**AI ASSIGNMENT REPORT**

How to run the programs:-

There are 3 separate programs for the 3 different algorithms that is Breadth-First Search, Depth-First Search and Uniform-Cost Search. There are also 5 text files containing the 5 different mazes. So we have to run each program for these 5 mazes in VS code. We need to open the maze files one by one for each algorithm and execute them. The output will contain directions like “left”, “right”, “up” or “down” indicating the path to be covered.

Explanation of the programs:-

Modelling a maze problem as a search problem involves representing the maze as a graph where the nodes correspond to different positions in the maze and the edges represent possible transitions between those positions.The goal is to find a path from the star position to the goal position while navigating through the maze’s ,obstacles and constraints.

Each position in the maze is a possible state. The state representation includes the coordinates(row and column) of the position.

Initial State: The starting position in the maze is the initial state from which the search begins. In our case it is “S”.

Goal State: The destination or goal position in the maze is the state that the search algorithm aims to reach. In our case it is ”G”.

Successor Function: The successor function generates new states by applying valid actions to the current state. It also checks whether the generated states are valid or not. In our case the successor function is *get\_neighbors().*

1. **Breadth-First Search:** BFS explores the maze in a breadth first manner that means it visits all the neighbouring cells of the current cell before moving on to the next level.

In the BFS algorithm a queue is used to store the cells to be explored. The initial state is added to the queue and then the algorithm iteratively removes a cell from the front of the queue, explores its neighbors and adds them to the back of the queue. BFS guarantees that the shortest path from the initial state to the goal state will be found.

1. **Depth-First Search:** DFS explores a maze in a depth first manner that means it visits one path as far as possible before backtracking and exploring other paths.

In DFS algorithm stack or recursion is used to store the cells to be explored (We have used recursion). The initial state is added to the stack or used as a initial recursive call and the the algorithm iteratively pops a cell from the stack or explores the current cell and recursively calls itself on its neighbors. DFS does not guarantee finding the shortest path as it may get stuck exploring a long path before considering shorter paths.

1. **Uniform Cost Search:** UCS is an extension of BFS that takes in the account the cost of each edge in the maze it explores the maze by considering the cost of reaching each cell and chooses the path with the lowest cost (We have taken edge weight as 1).

In the UCS algorithm a queue is used to store the cells to be explored with the priority based on the cost of reaching each cell. The initial state is added to the queue with the cost of 0 and then the algorithm iteratively removes a cell with the lowest cost, explores its neighbors and adds them to the queue with their updated costs. UCS guarantees finding the path with the lowest cost from the initial state to the goal state.

ASSUMPTIONS:

1. The maze is assumed to have a defined 2 dimensional array structure where each cell either open or blocked.
2. The actions and their outcomes are assumed to be deterministic that is taking a specific action from a particular state always leads to the same new state.
3. The agent can usually access complete information about the maze including its layout, obstacles, initial and goal positions.
4. The goal is generally find the shortest or the most optimal path often based on minimizing the total cost associated with the path.
5. This assumes that a single agent is navigating the maze.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Empty Cells** | **BFS** | **DFS** | **UCS** |
| 30 | 1.995 ms | 1.968 ms | 0.997 ms |
| 50 | 3.008 ms | 2.997 ms | 2.995 ms |
| 100 | 3.009 ms | 5.999 ms | 3.004 ms |
| 500 | 7.988 ms | 16.661 ms | 9.007 ms |
| 1000 | 14.011 ms | 78.996 ms | 14.992 ms |

Once the maze is represented as a graph or array and the problem is model as a search the search algorithms can be applied to find the solution.

**Execution Time of different algorithms**

**Graph Plots**

PREPARED BY

ANINDYA BAG (Roll: 2105260)

&

SOUHARDYA RAKSHIT (Roll: 21052198)

CS-31