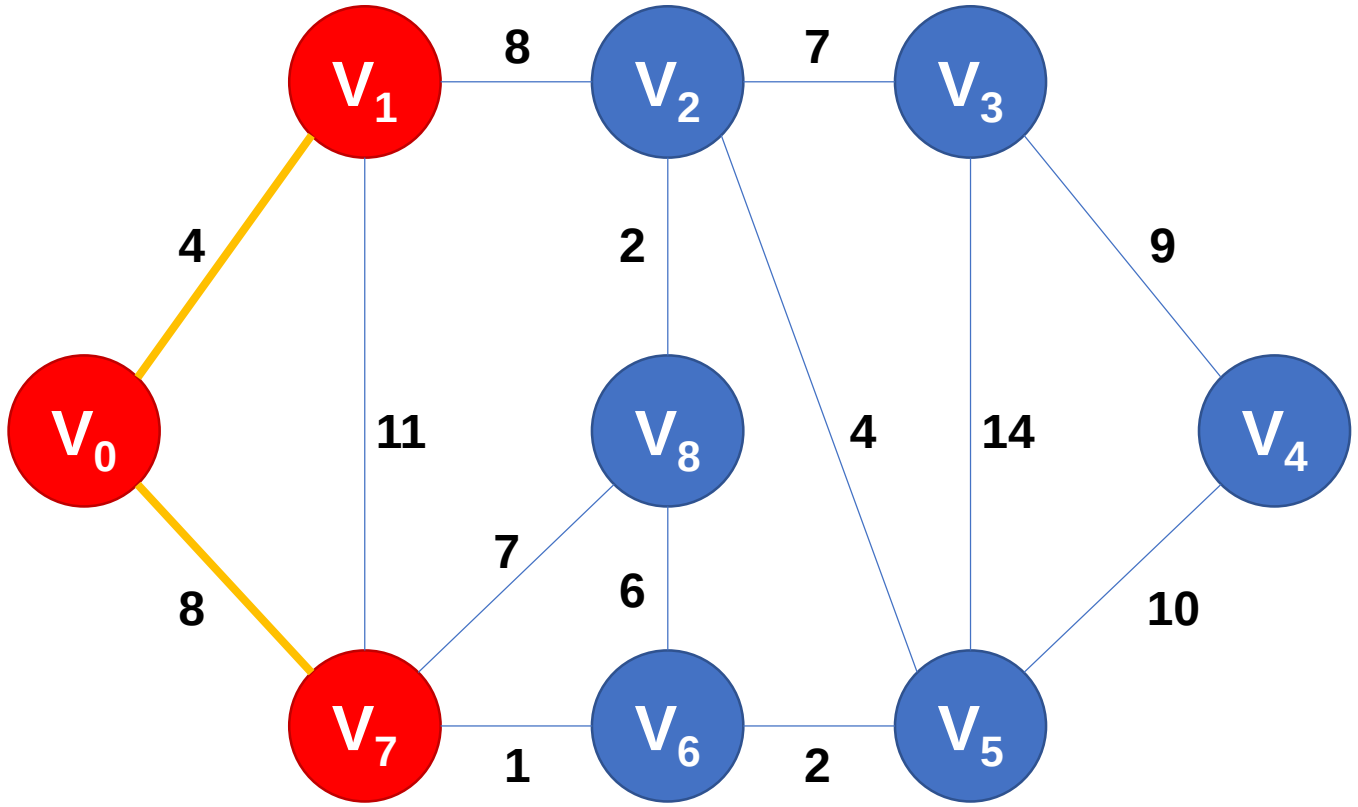
u

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1, v_7}

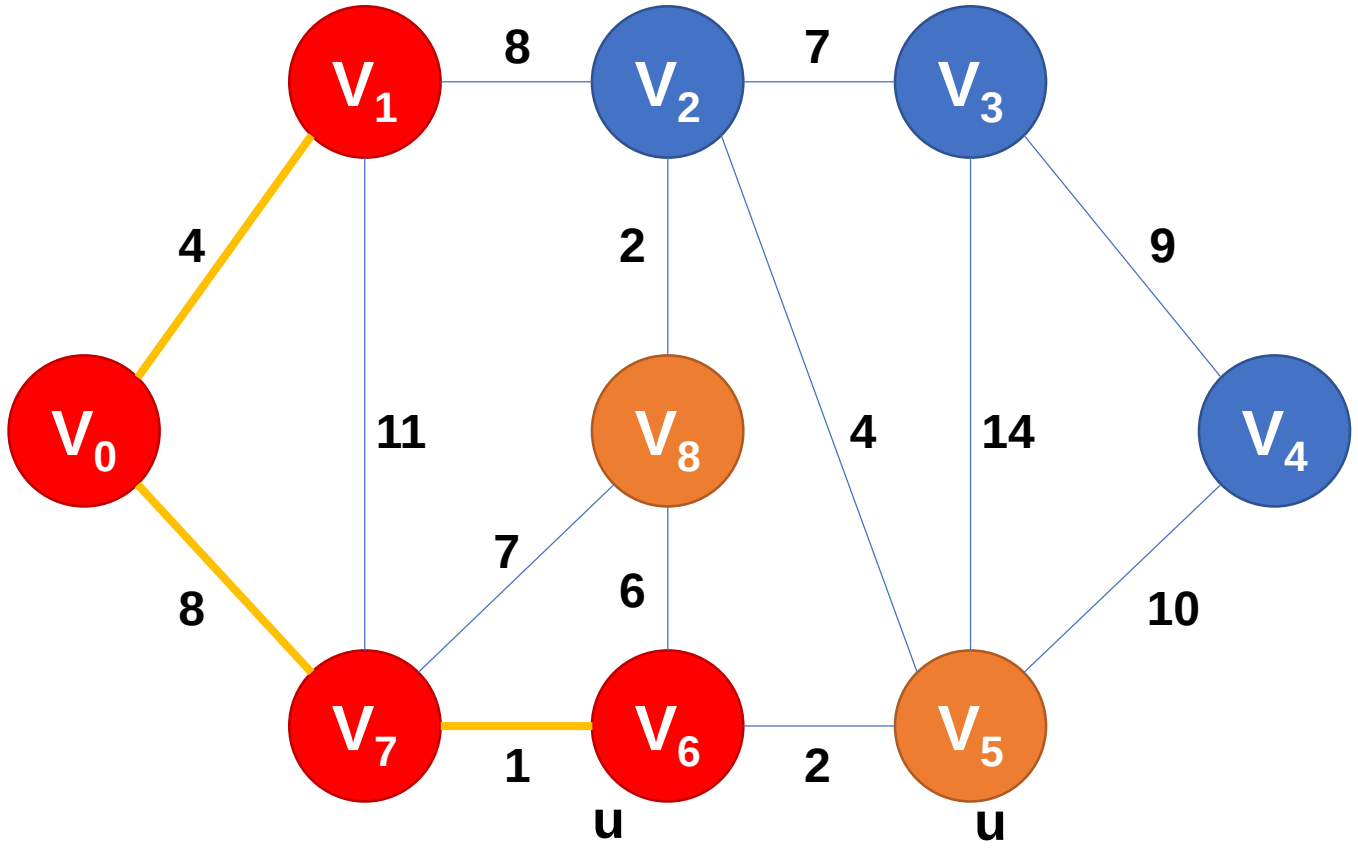


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	1	8	7

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1, v_7, v_6}

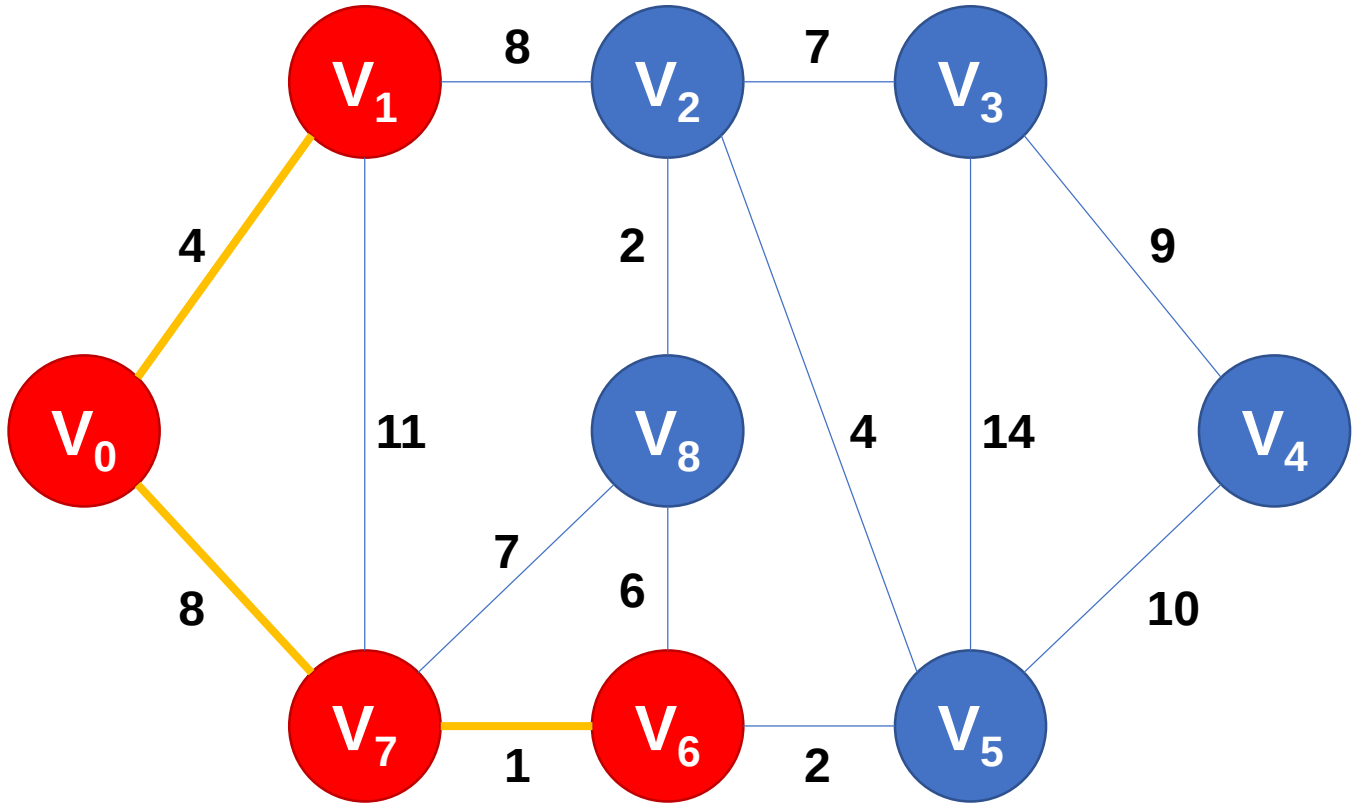


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	∞	1	8	7

