AI Assignment

REPORT

Daniel Krasovski | Artificial Intelligence | 30/04/2022
C18357323

Contents

How to run code: 2
DFS: 2
IDS: 2
Astar: 2
Encoding a maze 3
DF Algorithm 3
IDS Algorithm 4
A* Heuristics 5
A* implementation 6

How to run code:

To run the code, run the main.pl file. Maze 1 and maze 2 are in the mazes folder, and are imported in the main.pl file. To change which maze is being run, uncomment the maze you want to run, and comment the maze you don't want to run.

To run the different algorithms you write:

DFS:

solve_dfs(Path).

IDS:

ids(Path).

ASTAR:

astar(Path).

Encoding a maze

The mazes are created in the maze1.pl and maze2.pl files. They are created by specifying where the walls are, the size of the maze, and the start and goal positions

```
wall(2,2).

wall(2,3).

wall(2,4).

wall(3,4).

wall(4,4).

wall(4,3).

mazeSize(5,5).

start(pos(3,3)).

goal(pos(1,1)).
```

DF Algorithm

The depth first algorithm works by starting at the start position which is defined in the code. Then it moves to the next position following the move() function, so it will go up, until it cant no more, then down, left and right. It can no move back to a position it already was at, and stops at the goal position.

Output from running it on the 1st maze . From (3,3) to (1,5)

```
pos(3,2)pos(3,1)pos(2,1)pos(1,1)pos(1,2)pos(1,3)pos(1,4)pos(1,5)
Path = [down, [down, [left, [left, [up|...]]]]]
```

Output from running it on the 2nd maze from (1,1) to (8,6)

```
[debug] [8]  ?- solve_dfs(Path).
 pos(1,2) pos(1,3) pos(1,4) pos(1,5) pos(1,6) pos(1,7) pos(1,8) pos(1,9) pos(1,10) pos(2,10) pos(2,9) pos(2,8) pos(2,7) pos(2,6) pos(2,5) pos(3,5)
 pos(3,6) pos(3,7) pos(3,8) pos(3,9) pos(3,10) pos(4,10) pos(4,9) pos(4,8) pos(4,7) pos(4,6) pos(4,5) pos(5,5) pos(5,6) pos(5,7) pos(5,8) pos(5,9)
 pos(5,10) pos(6,10) pos(7,10) pos(7,9) pos(7,8) pos(8,8) pos(8,9) pos(8,10) pos(9,10) pos(9,9) pos(9,8) pos(10,8) pos(10,9) pos(10,10) pos(10,7)
pos(10,6) pos(10,5) pos(9,5) pos(9,6) pos(8,6)
Path = [up, [up, [up, [up, [up|...]]]]]
```

This algorithm works by attempting to brute force through the maze by checking every available path. This algorithm can also be improved by changing in which order it choses its path, for example, instead of up, down, left, right, it'll be up, right, down, left.

Here is the algorithm path on maze 2:

