CSE 4082 – Project#1

In this project, we have a defined maze and we try to solve this maze with uninformed and informed search methods like:

a. Depth First Search

b. Breadth First Search

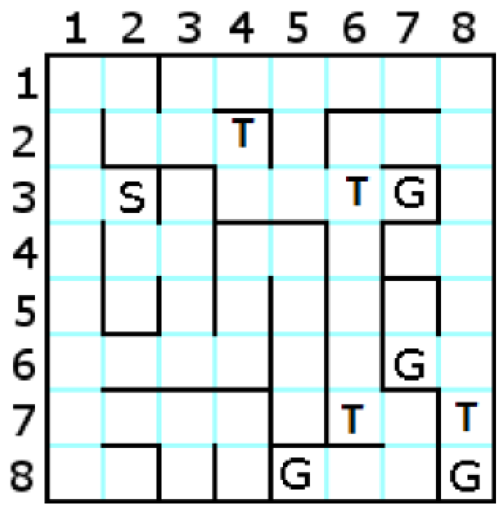
c. Iterative Deepening

d. Uniform Cost Search

e. Greedy Best First Search

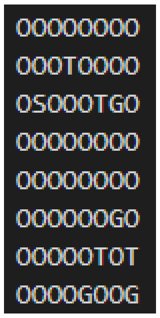
f. A* Heuristic Search

The given maze is:



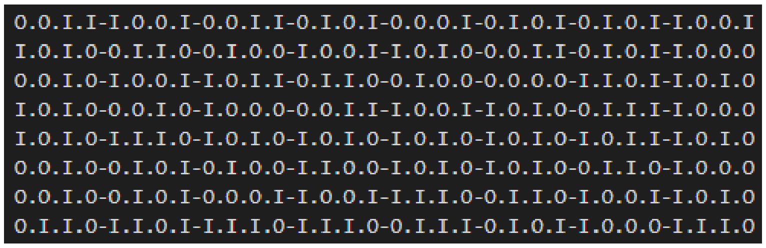
The maze is stored in a file for program in order to define walls and squares positions.

Squares stored in squares.txt as:



```
00000000
000T0000
0S000TG0
00000000
00000000
000000G0
00000T0T
0000G00G
```

Walls are stored in walls.txt as:



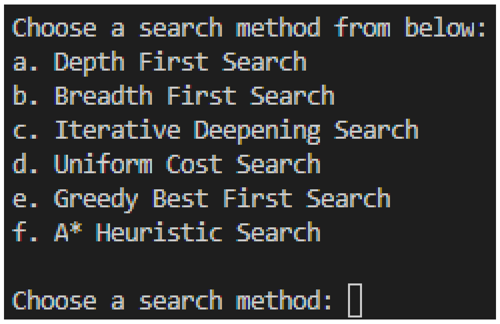
```
O.O.I.I-I.O.O.I-O.O.I.I-O.I.O.I-O.O.O.I-O.I.O.I-O.I.O.I-I.O.O.I
I.O.I.O-O.I.I.O-O.I.O.O-I.O.O.I-I.O.I.O-O.O.I.I-O.I.O.I-I.O.O.O
O.O.I.O-I.O.O.I-I.O.I.I-O.I.I.O-O.I.O.O-O.O.O.O-I.I.O.I-I.O.I.O
I.O.I.O-O.O.I.O-I.O.O.O-O.O.I.I-I.O.O.I-I.O.I.O-O.I.I.I-I.O.O.O
I.O.I.O-I.I.I.O-I.O.I.O-I.O.I.O-I.O.I.O-I.O.I.O-I.O.I.I-I.O.I.O
O.O.I.O-O.I.O.I-O.I.O.O-I.I.O.O-I.O.I.O-I.O.I.O-O.I.I.O-I.O.O.O
O.O.I.O-O.I.O.I-O.O.O.I-I.O.O.I-I.I.I.O-O.I.I.O-I.O.O.I-I.O.I.O
O.I.I.O-I.I.O.I-I.I.I.O-I.I.I.O-O.I.I.I-O.I.O.I-I.O.O.O-I.I.I.O
```

“O” represents there is no wall on this direction, and “I” represents a wall in this direction. Directions are order as EAST, SOUTH, WEST, and NORTH. This means that:

