Artificial Intelligence CS 512 Assignment 1

Pradeep Kumar 2019CSM1008

02 February 2020

1 Problem Statement

In this assignment, We have to find the path to Cercei's location in a very fast and efficient manner in the city of labyrinths for Melisandre. Environment for cair maze game is already created. We have to apply Breadth First Search (BFS), Depth First Search (DFS), Uniform Cost Search (UCS) and A* algorithms on given environment and tell the optimal path to Melisandre.

2 Basic Concepts and Implemented Algorithms

Using the coordinates of start position and goal position, we have to find path from start position to goal position. I implemented below algorithms to find the optimal path:

- Depth First Search
- Breadth First Search
- Uniform Cost Search
- A*

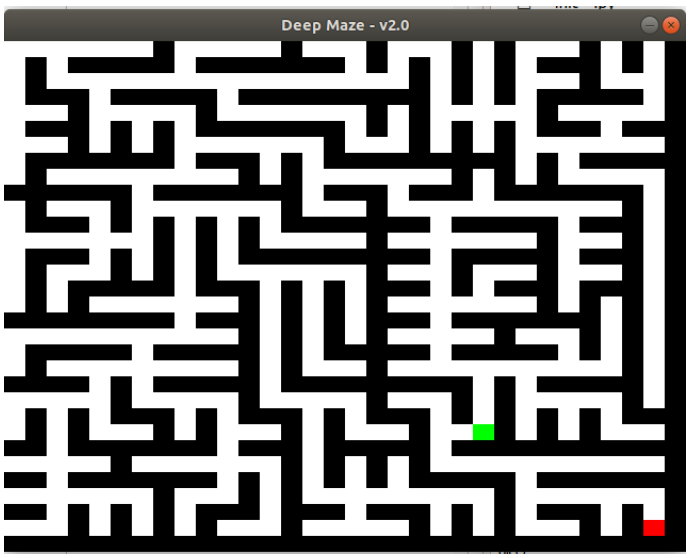


Figure 1: Cair Maze Game

In cair maze game, Green block is our current position, Red block is our goal position and Black block are walls.

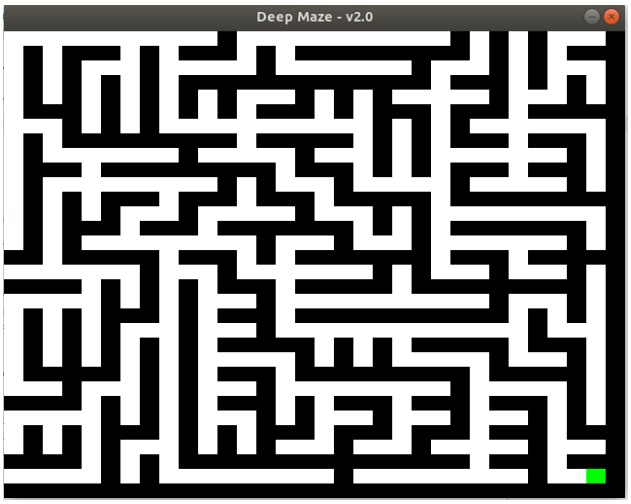
3 Results of each algorithm

In this experiment, I am running environment for 32x32 maze size. Environment is producing 2 output. First for 11x11 maze a=size and other for 32x32 maze size. By default, it is giving 2 outputs. For each output, my output format is:

- Start Position
- End(Goal) Position
- Optimal path from start to goal position
- Length of path
- Total required cost for path
- Time taken to run the algorithm (in microseconds)
- Memory consumed by data structure

3.1 Depth First Search

In this algorithm, We are traversing node(state) of graph in depth as much as possible. It can find the solution.



(a) DFS Output 32x32 Maze

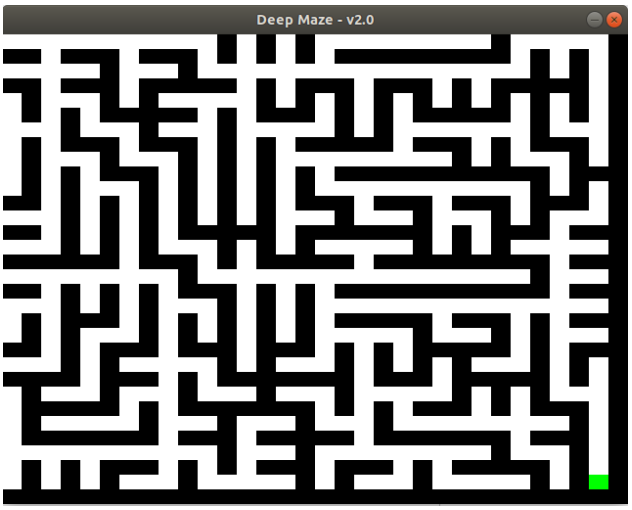
```
START ==>  (0, 0)

GOAL ==>  (30, 30)

PATH ==>  [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 6), (2, 6), (2, 7), (2, 8), (3, 8),
(4, 8), (4, 9), (4, 10), (5, 10), (6, 10), (7, 10), (8, 10), (8, 11), (8, 12), (7, 12), (6, 12), (6, 13),
(6, 14), (7, 14), (8, 14), (9, 14), (10, 14), (11, 14), (12, 14), (12, 15), (12, 16), (13, 16), (14, 16),
(14, 17), (14, 18), (14, 19), (14, 20), (15, 20), (16, 20), (17, 20), (18, 20), (19, 20), (20, 20), (20,
21), (20, 22), (21, 22), (22, 22), (23, 22), (24, 22), (24, 23), (24, 24), (25, 24), (26, 24), (27, 24),
(28, 24), (28, 25), (28, 26), (28, 27), (28, 28), (28, 29), (28, 30), (29, 30), (30, 30)]

PATH Length =  64

Total cost =  128

Total Expanded nodes =  470

Running Time:  12038.830001984024  microsecs

If each node is taking 1KB memory then maximum memory used by stack is 12 KB
```

