National University of Computer and Emerging Sciences

School of Computing Spring 2023 Lahore Campus

Artificial Intelligence

Assignment 1

(Deadline: 24th February, 2023 11:59 PM)

Submission: Combine all your code file(s) in one .zip file. Name the .zip file as ROLL_NUM_Section_01.zip (e.g. **20l-0001_A_01.zip**). Submit zip file on GCR within given deadline. Failure to submit according to above format would result in deduction of 10% marks per day.

Comments: Please ensure to provide proper comments in your code as it holds a significant weightage in the evaluation process. Effective commenting will aid in understanding the logic behind the code and increase the readability of the program.

Deadline: Deadline to submit assignment is **24th February, 2023 11:59 PM**. No submission will be considered for grading outside GCR. Correct and timely submission of assignment is responsibility of every student; hence no relaxation will be given to anyone.

Plagiarism: - Any form of academic dishonesty (e.g. plagiarism, copying code from other sources, sharing code with other students) is strictly prohibited and will result in disciplinary action. The individuals involved will get zero in this assignment and 50% deductions in assignments to come.

Viva Evaluation: - Students should be prepared to present their code and answer questions about their implementation during the viva evaluation. The viva evaluation will focus on your program logic and concepts used.

PART 1

Implementation of Blind Search Algorithm

Problem Statement:

Your task is to implement a maze solver using blind search algorithms (**Depth-First Search-DFS** and **Breadth-First Search-BFS**). You are given a 2D array representing a maze, where 0s represent open spaces and 1s represent walls. You need to find a path from the starting point to the goal point, where the starting point is at the top left corner of the maze (0,0) and the goal point is at the bottom right corner of the maze (n-1, m-1) where n and m are the dimensions of the maze.
Following is a sample Maze.

S	0	0	0	1	0	0	0	0	0
1	1	0	1	0	1	1	1	1	1
0	1	0	1	0	0	0	0	0	1
1	1	0	1	0	1	1	1	0	1
0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	1	1	1	1	1
0	0	0	1	0	0	0	0	0	1
1	1	1	1	0	1	1	1	0	1
0	0	0	0	0	1	0	0	0	0
0	1	1	1	1	1	0	1	1	G

1- **Reading the maze from a text file:** Write a function to read the maze from a text file. The function should take a filename as input and return a 2D array representing the maze. The maze can be represented using a binary matrix, where 0 represents an empty cell and 1 represents a wall.
2- **Implementing DFS and BFS**: Write a function for DFS and BFS algorithms. Both functions should take the maze as input and return a path to the goal state if there exists one, and -1 if there is no path to the goal state. Students should use the appropriate data structures and algorithms to implement DFS and BFS.
3- **Returning the path to the goal state:** The function should return a list of coordinates representing the path from the starting point to the goal state. If there is no path, the function should return -1.
4- **Bonus:** Design the maze using Python graphics library: Students can create a graphic representation of the maze using Python graphics library. The graphic representation of the maze should include a start and goal point, and the path returned by the DFS or BFS algorithm. Students can use any Python graphics library of their choice.

Note: It is important to note that a maze can have multiple goal states, which means that the DFS and BFS algorithms may produce different paths to reach different goal states. As a result, it is possible that the DFS algorithm may output one path while the BFS algorithm outputs another.

PART 2

Implementation of A* Search

Problem Statement:

Your task is to implement a maze solver using A* Algorithm. This time there are two types of walls, short walls and high walls. The user can jump over short walls. Your algorithm should return the path to the goal state, prioritizing the path with a minimum number of short walls or no walls.
To represent the maze, you can use 0, 1, and 2. Where 0 indicates no wall, 1 corresponds to a high wall, and 2 corresponds to a short wall. Your implementation should be able to read the maze from a text file.
Following is a sample Maze.

S	0	0	0	0	0	0	0	0	1
1	1	0	1	0	1	1	1	1	1
0	1	0	1	0	0	0	0	0	1
1	1	0	1	0	1	1	1	0	1
0	1	0	1	0	0	0	1	0	0
0	1	2	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1
1	1	1	1	0	1	1	1	0	1
0	0	0	0	0	1	0	0	0	1
0	1	1	1	1	1	0	1	0	G

In the above Maze, the green path should be returned as it has no short walls.

Best of Luck <3