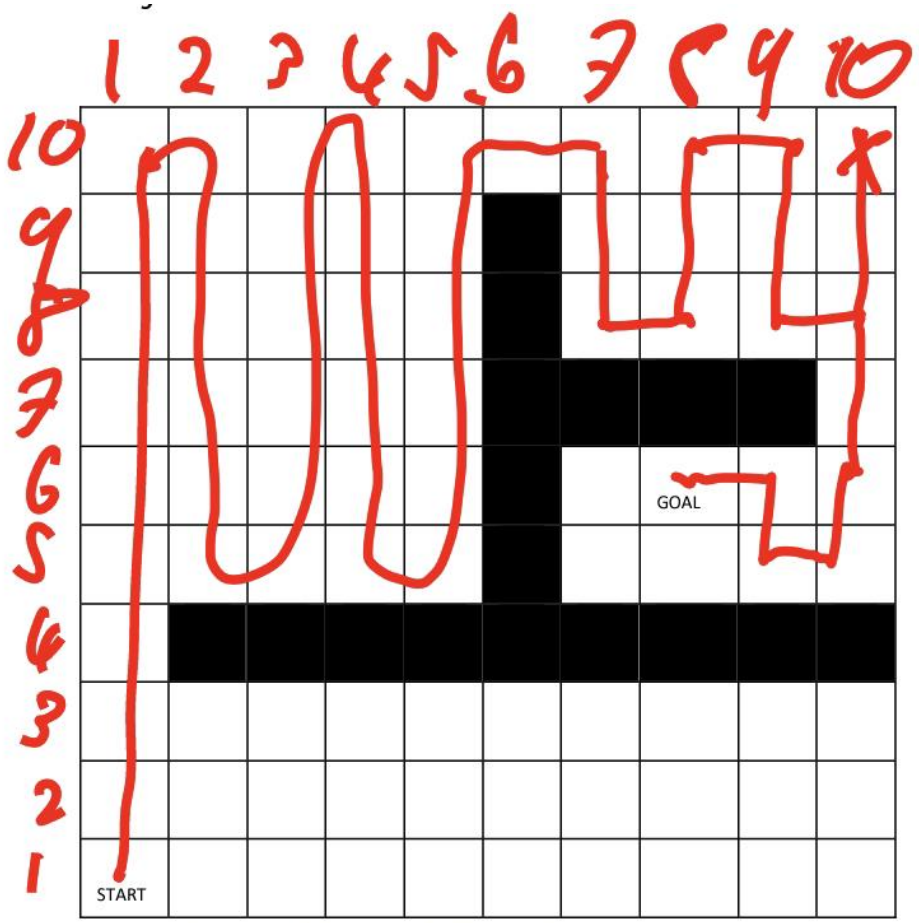

Figure 2 Sample maze 2

IDS Algorithm

This algorithm works exactly the same as the DFS except with a counter to count the depth and to make sure it doesn't go over the depth. Iterative Deepening search works when you want a BFS but don't have enough memory and can deal with the slower performance. The output I got for IDS algorithm is:

Maze 1:

```
?- ids(Path).
 pos(3,2) pos(3,1) pos(4,1) pos(4,2) pos(5,2) pos(5,3) pos(5,4) pos(5,5) pos(4,5) pos(3,5) pos(2,5) pos(1,5)
Depth is: 13
Path = [pos(1, 5), pos(2, 5), pos(3, 5), pos(4, 5), pos(5, 5), pos(5, 4), pos(5, 3), pos(5, 2), pos(..., ...)|...]
```

Maze2:

```
[debug]  ?- ids(Path).
 pos(1,2) pos(1,3) pos(1,4) pos(1,5) pos(1,6) pos(1,7) pos(1,8) pos(1,9) pos(1,10) pos(2,10) pos(2,9) pos(2,8) pos(2,7) pos(2,6) p
os(2,5) pos(3,5) pos(3,6) pos(3,7) pos(3,8) pos(3,9) pos(3,10) pos(4,10) pos(4,9) pos(4,8) pos(4,7) pos(4,6) pos(4,5) pos(5,5) pos
(5,6) pos(5,7) pos(5,8) pos(5,9) pos(5,10) pos(6,10) pos(7,10) pos(7,9) pos(7,8) pos(8,8) pos(8,9) pos(8,10) pos(9,10) pos(9,9) po
s(9,8) pos(10,8) pos(10,9) pos(10,10) pos(10,7) pos(10,6) pos(10,5) pos(9,5) pos(9,6) pos(8,6)
Depth is: 51
Path = [pos(8, 6), pos(9, 6), pos(9, 5), pos(10, 5), pos(10, 6), pos(10, 7), pos(10, 8), pos(9, 8), pos(..., ...)|...] ,
```

The path is the same as DFS however it can be improved with changing the move order.
As seen here:

```
?- ids(Path).
 pos(1,2) pos(1,3) pos(1,4) pos(1,5) pos(1,6) pos(1,7) pos(1,8) pos(1,9) pos(1,10) pos(2,10) pos(3,10) pos(4,10) pos(5,10) pos(6,1
0) pos(7,10) pos(8,10) pos(9,10) pos(10,10) pos(10,9) pos(10,8) pos(10,7) pos(10,6) pos(10,5) pos(9,5) pos(9,6) pos(8,6)
Depth is: 27
Path = [pos(8, 6), pos(9, 6), pos(9, 5), pos(10, 5), pos(10, 6), pos(10, 7), pos(10, 8), pos(10, 9), pos(..., ...)|...] 
```

With the depth being 27 and the path taken:

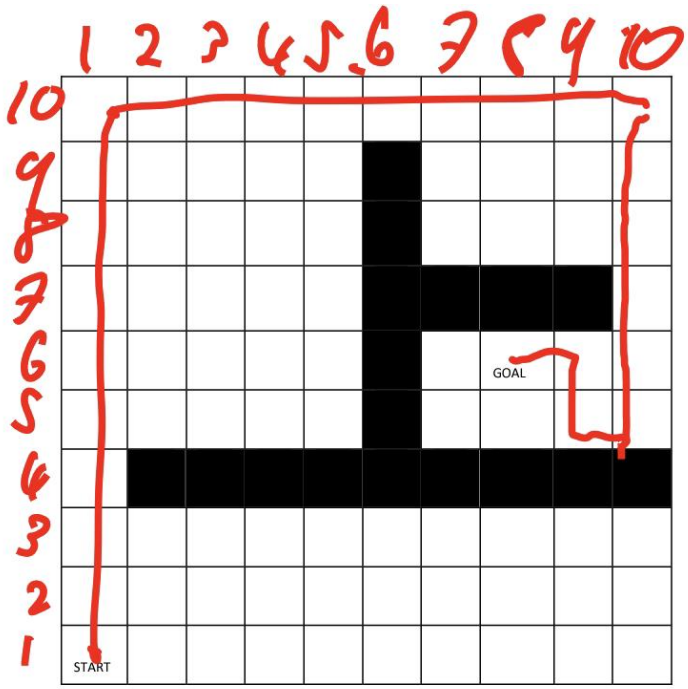

Figure 2 Sample maze 2

The exact same output will be also for the DFS algorithm.

