FIS PROJECT 1

Soniya Rode
sr5055

Problem Statement: To implement a maze solving program using IDS and A* (using four heuristics) and, to understand how different heuristics work on the same problem and how well each approach works.

Executing the program:
Run the project1.py file with the command : python3 project1.py File_name.txt
File_name can be maze.txt or mazeAstar.txt or mazeAstart2.txt (Bigger maze, takes time to execute)
If no command line args are passed, default file maze.txt is read

State Space Description:
The state space consists of a grid maze, a binary matrix where fields containing a 0 are accessible whereas fields containing 1 are blocked. The single elements of the maze are denoted by (i,j). We assume the start and goal position to always be in the upper-left and lower-right, respectively.

State Description: Any position on the maze containing zero is a state. Not all states are reachable from the initial state. Thus making some mazes un-solvable.

State space Representation: A position in a maze is given by a tuple (i, j) where $1 \leq i \leq$ height, $1 \leq j \leq$ width.
Initial state - The left-most top corner of the maze is the initial position which is at (0,0).
Goal state - The right-most lower corner of the maze is the goal state which is at (height-1,width-1) where height, width are the height and width of the maze

Actions: Movements in 4 directions: up, down, left and right are allowed.You can move only If the adjacent field in any of the movement direction is free. That is you can only move to an adjacent postion if there is a zero present.

• Transition model:
The aim is to find the shortest solving sequence for the given maze.
After each iteration, to move a step closer to the goal position or return if no solution exists.

Steps :
1. Create a graph of all the nodes reachable from the start node (start state) while adding edges to the graph and the current edge's neighbours in a list called connectedTo.
2. For all the neighbours of the start node/current node choose a neighbour which gets us a step closer to the goal state.

Different methods can be used to get to the goal state, DFS, BFS, IDS, DLS, uniform cost search, A* etc.(The path cost of the solution will be different for different algorithms used)
Continue step 2 until the goal node is reached i.e the maze exit. If the goal state is not reachable from the start node, return path not found

IDS:

- IDDFS(src, target)

Initialize the depth as 0 and go on increasing the depth till target is found or max depth is reached.

```
depth=0
While true
    path=__findPathDLS(start,end,visited,depth)
      If path is found, break the loop
    if path!=None:
            break
      else:
        depth+=1
return path
```

- DLS(src, target, depth)

```
    if (src == target)
        return true
```

If depth limit is reached stop recursion

```
if (limit <= 0)
    return false;

for each adjacent node of src
    if DLS(i, target, depth-1)
        return true
else
    return false
```

A* :

Step 1: Create a graph from the maze by adding all the nodes having '0' and add its corresponding neighbours in the neighbour's list.
Step 2: Initialize two lists, open and close. Open to store nodes to be checked and close to store the finalised nodes.

Step 3: Add start node to the open list, exit if its = null
Step 4: while open list has nodes :
Remove the node having a min value of f (f=g+h) and add it to the closed list, set it as the current node
Add its neighbours which are not in the closed and open list.
Step 5: check if the current node is equal to goal, if yes then exit else continue to step 4

• **Goal test:** To reach the goal state (Maze Exit) using a short possible path.

• **Path cost**: Each step costs 1, so the path cost is the number of steps in the path

Heuristics for A* algorithm :
• The heuristic can be used to control A*'s behaviour.
• If h(n) is exactly equal to the cost of moving from current_node to the goal, then A* will only choose the best path and never expand any other node, making it very fast and efficient.
• If h(n) =0, then A* turns into Dijkstra's Algorithm, which is guaranteed to find the shortest path.
• Having a good heuristic allows A* to expand very few nodes and find the optimal path.
• Good heuristics for grid matrix are distance heuristics, thus the maze solver is implemented using distance heuristics.

Manhattan distance: It is said to be an "admissible" heuristic for square grids which allow movements in 4 directions. It is the sum of absolute values of differences in the goal's x and y coordinates and the current nodel's x and y coordinates respectively.Manhattan distance is the sum of all the real distances between current and goal where each distance are always the straight line distance

Diagonal distance: If movements in 8 directions are allowed, i.e. diagonal movements are also allowed then diagonal heuristic will result in lower path cost as compared to manhattan. But since only 4 directional movements are allowed, this heuristic does not show a significant difference in path cost.

Euclidean Distance: If movement at any angle is allowed, then straight line distance is a good heuristic to be used.It is shorted than manhattan and diagonal distance thus gives the shortest path incomparasion but takes longer to run.

Squared Euclidean Distance: This heuristic function turns into a greedy BFS for longer distances and into Dijkstra's algorithm for shorter distances.

Random number Heuristic Function: This function is a bad heuristic since it assigns random h values to the nodes, chances of selecting the optimal node is very less resulting in higher path cost.

Performance Metrics :

Manhattan distance metrics perform better than other distance metrics in A* algorithm allowing movements in 4 directions only. Another reason of better performance of Manhattan distance is the fast calculation of all parameters as well as the generation of the fast shortest path.Since diagonal movements are not allowed, diagonal distance heuristic does not perform that well in this case. But the choice of the better Heuristic function will change with the problem and allowed directional movements.Random function is a bad heuristic.