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1, v_7, v_6}

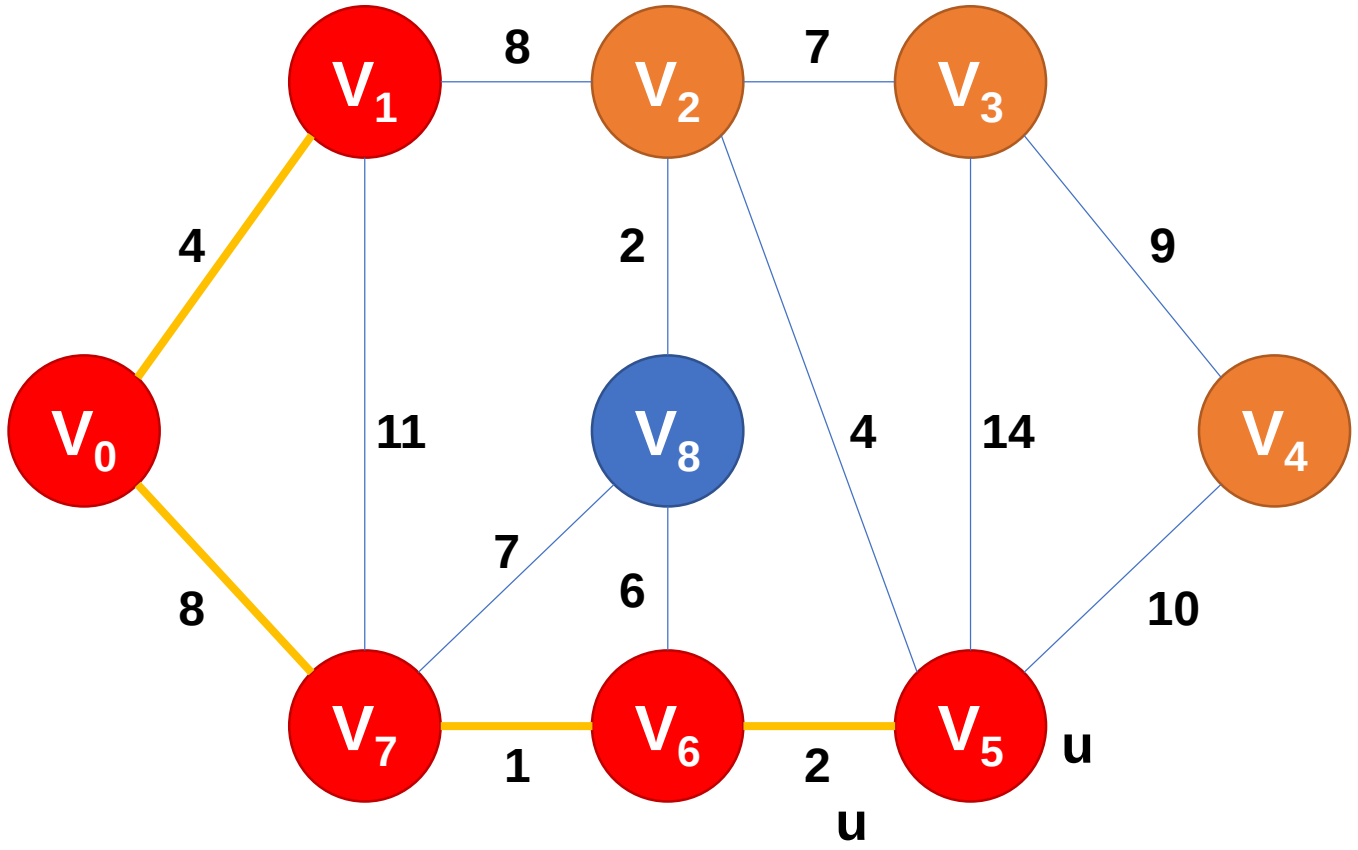


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

$mstSet=\{v_0, v_1, v_7, v_6, v_5\}$

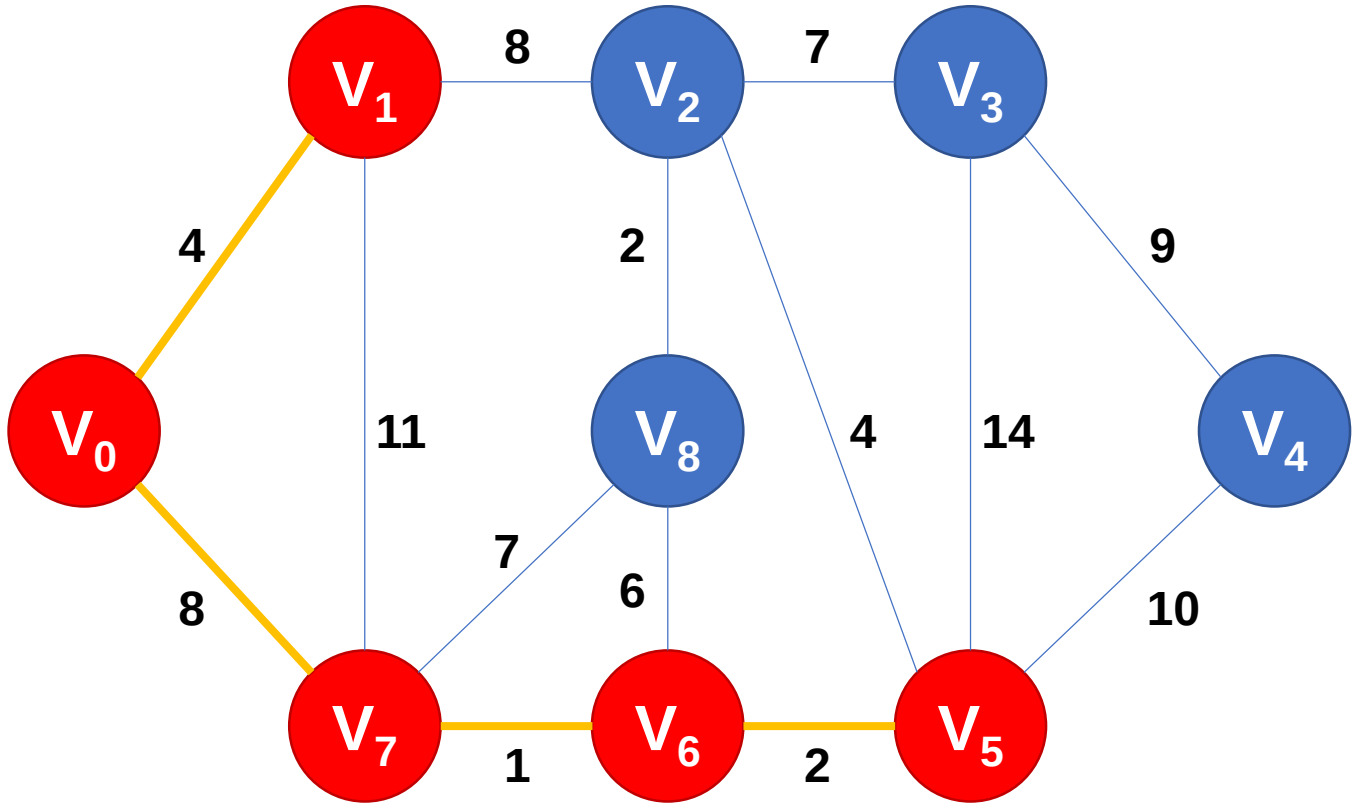


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	8	∞	∞	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={v_0, v_1, v_7, v_6, v_5}

