Kruskal’s Minimum Spanning Tree (MST) Algorithm

A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. To learn more about Minimum Spanning Tree, refer to **this article**.

Introduction to Kruskal’s Algorithm:

Here we will discuss **Kruskal’s algorithm** to find the MST of a given weighted graph.

In Kruskal’s algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first at the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a **Greedy Algorithm**.

How to find MST using Kruskal’s algorithm?

Below are the steps for finding MST using 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 the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

***Step 2** uses the Union-Find algorithm to detect cycles.*

So we recommend reading the following post as a prerequisite.

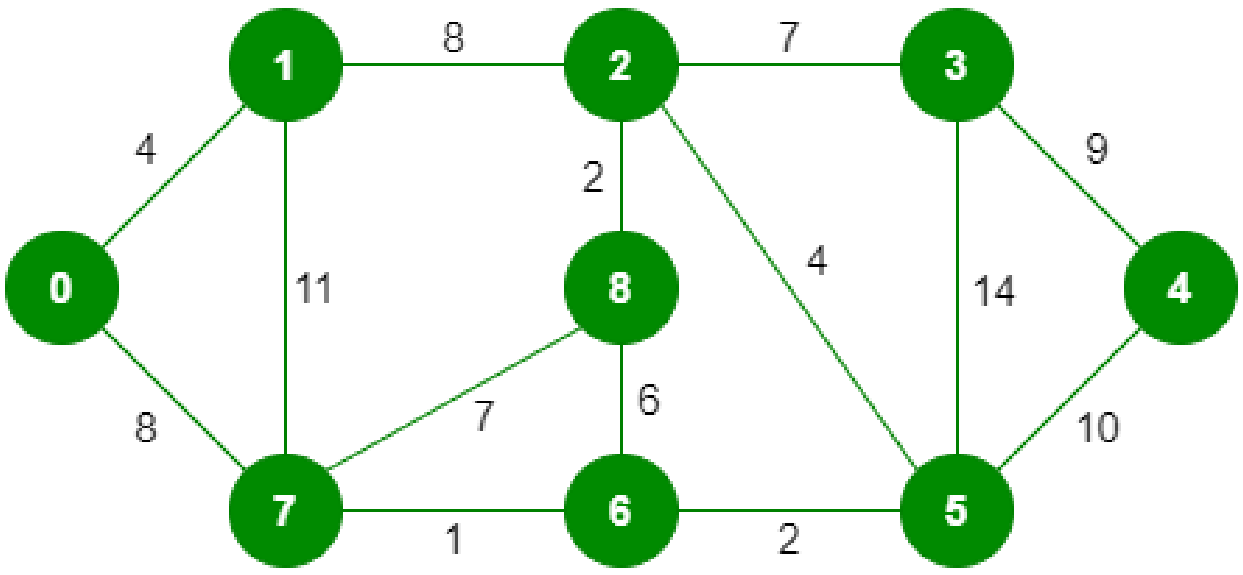
- *Union-Find Algorithm | Set 1 (Detect Cycle in a Graph)*
- *Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)*

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example:

Illustration:

Below is the illustration of the above approach:

Input Graph:



