Breadth First Search

In [1]:

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

visited = []  # list for visited nodes
queue = []    # initialize a queue

def bfs(visited,graph,node):  # function for bfs
    visited.append(node)
    queue.append(node)

    while queue:        # creating loop to visit each node
        m=queue.pop(0)
        print(m,end=" ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is the BFS traversal")
bfs(visited,graph,'5') # function calling
```

```
Following is the BFS traversal
5 3 7 2 4 8
```

Depth First Search

In [2]:

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

visited = set()  # to keep track of visited nodes

def dfs(visited,graph,node):  # function for dfs
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph [node]:
            dfs(visited,graph,neighbour)
print("Following is the DFS traversal")
dfs(visited,graph,'5')
```

```
Following is the DFS traversal
5
3
2
4
8
7
```

Selection Sort

In [9]:

```
def sort(nums): # function for selection sort
    for i in range(5): # higher index
        minpos = i     # variable to hold min position
        for j in range(i,6):  # sorted array
            if nums[j] < nums[minpos]:
                minpos = j # nw position

        temp = nums[i] # for swapping index value i with minpos
        nums[i] = nums[minpos]
        nums[minpos] = temp

        #print(nums)

nums = [5,3,8,6,7,2]
sort(nums)
print(nums)
```

```
[2, 3, 5, 6, 7, 8]
```

Job Scheduling

In [10]:

```
def printJobScheduling(arr,t): # function to schedule the jobs take 2 arguments array and no. of jobs to schedu
    n = len(arr)   # length of array
    for i in range(n):
        for j in range(n-1-i):  # Sort all jobs according to decreasing order of profit
            if arr[j][2] < arr[j+1][2]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    result = [False]* t # To keep track of free time slots
    job = ['-1']* t      # To store result (Sequence of jobs)

    for i in range(len(arr)):  # Iterate through all given jobs
        for j in range(min(t-1,arr[i][1]-1), -1, -1): # Find a free slot for this job (Note that we start from t
            if result[j] is False:   # Free slot found
                result[j] = True
                job[j] = arr[i][0]
                break
    print(job)   # print the sequence

if __name__ == '__main__':
    arr =  [                # Job Array
        ['a', 2, 15],
        ['b', 1, 27],
        ['c', 2, 10],
        ['d', 1, 100],
        ['e', 3, 150]
    ]

    print("Following is maximum profit sequence of jobs")
    printJobScheduling(arr,3)  # Function Call
```

```
Following is maximum profit sequence of jobs
['d', 'a', 'e']
```

Prim's Algorithm

In [5]:

```python
# Prims's Algorithm

INF = 9999999
# number of vertices in graph
N = 5
#creating graph by adjacency matrix method
G = [[0, 19, 5, 0, 0],
     [19, 0, 5, 9, 2],
     [5, 5, 0, 1, 6],
     [0, 9, 1, 0, 1],
     [0, 2, 6, 1, 0]]

selected_node = [0, 0, 0, 0, 0]

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(str(a) + "-" + str(b) + ":" + str(G[a][b]))
    selected_node[b] = True
    no_edge += 1
```

```
Edge : Weight

0-2:5
2-3:1
3-4:1
4-1:2
```

Dijkstra's Algorithm

In [6]:

```
# takes the graph and the starting node
# returns a list of distances from the starting node to every other node
from numpy import Inf
def Dijkstra(graph, start):
    l = len(graph)

    # initialize all node distances as infinite
    dist = [Inf for i in range(l)]

    # set the distance of starting node as 0
    dist[start] = 0

    # create a list that indicates if a node is visited or not
    vis = [False for i in range(l)]

    # iterate over all the nodes
    for i in range(l):

        # set u=-1 to indicate a current starting node
        u = -1

        # iterate over all the nodes to check the status of the visit
        for x in range(l):
            # now if the 'x' node is not visited yet or the distance we have currently for it is less than the c
            if not vis[x] and (u == -1 or dist[x] < dist[u]):
                u = x

        # check if we have visited all the nodes or we haven't reached the node
        if dist[u] == Inf:
            break

        # set the currently running node as visited
        vis[u] = True

       # now if the distance of the current node + the distance to the node we're visiting is less than the pric
        for v, d in graph[u]:
            if dist[u] + d < dist[v]:
                dist[v] = dist[u] + d

    # now at last return the list which contains the shortest path to each node from that given node
    return dist

graph = {
    0: [(1, 1)],
    1: [(0, 1), (2, 2), (3, 3)],
    2: [(1, 2), (3, 1), (4, 5)],
    3: [(1, 3), (2, 1), (4, 1)],
    4: [(2, 5), (3, 1)]
}
print("Dijstra algorithm")
Dijkstra(graph,0)
```

```
Dijstra algorithm
```

Out[6]:

```
[0, 1, 3, 4, 5]
```

Chatbot

In [7]:

```
from tkinter import *
root = Tk()
root.title("Chatbot")
def send():
    send = "You -> "+e.get()
    txt.insert(END, "\n"+send)
    user = e.get().lower()
    if(user == "hello"):
        txt.insert(END, "\n" + "Bot -> Hi")
    elif(user == "hi" or user == "hii" or user == "hiiii"):
        txt.insert(END, "\n" + "Bot -> Hello")
    elif(e.get() == "how are you"):
        txt.insert(END, "\n" + "Bot -> fine! and you")
    elif(user == "fine" or user == "i am good" or user == "i am doing good"):
        txt.insert(END, "\n" + "Bot -> Great! how can I help you.")
    else:
        txt.insert(END, "\n" + "Bot -> Sorry! I dind't got you")
    e.delete(0, END)
txt = Text(root)
txt.grid(row=0, column=0, columnspan=2)
e = Entry(root, width=100)
e.grid(row=1, column=0)
send = Button(root, text="Send", command=send).grid(row=1, column=1)
root.mainloop()
```

In [ ]:

```

```