```
# CAP 6635 Artificial Intelligence, 2023 Summer
# June. 13 2023, X. Zhu
# Informed Search
# code credit: Adapted from following github project with revision
# https://gist.github.com/Nicholas-Swift/003e1932ef2804bebef2710527008f44

# Modified by Matthew Acs for HW 2


import random
import numpy as np
class Node():
    """A node class for A* Pathfinding"""

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position

    def __hash__(self):               #<-- added a hash method
        return hash(self.position)

# method ='AStar', 'GBF', 'UCS'
# 'AStar': A-star search, 'GBF': greedy best first, 'UCS': uniform cost search
def InformedSearch(maze, start, end, method='Astar'):
    """Returns a list of tuples as a path from the given start to the given end in the given maze"""

    # Create start and end node
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    # Initialize both open and closed list
    open_list = []
    closed_list = set()                # <-- closed_list must be a set

    # Add the start node
    open_list.append(start_node)

    # Loop until you find the end
    expanded_nodes=0
    queue_size=0
    while len(open_list) > 0:

        # Get the current node
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        # Pop current off open list, add to closed list
        open_list.pop(current_index)
        closed_list.add(current_node)     # <-- change append to add

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return(expanded_nodes,queue_size,path[::-1]) # Return reversed path

        # update expanded nodes, and update maximum queuze size
        expanded_nodes=expanded_nodes+1
        if(len(open_list)>queue_size):
            queue_size=len(open_list)  # check maximum queue size
```

```
        # Generate children
        children = []
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

            # Get node position
            node_position = (current_node.position[0] + new_position[0], current_node.position[1] + new_position[1])

            # Make sure within range
            if node_position[0] > (len(maze) - 1) or node_position[0] < 0 or node_position[1] > (len(maze[len(maze)-1]) -1) or noc
                continue

            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            # Create new node
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
        for child in children:

            # Child is on the closed list
            if child in closed_list:              # <-- remove inner loop so continue takes you to the end of the outer loop
                continue

            # Create the f, g, and h values
            child.g = current_node.g + np.sqrt(np.square(child.position[0] - current_node.position[0])+np.square(child.position[1]
            child.h = np.sqrt(np.square(child.position[0] - end_node.position[0])+np.square(child.position[1] - end_node.position[
            if method=='AStar':
                child.f = child.g + child.h
            elif method=='GBF':
                child.f=child.h
            elif method=='UCS':
                child.f=child.g


            # Child is already in the open list
            childAlreadyExist=False
            for open_node in open_list:
                if child == open_node and child.g >=open_node.g:
                    childAlreadyExist=True
                    break

            # Add the child to the open list if Chinld not in the open list
            if(not childAlreadyExist):
                open_list.append(child)


def pathLength(path):
    dis=0
    for i in range(len(path)-1):
        x1=path[i][0]
        y1=path[i][1]
        x2=path[i+1][0]
        y2=path[i+1][1]
        dis=dis+np.sqrt(np.square(x1-x2)+np.square(y1-y2))
    return(dis)


# searchMethod: 'AStar' or 'GBF' or 'UCS'
def mazeRunner(start,end,maze,searchMethod='AStar'):
    #force start and end positions to be reachable.
    maze[start[0]][start[1]]=0
    maze[end[0]][end[1]]=0
    expanded_nodes,queue_size,path = InformedSearch(maze, start, end, searchMethod)  # 'AStar' or 'GBF' or 'UCS'
    print("\r\n%s search path length: %f"%(searchMethod,pathLength(path)))
    return(expanded_nodes,queue_size,path)

# k.    Using Figure 7 as the game field, and set initial state as [0, 0] and goal state as [9, 9].
# Beginning of changes for assignment
maze = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
start = (0, 0)
end = (9, 9)
# End of changes for assignment

print(mazeRunner(start,end,maze,'AStar'))
print(mazeRunner(start,end,maze,'GBF'))
print(mazeRunner(start,end,maze,'UCS'))
```

```
AStar search path length: 14.485281
(77, 59, [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (5, 6), (5, 7), (5, 8), (6, 9), (7, 9), (8, 9), (9, 9)])

GBF search path length: 15.313708
(13, 29, [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (6, 7), (5, 8), (6, 9), (7, 9), (8, 9), (9, 9)])

UCS search path length: 14.485281
(101, 22, [(0, 0), (0, 1), (0, 2), (0, 3), (1, 4), (2, 5), (3, 6), (4, 7), (5, 8), (6, 9), (7, 9), (8, 9), (9, 9)])
```

```
# Create random maze environment with controled obstacles
# m x m maze, p
def createMaze(m,p):
    maze = np.arange(m*m).reshape(m,m)
    for mI in range(m):
        for aI in range(m):
            xy=random.random()
            if xy<p:
                maze[mI][aI] = 1
            else:
                maze[mI][aI] = 0
    #print(maze)
    return(maze)


start = (0, 0)
end = (99, 99)
m,p=100,0.3
maze=createMaze(m,p)
print(maze)
print(mazeRunner(start,end,maze,'AStar'))
print(mazeRunner(start,end,maze,'GBF'))
print(mazeRunner(start,end,maze,'UCS'))
```

```
[[1 0 0 ... 0 0 0]
 [1 0 0 ... 0 1 0]
 [1 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [1 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]]

AStar search path length: 148.208153
(2598, 567, [(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (9, 4), (10, 4), (11, 5), (12, 6), (12,

GBF search path length: 154.450793
(122, 260, [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 5), (7, 6), (8, 6), (9, 7), (9, 8), (10, 9), (10, 10), (11,

UCS search path length: 148.208153
(7154, 141, [(0, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 2), (6, 3), (7, 3), (8, 3), (9, 3), (10, 4), (11, 5), (11, 6), (12,
```

✓ 0s completed at 12:22 AM