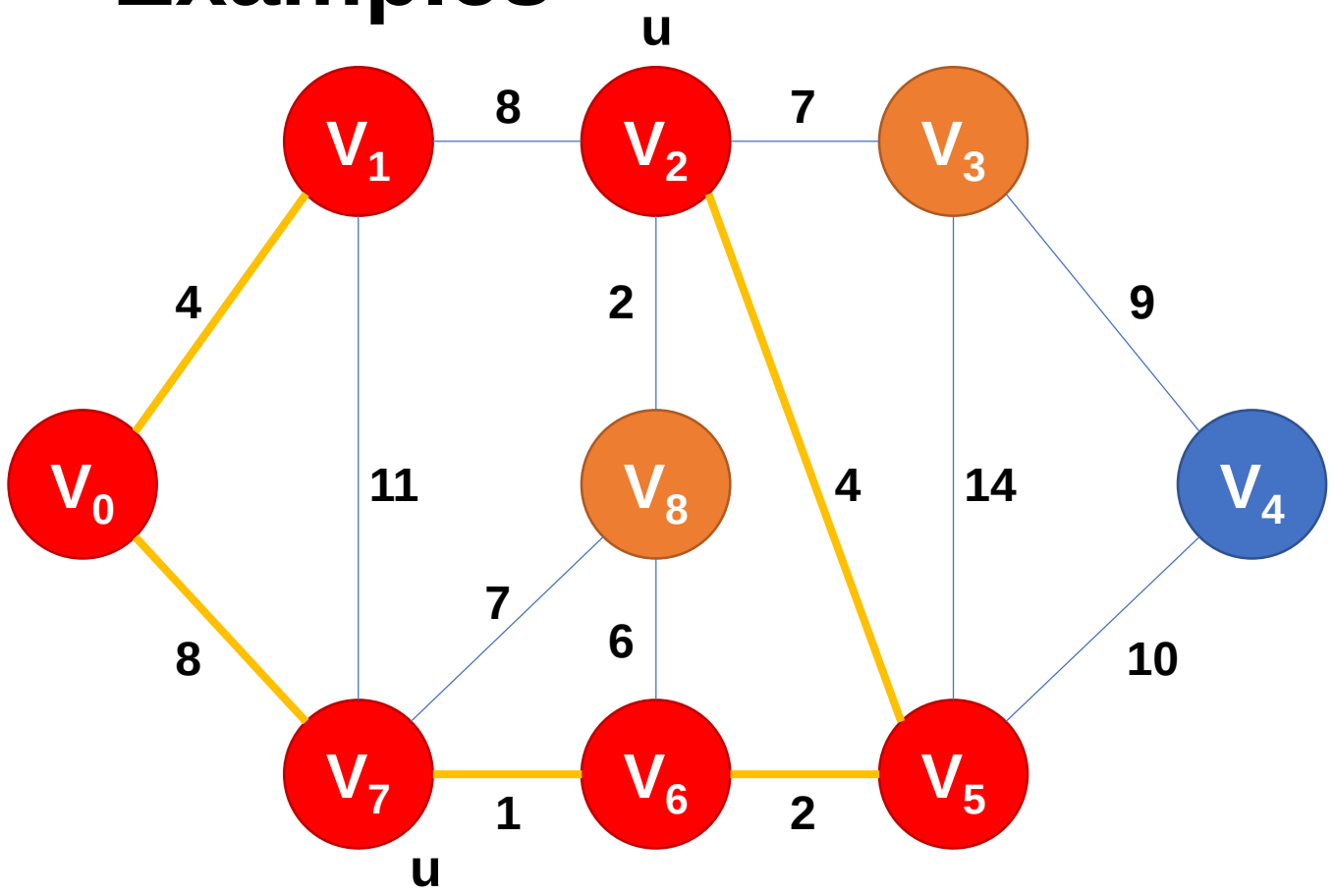


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	14	10	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

