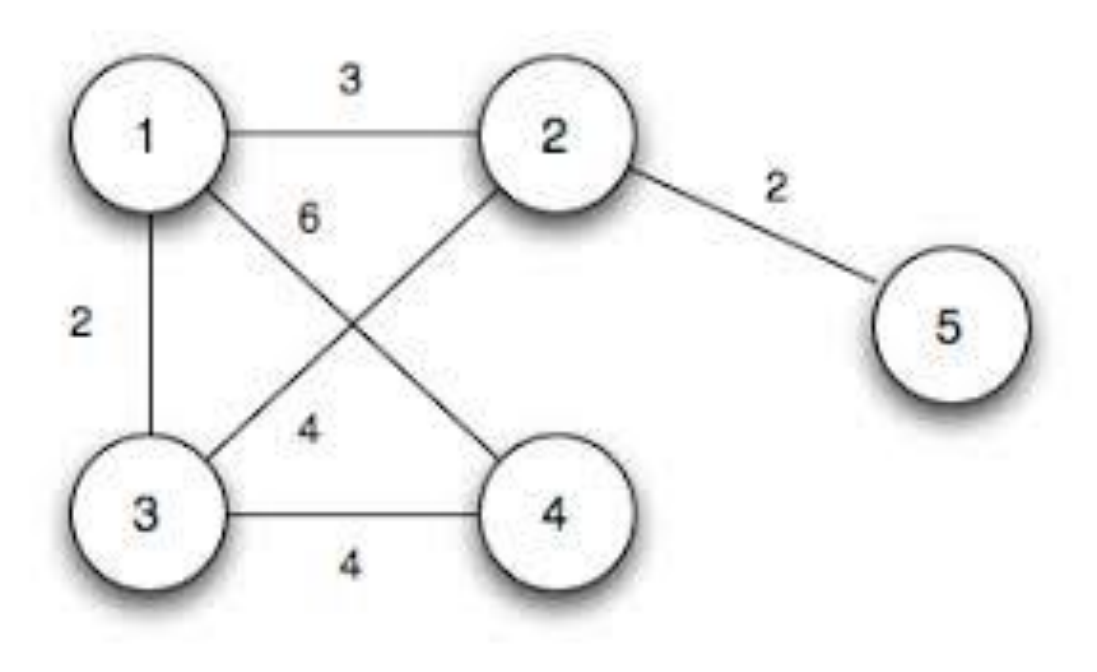


CTA PRACTICAL 5

20MCED08

Practical 5

Implement Kruskal's algorithm to find MST using greedy approach.

Introduction

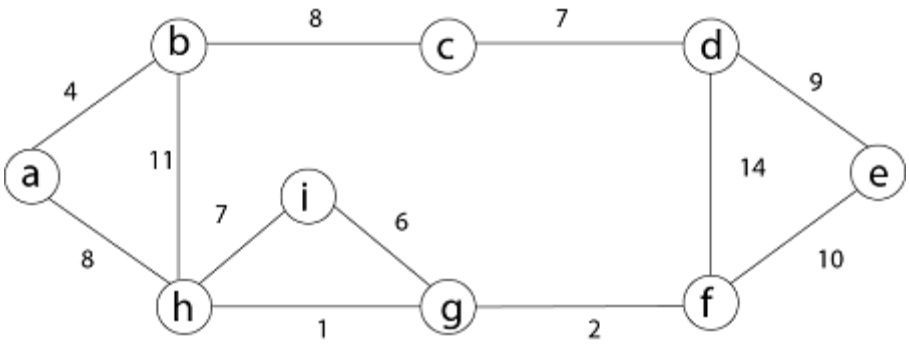
An algorithm to construct a Minimum Spanning Tree for a connected weighted graph. It is a Greedy Algorithm. The Greedy Choice is to put the smallest weight edge that does not because a cycle in the MST constructed so far.

Greedy method:

Steps for finding MST using Kruskal's Algorithm:

1. Arrange the edge of G in order of increasing weight.
2. Starting only with the vertices of G and proceeding sequentially add each edge which does not result in a cycle, until (n - 1) edges are used.
3. EXIT.

For Example: (IN THIS PRACTICAL) Find the Minimum Spanning Tree of the following graph using Kruskal's algorithm.



