NAME: M ZOHAB AHMER
REG NO. : FA20-BCS-038
SEM/SEC : 5/B
SUBJECT : AI LAB
DATED : 30-09-2022

LAB ASSIGNMENT

Lab Task 1:

imagine going from Arad to Bucharest in the following map. Your goal is to minimize the distance mentioned in the map during your travel. Implement a depth first search to find the corresponding path.

SOLUTION:-

PATH:- ARAD, TIMISOARA, LUGOJ, MEHADIA, DOBRETA, CRAIOVA, PITESTI, BURCHAREST.

```
class node:
def __init__(self, name):
self.explored = 0
self.name = name
self.neighbours = {}
nodes = {}
start = 'Arad'
goal = 'Bucharest'
explored = []
frontier = []
path = []
f = open("input.txt", "rb")
for line in f:
line = line.strip()
node1, node2, distance = line.split(",")
if node1 not in nodes:
nodes[node1] = node(node1)
if node2 not in nodes:
nodes[node1] = node(node1)
if node2 not in nodes:
nodes[node2] = node(node2)
nodes[node1].neighbours[node2] = distance
nodes[node2].neighbours[node1] = distance
def initFrontier():
frontier.append(start)
nodes[start].parent = ''
def choosenode():
node = frontier.pop()
if testgoal(node):
print goal
pathcost = calpath(goal)
print "path cost is {}".format(pathcost)
print "path selected is {}".format(path)
exit()
return node
def calpath(cnode):
path.append(cnode)
if nodes[cnode].parent == '':
return 0
else:
cparent = nodes[cnode].parent
pathcost = calpath(cparent)+int(nodes[cnode].neighbours[cparent])
return pathcost
def testgoal(curnode)
if curnode == goal:
return True
return False
def graphsearch():
if not frontier:
print "failure"
exit()
curnode = choosenode()
nodes[curnode].explored = 1
explored.append(curnode)
for neighbour in nodes[curnode].neighbours.keys():
```

```
if neighbour in frontier:
Continue
frontier.append(neighbour)
nodes[neighbour].parent = curnode
initFrontier()
while True:
graphsearch()
print frontier
```

Lab Task 2:

Generate a list of possible words from a character matrix

Given an M × N boggle board, find a list of all possible words that can be formed by a sequence of adjacent characters on the board.

We are allowed to search a word in all eight possible directions, i.e., North, West, South, East, North East, North-West, South-East, South-West, but a word should not have multiple instances of the same cell.

Consider the following the traditional 4 × 4 boggle board. If the input dictionary is [START, NOTE, SAND, STONED], the valid words are [NOTE, SAND, STONED].

SOLUTION:-

```
class Trie:

    def __init__(self):
        self.character = {}
        self.isLeaf = False


def insert(root, s):

    curr = root

    for ch in s:
                curr = curr.character.setdefault(ch, Trie())

    curr.isLeaf = True


row = [-1, -1, -1, 0, 1, 0, 1, 1]
col = [-1, 1, 0, -1, -1, 1, 0, 1]


def isSafe(x, y, processed, board, ch):
    return (0 <= x < len(processed)) and (0 <= y < len(processed[0])) and \
           not processed[x][y] and (board[x][y] == ch)


def searchBoggle(root, board, i, j, processed, path, result):

    if root.isLeaf:

        result.add(path)

        processed[i][j] = True

       for key, value in root.character.items():

                for k in range(len(row)):


            if isSafe(i + row[k], j + col[k], processed, board, key):
                searchBoggle(value, board, i + row[k], j + col[k],
                             processed, path + key, result)
    processed[i][j] = False


def searchInBoggle(board, words):
    result = set()

    if not board or not len(board):
        return

    root = Trie()
    for word in words:
        insert(root, word)

    (M, N) = (len(board), len(board[0]))

    processed = [[False for x in range(N)] for y in range(M)]


    for i in range(M):
        for j in range(N):
            ch = board[i][j]
            if ch in root.character:
                searchBoggle(root.character[ch], board, i, j, processed, ch, result)

    return result


if __name__ == '__main__':
    board = [
        ['M', 'S', 'E', 'F'],
        ['R', 'A', 'T', 'D'],
        ['L', 'O', 'N', 'E'],
        ['K', 'A', 'F', 'B']
    ]

    words = ['START', 'NOTE', 'SAND', 'STONED']
    searchInBoggle(board, words)

    validWords = searchInBoggle(board, words)
    print(validWords)
```