O.O.I.I means:

- O: EAST, No Wall
- O: SOUTH, No Wall
- I: WEST, Wall
- I: NORTH, Wall

When the program is started, it shows a menu as below:



```
Choose a search method from below:
a. Depth First Search
b. Breadth First Search
c. Iterative Deepening Search
d. Uniform Cost Search
e. Greedy Best First Search
f. A* Heuristic Search

Choose a search method: 
```

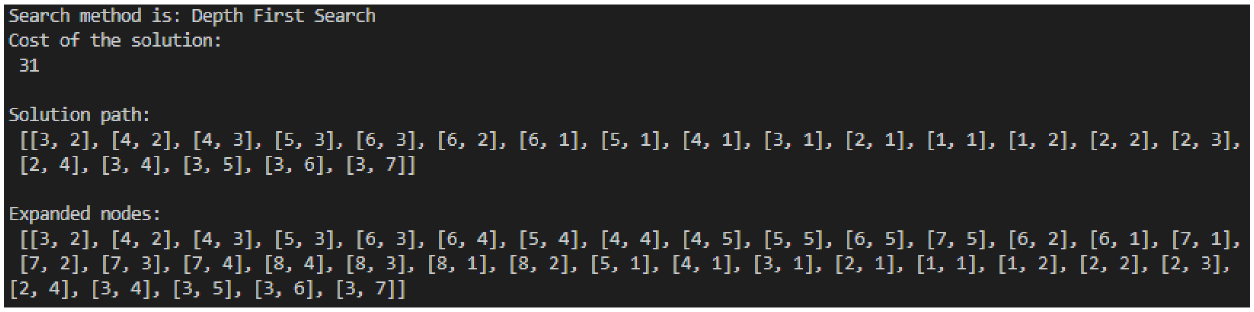
User can select which search method will be run on the maze solving problem by typing a, b, c, d, e, and f characters. For example, after typing ‘a’ and pressing return key, the program will run and it uses “Depth First Search” algorithm and print the result on the screen.

Result outputs contains:

- Costof the solution
- Solution path which means the agent movements to squares from start to goal squares
- Expanded nodes which means which squares are explored while the agent searchs goal

The outputs are:

1. Depth First Search

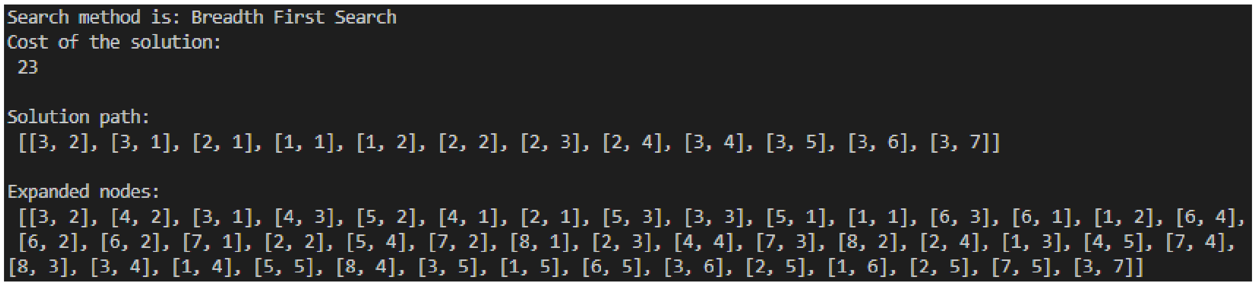


```
Search method is: Depth First Search
Cost of the solution:
 31

Solution path:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 2], [6, 1], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3],
 [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 4], [5, 4], [4, 4], [4, 5], [5, 5], [6, 5], [7, 5], [6, 2], [6, 1], [7, 1],
 [7, 2], [7, 3], [7, 4], [8, 4], [8, 3], [8, 1], [8, 2], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3],
[2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]
```