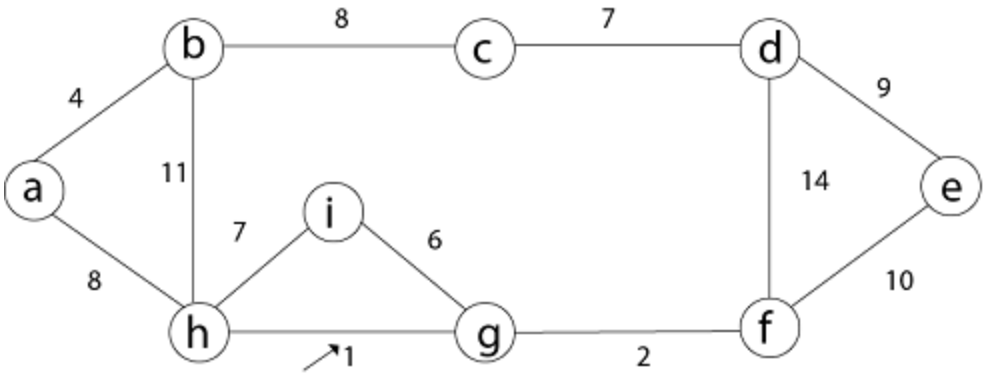
Solution: First we initialize the set A to the empty set and create |v| trees, one containing each vertex with MAKE-SET procedure. Then sort the edges in E into order by non-decreasing weight.

There are 9 vertices and 12 edges. So MST formed (9-1) = 8 edges

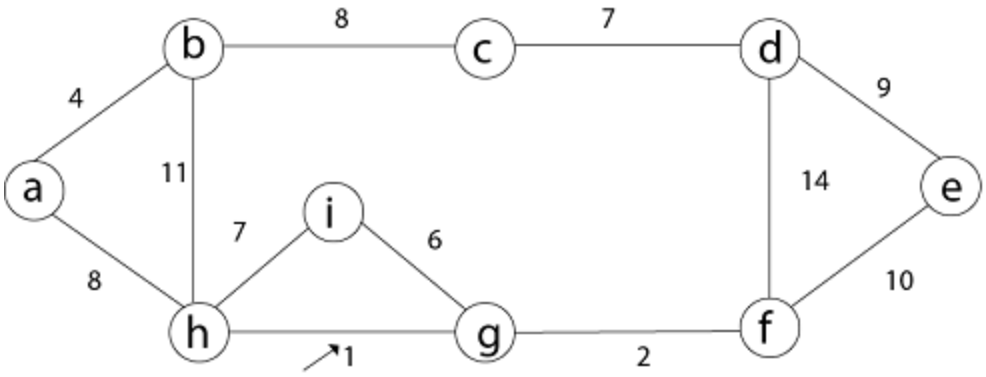
Weight	Source	Destination
1	h	g
2	g	f
4	a	b
6	i	g
7	h	i
7	c	d
8	b	c
8	a	h
9	d	e
10	e	f
11	b	h
14	d	f

Now, check for each edge (u, v) whether the endpoints u and v belong to the same tree. If they do then the edge (u, v) cannot be supplementary. Otherwise, the two vertices belong to different trees, and the edge (u, v) is added to A, and the vertices in two trees are merged in by union procedure.

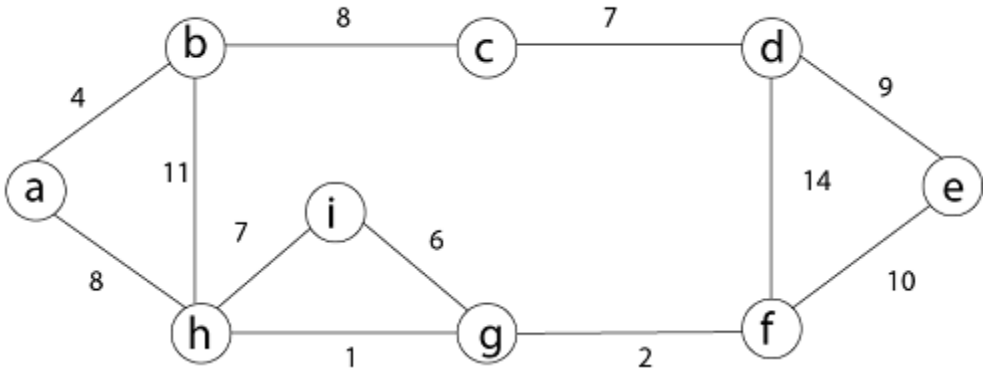
Step1: So, first take (h, g) edge



Step 2: then (g, f) edge.

