Practical -1

Name- Bhavesh Kewalramani

Roll No.- A-25

Section- A

Semester- 6th

Shift- 1st

Aim:

Write a program to implement Breadth First Search. Take a weighted graph and start/goal node an input. Your job is to find goal node. Print the total cost and path.

Code:

```
import time



#numbers assigned to cities

cities={

    0: 'Oradea',

    1: 'Zerind',

    2: 'Arad',

    3: 'Sibiu',

    4: 'Timisoara',

    5: 'Lugoj',

    6: 'Mehadia',

    7: 'Drobeta',
```

```
    8: 'Fagaras',
    9: 'Rimnicu Vilcea',
    10: 'Craiova',
    11: 'Pitesti',
    12: 'Bucharest'
}

#Connection between cities
graph={
    0:[1,3],
    1:[2],
    2:[3,4],
    3:[8,9],
    4:[5],
    5:[6],
    6:[7],
    7:[10],
    8:[12],
    9:[10,11],
    10:[11],
```

```
    11:[12],
    12:[]
}

#cost associated with the edges
cost=[[0  , 71 , 0  , 151, 0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  ],
     [71 , 0  , 75 , 0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  ],
     [0  , 75 , 0  , 140, 118, 0  , 0  , 0  , 0  , 0  , 0  , 0  ],
     [151, 0  , 140, 0  , 0  , 0  , 0  , 0  , 99 , 80 , 0  , 0  ],
     [0  , 0  , 118, 0  , 0  , 111, 70 , 0  , 0  , 0  , 0  , 0  ],
     [0  , 0  , 0  , 0  , 111, 0  , 70 , 0  , 75 , 0  , 0  , 0  ],
     [0  , 0  , 0  , 0  , 70 , 70 , 0  , 75 , 0  , 0  , 0  , 0  ],
     [0  , 0  , 0  , 0  , 0  , 0  , 75 , 0  , 0  , 0  , 120, 0  ],
     [0  , 0  , 0  , 99 , 0  , 75 , 0  , 0  , 0  , 0  , 0  , 211],
     [0  , 0  , 0  , 80 , 0  , 0  , 0  , 0  , 0  , 0  , 146, 97 ],
     [0  , 0  , 0  , 0  , 0  , 0  , 0  , 120, 0  , 146, 0  , 138],
     [0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  , 97 , 138, 0  , 101],
     [0  , 0  , 0  , 0  , 0  , 0  , 0  , 0  , 211, 0  , 0  , 101, 0  ]]

#convert dictionary values to list
```

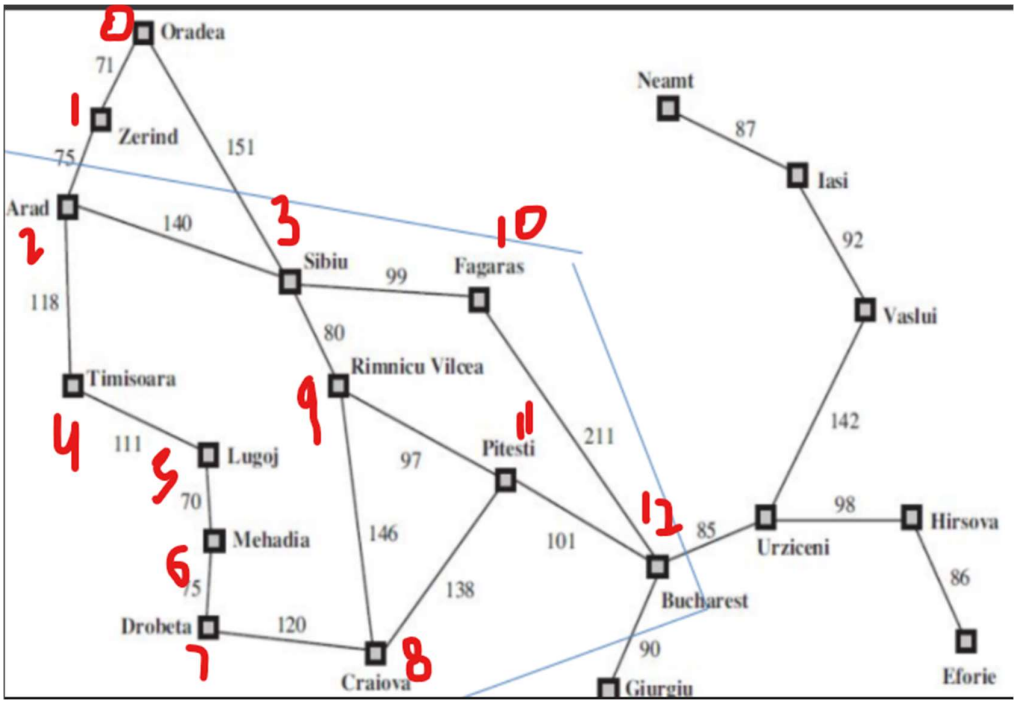
```
g=list(graph.values())


#starting node

source=2


#goal node

destination=12


start = time.time()

#queue to hold visited paths

queue = []


#boolean list to check if path is already visited or not

visited = [False]*13


#list to store possible path

path = []


#appending the starting node to the path

path.append(source)
```

```
#append the visited paths in the queue
queue.append(path.copy())

#marking the visisted vertices to true
visited[source]=True

#checking all the paths until destination node is reached
time.sleep(1)
while queue:
    path = queue.pop(0)
    last_node = path[len(path) - 1]

    if (last_node == destination):
        break

    for i in g[last_node]:
        if (visited[i]==False):
            newpath = path.copy()
            newpath.append(i)
```

```
            queue.append(newpath)
            visited[i]=True

#print the associated cities
time.sleep(1)
for i in range(len(path)-1):
    print(cities[path[i]],end=" ")
    print("->",end=" ")
print(cities[path[i+1]])

#calculate the total cost using cost matrix
time.sleep(1)
total = 0

for i in range(len(path)-1):
    total+=cost[path[i]][path[i+1]]
print("Total Cost : ",total)

end = time.time()
```

print("Total time taken : ",end-start)

Input:



Output:

Arad -> Sibiu -> Fagaras -> Bucharest
Total Cost : 450
Total time taken : 3.018648862838745

```
Arad -> Sibiu -> Fagaras -> Bucharest
Total Cost :  450
Total time taken :  3.018648862838745
```