2. Breadth First Search

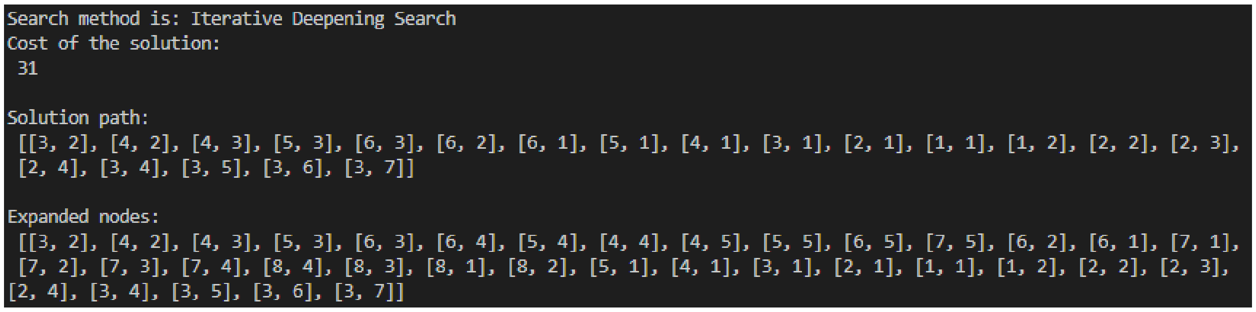


```
Search method is: Breadth First Search
Cost of the solution:
 23

Solution path:
 [[3, 2], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3], [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [3, 1], [4, 3], [5, 2], [4, 1], [2, 1], [5, 3], [3, 3], [5, 1], [1, 1], [6, 3], [6, 1], [1, 2], [6, 4],
 [6, 2], [6, 2], [7, 1], [2, 2], [5, 4], [7, 2], [8, 1], [2, 3], [4, 4], [7, 3], [8, 2], [2, 4], [1, 3], [4, 5], [7, 4],
[8, 3], [3, 4], [1, 4], [5, 5], [8, 4], [3, 5], [1, 5], [6, 5], [3, 6], [2, 5], [1, 6], [2, 5], [7, 5], [3, 7]]
```

3. Iterative Deepening Search

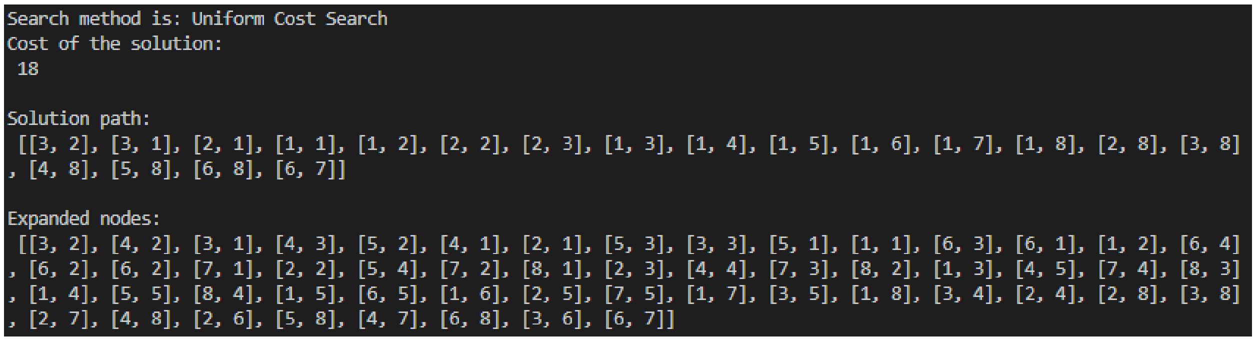


```
Search method is: Iterative Deepening Search
Cost of the solution:
 31

Solution path:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 2], [6, 1], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3],
 [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 4], [5, 4], [4, 4], [4, 5], [5, 5], [6, 5], [7, 5], [6, 2], [6, 1], [7, 1],
 [7, 2], [7, 3], [7, 4], [8, 4], [8, 3], [8, 1], [8, 2], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3],
[2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]
```