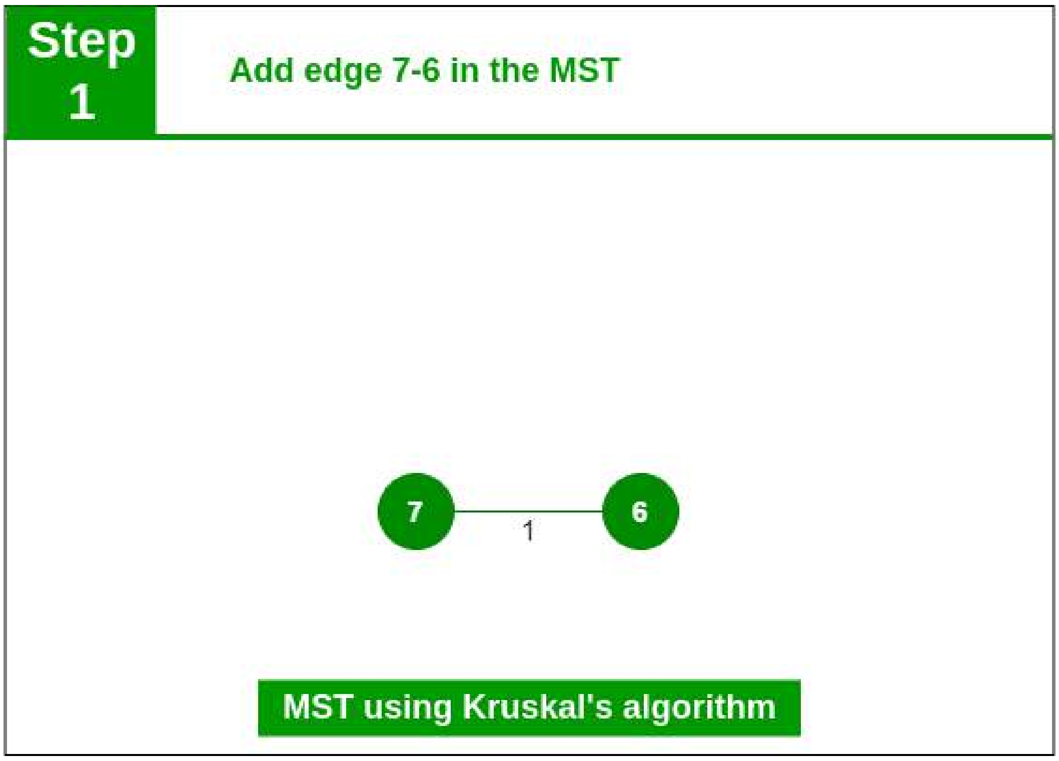
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.
After sorting:

Weight	*Source*	*Destination*
1	*7*	*6*

2	*8*	*2*
2	*6*	*5*
4	*0*	*1*
4	*2*	*5*
6	*8*	*6*
7	*2*	*3*
7	*7*	*8*
8	*0*	*7*
8	*1*	*2*
9	*3*	*4*
10	*5*	*4*
11	*1*	*7*
14	*3*	*5*

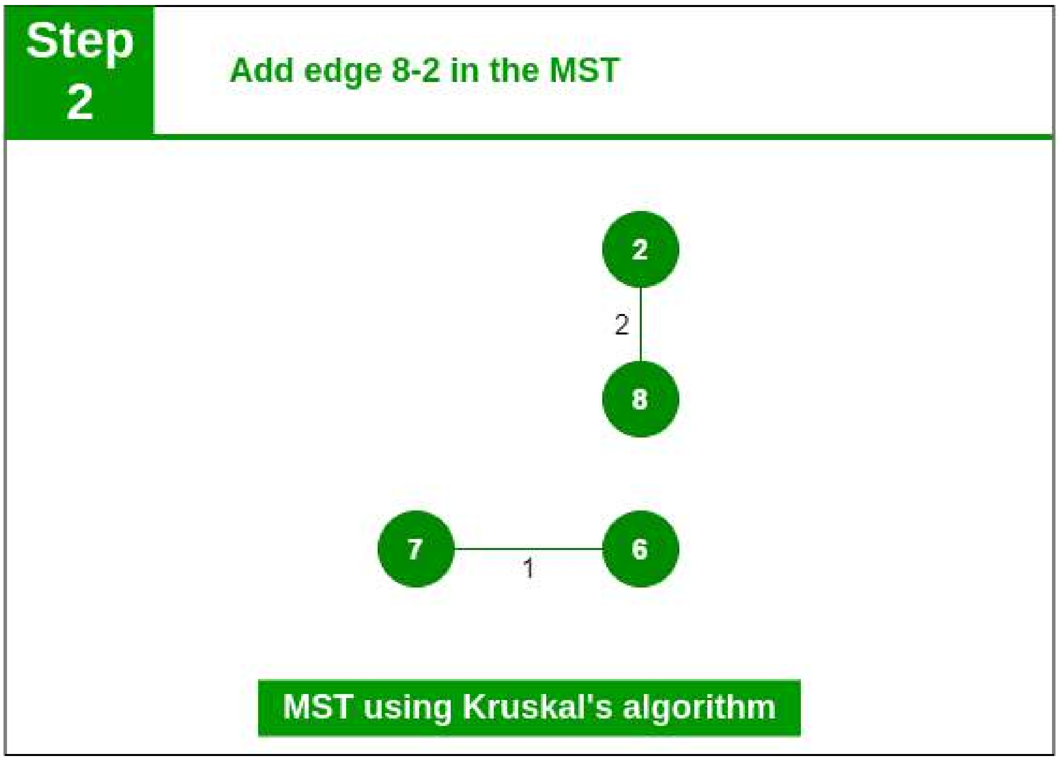
Now pick all edges one by one from the sorted list of edges