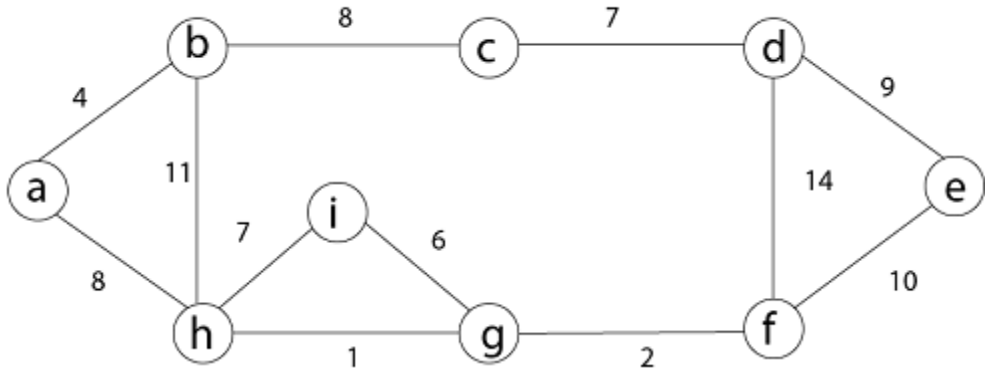


Step 3: then (a, b) and (i, g) edges are considered, and the forest becomes



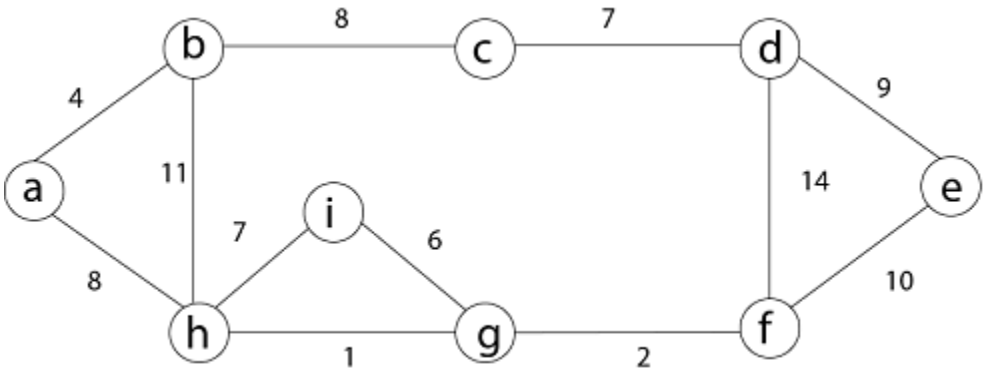
Step 4: Now, edge (h, i). Both h and i vertices are in the same set. Thus it creates a cycle. So this edge is discarded.

Then edge (c, d), (b, c), (a, h), (d, e), (e, f) are considered, and the forest becomes.



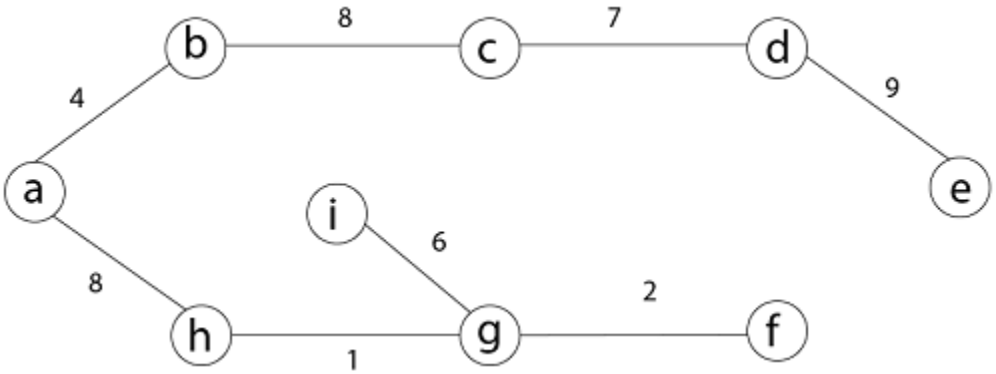
Step 5: In (e, f) edge both endpoints e and f exist in the same tree so discarded this edge. Then (b, h) edge, it also creates a cycle.