$mstSet=\{v_0, v_1, v_7, v_6, v_5, v_2\}$

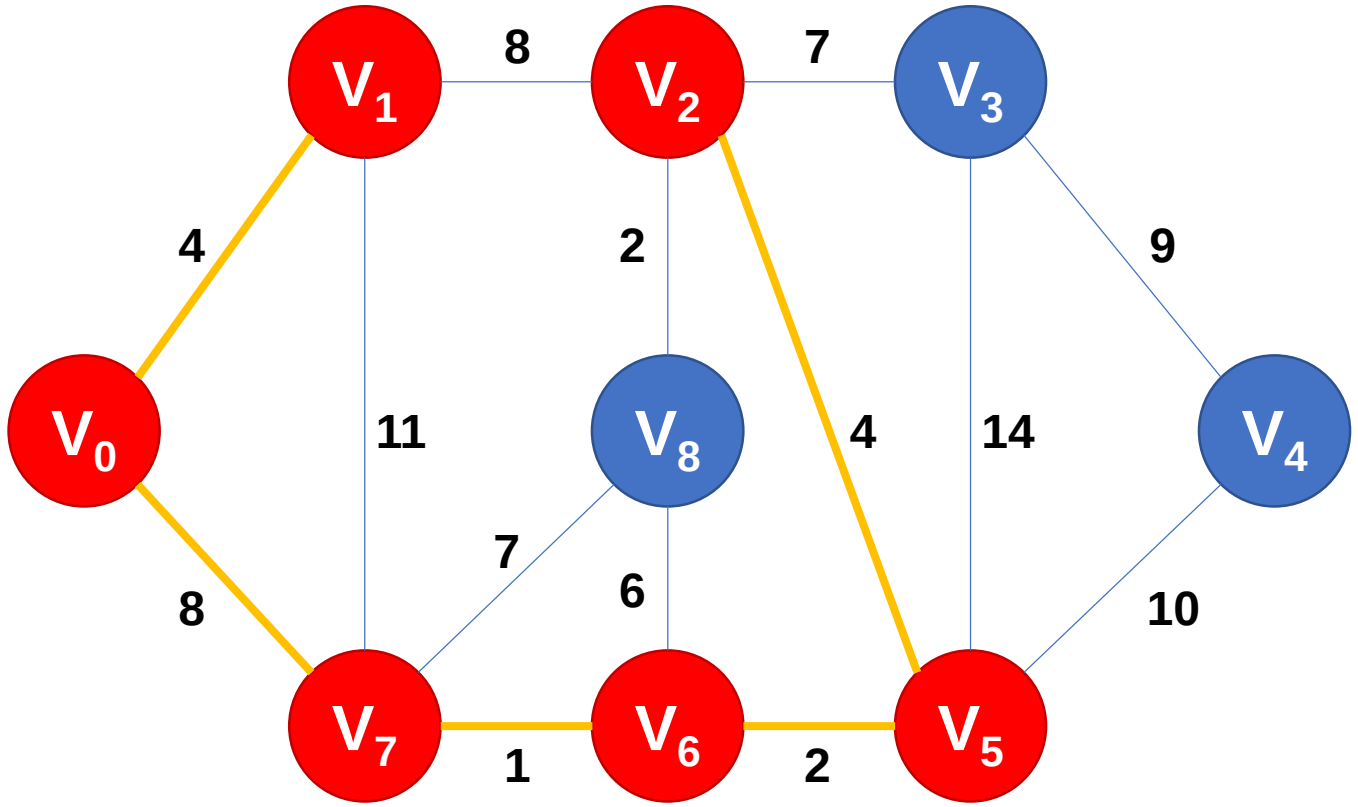


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	14	10	2	1	8	6

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

$mstSet=\{v_0, v_1, v_7, v_6, v_5, v_2\}$

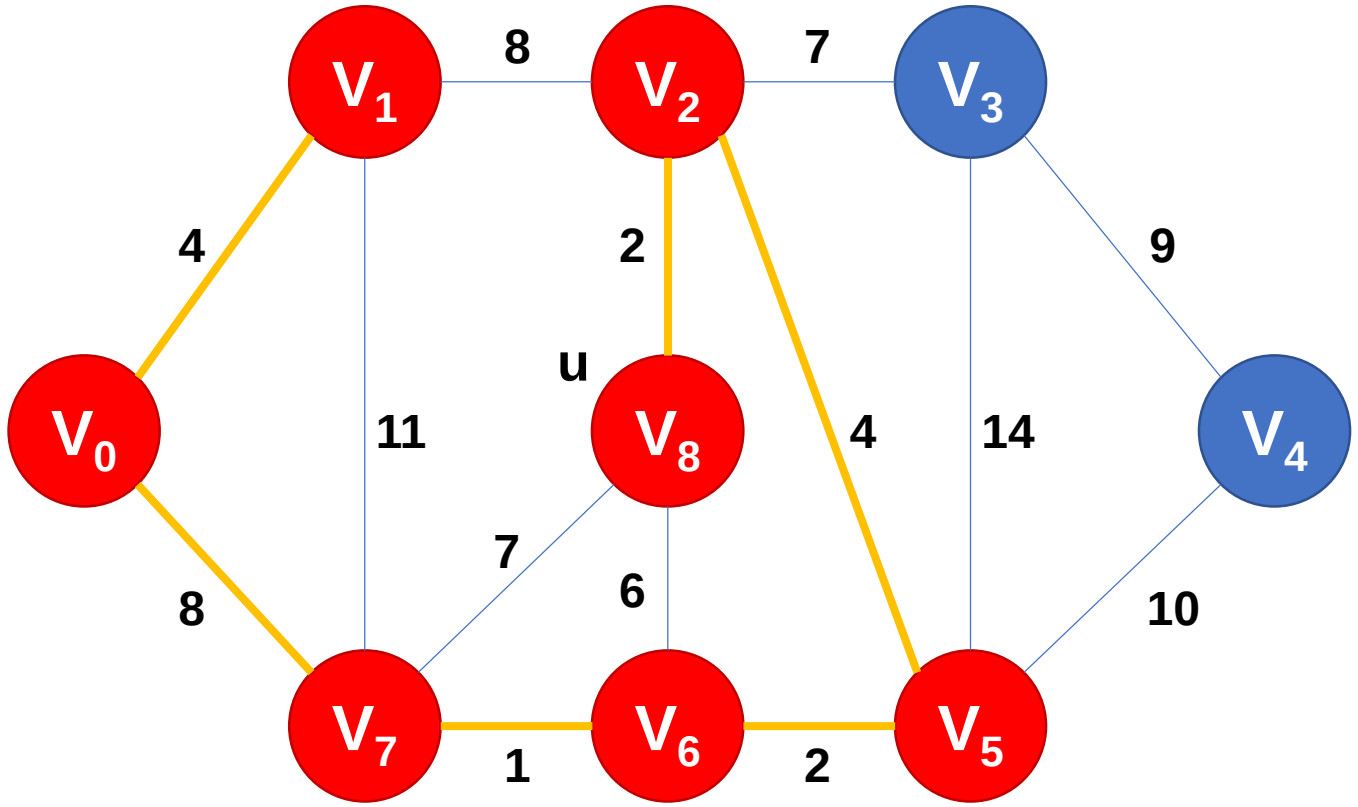


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8$}