However if we set a limit on the depth to something lower than what is required, the maze will stop running once it reaches the depth .

A* Heuristics

For the A* heuristic I went with the Manhattan distance. This is because we can not move diagonally. If we could, then the Euclidean would have been chosen. Manhattan distance

works by finding the shortest distance between 2 points as |x1-y1| + |x2-y2|. Here is the code for the Manhattan distance in prolog:

```
% A* search manhattan distance heuristic
h(pos(X,Y),W):-
    goal(pos(X_goal,Y_goal)),
    W is abs(X-X_goal) + abs(Y-Y_goal).
```

Where goal is the end position to reach, and W is the Weight, which is calculated on the distance between both the X and Y coordinates.

A* implementation

A * was implemented by finding the cost for each move, and then choosing the lowest available cost, if there were moves with the same cost, then the one with the higher priority was given(Up in this case).

This is the function to find the cost for the next move

```
move_cost(Position,Move,Cost):-
    move(Position,Move),
    changePos(Position,Move,Next),
    valid(Next),
    h(Next,H),
    Cost is H.
```

This function gets called in the lowest_cost function to find the lowest cost available move.

```
%function to find the lowest cost move
lowest_cost(Position,Move,Cost):-
    move_cost(Position,Move,Cost),
    move_cost(Position,Move_new,Cost_new),
    Cost_new > Cost.
lowest_cost(Position,Move,Cost):-
    move_cost(Position,Move,Cost),
    move_cost(Position,Move_new,Cost_new),
    Cost_new >= Cost.
```

In the first function, it returns the move if there is a clear low cost move, in the 2nd function, it returns the move if two or more moves will give the same cost.

The code gets ran the same with as the other 2 algorithms however the movement is dictated by the lowest cost. This is how it runs on the 2nd maze:

```
[debug]  ?- astar(Path).
pos(1,2)11
pos(1,3)10
pos(1,4)9
pos(1,5)8
pos(1,6)7
pos(2,6)6
pos(3,6)5
pos(4,6)4
pos(5,6)3
pos(5,7)4
pos(5,8)5
pos(5,9)6
pos(5,10)7
pos(6,10)6
pos(7,10)5
pos(8,10)4
pos(8,9)3
pos(8,8)2
pos(9,8)3
pos(9,9)4
pos(9,10)5
pos(10,10)6
pos(10,9)5
pos(10,8)4
pos(10,7)3
pos(10,6)2
pos(9,6)1
pos(8,6)0
```

With the number beside the pos(), being the distance away from the end goal.

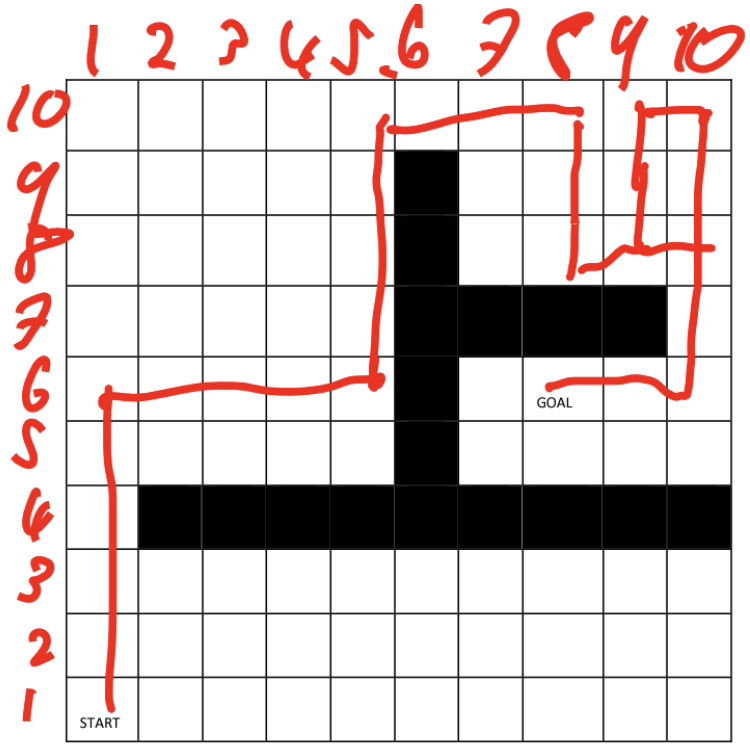

Figure 2 Sample maze 2

When the movement directions priorities are not set to the optimal, the A* algorithm would be the best 100% of the time, however if the movements put into optimal priorities, then the DFS and IDS can be just as good as the A* algorithm