Kruskal's Minimum Spanning Tree Algorithm

What is Minimum Spanning Tree?

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. 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. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

What are the applications of the *Minimum Spanning Tree?*

See this for applications of MST.

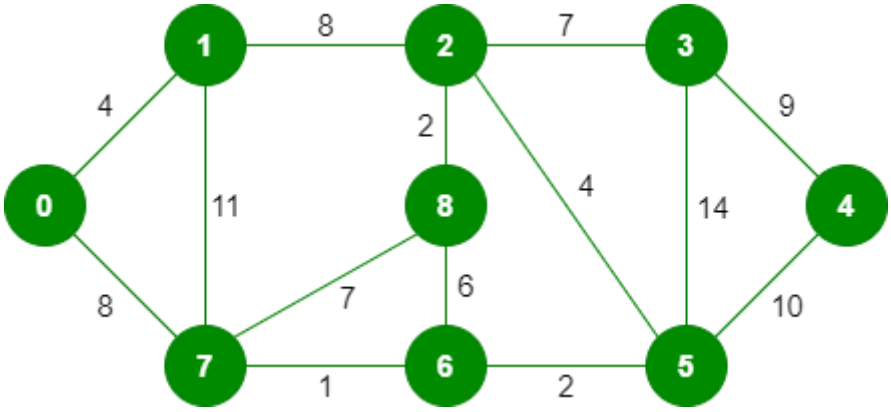
Below are the steps for finding MST using Kruskal's algorithm

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 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)

The algorithm is a Greedy Algorithm. 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: Consider the below input graph.



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

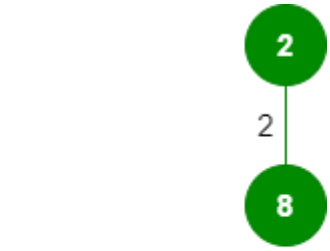
```
After sorting:

Weight   Src    Dest
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
```

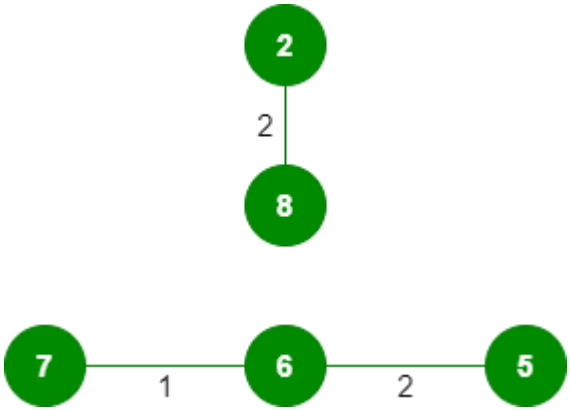
Now pick all edges one by one from the sorted list of edges

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

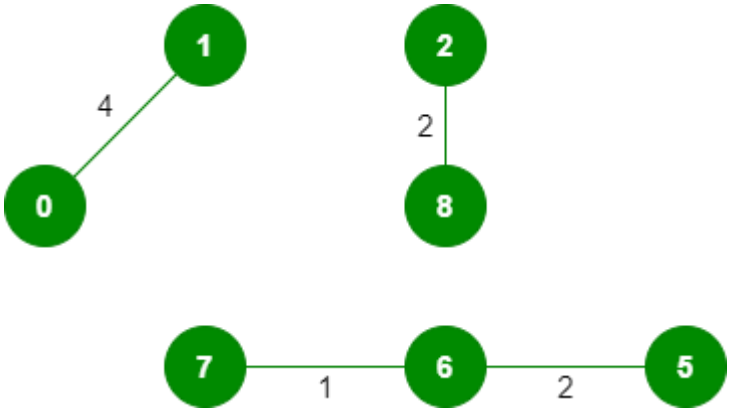
2. *Pick edge 8-2:* No cycle is formed, include it.



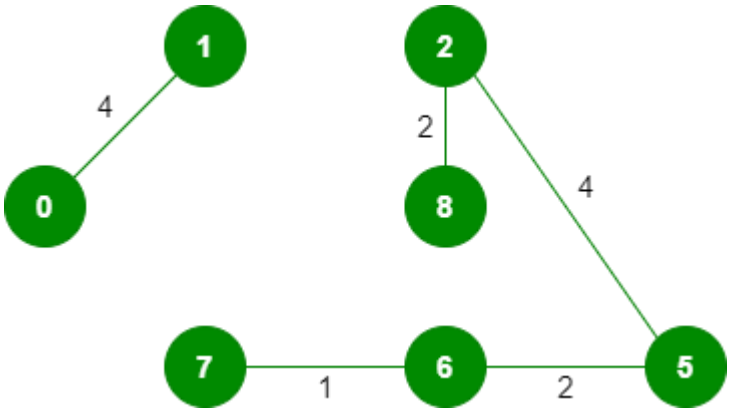
3. *Pick edge 6-5:* No cycle is formed, include it.