Step 6: After that edge (d, f) and the final spanning tree is shown as in dark lines.



Step 7: This step will be required Minimum Spanning Tree because it contains all the 9 vertices and (9 - 1) = 8 edges

1. e → f, b → h, d → f [cycle will be formed]



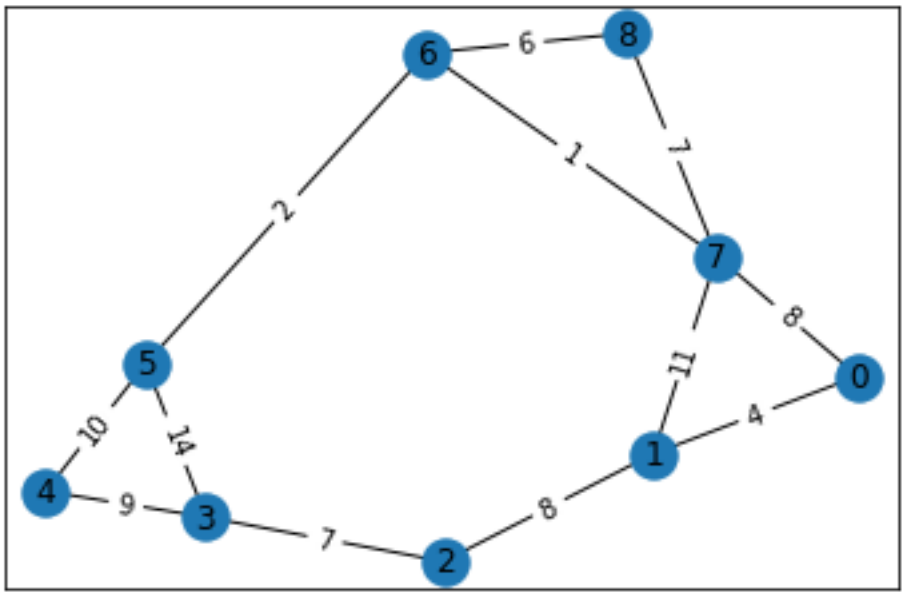
1) Making Change Problem Using Greedy Method

```
class DisjointSet:

    parent = {}
    def __init__(self, N):
        # create N disjoint sets (one for each vertex)
        for i in range(N):
            self.parent[i] = i #|1|2|2|2|

    # Find the root of the set in which element k belongs
    def Find(self, k):

        # if k is root
        if self.parent[k] == k:
            return k

        # recur for parent until we find root
        return self.Find(self.parent[k])

    # Perform Union of two subsets
    def Union(self, a, b):

        # find root of the sets in which elements
        # x and y belongs
        x = self.Find(a)
        y = self.Find(b)

        self.parent[x] = y


# construct MST using Kruskal's algorithm
def KruskalAlgo(edges, N):
    # stores edges present in MST
    MST = []
    ds = DisjointSet(N)


    index = 0

    # MST contains exactly V-1 edges
    while len(MST) != N - 1:

        # consider next edge with minimum weight from the graph
        (src, dest, weight) = edges[index]
        index = index + 1

        # find root of the sets to which two endpoint
        # vertices of next_edge belongs
        x = ds.Find(src)
        y = ds.Find(dest)
# [(0 1 1),(3 2 1)(1 3 2)(2 0 2)]   x,y = [(0,1)(3,2)(1,2)(2,2)]
        # different connected components and can be included in MST
        if x != y:
            MST.append((src, dest, weight))
            ds.Union(x, y)
            #print(ds)
```

```
    return MST


import networkx as nx
G = nx.Graph()
if __name__ == '__main__':
    edges = []
    N = 0
    choice = int(input("Press 1 to use default graph\nPress 2 to use custom graph\n"))
    if choice == 1:

        N = 4
        edges = [
#             (1, 2, 7), (1, 4, 6), (4, 2, 9), (4, 3, 8), (2, 3, 6)
            (0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)
        ]

    elif choice == 2:
        N = int(input("Enter the number of vertices = "))
        E = int(input("Enter the number of edges = "))

        for i in range(E):
            inValue = input("Enter the tuple (E.g u,v,w)")
            edges.append(tuple(list(map(int, inValue.strip().split()))))
            #print(edges)

    G.add_weighted_edges_from(edges,weight="weight")
    #nx.draw_networkx(G, with_labels = True)
    pos=nx.spring_layout(G) # pos = nx.nx_agraph.graphviz_layout(G)
    nx.draw_networkx(G,pos)
    labels = nx.get_edge_attributes(G,'weight')
    nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
    # sort edges by increasing weight
    edges.sort(key=lambda x: x[2])

    # Number of vertices in the graph
    mcst = 0

    # construct graph
    #print(N)
    #print(edges)

    mst = KruskalAlgo(edges, N)
    #print(mst)
    for u,v,w in mst:
        mcst+=w


    print("\nPrinting the Graph: ")
```