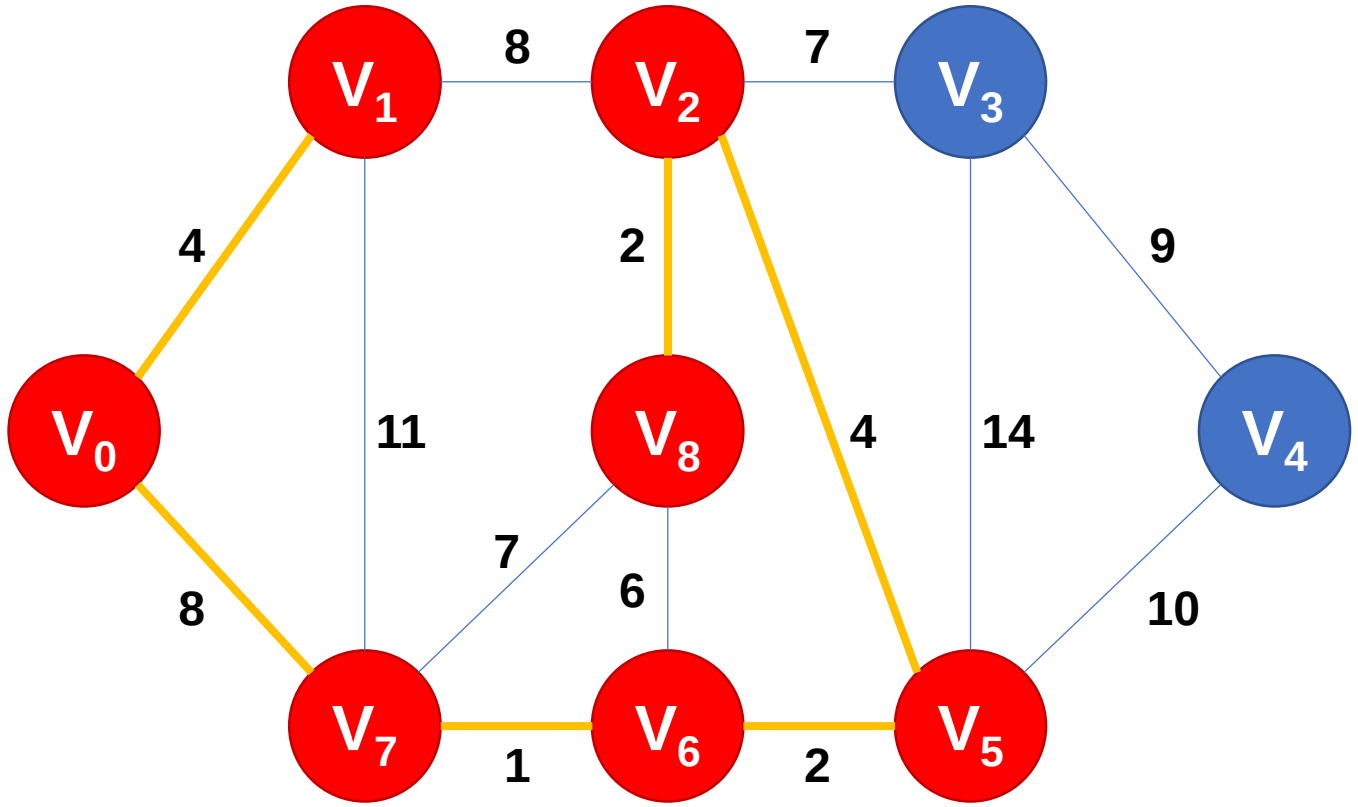
u

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8$}

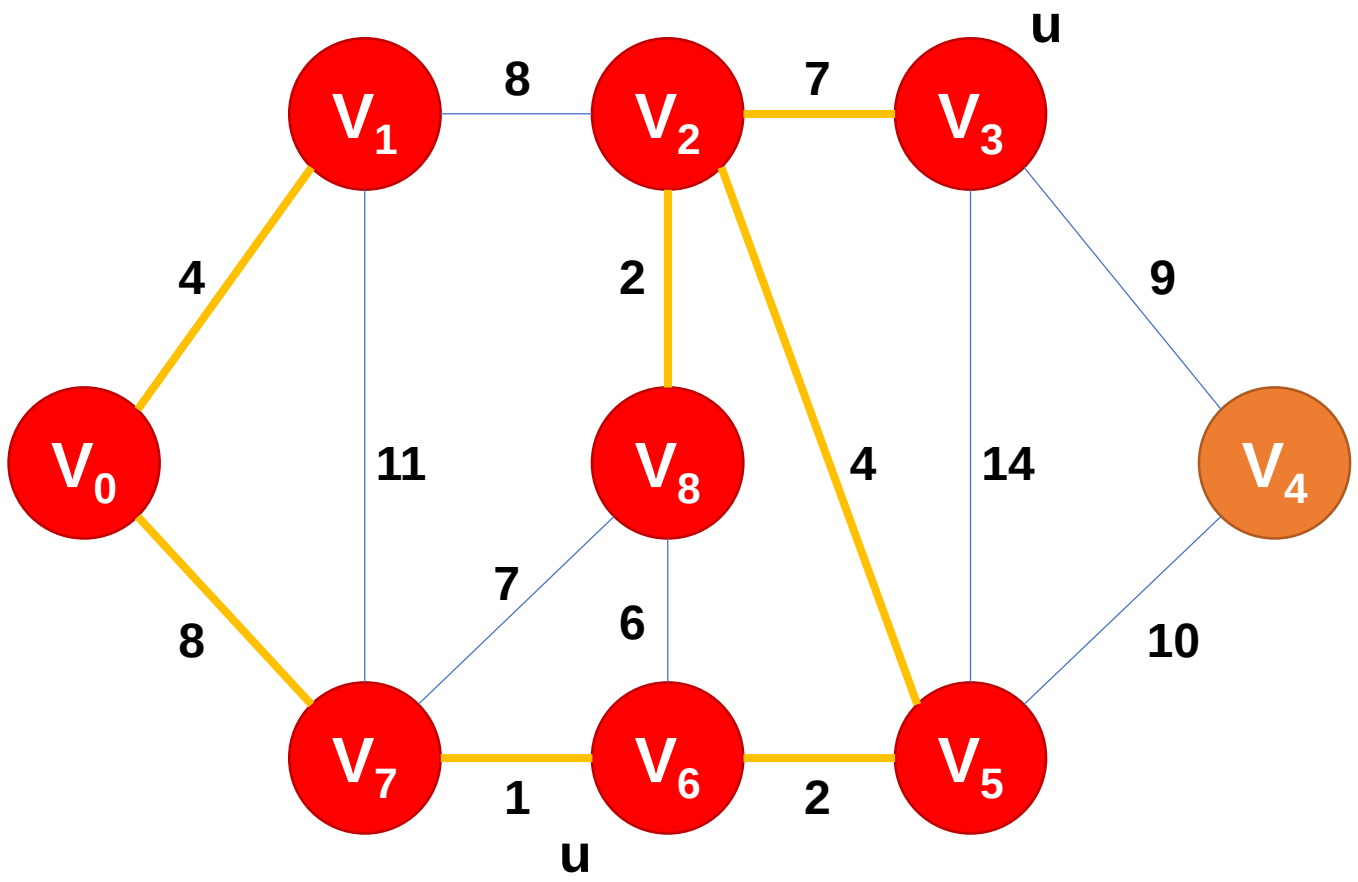


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8, v_3$}

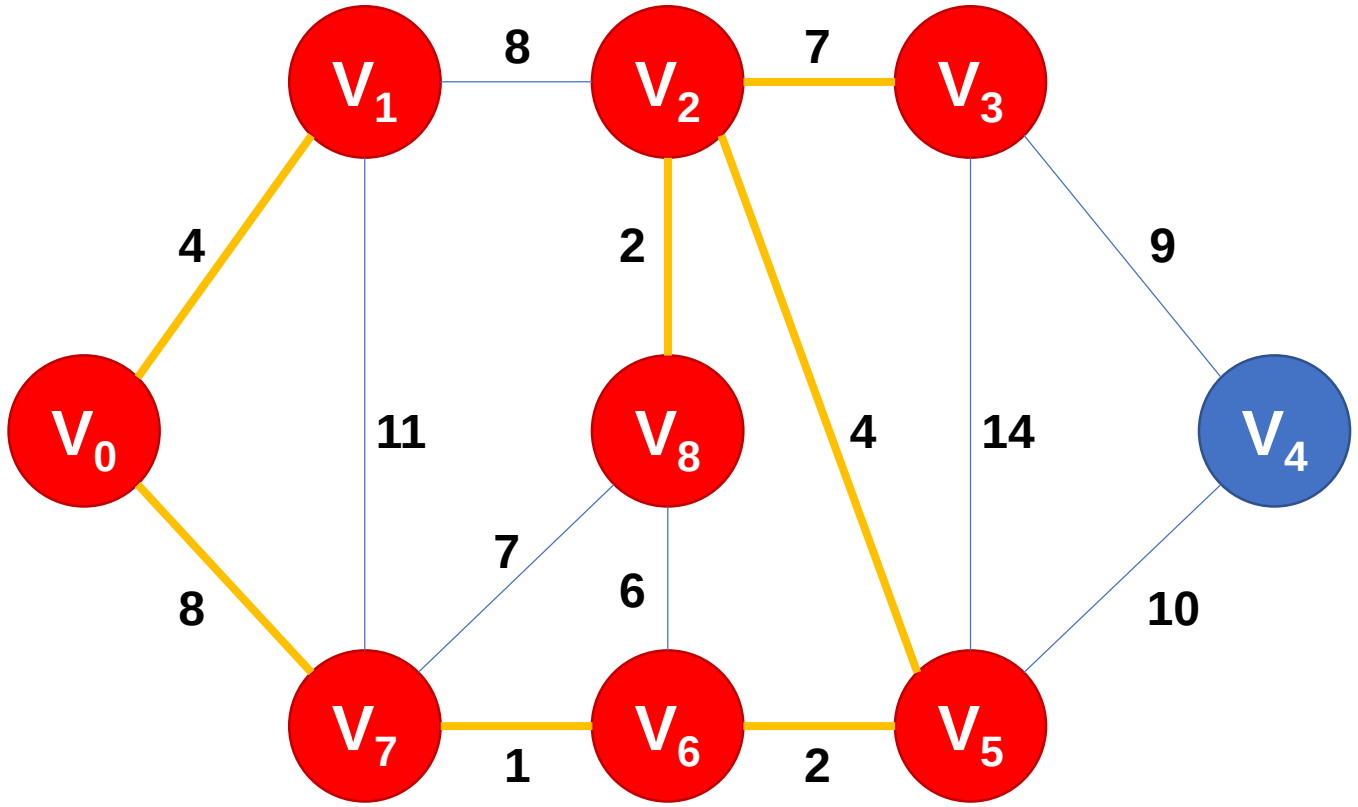


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	10	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8, v_3$}