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

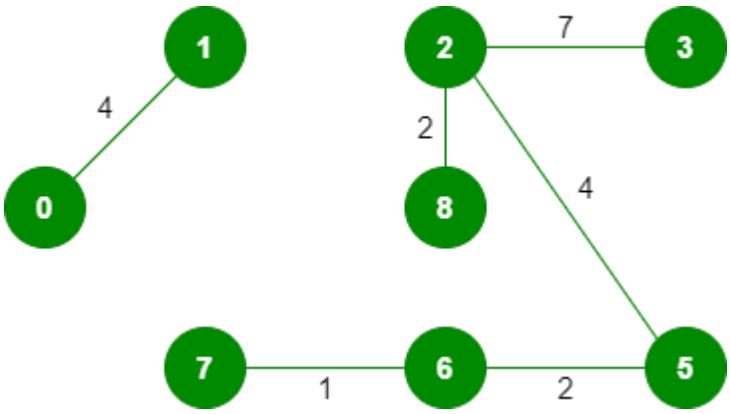


5. *Pick edge 2-5:* No cycle is formed, include it.

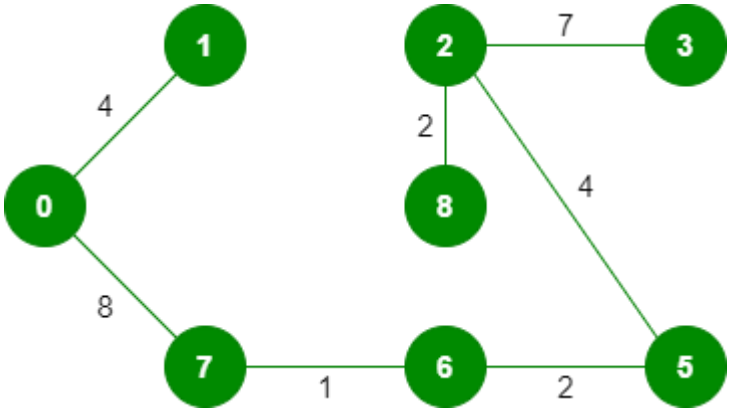


6. *Pick edge 8-6:* Since including this edge results in the cycle, discard it.

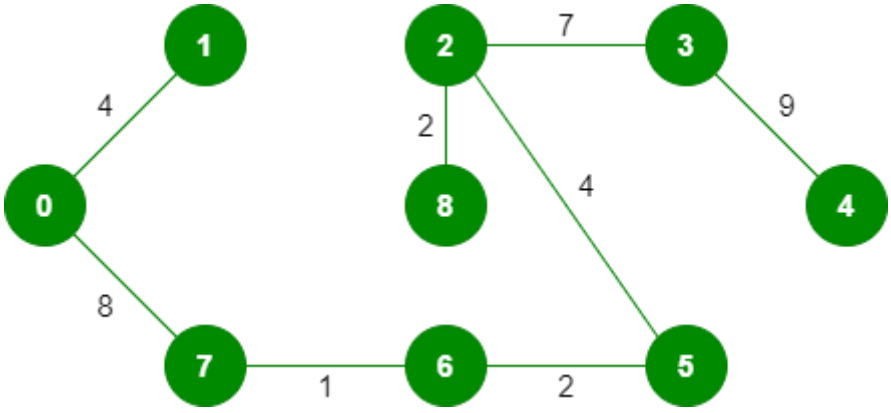
7. *Pick edge 2-3:* No cycle is formed, include it.



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



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



Since the number of edges included equals (V – 1), the algorithm stops here.

```
class Graph:

    def __init__(self, vertices):
        self.V = vertices  # No. of vertices
        self.graph = []  # default dictionary
        # to store graph
```

```
    # function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def getGraph(self):
        return self.graph


def krushKalAlgorithmMST(graph, v):
    graph = sorted(graph, key=lambda x: x[2])
    parent = [i for i in range(v + 1)]
    rank = [1 for i in range(v + 1)]
    ans = 0
    for i in range(len(graph)):
        u, v, w = graph[i]
        if union(u, v, parent, rank):
            ans += w
    return ans


def union(x, y, parent, rank):
    lx = find(x, parent)
    ly = find(y, parent)

    if lx != ly:
        if rank[lx] < rank[ly]:
            parent[lx] = ly
        elif rank[lx] > rank[ly]:
            parent[ly] = lx
        else:
            parent[lx] = ly
            rank[ly] += 1
        return True
    else:
        return False


def find(x, parent):
    if parent[x] == x:
        return x
    temp = find(parent[x], parent)
    parent[x] = temp
    return temp
```

```
graph = Graph(7)
graph.addEdge(1, 2, 10)
graph.addEdge(2, 3, 10)
graph.addEdge(3, 4, 10)
graph.addEdge(1, 4, 40)
graph.addEdge(4, 5, 2)
graph.addEdge(5, 6, 3)
graph.addEdge(6, 7, 3)
graph.addEdge(5, 7, 8)
# print(type(graph))
g = graph.getGraph()
print(krushKalAlgorithmMST(g, 7))

# graph.addEdge(0 ,1 ,10)
# graph.addEdge(1 ,2 ,10)
# graph.addEdge(2 ,3 ,10)
# graph.addEdge(0 ,3 ,40)
# graph.addEdge(3 ,4 ,2)
# graph.addEdge(4 ,5 ,3)
# graph.addEdge(5 ,6 ,3)
# graph.addEdge(4 ,6 ,8)
```