Step 1: *Pick edge 7-6. No cycle is formed, include it.*



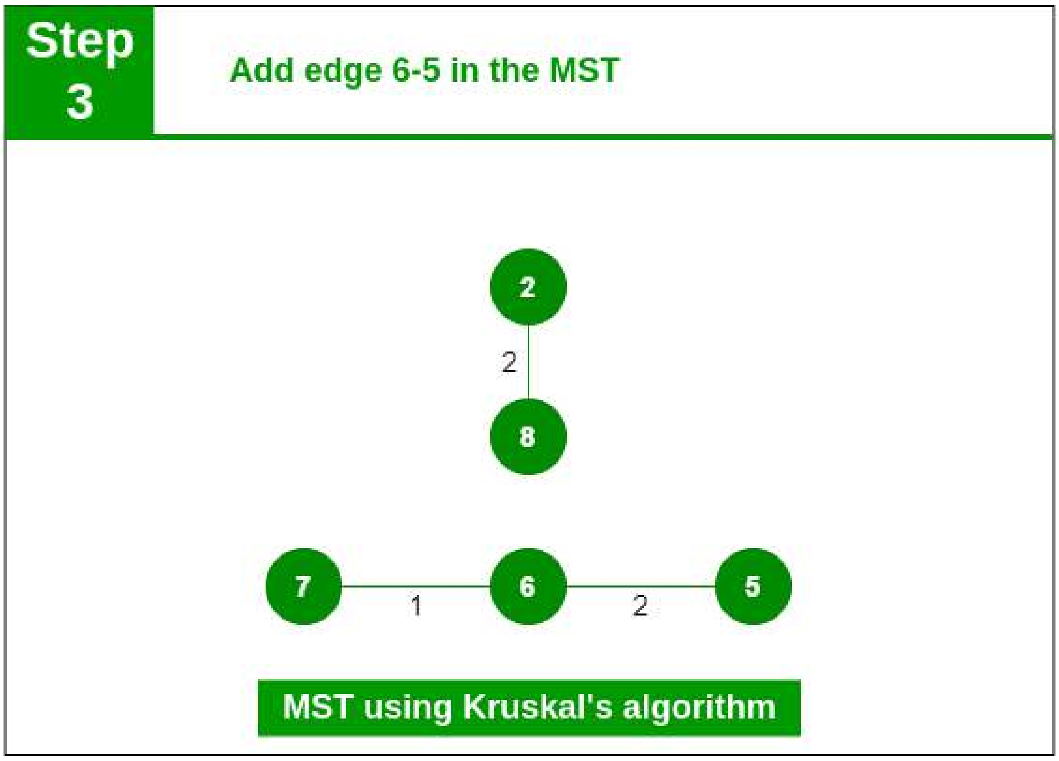
Add edge 7-6 in the MST

Step 2: *Pick edge 8-2. No cycle is formed, include it.*



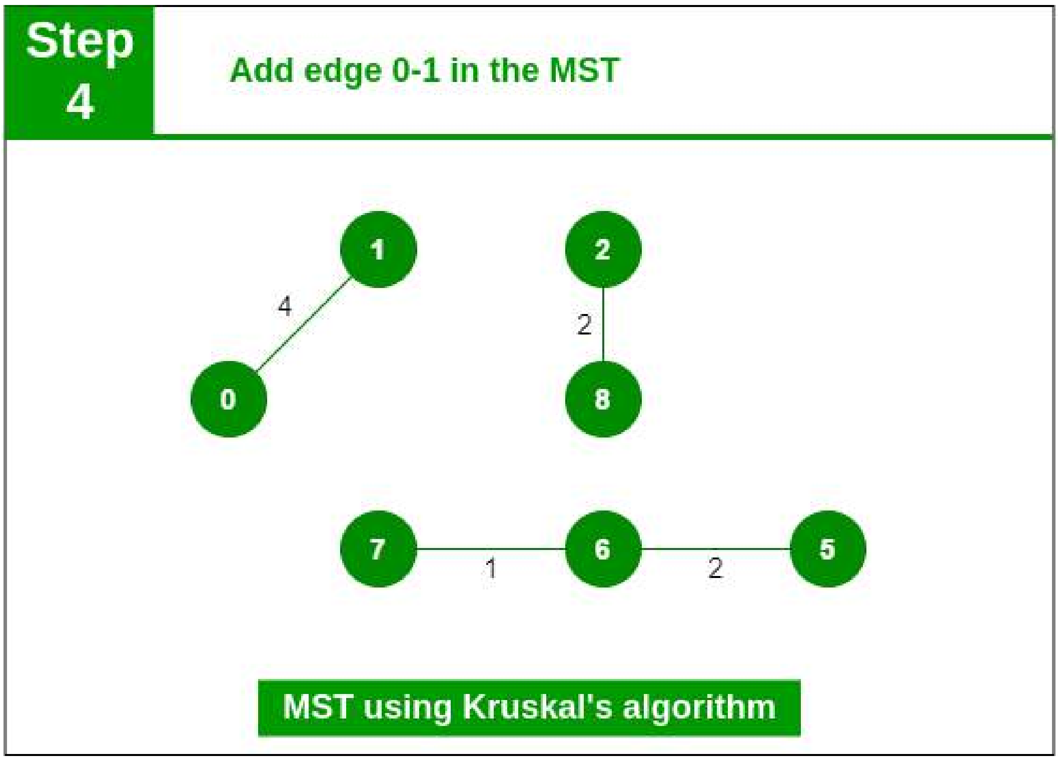