Output for rs = 93:

```
Press 1 to use default graph
Press 2 to use custom graph
2
Enter the number of vertices = 9
Enter the number of edges = 12
Enter the tuple (E.g u,v,w)0 1 4
Enter the tuple (E.g u,v,w)0 7 8
Enter the tuple (E.g u,v,w)1 7 11
Enter the tuple (E.g u,v,w)7 8 7
Enter the tuple (E.g u,v,w)8 6 6
Enter the tuple (E.g u,v,w)6 7 1
Enter the tuple (E.g u,v,w)1 2 8
Enter the tuple (E.g u,v,w)2 3 7
Enter the tuple (E.g u,v,w)5 6 2
Enter the tuple (E.g u,v,w)3 5 14
Enter the tuple (E.g u,v,w)3 4 9
Enter the tuple (E.g u,v,w)5 4 10

Printing the Graph:
```



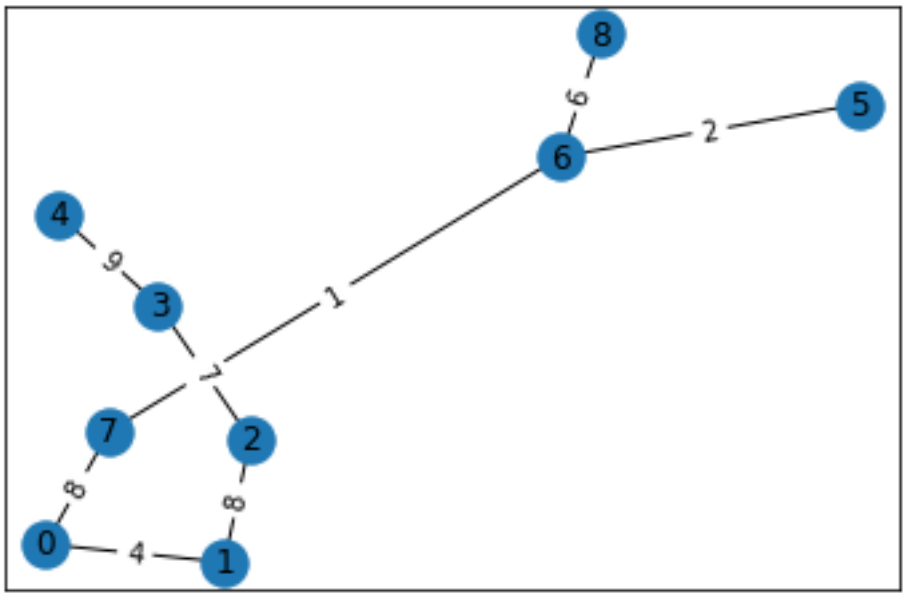
```
print("Minimum Cost Spanning Tree  = ",mcst)
```

```
Minimum Cost Spanning Tree  =  45
```

```
print("\nEdges in Minimum Spanning Tree  = ",mst)
```

```
Edges in Minimum Spanning Tree  =  [(6, 7, 1), (5, 6, 2), (0, 1, 4), (8, 6, 6),
(2, 3, 7), (0, 7, 8), (1, 2, 8), (3, 4, 9)]
```

```
mstg = nx.Graph()
print("\nPrinting the Minimum Spanning Tree\n")
# print(mst)
mstg.add_weighted_edges_from(mst,weight="weight")
#nx.draw_networkx(G, with_labels = True)
pos=nx.spring_layout(mstg) # pos = nx.nx_agraph.graphviz_layout(G)
nx.draw_networkx(mstg,pos)
labels = nx.get_edge_attributes(mstg,'weight')
nx.draw_networkx_edge_labels(mstg,pos,edge_labels=labels)
```



Observation:

Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take at most O(LogV) time. So overall complexity is O(ELogE + ELogV) time.