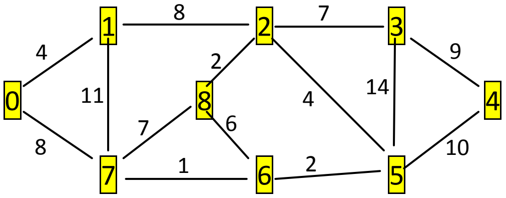
Kruskal's Minimum Spanning Tree Algorithm:

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

Steps for finding MST using Kruskal's algorithm

1. Remove all loops.
2. Remove all parallel edges and keep the edge that has least weight.
3. Sort all the edges in non-decreasing (Ascending) order of their weight.
4. 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.
5. Repeat step #4 until there are (V-1) edges in the spanning tree.

EXAMPLE:

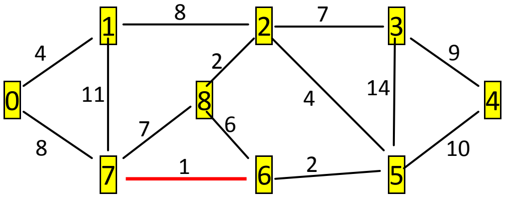


S: Source
D: Destination
W: Weight

S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

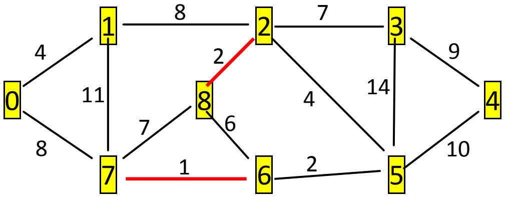
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



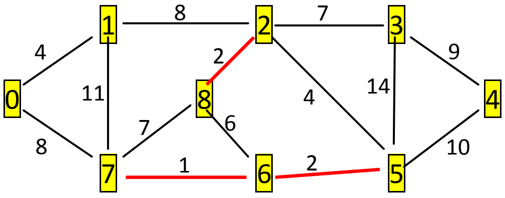
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



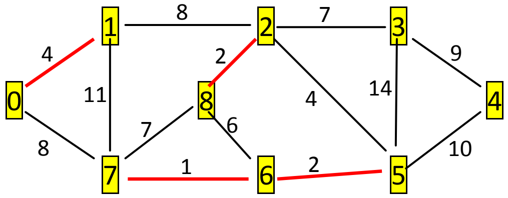
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



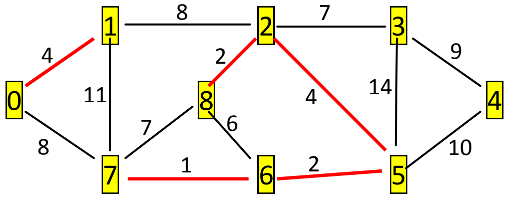
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



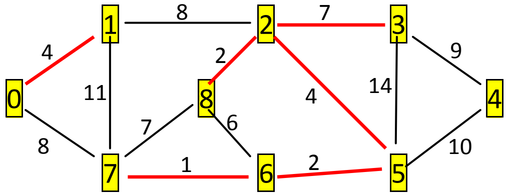
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



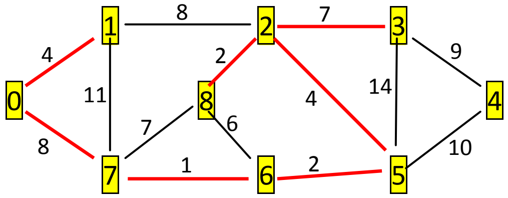
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

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



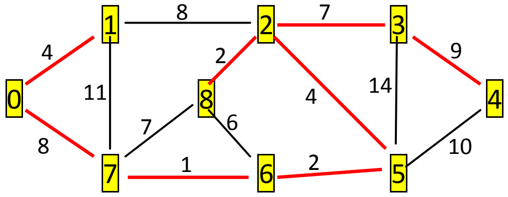
S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

Since 7-8 results in cycle, discard it.
Pick edge 0-7: No cycle is formed, include it.