4. Uniform Cost Search

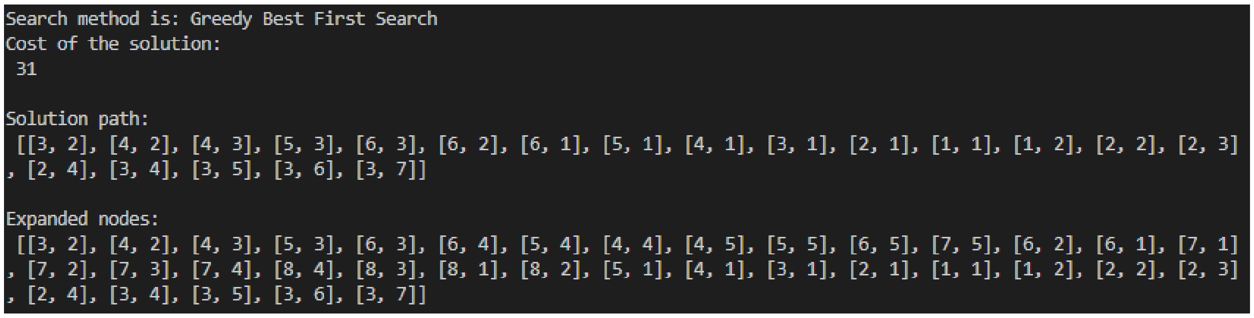


```
Search method is: Uniform Cost Search
Cost of the solution:
 18

Solution path:
 [[3, 2], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 8], [3, 8]
, [4, 8], [5, 8], [6, 8], [6, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [3, 1], [4, 3], [5, 2], [4, 1], [2, 1], [5, 3], [3, 3], [5, 1], [1, 1], [6, 3], [6, 1], [1, 2], [6, 4]
, [6, 2], [6, 2], [7, 1], [2, 2], [5, 4], [7, 2], [8, 1], [2, 3], [4, 4], [7, 3], [8, 2], [1, 3], [4, 5], [7, 4], [8, 3]
, [1, 4], [5, 5], [8, 4], [1, 5], [6, 5], [1, 6], [2, 5], [7, 5], [1, 7], [3, 5], [1, 8], [3, 4], [2, 4], [2, 8], [3, 8]
, [2, 7], [4, 8], [2, 6], [5, 8], [4, 7], [6, 8], [3, 6], [6, 7]]
```

5. Greedy Best First Search

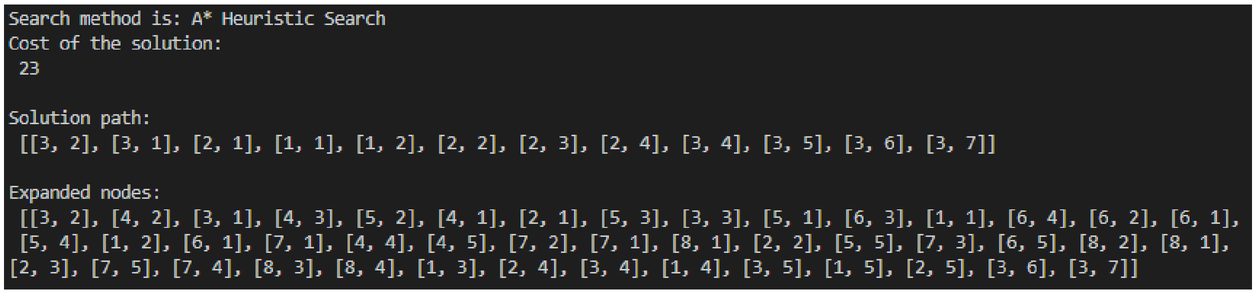


```
Search method is: Greedy Best First Search
Cost of the solution:
 31

Solution path:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 2], [6, 1], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3]
, [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [4, 3], [5, 3], [6, 3], [6, 4], [5, 4], [4, 4], [4, 5], [5, 5], [6, 5], [7, 5], [6, 2], [6, 1], [7, 1]
, [7, 2], [7, 3], [7, 4], [8, 4], [8, 3], [8, 1], [8, 2], [5, 1], [4, 1], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3]
, [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]
```

