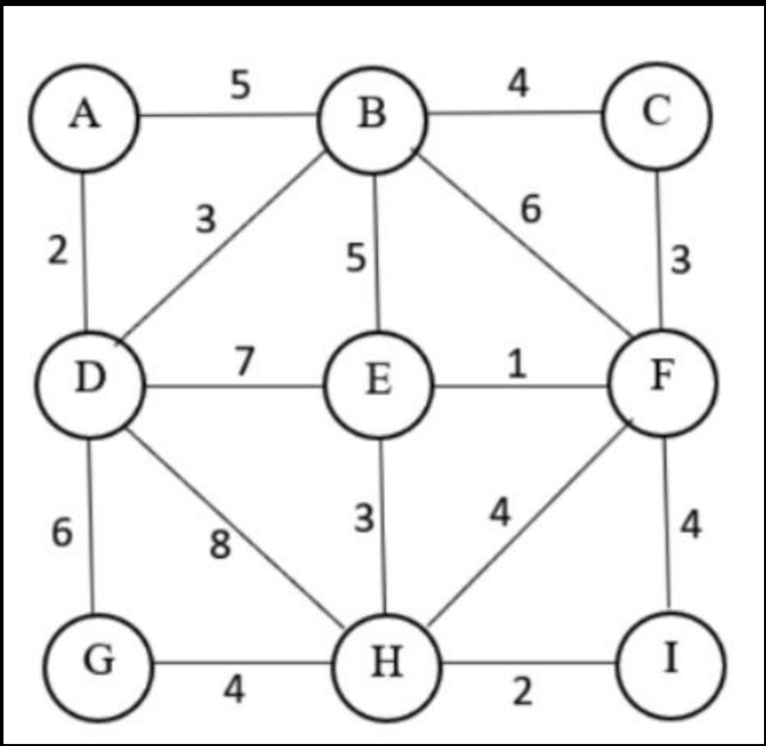
Name: Girish S

Roll No.: AM.EN.U4AIE22044

-22AIE203-
DATA STRUCTURES AND ALGORITHMS -2

ASSIGNMENT 4 – Minimum Spanning Tree

1. **Find the minimum spanning tree for the graph given below using both Prim's and Kruskal's algorithms.**



Graph Representation using class

```python
## [ Minimum Spanning Tree ]

class Graph:
    def __init__(self, adj_dic={}):
        self.adj_dic = adj_dic

    def child(self, parent):
        return self.adj_dic[parent]

    def vertices(self):
        return list(self.adj_dic)

    def Edges(self):
        edges = {(i, j): self.adj_dic[i][j] for i in self.vertices() for j in self.child(i)}
        return edges

    def PathWeight(self, u, v):
        return self.adj_dic[u][v]
```

```python
    def addVertices(self, *v):
        for i in v:
            self.adj_dic[i] = {}
    def addEdges(self, d={}):# addEdges({("A", "B"):2})
        for i in d:
            if i[0] not in self.adj_dic and i[1] not in self.adj_dic:
                self.adj_dic[i[0]] = {i[1]:d[i]}
                self.adj_dic[i[1]] = {i[0]:d[i]}
            elif i[0] not in self.adj_dic and i[1] in self.adj_dic:
                self.adj_dic[i[0]] = {i[1]:d[i]}
                self.adj_dic[i[1]][i[0]] = d[i]
            elif i[1] not in self.adj_dic and i[0] in self.adj_dic:
                self.adj_dic[i[1]] = {i[0]:d[i]}
                self.adj_dic[i[0]][i[1]] = d[i]
            else:
                self.adj_dic[i[1]][i[0]] = d[i]
                self.adj_dic[i[0]][i[1]] = d[i]
    def __str__(self):
        return str(self.adj_dic).replace("}, ", "}\n")[1:-1].replace("'", "")
```