Add edge 8-2 in the MST

Step 3: *Pick edge 6-5. No cycle is formed, include it.*



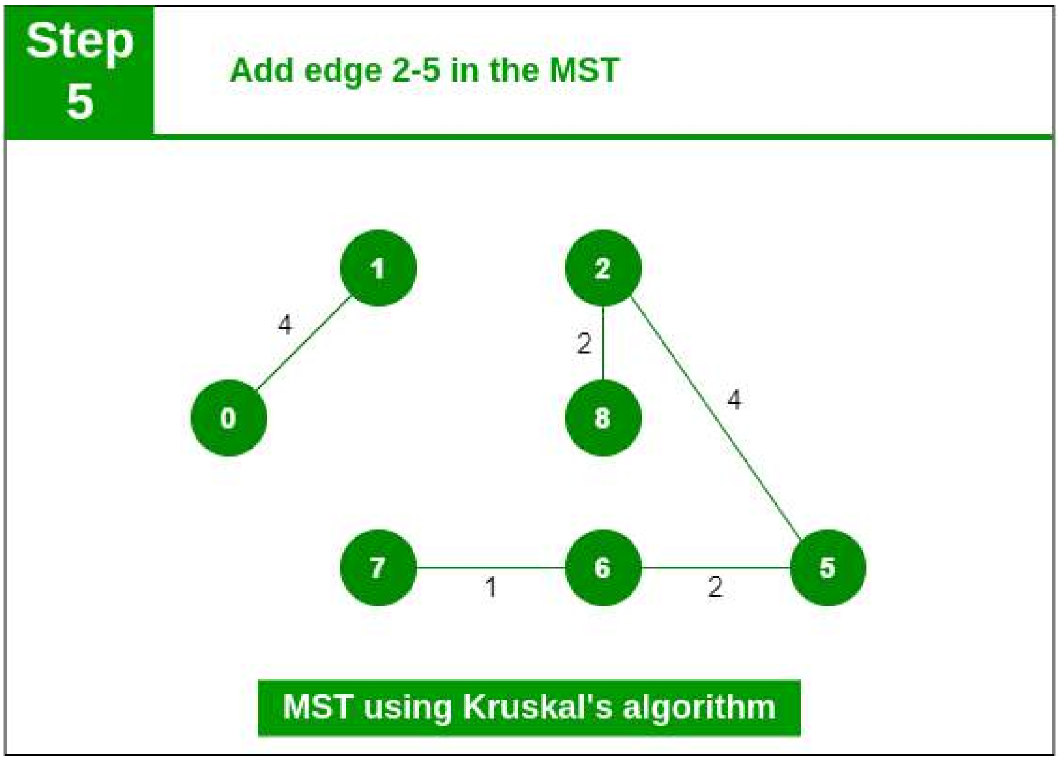
Add edge 6-5 in the MST

Step 4: *Pick edge 0-1. No cycle is formed, include it.*



