Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_\_\_\_\_

LIST OF TASKS

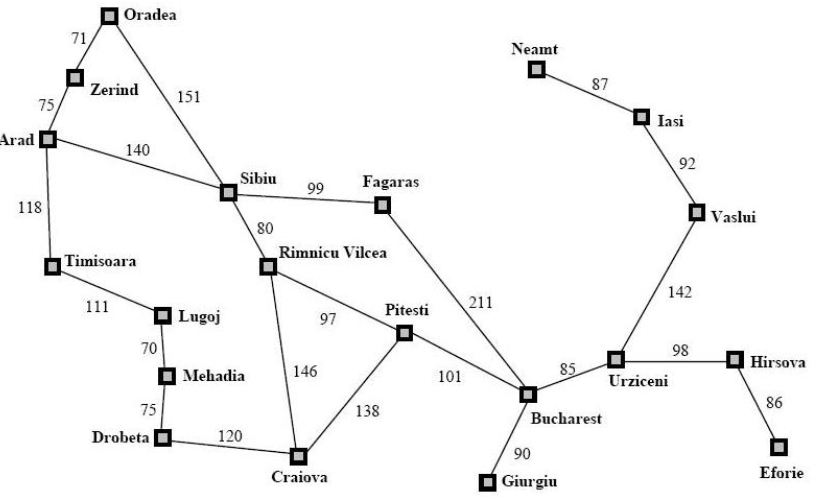
|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Submitted On:

\_\_\_\_\_\_\_\_\_\_\_\_

(Date: DD/MM/YY)

**Task No. 1:** Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST.



**Solution:**

import queue as Q

def search(graph, start, end):

if start not in graph:

raise TypeError(str(start) + ' not found in graph !')

return

if end not in graph:

raise TypeError(str(end) + ' not found in graph !')

return

queue = Q.PriorityQueue()

queue.put((0, [start]))

while not queue.empty():

node = queue.get()

current = node[1][len(node[1]) - 1]

if end in node[1]:

print("Path found: " + str(node[1]) + ", Cost = " + str(node[0]))

break

cost = node[0]

for neighbor in graph[current]:

temp = node[1][:]

temp.append(neighbor)

queue.put((cost + graph[current][neighbor], temp))

def readGraph():

lines = int( input("Give input for searching : ") )

graph = {}

for line in range(lines):

line = input()

tokens = line.split()

node = tokens[0]

graph[node] = {}

for i in range(1, len(tokens) - 1, 2):

# print(node, tokens[i], tokens[i + 1])

# graph.addEdge(node, tokens[i], int(tokens[i + 1]))

graph[node][tokens[i]] = int(tokens[i + 1])

return graph

def main():

graph = readGraph()

search(graph, 'Arad', 'Bucharest')

main()

**Output:**

