Kruskal's Minimum Spanning Tree

Definition of Minimum Spanning Tree: A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a *connected, edge-weighted undirected* graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

Reference: https://en.wikipedia.org/wiki/Minimum_spanning_tree

Weight of a spanning tree: the sum of weights given to each edge of the spanning tree.

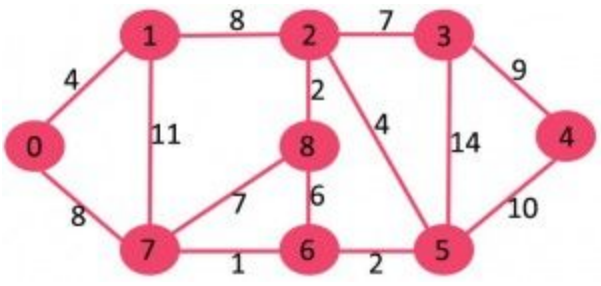
Number of Edges in a Minimum Spanning Tree: A minimum spanning tree has (N_V-1) edges, where N_V is the number of vertices in the given graph.

Kruskal's Minimum Spanning Tree operation principle: Kruskal's algorithm functions as follows:

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 (N_V-1) edges in the spanning tree.

Reference:
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

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.

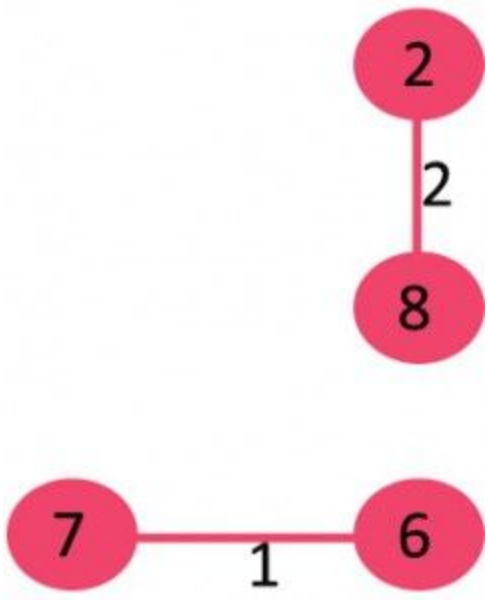
`After sorting:`

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

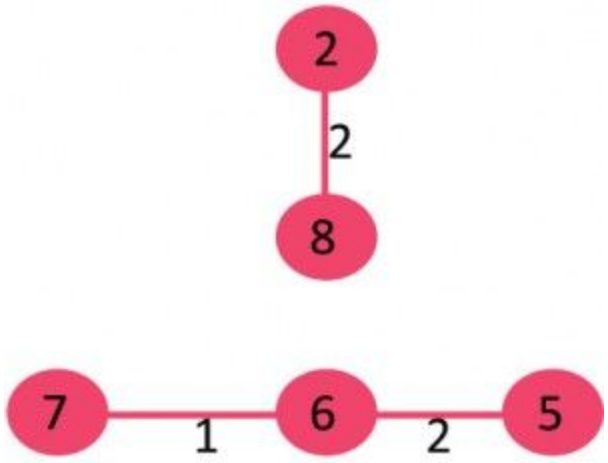
Now pick all edges one by one from sorted list of edges

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

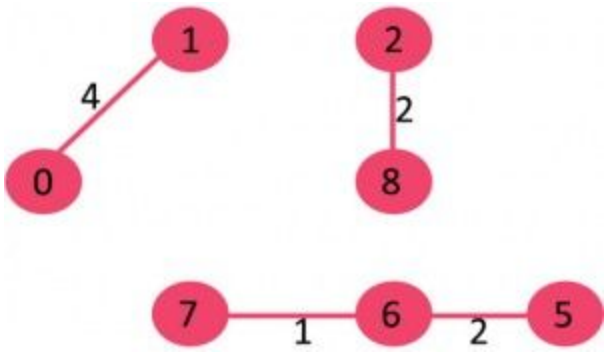
2. Pick edge 8-2: No cycle is formed, include it.