6. A* Heuristic Search



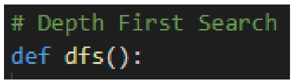
```
Search method is: A* Heuristic Search
Cost of the solution:
 23

Solution path:
 [[3, 2], [3, 1], [2, 1], [1, 1], [1, 2], [2, 2], [2, 3], [2, 4], [3, 4], [3, 5], [3, 6], [3, 7]]

Expanded nodes:
 [[3, 2], [4, 2], [3, 1], [4, 3], [5, 2], [4, 1], [2, 1], [5, 3], [3, 3], [5, 1], [6, 3], [1, 1], [6, 4], [6, 2], [6, 1],
 [5, 4], [1, 2], [6, 1], [7, 1], [4, 4], [4, 5], [7, 2], [7, 1], [8, 1], [2, 2], [5, 5], [7, 3], [6, 5], [8, 2], [8, 1],
[2, 3], [7, 5], [7, 4], [8, 3], [8, 4], [1, 3], [2, 4], [3, 4], [1, 4], [3, 5], [1, 5], [2, 5], [3, 6], [3, 7]]
```

Used methods for each search algorithm:

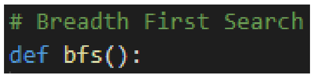
1. Depth First Search



```python
# Depth First Search
def dfs():
```

It uses this function as main body for itself. This function defines starting position, explores possible positions from current position and moves to a possible square with obeying the move rule (East first, North last) and moves to another possible squares of the new square. It goes deeper.

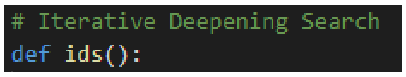
2. Breadth First Search



```python
# Breadth First Search
def bfs():
```

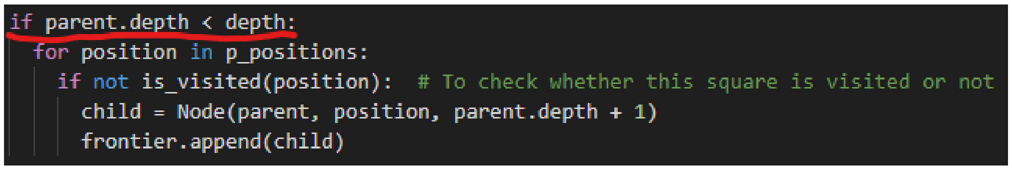
It uses this function as main body for itself. This function defines starting position, explores possible positions from current position and moves to a possible square with obeying the move rule (East first, North last) and moves to another possible squares of the previous square. It traverses widely not deeply as DFS.

3. Iterative Deepening Search



```
# Iterative Deepening Search
def ids():
```

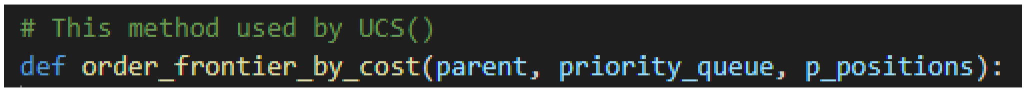
It runs like DFS with one different aspect:



```
if parent.depth < depth:
    for position in p_positions:
        if not is_visited(position):  # To check whether this square is visited or not
            child = Node(parent, position, parent.depth + 1)
            frontier.append(child)
```

It checks depth and runs only in strict depth level. When it cannot find, it increases depth level by one.

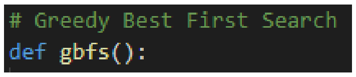
4. Uniform Cost Search
   It uses a different function from above search methods:



```
# This method used by UCS()
def order_frontier_by_cost(parent, priority_queue, p_positions):
```

This method takes all possible squares and order them according to movement cost. Then returns ordered frontier and main function takes next possible action from the ordered frontier list.

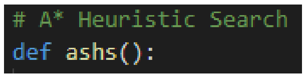
5. Greedy Best First Search



```
# Greedy Best First Search
def gbfs():
```

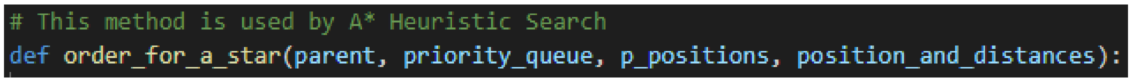
After finding possible movements to another square from current square, this algorithms calculates total distance to goals from that square and puts least cost square to first position in order to move to it first.

6. A* Heuristic Search



```
# A* Heuristic Search
def ashs():
```

It uses BFS heuristic and UCS cost calculation method. After combining their calculations with this method below;



```
# This method is used by A* Heuristic Search
def order_for_a_star(parent, priority_queue, p_positions, position_and_distances):
```

it takes a square which has the least "cost of square + distance to goals" value and moves to this square.