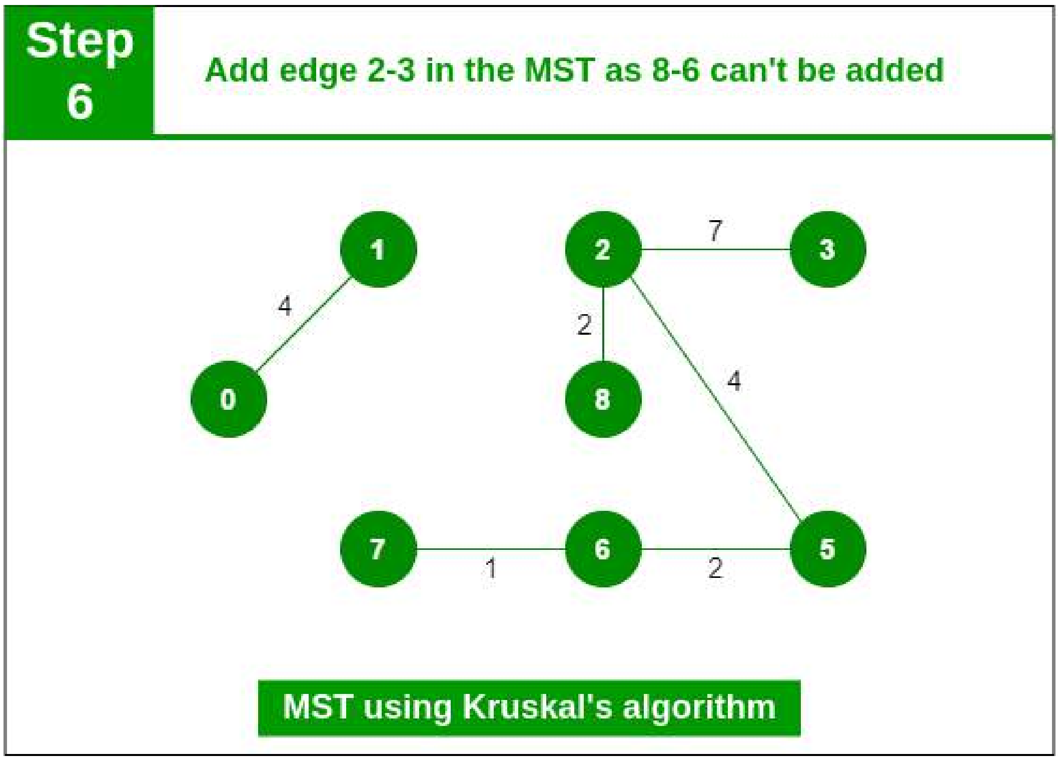
Add edge 0-1 in the MST

Step 5: *Pick edge 2-5. No cycle is formed, include it.*



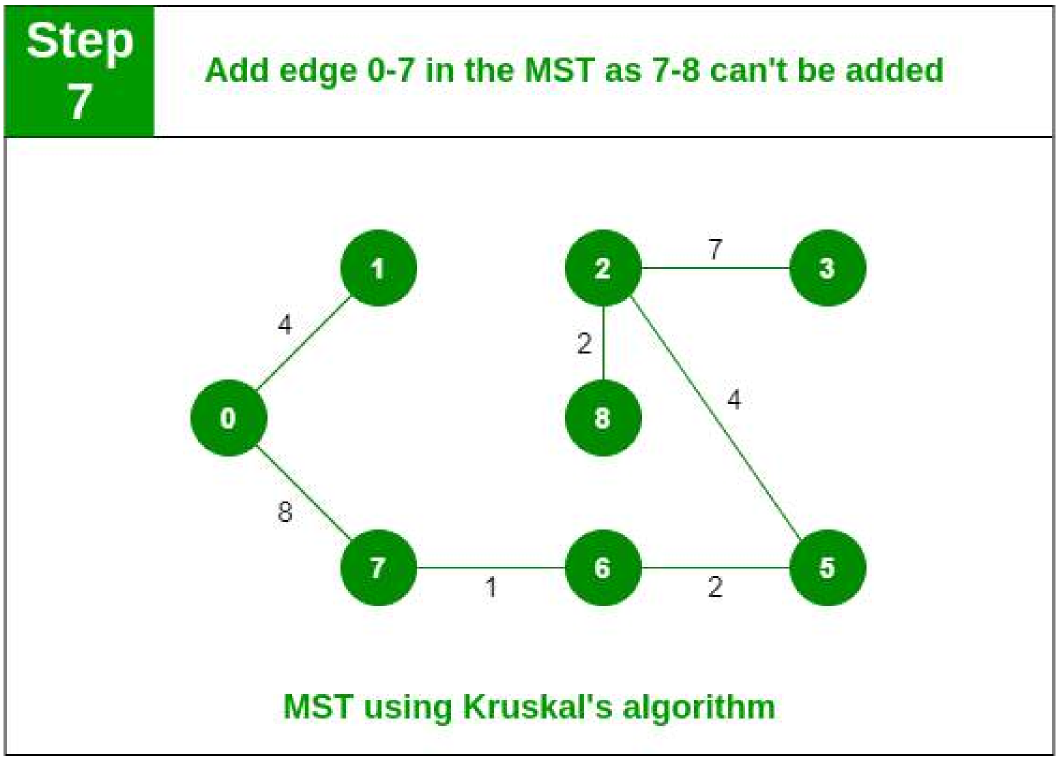
Add edge 2-5 in the MST

Step 6: *Pick edge 8-6. Since including this edge results in the cycle, discard it. Pick edge 2-3: No cycle is formed, include it.*


Step 6
Add edge 2-3 in the MST as 8-6 can't be added
1
2
7
3
4
2
0
8
4
7
6
5
1
2
MST using Kruskal's algorithm

