# Graph Algorithms and Related Data Structures

## Project 2


## Nisha Ramrakhyani (801208678)
## Zalak Panchal (801196881)


**Problem 1: Single-source Shortest Path Algorithm**

Dijkstra Algorithm:

In [1]:

```
import heapq
import collections

def shortestpath(edges,startnode,destnode,graph_check):

    graph = collections.defaultdict(set)

    if graph_check == 'D':
        for l,r,c in edges:
            graph[l].add((c,r))
    else:
        for l,r,c in edges:
            graph[l].add((c,r))
            graph[r].add((c,l))


    queue, visited = [(0, startnode, [])], set()

    heapq.heapify(queue)

    while queue:
        (cost, node, path) = heapq.heappop(queue)
        if node not in visited:
            visited.add(node)
            path = path + [node]
            if node == destnode:
                return (cost, path)
            for c, neighbour in graph[node]:

                if neighbour not in visited:
                    heapq.heappush(queue, (cost+int(c), neighbour, path))

    return float("inf")
def main():
    edges = []
    nodes=[]
    file=open("Dijkstra_Undirected_Graph/Input4.txt", "r+")
    lines=file.readlines()
    for line in lines:
        edges.append(line.split())

    first = list(edges[0])
    last= list(edges[-1])
    vertices = first[0]
    numofedges=first[1]
    g_check= first[2]
    edges.pop(0)


    if(len(last)==1):
        startnode=last[0]
        edges.pop()
    else:
        firstnode=list(edges[0])
        startnode=firstnode[0]
```

```
    print('\nSource vertex: '+str(startnode)+'\n')
    print('Nodes: '+str(vertices)+'\n')
    print('Edges: '+str(numofedges)+'\n')

    if g_check == 'U':
        print('Undirected Graph')
    else:
        print('Directed Graph')


    for i in edges:
        nodes.append(i[0])
    nodes.extend(y[1] for y in edges)
    destnode=list(set(nodes))
    for i in destnode:
        print ("\n"+str(startnode)+" -> "+str(i)+":")
        result = shortestpath(edges, startnode, i,g_check)
        print('Path Cost: '+str(result[0]))
        print('Path: '+str(result[1]))

if __name__== "__main__":
    main()
```

```
Source vertex: 0

Nodes: 9

Edges: 14

Undirected Graph

0 -> 1:
Path Cost: 4
Path: ['0', '1']

0 -> 8:
Path Cost: 14
Path: ['0', '1', '2', '8']

0 -> 4:
Path Cost: 21
Path: ['0', '7', '6', '5', '4']

0 -> 2:
Path Cost: 12
Path: ['0', '1', '2']

0 -> 3:
Path Cost: 19
Path: ['0', '1', '2', '3']

0 -> 0:
Path Cost: 0
Path: ['0']

0 -> 6:
```

```
Path Cost: 9
Path: ['0', '7', '6']

0 -> 7:
Path Cost: 8
Path: ['0', '7']

0 -> 5:
Path Cost: 11
Path: ['0', '7', '6', '5']
```

### Problem 2: Minimum Spanning Tree Algorithm

Prim's Algorithm:

In [4]:

```
#Program to find the Minimum Spanning Tree using Prim's Algorithm and Heap Da

import sys

#Implementation of Heap
class Heap:
    s = 0
    a = []
    p = {}
    key = {}

    def __init__(self):
        self.s = -1

    def heapify_up(self, i):
        while i>0:
            j = i//2
            if self.key[self.a[i]] < self.key[self.a[j]]:
                temp = self.a[i]
                self.a[i] = self.a[j]
                self.a[j] = temp
                self.p[self.a[i]] = i
                self.p[self.a[j]] = j
                i = j
            else:
                break

    def heapify_down(self,i):
        j=-1
        while 2*i <= self.s:
            if 2*i == self.s or self.key[self.a[2*i]] <= self.key[self.a[2*i
                j = 2*i
            else:
                j = 2*i + 1
            if self.key[self.a[j]] < self.key[self.a[i]]:
                temp = self.a[i]
                self.a[i] = self.a[j]
                self.a[j] = temp
                self.p[self.a[i]] = i
                self.p[self.a[j]] = j
                i = j
            else:
                break

    def decrease_key(self, v, key_value):
        self.key[v] = key_value
        self.heapify_up(self.p[v])

    def extract_min(self):
        ret = self.a[0]
        self.a[0]=self.a[self.s]
        self.p[self.a[0]] = 0
        self.s-=1
        if self.s>=0:
            self.heapify_down(0)
        return ret
```

```
    def insert(self, v, key_value):
        self.a.append(v)
        self.s +=1
        self.p[v] = self.s
        self.key[v] = key_value
        self.heapify_up(self.s)

    def printdata(self):
        print("Value of array a: ", self.a)
        print("Value of p: ", self.p)
        print("Value of key: ", self.key)

#Reading data from file
f = open("MST/input2_MST.txt")
lines = f.readlines()
f.close()


#Global variables for the Minimum Spanning Tree
Q = Heap()
n,m = map(int,lines[0].strip().split(" "))
edges = [[-1 for x in range(0,n+1)] for y in range(0,n+1)]
d = {}
pi = {}
S = []
V = []
total_weight = 0
edges_mst = [None]*(n-1)

#Add all nodes to the list V
for i in range(1,n+1):
    V.append(i)

#Add all edges to the edges matrix
for i in range(0,m):
    p,q,r = map(int,lines[i+1].strip().split(" "))
    edges[p][q] = r
    edges[q][p] = r #Adding the edges for both because it is an undirected gr

#Choosing the Arbitrary vertex as "Vertex 1"
d[1] = 0
Q.insert(1,d[1])

#Inserting the Infinity value for all other vertices
for i in range(1,n+1):
    d[i] = sys.maxsize
    Q.insert(i,d[i])

#Finding the Minimum Spanning Tree
while set(S)!=set(V):
    u = Q.extract_min()
    S.append(u)
    left = list(set(V) - set(S))
    for v in left:
        if edges[u][v] != -1:
            if edges[u][v] < d[v]:
```

```
                d[v] = edges[u][v]
                Q.decrease_key(v,d[v])
                pi[v] = u

#Adding the list of edges into a string array and calculating total weight
i = 0
for v in list(set(V) - set({1})):
    total_weight+=edges[v][pi[v]]
    if v < pi[v]:
        edges_mst[i] = "   " +str(v) + " - " + str(pi[v]) + "    \t" + str(ed
    else:
        edges_mst[i] = "   " +str(pi[v]) + " - " + str(v) + "    \t" + str(ed
    i+=1

print("Total cost of Minimum Spanning Tree (MST):",str(total_weight)+"\n")
print("Tree Edges   Cost \n")
for i in range(0, n-1):
    print(edges_mst[i])
```

```
Total cost of Minimum Spanning Tree (MST): 37

Tree Edges   Cost 

   1 - 2        4
   3 - 6        4
   3 - 4        7
   4 - 5        9
   6 - 7        2
   7 - 8        1
   1 - 8        8
   3 - 9        2
```