(b) DFS Output for 32x32 maze

Figure 2: DFS

3.2 Breadth First Search

In this algorithm, We are traversing node(state) of graph in level wise level. We are expanding all possible valid path from start to goal. It can find the solution.



(a) BFS Output 11x11 Maze

```
START ==>  (0, 0)

GOAL ==>  (30, 30)

PATH ==>  [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4), (4, 5), (4, 6), (5, 6),
(6, 6), (6, 7), (6, 8), (7, 8), (8, 8), (8, 9), (8, 10), (8, 11), (8, 12), (8, 13), (8, 14), (9, 14), (10,
14), (10, 15), (10, 16), (11, 16), (12, 16), (13, 16), (14, 16), (15, 16), (16, 16), (16, 17), (16, 18),
(17, 18), (18, 18), (19, 18), (20, 18), (21, 18), (22, 18), (23, 18), (24, 18), (25, 18), (26, 18), (27,
18), (28, 18), (28, 19), (28, 20), (28, 21), (28, 22), (28, 23), (28, 24), (29, 24), (30, 24), (30, 25),
(30, 26), (30, 27), (30, 28), (30, 29), (30, 30)]

PATH Length =  60

Total cost =  120

Total Expanded nodes =  510

Running Time:  20264.787002815865  microsecs

If each node is taking 1KB memory then maximum memory used by queue is 16 KB
```

(b) BFS Output for 32x32 maze

Figure 3: BFS

3.3 Uniform Cost Search

In this algorithm, We are finding all possible path based on cost of path and storing in priority queue and selecting minimum cost path. It can find the optimal cost path.

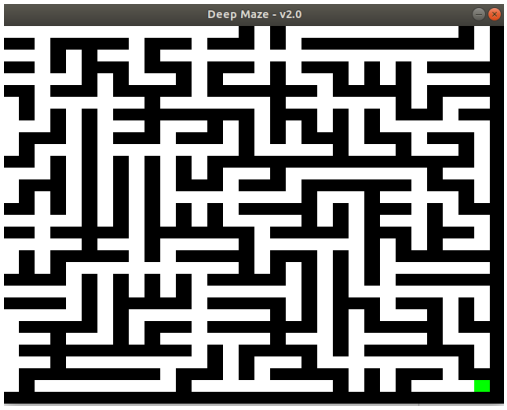
3.4 A* algorithm

In this algorithm, We are estimating(heuristic) the cost of all possible paths from current state and choosing minimum total cost path from priority queue. It can find the optimal path same as UCS algorithm

4 Observation

5 A* algorithm for New Environment

A* is complete and optimal algorithm. With few changes, We can use A* algorithm on new

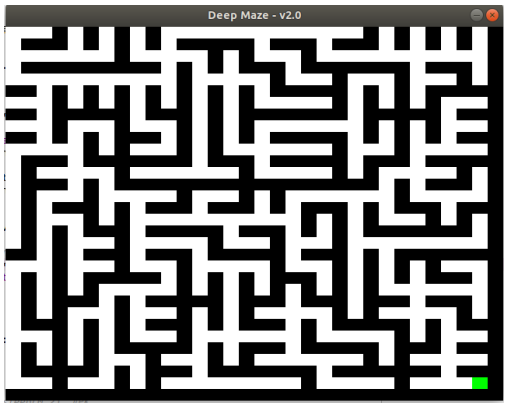


(a) UCS Output 11x11 Maze

```
START ==>  (0, 0)

GOAL ==>  (30, 30)

PATH ==>  [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 6), (4, 6), (5, 6),
(6, 6), (6, 7), (6, 8), (6, 9), (6, 10), (7, 10), (8, 10), (9, 10), (10, 10), (10, 11), (10, 12), (10,
13), (10, 14), (11, 14), (12, 14), (13, 14), (14, 14), (14, 15), (14, 16), (15, 16), (16, 16), (16, 17),
(16, 18), (17, 18), (18, 18), (19, 18), (20, 18), (20, 19), (20, 20), (20, 21), (20, 22), (20, 23), (20,
24), (20, 25), (20, 26), (20, 27), (20, 28), (21, 28), (22, 28), (23, 28), (24, 28), (25, 28), (26, 28),
(27, 28), (28, 28), (28, 29), (28, 30), (29, 30), (30, 30)]

PATH length =  60

Total cost =  90

Total Expanded nodes =  509

Running Time:  19241.592999605928  microsecs

If each node is taking 1KB memory then maximum memory used by priority queue is 18 KB
```