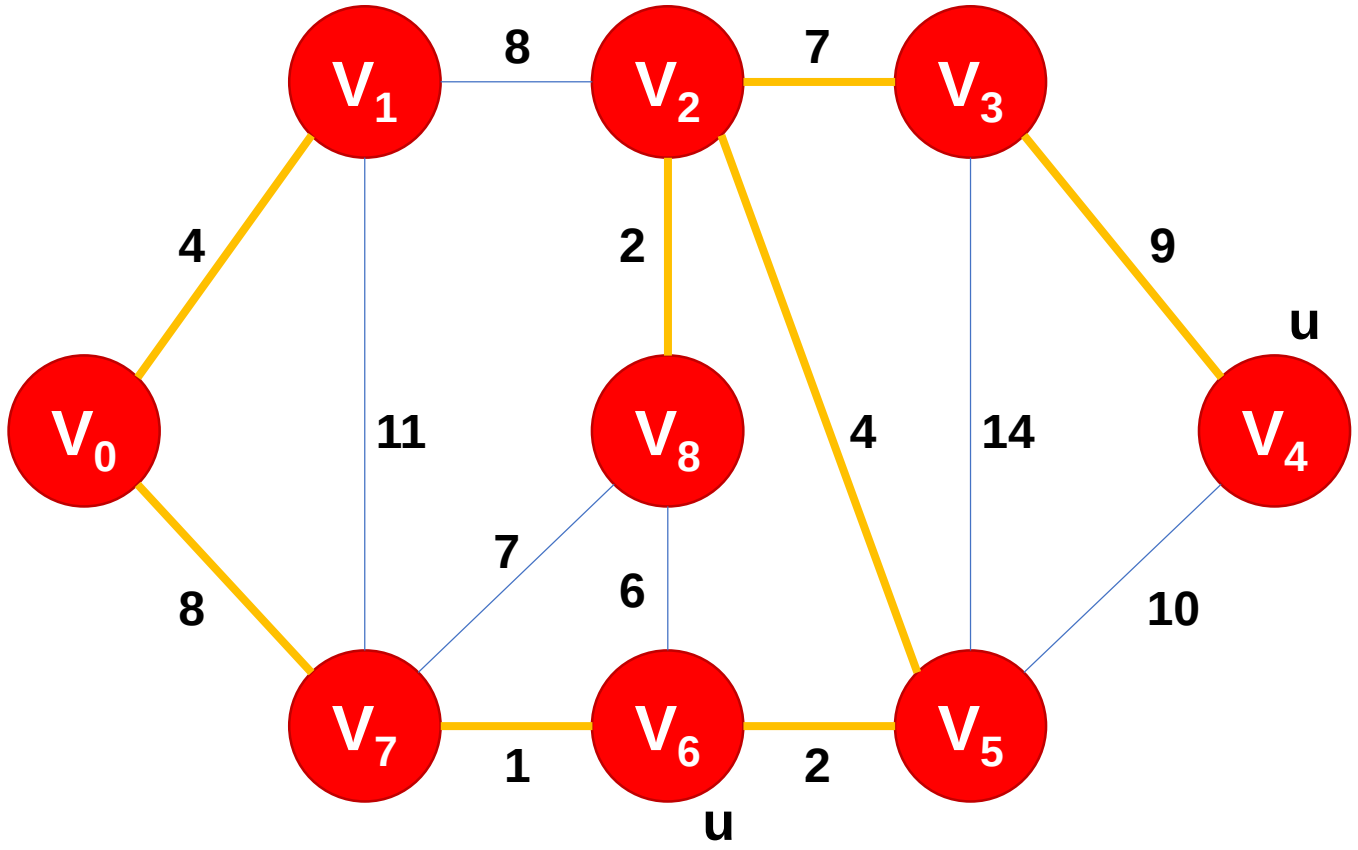


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8, v_3, v_4$}

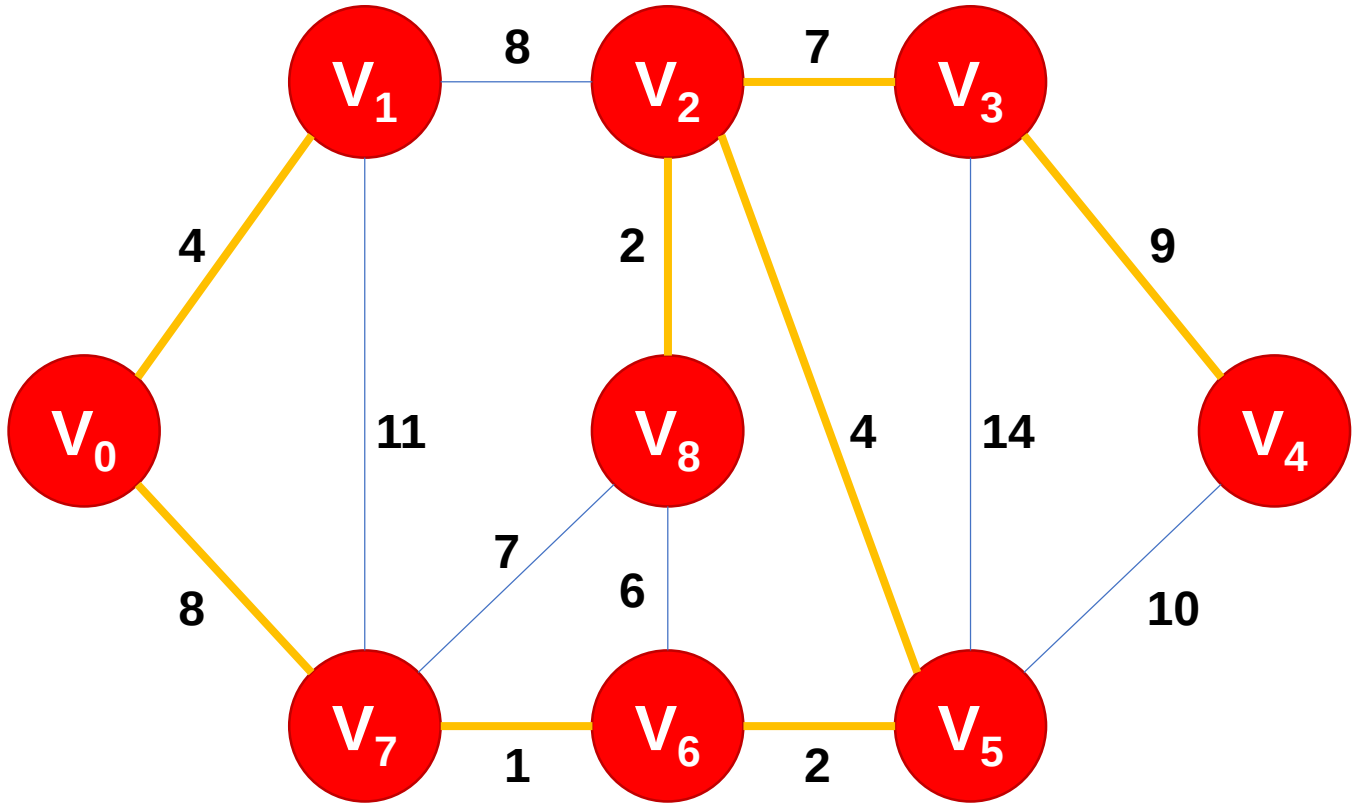


Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

mstSet={$v_0, v_1, v_7, v_6, v_5, v_2, v_8, v_3, v_4$}



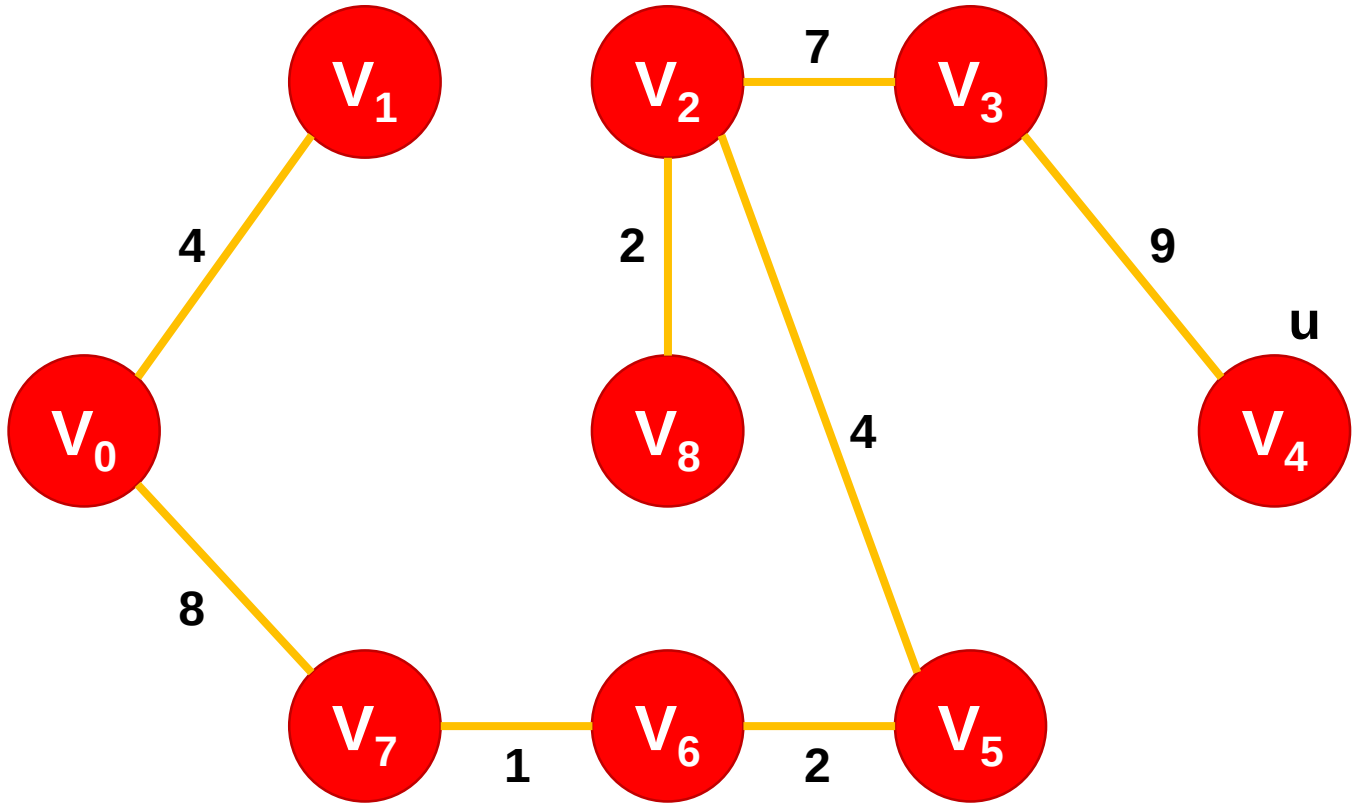
Terminate!!

Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   a. Pick a vertex u which is not there in mstSet and has minimum key value.
   b. Include u to mstSet.
   c. Update key value of all adjacent vertices of u.

Examples

Weights of MST = 37



Ver.	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Key	0	4	4	7	9	2	1	8	2

Time Complexity

- Time Complexity of Prim's Algorithm is $O(V^2)$ if adjacency matrix is used.
- If adjacency list is used, then the time complexity of Prim's algorithm can be reduced to O(E Log V) with the help of binary heap and O(E + V Log V) with the help of Fibonacci heap.

Pseudocode

```
PRIM(V, E, w, r)
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)      ▷ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))
```

Pseudocode (when we use Binary Heap)

```
PRIM(V, E, w, r)
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)      ▷ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)                          ——— O(log V)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))  ——— O(log V)
```

→ O(E log V + V log V) = O(E log V)

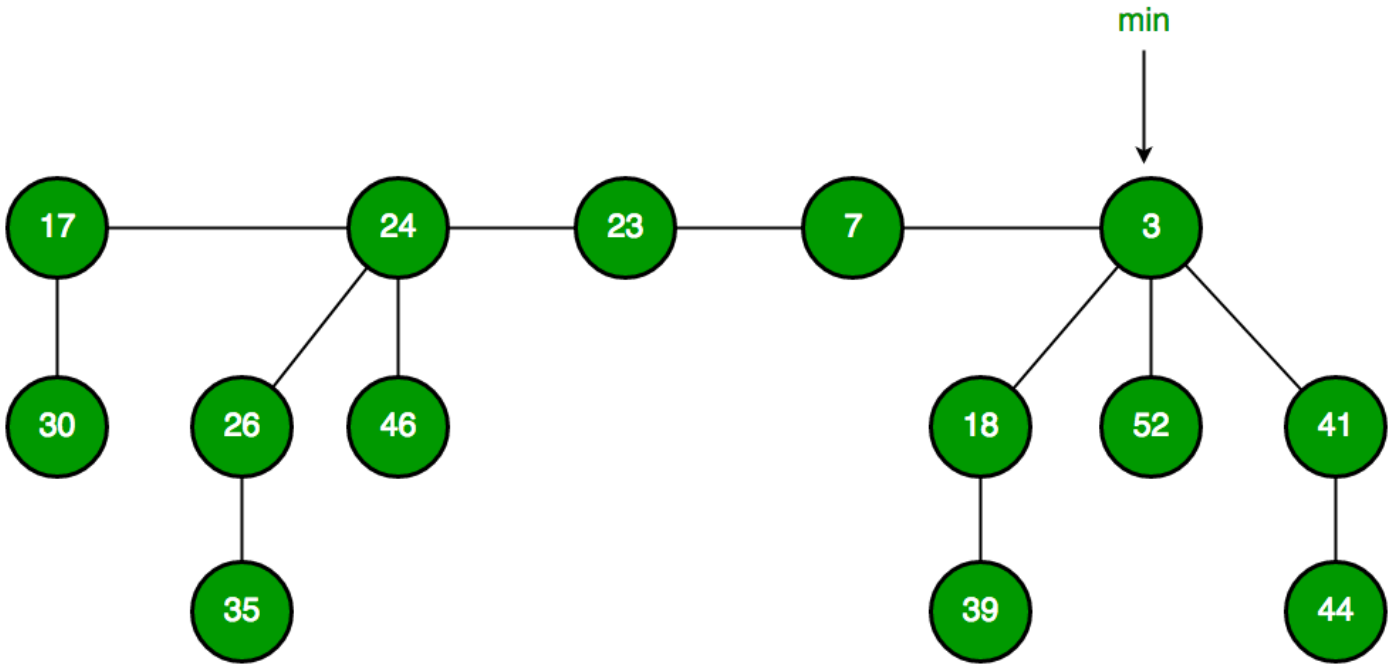
Pseudocode (when we use Fibonacci Heap)

```
PRIM(V, E, w, r)
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)        ▷ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)                          ——— O(log V)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))  ——— O(1)
```

→ O(V log V + E)

Pseudocode (when we use Fibonacci Heap)