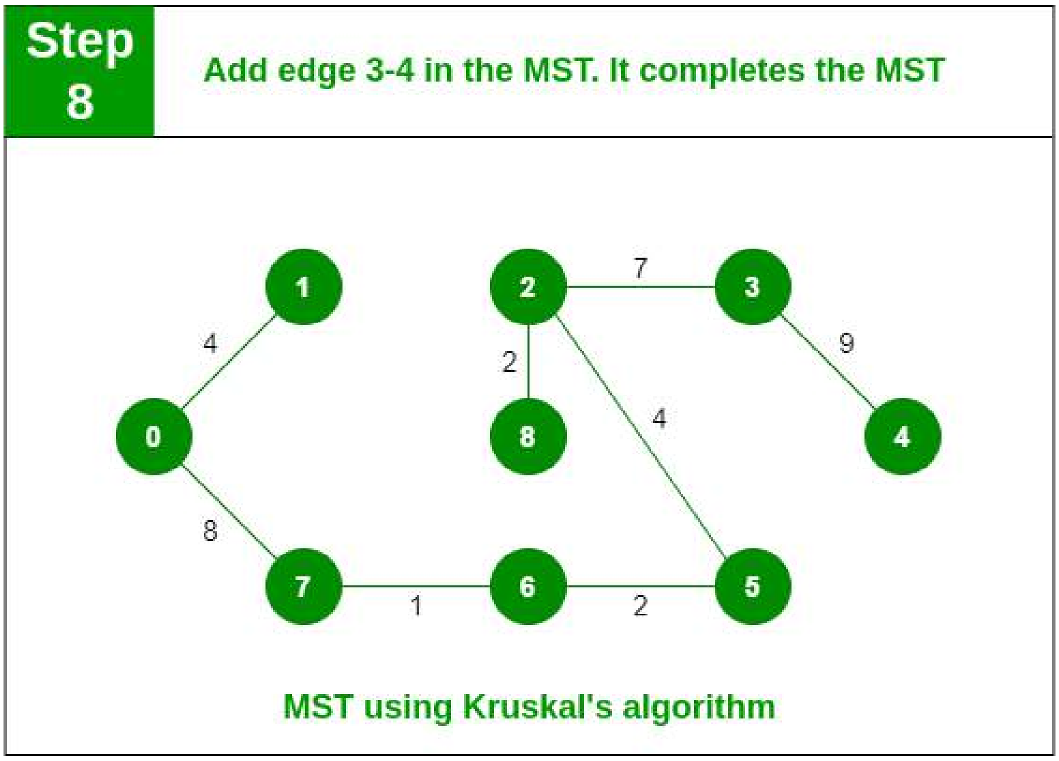
Add edge 2-3 in the MST

Step 7: *Pick edge 7-8. Since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed, include it.*


Step 7
Add edge 0-7 in the MST as 7-8 can't be added
1
2
7
3
4
2
4
0
8
8
7
1
6
2
5
MST using Kruskal's algorithm

Add edge 0-7 in MST

Step 8: *Pick edge 1-2. Since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed, include it.*


Step 8
Add edge 3-4 in the MST. It completes the MST
1
2
7
3
4
2
9
4
0
8
4
8
7
1
6
2
5
MST using Kruskal's algorithm

Add edge 3-4 in the MST

Note: *Since the number of edges included in the MST equals to (V – 1), so the algorithm stops here*

```
    return 0;
}
```