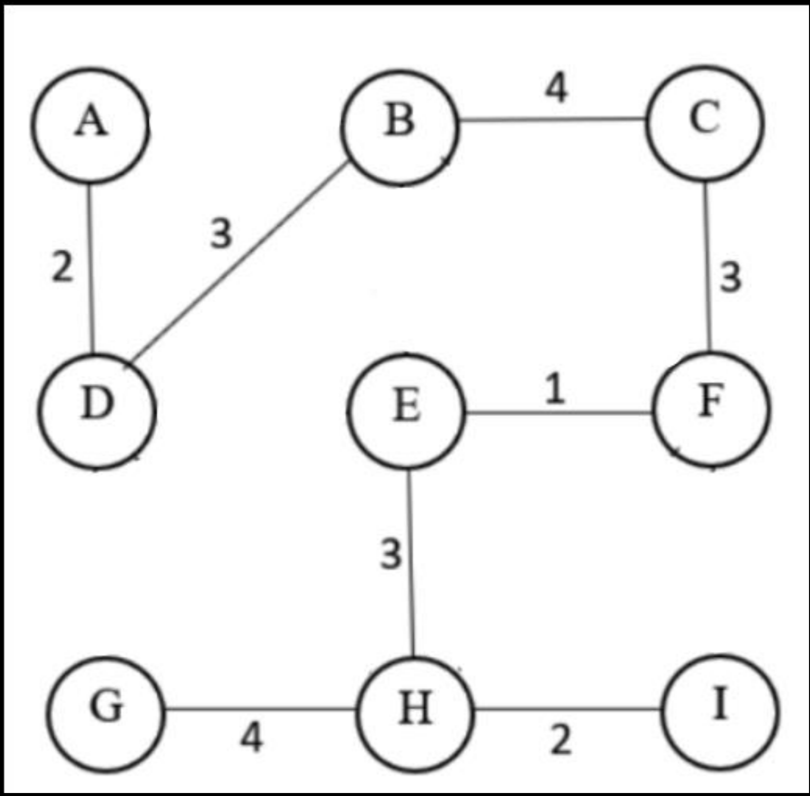
```python
dic = {
        'A': {'B': 5, 'D': 2},
        'B': {'A': 5, 'C': 4, 'D': 3, 'E': 5, 'F': 6},
        'C': {'B': 4, 'F': 3},
        'D': {'A': 2, 'B': 3, 'E': 7, 'G': 6, 'H':8},
        'E': {'B': 5, 'D': 7, 'F': 1, 'H': 3},
        'F': {'B': 6, 'C': 3, 'E': 1, 'H': 4, 'I': 4},
        'G': {'D': 6, 'H': 4},
        'H': {'D': 8, 'E': 3, 'F': 4, 'G': 4, 'I': 2},
        'I': {'F': 4, 'H': 2}
}

graph = Graph(dic)
```

Prim’s Algorithm for Minimum Spanning Tree

```python
def Prims(graph, source):
    spanTree = Graph()
    spanTree.addVertices(source)
    Leafs = [source]
    Edges = []
    pathCost = 0
    while spanTree.vertices()!=graph.vertices():
        availPaths = []
        for i in Leafs:
            children = graph.child(i)
            for j in children:
                if j in Leafs:
                    continue
                if {i, j} not in availPaths:
                    availPaths.append({i, j})
        paths = {tuple(i): graph.PathWeight(tuple(i)[0], tuple(i)[1]) for i in availPaths}
        if paths == {}:
            break
        minPath = min(paths, key=lambda k: paths[k])
        if minPath[0] in Leafs:
            Leafs.append(minPath[1])
        else:
            Leafs.append(minPath[0])
        Edges.append(minPath)
        spanTree.addEdges({minPath:paths[minPath]})
        pathCost+=paths[minPath]
    print(spanTree)
    print(f"Path Cost : {pathCost}")
    return spanTree, pathCost
```

```
A: {B: 5, D: 2}
B: {A: 5, C: 4, D: 3, E: 5, F: 6}
C: {B: 4, F: 3}
D: {A: 2, B: 3, E: 7, G: 6, H: 8}
E: {B: 5, D: 7, F: 1, H: 3}
F: {B: 6, C: 3, E: 1, H: 4, I: 4}
G: {D: 6, H: 4}
H: {D: 8, E: 3, F: 4, G: 4, I: 2}
I: {F: 4, H: 2}
```

```
A: {D: 2}
D: {A: 2, B: 3}
B: {D: 3, C: 4}
C: {B: 4, F: 3}
F: {C: 3, E: 1}
E: {F: 1, H: 3}
H: {E: 3, I: 2, G: 4}
I: {H: 2}
G: {H: 4}
Path Cost : 22
```



Kruskal’s Algorithm for Minimum Spanning Tree

```python
def find_set(parent, sets):
     if sets[parent] != parent:
          sets[parent] = find_set(sets[parent], sets)
     return sets[parent]

def union_sets(u, v, sets):
     root_u = find_set(u, sets)
     root_v = find_set(v, sets)
     sets[root_u] = root_v
```

```python
def kruskal(graph):
     spanTree = Graph()
     sets = {v: v for v in graph.vertices()}
     Edges = sorted(graph.Edges().items(), key=lambda x: x[1])
     pathCost = 0
     for (u, v), weight in Edges:
          if find_set(u, sets) != find_set(v, sets):
               spanTree.addEdges({(u, v):weight})
               pathCost+=weight
               union_sets(u, v, sets)
     print(spanTree)
     print(f"Path Cost : {pathCost}")
     return spanTree, pathCost


kruskal(graph)
```

```
E: {F: 1, H: 3}
F: {E: 1, C: 3}
A: {D: 2}
D: {A: 2, B: 3}
H: {I: 2, E: 3, G: 4}
I: {H: 2}
B: {D: 3, C: 4}
C: {F: 3, B: 4}
G: {H: 4}
Path Cost : 22
```