Fibonacci Heap



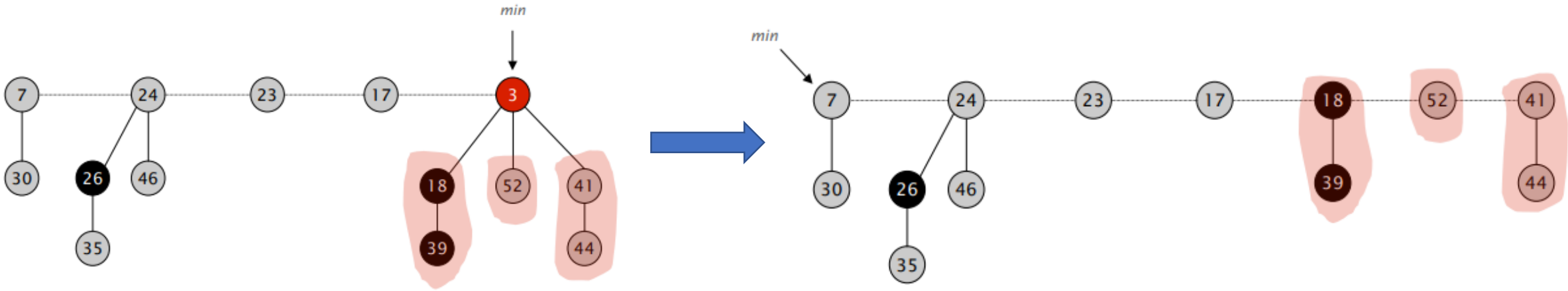
Pseudocode (when we use Fibonacci Heap)

INSERT(Q, n): O(1)

Pseudocode (when we use Fibonacci Heap)

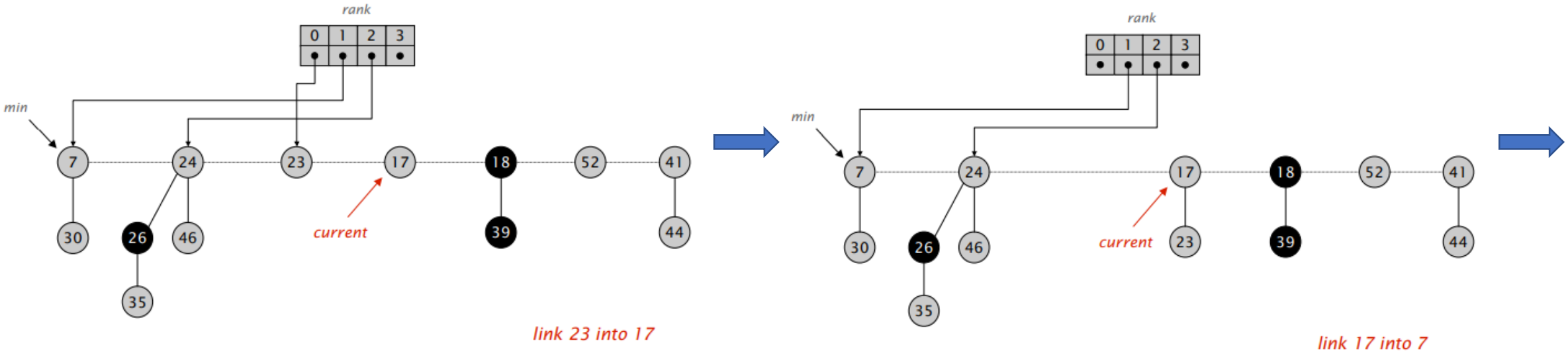
EXTRACT-MIN(Q): O(log n)

1. Delete min; meld its children into root list; update min



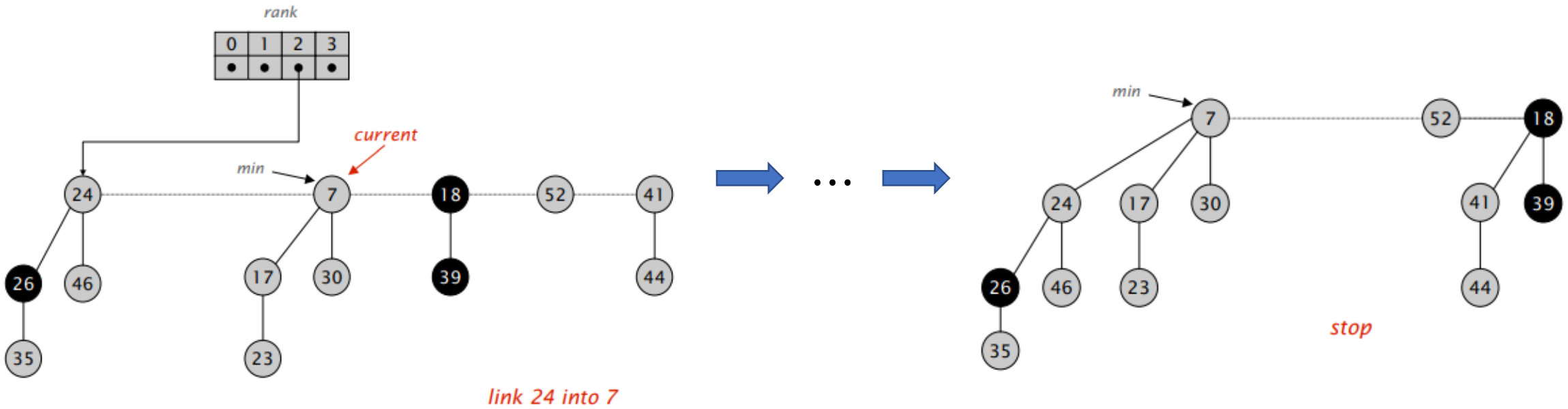
Pseudocode (when we use Fibonacci Heap)

2. **Consolidate trees so that no two roots have same rank (# of children).**



Pseudocode (when we use Fibonacci Heap)

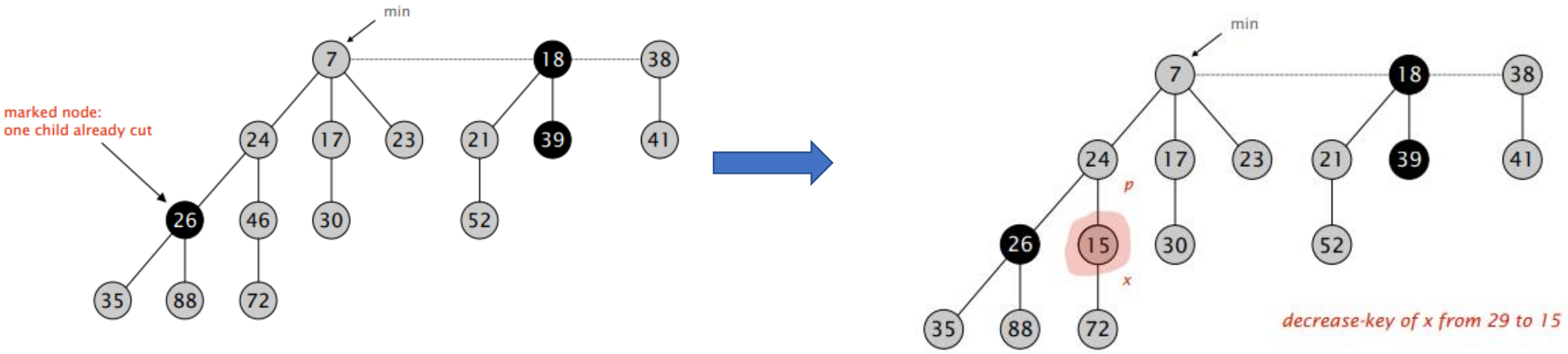
2. **Consolidate trees so that no two roots have same rank (# of children).**



Pseudocode (when we use Fibonacci Heap)

DECREASE-KEY(Q, v, w(u,v)): O(1)

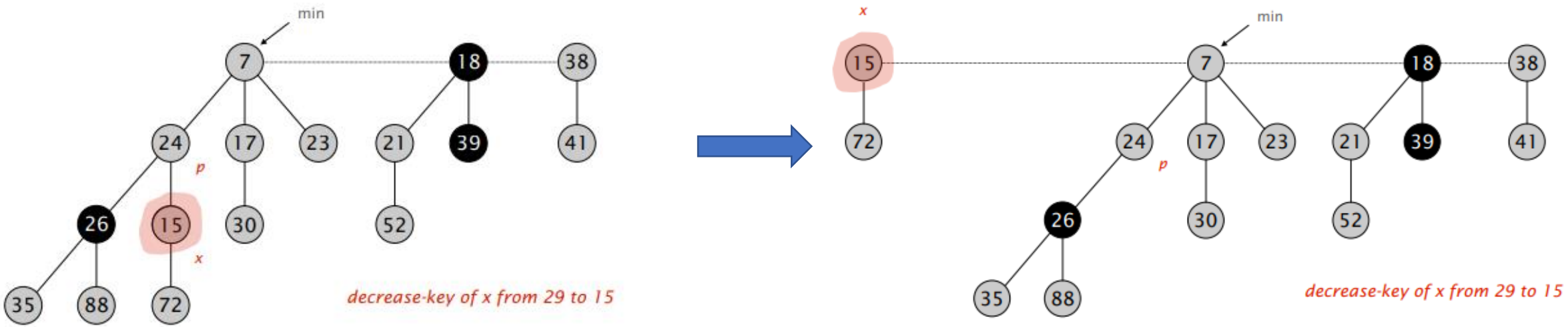
1. **Decrease key of x**



Pseudocode (when we use Fibonacci Heap)

