NMTTNT_Tuan1_19110413

November 13, 2021

[ ]:
```
from queue import Queue, PriorityQueue
from collections import defaultdict
```

[ ]:
```
f=open("input BFS-DFS.txt","r")
verticles = f.readline()
start,goal = [int(num) for num in f.readline().split(' ')]
adj_matrix = [[int(num) for num in line.split(' ')] for line in f if line.
 ↪strip("\n") != "\n"]
f.close
```

[ ]:
```
<function TextIOWrapper.close()>
```

[ ]:
```
# converts from adjacency matrix to adjacency list
def adjm_to_adjl(a):
    adjList = defaultdict(list)
    for i in range(len(a)):
        for j in range(len(a[i])):
                if a[i][j]>= 1:
                    adjList[i].append(j)
    return adjList
```

[ ]:
```
def bfs(graph, start, goal):
    queue = []
    # push the first path into the queue
    queue.append([start])
    while queue:
    # can't find path to goal
        if (len(queue)==0):
            raise Exception("No way Exception")
    # get the first path from the queue
        path = queue.pop(0)
        # get the last node from the path
        node = path[-1]
        # path found
        if node == goal:
            return path
        # explore path and push it into the queue
        for adj in graph.get(node, []): # explore path of graph
```

```
            explored = list(path) # remember node visited
            explored.append(adj)
            queue.append(explored) #push path explored into queue
```

[ ]:
```
def dfs(graph, start, goal):
    stack = [(start, [start])] #push the first node and path into stack
    explored = set()
    while stack:
        if len(stack) == 0: # can't find path to goal
            raise Exception("No way Exception")
        (current, path) = stack.pop() # get the last (node and path) from stack
        if current not in explored: #check node was in explored
            if current == goal: #path found
                return path
            #remember node explored
            explored.add(current)
            #push node and path explore into stack
            for adj in graph[current]:
                stack.append((adj, path + [adj]))
```

[ ]:
```
graph = adjm_to_adjl(adj_matrix)
graph
```

[ ]:
```
defaultdict(list,
            {0: [1, 2, 3],
             1: [6],
             2: [5, 7],
             3: [4],
             4: [13],
             5: [11],
             6: [2],
             7: [8, 9],
             8: [10],
             10: [9, 14],
             11: [12],
             12: [13],
             13: [10],
             14: [15],
             15: [16],
             16: [17]})
```

[ ]:
```
solution = bfs(graph,start,goal)
print("BFS solution: ",*solution)
```

```
BFS solution:  0 2 7 8 10 14 15 16 17
```

[ ]:
```
solution2 = dfs(graph,start,goal)
print("DFS solution: ",*solution2)
```

```
DFS solution:  0 3 4 13 10 14 15 16 17
```

[ ]:
```
f=open("input UCS.txt","r")
verticles = f.readline()
start,goal = [int(num) for num in f.readline().split(' ')]
adj_matrix = [[int(num) for num in line.split(' ')] for line in f if line.
 ↪strip("\n") != "\n"]
```

[ ]:
```
def adjm_to_adjlUCS(a):
    adjList = defaultdict(list)
    for i in range(len(a)):
        for j in range(len(a[i])):
                if a[i][j]>= 1:
                    adjList[i].append((a[i][j],j))

    return adjList
```

[ ]:
```
graph = adjm_to_adjlUCS(adj_matrix)
graph
```

[ ]:
```
defaultdict(list,
            {0: [(50, 1), (350, 2), (300, 3)],
             1: [(600, 6)],
             2: [(100, 5), (900, 7)],
             3: [(1300, 4)],
             4: [(1400, 13)],
             5: [(700, 11)],
             6: [(800, 2)],
             7: [(790, 8), (300, 9)],
             8: [(1200, 10)],
             10: [(800, 9), (400, 14)],
             11: [(950, 12)],
             12: [(600, 13)],
             13: [(1300, 10)],
             14: [(1300, 15)],
             15: [(770, 16)],
             16: [(1200, 17)]})
```

[ ]:
```
def UCS(graph,start,goal):
    queue = PriorityQueue()
    queue.put((0,[start])) #put first cost and path to queue

    while queue:
        if queue.empty():
            raise Exception("No way Exception")
```

```
        cost,path = queue.get() #get least cost path in queue
        current = path[-1] #get last node in path
        #path found
        if current == goal:
            return path,cost
        # explore path and push it into the queue
        for adj in graph.get(current,[]):  # explore path of graph
            explored = list(path) # remember node visited
            totalCost = cost + adj[0] # calculated total cost of path explore
            explored.append(adj[1])
            # push path and total cost into queue
            queue.put((totalCost,explored))
```

[ ]:
```
path,cost = UCS(graph,start,goal)
print("UCS solution: ",'[',*path,']',"Total Cost",cost)
```

```
UCS solution:  [ 0 2 7 8 10 14 15 16 17 ] Total Cost 6910
```