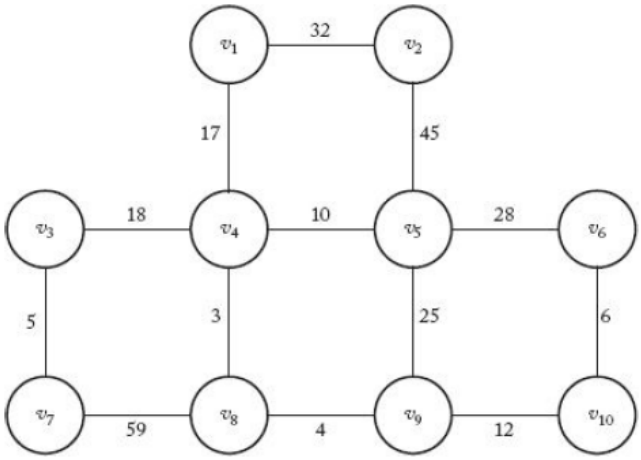
COMP319-004 Algorithm 1

Assignment #3

마감: 6/3 (금) 23:59

성명 (학번): 이인영 2016114463



1. Prim 알고리즘으로, Minimum Spanning Tree를 찾는 과정을 보여주세요. (50pt)

```python
# Prim's Algorithm in Python

INF = 9999999
# number of vertices in graph
N = 10
#creating graph by adjacency matrix method
G = [[0, 32, 0, 17, 0, 0, 0, 0, 0, 0],
     [32, 0, 0, 0, 45, 0, 0, 0, 0, 0],
     [0, 0, 0, 18, 0, 0, 5, 0, 0, 0],
     [17, 0, 18, 0, 10, 0, 0, 3, 0, 0],
     [0, 45, 0, 10, 0, 28, 0, 0, 25, 0],
     [0, 0, 0, 0, 28, 0, 0, 0, 0, 6],
     [0, 0, 5, 0, 0, 0, 0, 59, 0, 0],
     [0, 0, 0, 3, 0, 0, 59, 0, 4, 0],
     [0, 0, 0, 0, 25, 0, 0, 4, 0, 12],
     [0, 0, 0, 0, 0, 6, 0, 0, 12, 0]]

selected_node = [0]*N

no_edge = 0

selected_node[0] = True

# printing for edge and weight
print("Edge : Weight\n")
while (no_edge < N - 1):

    minimum = INF
    a = 0
    b = 0
    for m in range(N):
        if selected_node[m]:
            for n in range(N):
                if ((not selected_node[n]) and G[m][n]):
                    # not in selected and there is an edge
                    if minimum > G[m][n]:
                        minimum = G[m][n]
                        a = m
                        b = n
    print("v" + str(1+a) + "-" + "v"+ str(1+b) + ":" + str(G[a][b]))
    selected_node[b] = True
    no_edge += 1
```

```
Edge : Weight

v1-v4:17
v4-v8:3
v8-v9:4
v4-v5:10
v9-v10:12
v10-v6:6
v4-v3:18
v3-v7:5
v1-v2:32
```

2. Kruskal 알고리즘으로, Minimum Spanning Tree를 찾는 과정을 보여주세요. (50pt)

```python
#Initializing the Graph Class
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
        self.nodes = []
        self.MST = []

    def addEdge(self, s, d, w):
        self.graph.append([s, d, w])

    def addNode(self, value):
        self.nodes.append(value)

    def printSolution(self,s,d,w):
        print("Edge : Weight\n")
        for s, d, w in self.MST:
            print("%s-%s:%s" % (s, d, w))

    def kruskalAlgo(self):
        i, e = 0, 0
        ds = DisjointSet(self.nodes)
        self.graph = sorted(self.graph, key=lambda item: item[2])
        while e < self.V - 1:
            s, d, w = self.graph[i]
            i += 1
            x = ds.find(s)
            y = ds.find(d)
            if x != y:
                e += 1
                self.MST.append([s,d,w])
                ds.union(x,y)
        self.printSolution(s,d,w)

#Implementing Disjoint Set data structure and its functions
class DisjointSet:
    def __init__(self, vertices):
        self.vertices = vertices
        self.parent = {}
        for v in vertices:
            self.parent[v] = v
        self.rank = dict.fromkeys(vertices, 0)

    def find(self, item):
        if self.parent[item] == item:
            return item
        else:
            return self.find(self.parent[item])

    def union(self, x, y):
        xroot = self.find(x)
        yroot = self.find(y)
        if self.rank[xroot] < self.rank[yroot]:
            self.parent[xroot] = yroot


        elif self.rank[xroot] > self.rank[yroot]:
            self.parent[yroot] = xroot
        else:
            self.parent[yroot] = xroot
            self.rank[xroot] += 1

    #Function to implement Kruskal's Algorithm


g = Graph(10)
for i in range(1,11) :
    g.addNode("v"+str(i))
g.addEdge("v1", "v2", 32)
g.addEdge("v1", "v4", 17)
g.addEdge("v2", "v5", 45)
g.addEdge("v3", "v4", 18)
g.addEdge("v3", "v7", 5)
g.addEdge("v4", "v5", 10)
g.addEdge("v4", "v8", 3)
g.addEdge("v5", "v6", 28)
g.addEdge("v5", "v9", 25)
g.addEdge("v6", "v10", 6)
g.addEdge("v7", "v8", 59)
g.addEdge("v8", "v9", 4)
g.addEdge("v9", "v10", 12)

g.kruskalAlgo()
```

```
Edge : Weight

v4-v8:3
v8-v9:4
v3-v7:5
v6-v10:6
v4-v5:10
v9-v10:12
v1-v4:17
v3-v4:18
v1-v2:32
```

- 끝 -