[heap order violated]

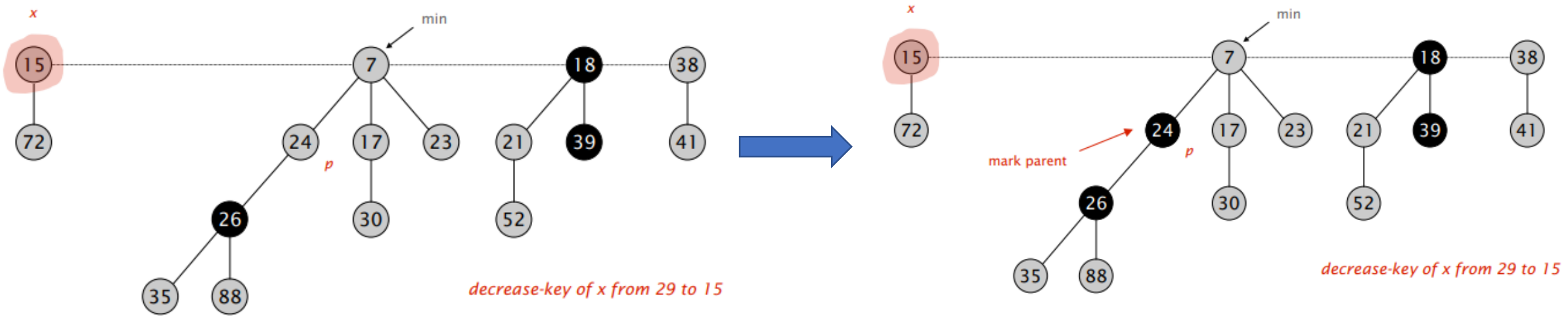
2. **Cut tree rooted at x, meld into root list, and unmark.**



Pseudocode (when we use Fibonacci Heap)

[heap order violated]

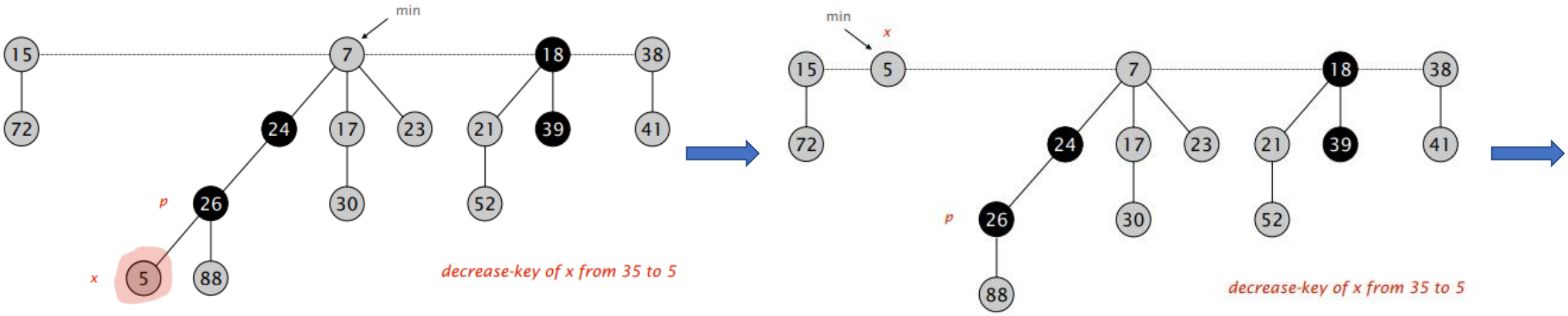
3. **If parent p of x is unmarked (hasn't yet lost a child), mark it;**



Pseudocode (when we use Fibonacci Heap)

[heap order violated]

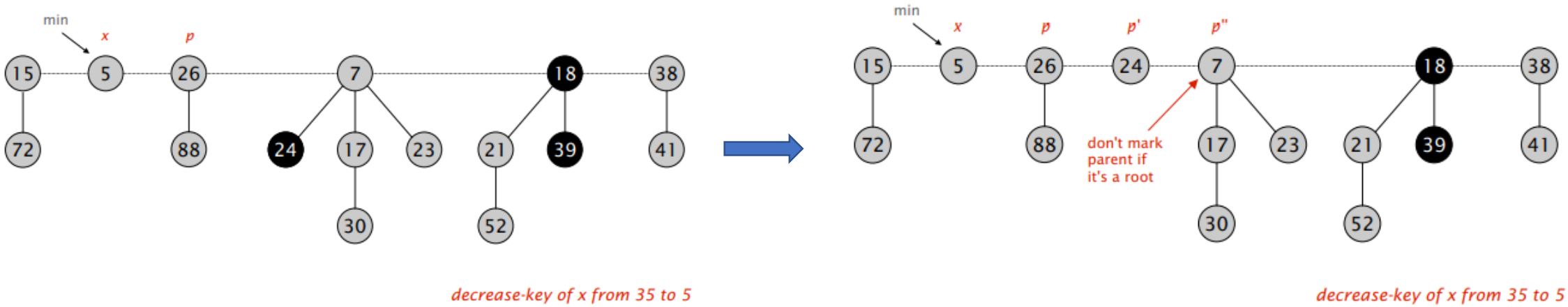
3. Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Pseudocode (when we use Fibonacci Heap)

[heap order violated]

3. Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Implementation (1)

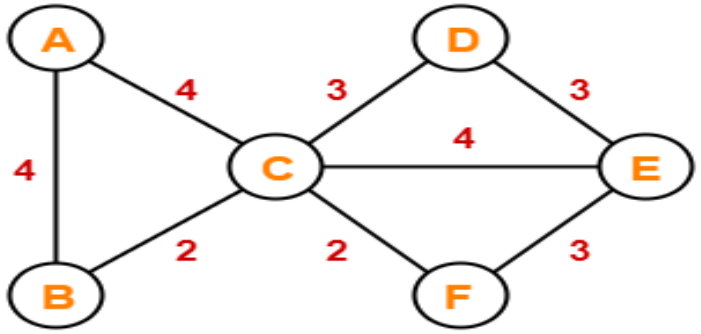
```
// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V])
{
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";
}
```

```
// Driver code
int main()
{
    /* Let us create the following graph
        2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | / \ |
    (3)-------(4)
            9     */
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };

    // Print the solution
    primMST(graph);

    return 0;
}
```

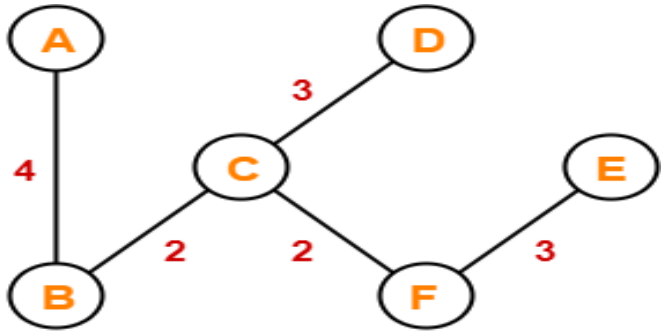
Implementation (2)

```
// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];

    // To represent set of vertices not yet included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first vertex.
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++)
    {
        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent vertices of m
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, graph);
}
```

Kruskal’s Algorithm vs Prim’s Algorithm

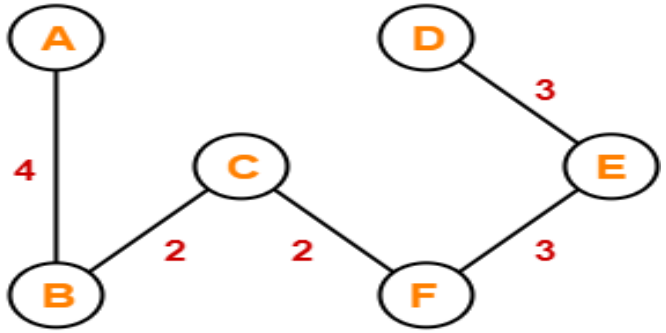


Given Graph



Result from Prim's Algorithm

(Cost = 14 units)



Result from Kruskal's Algorithm

(Cost = 14 units)

Kruskal's Algorithm vs Prim's Algorithm

Kruskal's Algorithm	Prim's Algorithm
Start to build MST from minimum edge in the graph	Start to build MST from any vertices
Disjoint Set	Adjacency Matrix, Binary Heap, Fibonacci Heap
Run faster in sparse graphs	Run faster in dense graphs
O(E Log V)	O(E Log V) with Binary Heap, O(E + V Log V) with Fibonacci Heap
Traverse the edge only once. Based on cycle, it will either reject it or accept it	Traverse the one node several time in order to get minimum distance
Greedy Algorithm	Greedy Algorithm

Reference

- Charles Leiserson and Piotr Indyk, “*Introduction to Algorithms”,* September 29, 2004

- https://www.geeksforgeeks.org

- https://www.cs.princeton.edu/~wayne/teaching/fibonacci-heap.pdf