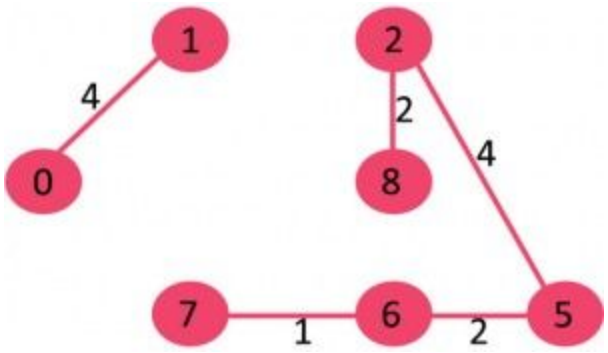
3. Pick edge 6-5: No cycle is formed, include it.



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

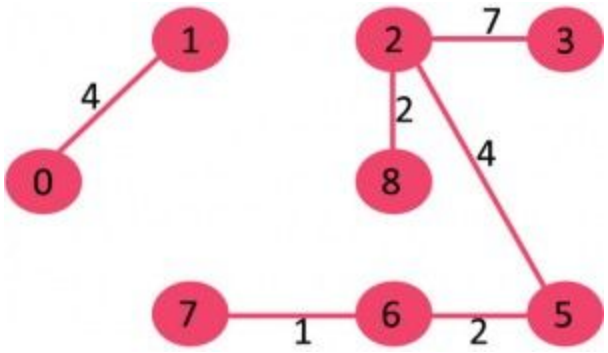


5. Pick edge 2-5: No cycle is formed, include it.



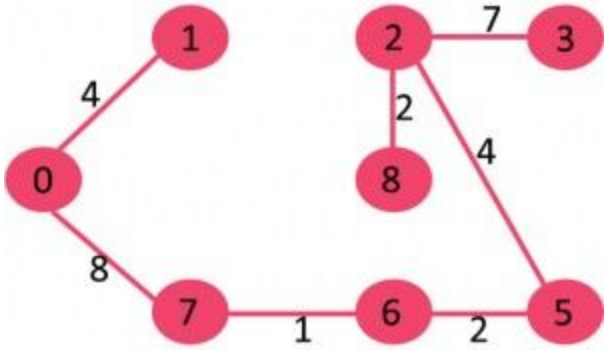
6. Pick edge 8-6: Since including this edge results in cycle, discard it.

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



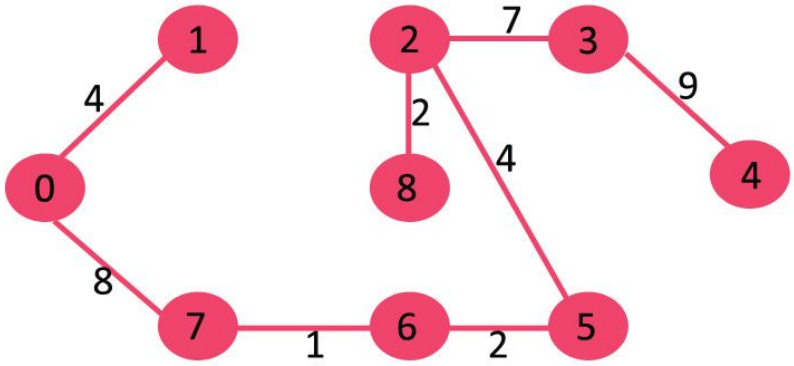
8. Pick edge 7-8: Since including this edge results in cycle, discard it.

9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in cycle, discard it.

11. Pick edge 3-4: No cycle is formed, include it.



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

Task: Considering the following code subset for the graph data structure and the further provided driving program - implement the `KruskalMST(graph)` method.

```
// C++ program for Kruskal's algorithm to find Minimum Spanning Tree
// of a given connected, undirected and weighted graph
#include <bits/stdc++.h>
using namespace std;

// 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;
}

/*
Your Code Here
*/

// Driver code
int main()
{
    /* Let us create following weighted graph
             10
        0--------1
```

```
        | \ |
    6| 5\ |15
        | \ |
        2--------3
            4 */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    Graph* graph = createGraph(V, E);


    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}

// This code is contributed by rathbhupendra
```

References:

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/ which also references

http://www.ics.uci.edu/~eppstein/161/960206.html

http://en.wikipedia.org/wiki/Minimum_spanning_tree