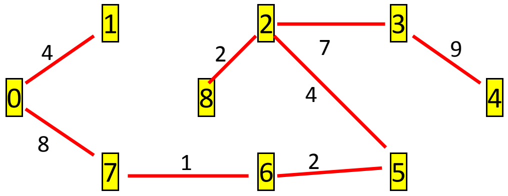


S	7	8	6	0	2	8	2	7	0	1	3	5	1	3
D	6	2	5	1	5	6	3	8	7	2	4	4	7	5
W	1	2	2	4	4	6	7	7	8	8	9	10	11	14

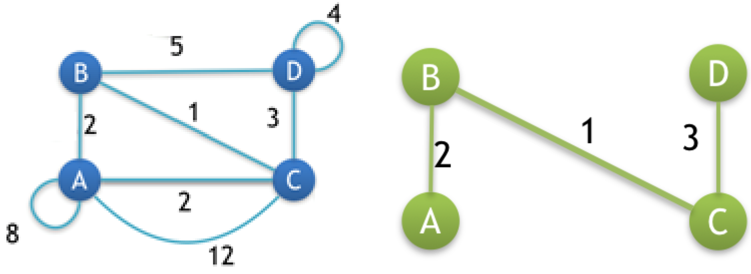
Since 1-2 results in cycle, discard it.
Pick edge 3-4: No cycle is formed, include it.



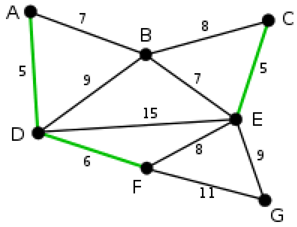
Vertices: 9
Edges selected: 8



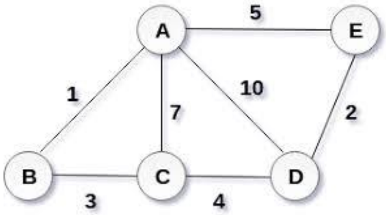
Task 1:



Task 2:



Task 3:

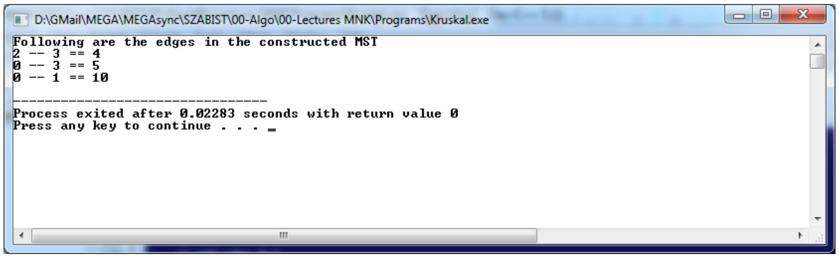


Programming/Implementation Task: Use the following code to find out the MSTs of the example discussed today.

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

class Edge
{
   public:
   int src, dest, weight;
};

class Graph
{
   public:
   int V, E;
   Edge* edge;
};
```

```
Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;

    graph->edge = new Edge[E];

    return graph;
}

class subset
{
    public:
    int parent;
    int rank;
};

int find(subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int myComp(const void* a, const void* b)
{
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}

void KruskalMST(Graph* graph)
{
    int V = graph->V;
    Edge result[V];
    int e = 0;
    int i = 0;

    // 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);

    subset *subsets = new subset[( V * sizeof(subset) )];

    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    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
    }

    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;
}
```

```
int main()
{
   int V = 4;
   int E = 5;
   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;
}
```


D:\GMail\MEGA\MEGAsync\SZABIST\00-Algo\00-Lectures MNK\Programs\Kruskal.exe

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

--------------------------------
Process exited after 0.02283 seconds with return value 0
Press any key to continue . . . _
```