National University of Computer and Emerging Sciences

Artifical Intelligence Lab Manual

Department of Computer Science

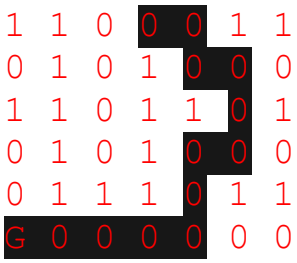
PART 1

Implementation of Blind Search Algorithm

Problem Statement:

Your task is to implement a cube solver using blind search algorithms (**Depth-First Search-DFS** and **Breadth-First Search-BFS**). You are given a 2D array representing a cube, 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 cube (0,0) and the goal point is at the bottom right corner of the cube (n-1, m-1) where n and m are the dimensions of the cube. Following is a sample Cube.

1 0 S 0 1 0 0



- **Reading the cube from a text file:** Write a function to read the cube from a text file. The function should take a filename as input and return a 2D array representing the cube. The cube can be represented using a binary matrix, where 0 represents an empty cell and 1 represents a wall.
- **Implementing DFS and BFS**: Write a function for DFS and BFS algorithms. Both functions should take the cube 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.
- **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.

Note: It is important to note that a cube 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.

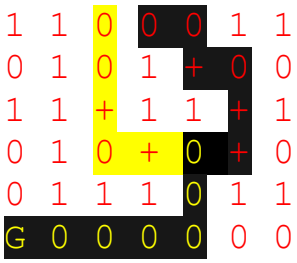
PART 2

Implementation of A* Search

Problem Statement:

Your task is to implement a cube 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 cube, 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 cube from a text file.

Following is a sample Cube.



In the above Cube, the yellow path should be returned as it has only 2 short walls. Other has 3.