(b) UCS Output for 32x32 maze

Figure 4: UCS

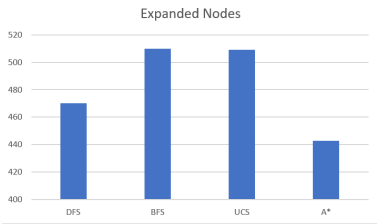


(a) A* Output 11x11 Maze

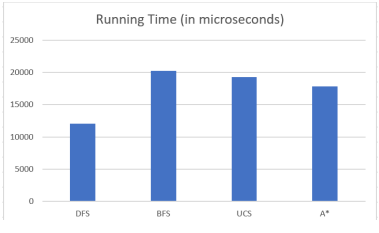
```
START ==>  (0, 0)

GOAL ==>  (30, 30)

PATH ==>  [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9),
(2, 10), (3, 10), (4, 10), (4, 11), (4, 12), (5, 12), (6, 12), (6, 13), (6, 14), (7, 14), (8, 14), (8,
15), (8, 16), (9, 16), (10, 16), (11, 16), (12, 16), (13, 16), (14, 16), (15, 16), (16, 16), (17, 16),
(18, 16), (18, 15), (18, 14), (19, 14), (20, 14), (21, 14), (22, 14), (22, 15), (22, 16), (22, 17), (22,
18), (23, 18), (24, 18), (24, 19), (24, 20), (24, 21), (24, 22), (25, 22), (26, 22), (26, 23), (26, 24),
(27, 24), (28, 24), (28, 25), (28, 26), (28, 27), (28, 28), (28, 29), (28, 30), (29, 30), (30, 30)]

PATH length =  64

Total cost =  94

Total Expanded nodes =  443

Running Time:  17825.12000136194  microsecs

If each node is taking 1KB memory then maximum memory used by priority queue is 21 KB
```

(b) A* Output for 32x32 maze

Figure 5: A* search Algorithm



(a) No. of expanded



(b) Running Time (in microseconds)

(c) Total Memory (in KB)

Figure 6: A* search Algorithm

environment. For the new environment, I am using 8-puzzle game in which we have to solve the puzzle by moving blank block in left, right, up and down directions.
Link: https://github.com/AZstudy/gym-sliding-puzzle

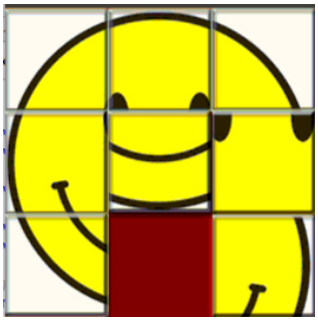


Figure 7: 8-puzzle Game

5.1 Results

```
START ==>  (0, 2, 5, 4, 1, 8, 3, 6, 7)

GOAL ==>  (0, 1, 2, 3, 4, 5, 6, 7, 8)

PATH ==>  [(0, 2, 5, 4, 1, 8, 3, 6, 7), (0, 2, 8, 4, 1, 5, 3, 6, 7), (0, 8, 2, 4, 1, 5, 3, 6, 7), (0, 1,
2, 4, 8, 5, 3, 6, 7), (0, 1, 2, 8, 4, 5, 3, 6, 7), (0, 1, 2, 3, 4, 5, 8, 6, 7), (0, 1, 2, 3, 4, 5, 6, 8,
7), (0, 1, 2, 3, 4, 5, 6, 7, 8)]

PATH length =  7

List of Actions =  [2, 0, 3, 0, 3, 1, 1]

Total Actions =  7

Total Expanded nodes =  12

Running Time:  382.47300108196214  microsecs

If each node is taking 1KB memory then maximum memory used by priority queue is 10 KB
```

(a) Output of 8-puzzle using A*

```
[0, 2, 8, 4, 1, 5, 3, 6, 7]
[0, 8, 2, 4, 1, 5, 3, 6, 7]
[0, 1, 2, 4, 8, 5, 3, 6, 7]
[0, 1, 2, 8, 4, 5, 3, 6, 7]
[0, 1, 2, 3, 4, 5, 8, 6, 7]
[0, 1, 2, 3, 4, 5, 6, 8, 7]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

(b) Sequence of states

Figure 8: 8-puzzle Game Results