C++

```
// C++ program for the above approach

#include <bits/stdc++.h>
using namespace std;

// DSU data structure
// path compression + rank by union
class DSU {
    int* parent;
    int* rank;

public:
    DSU(int n)
    {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++) {
            parent[i] = -1;
            rank[i] = 1;
        }
    }

    // Find function
    int find(int i)
    {
        if (parent[i] == -1)
            return i;

        return parent[i] = find(parent[i]);
    }

    // Union function
    void unite(int x, int y)
    {
        int s1 = find(x);
        int s2 = find(y);

        if (s1 != s2) {
            if (rank[s1] < rank[s2]) {
                parent[s1] = s2;
            }
            else if (rank[s1] > rank[s2]) {
                parent[s2] = s1;
            }
            else {
                parent[s2] = s1;
                rank[s1] += 1;
            }
        }
```

```cpp
    }
};

class Graph {
    vector<vector<int> > edgelist;
    int V;

public:
    Graph(int V) { this->V = V; }

    // Function to add edge in a graph
    void addEdge(int x, int y, int w)
    {
        edgelist.push_back({ w, x, y });
    }

    void kruskals_mst()
    {
        // Sort all edges
        sort(edgelist.begin(), edgelist.end());

        // Initialize the DSU
        DSU s(V);
        int ans = 0;
        cout << "Following are the edges in the "
                "constructed MST"
             << endl;
        for (auto edge : edgelist) {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            // Take this edge in MST if it does
            // not forms a cycle
            if (s.find(x) != s.find(y)) {
                s.unite(x, y);
                ans += w;
                cout << x << " -- " << y << " == " << w
                     << endl;
            }
        }
        cout << "Minimum Cost Spanning Tree: " << ans;
    }
};

// Driver code
int main()
{
    Graph g(4);
    g.addEdge(0, 1, 10);
    g.addEdge(1, 3, 15);
    g.addEdge(2, 3, 4);
    g.addEdge(2, 0, 6);
    g.addEdge(0, 3, 5);

    // Function call
    g.kruskals_mst();
```

```
<script>
// JavaScript implementation of the krushkal's algorithm.

function makeSet(parent,rank,n)
{
    for(let i=0;i<n;i++)
    {
        parent[i]=i;
        rank[i]=0;
    }
}

function findParent(parent,component)
{
    if(parent[component]==component)
        return component;

    return parent[component] = findParent(parent,parent[component]);
}

function unionSet(u, v, parent, rank,n)
{
    //this function unions two set on the basis of rank
    //as shown below
    u=findParent(parent,u);
    v=findParent(parent,v);

    if(rank[u]<rank[v])
    {
        parent[u]=v;
    }
    else if(rank[u]<rank[v])
    {
        parent[v]=u;
    }
    else
    {
        parent[v]=u;
        rank[u]++;//since the rank increases if the ranks of two sets are same
    }
}

function kruskalAlgo(n, edge)
{
    //First we sort the edge array in ascending order
    //so that we can access minimum distances/cost
    edge.sort((a, b)=>{
        return a[2] - b[2];
    })
    //inbuilt quick sort function comes with stdlib.h
    //go to https://www.geeksforgeeks.org/comparator-function-of-qsort-in-c/
    //if there is any doubt regarding the function
    let parent = new Array(n);
    let rank = new Array(n);
    makeSet(parent,rank,n);//function to initialize parent[] and rank[]

    let minCost=0;//to store the minimun cost

    document.write("Following are the edges in the constructed MST");
    for(let i=0;i<n;i++)
    {
        let v1=findParent(parent,edge[i][0]);
        let v2=findParent(parent,edge[i][1]);
        let wt=edge[i][2];

        if(v1!=v2)//if the parents are different that means they are in
                  //different sets so union them
        {
            unionSet(v1,v2,parent,rank,n);
            minCost+=wt;
            document.write(edge[i][0] + " -- " + edge[i][1] + " == " + wt);
        }
    }

    document.write("Minimum Cost Spanning Tree:",minCost);
}


//Here 5 is the number of edges, can be asked from the user
//when making the graph through user input
//3 represents the no of index positions for storing u --> v(adjacent vertices)
//and its cost/distance;
let edge = [
        [0,1,10],
        [0,2,6],
        [0,3,5],
        [1,3,15],
        [2,3,4]
];

kruskalAlgo(5,edge);

// The code is contributed by Arushi Jindal.
</script>
```

Output

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
```

Time Complexity: O(E * logE) or O(E * logV)

- Sorting of edges takes O(E * logE) time.

- After sorting, we iterate through all edges and apply the find-union algorithm. The find and union operations can take at most O(logV) time.
- So overall complexity is O(E * logE + E * logV) time.
- The value of E can be at most $O(V^2)$, so O(logV) and O(logE) are the same. Therefore, the overall time complexity is O(E * logE) or O(E*logV)

Auxiliary Space: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

This article is compiled by Aashish Barnwal and reviewed by the GeeksforGeeks team. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 31 Mar, 2023