In [4]:

```
def  uniform_cost_search(goal, start):

    global graph,cost
    answer = []

    queue = []

    for i in range(len(goal)):
        answer.append(10**8)

    queue.append([0, start])

    visited = {}

    count = 0

    while (len(queue) > 0):

        queue = sorted(queue)
        p = queue[-1]

        del queue[-1]

        p[0] *= -1

        if (p[1] in goal):

            index = goal.index(p[1])

            if (answer[index] == 10**8):
                count += 1

            if (answer[index] > p[0]):
                answer[index] = p[0]

            del queue[-1]

            queue = sorted(queue)
            if (count == len(goal)):
                return answer

        if (p[1] not in visited):
            for i in range(len(graph[p[1]])):

                queue.append( [(p[0] + cost[(p[1], graph[p[1]][i])])* -1, graph[p[1]][i]]

        visited[p[1]] = 1

    return answer

if __name__ == '__main__':

    graph,cost = [[] for i in range(8)],{}

    graph[0].append(1)
    graph[0].append(3)
    graph[3].append(1)
    graph[3].append(6)
    graph[3].append(4)
    graph[1].append(6)
    graph[4].append(2)
    graph[4].append(5)
    graph[2].append(1)
    graph[5].append(2)
    graph[5].append(6)
    graph[6].append(4)

    cost[(0, 1)] = 2
    cost[(0, 3)] = 5
    cost[(1, 6)] = 1
    cost[(3, 1)] = 5
    cost[(3, 6)] = 6
    cost[(3, 4)] = 2
    cost[(2, 1)] = 4
    cost[(4, 2)] = 4
    cost[(4, 5)] = 3
    cost[(5, 2)] = 6
    cost[(5, 6)] = 3
    cost[(6, 4)] = 7

    goal = []

    goal.append(6)

    answer = uniform_cost_search(goal, 0)

    print("Minimum cost from 0 to 6 is = ",answer[0])
```

```
Minimum cost from 0 to 6 is =  3
```

In [5]:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

```
Following is the Depth-First Search
5
3
2
4
8
7
```

In [7]:

```
from collections import defaultdict

class Graph:

    def __init__(self,vertices):

        self.V = vertices

        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def DLS(self,src,target,maxDepth):

        if src == target : return True

        if maxDepth <= 0 : return False

        for i in self.graph[src]:
                if(self.DLS(i,target,maxDepth-1)):
                    return True
        return False

    def IDDFS(self,src, target, maxDepth):

        for i in range(maxDepth):
            if (self.DLS(src, target, i)):
                return True
        return False

g = Graph (7);
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(1, 4)
g.addEdge(2, 5)
g.addEdge(2, 6)

target = 6; maxDepth = 3; src = 0

if g.IDDFS(src, target, maxDepth) == True:
    print ("Target is reachable from source " + "within max depth")
else :
    print ("Target is NOT reachable from source " + "within max depth")
```

```
Target is reachable from source within max depth
```