CS451 HW1(Solving N-Puzzle Problem with Search Algorithms) Report

Onur YILMAZ S009604

0. Introduction

The 8 Puzzle is a simple and fun game. It can be solved by heuristics search such as “A*” and uninformed search (also blind search) methods such as “BFS,DFS,UCS”. Heuristics are examined to allow the algorithm to find the optimal solution while examining as few states as possible, on the other hand, blind search methods try to reach the goal state from the start state differ only by the order or length of actions. This algorithms aim that showing the heuristics search algorithm and blind search algorithms performance differences using various parameters.

1. Implementation

This solution is implemented using Python3. It includes seven classes, Graph, Node, BFSearch, DFSearch, UCSearch, AStarSeatch and main file. Eight Puzzle contains the main function. It begins by asking grid size, then it takes two strings, on separate lines as initial state and goal state from standard input. They are both of the form number from 1 to 8. Then, If the input is correct, the next function asks the user for choosing a search method to solve the problem. If the problem is solvable, the number of visited nodes is printed as well as depth of graph and elapsed time. After printing the results, the program gives the menu again for another search algorithm implementation.

EXAMPLE SCENARIO:

```
Enter the grid size:
8
Enter initial state:
0 2 3
1 4 5
8 7 6
Enter goal state:
1 2 3
8 0 4
7 6 5

-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

1
Match found!
Num. of visited nodes: 59
Depth of graph: 6
Elapsed time: 0.0076770782470703125 secs.

-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit
```

```
-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

1
Match found!
Num. of visited nodes: 59
Depth of graph: 6
Elapsed time: 0.0076770782470703125 secs.

-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

2
Match found!
Num. of visited nodes: 1807
Depth of graph: 1320
Elapsed time: 0.04788613319396973 secs.
```

```
Enter the grid size:
3
Enter initial state:
0 2 3
1 4 5
8 7 6
Enter goal state:
1 2 3
8 0 4
7 6 5

-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

4
Match found!
Num. of visited nodes: 7
Depth of graph: 6
Elapsed time: 0.0010380744934082031 secs.

-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

5

Process finished with exit code 0
```

2. Details and Results

- Uninformed Search Methods:

a. **Breadth First Search**: Breadth-first search algorithm starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.This algorithm can be implemented by using queue data structure. That is, it works based on the first in first out (FIFO) principle. The time complexity is O(b^d+1) and space complexity is O(b^d+1), where ‘b’ is a branching factor and ‘d’ is the solution depth.

```
Enter the grid size:
8
Enter initial state:
1 2 3
0 4 5
6 7 8
Enter goal state:
1 2 3
4 5 0
6 7 8
```

```
-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

1
Match found!
Num. of visited nodes: 8
Depth of graph: 2
Elapsed time: 0.000827789306640625 secs.
```

b. **Depth First Search**: Depth-first search algorithm starts at the root node and explores as far as possible along each branch before backtracking. This algorithm can be implemented by using stack data structure. That is, it works based on the last In first out (LIFO) principle. The time complexity is O(b^m) and space complexity is O(b^m), where ‘b’ is a branching factor and ‘m’ is maximum depth.

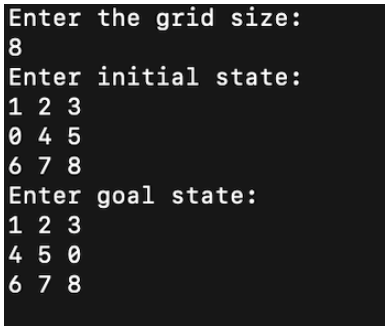
i. Depth First Search algorithms can find an optimal path solution for a few number of states(depth). However, after examining a great number of advanced states, eight puzzle games are unable to be solved by this search algorithm due to timeout.

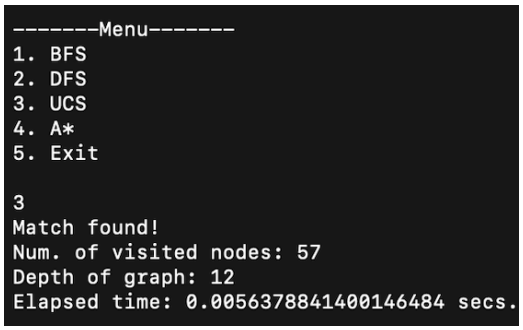
```
Enter the grid size:
8
Enter initial state:
1 2 3
0 4 5
6 7 8
Enter goal state:
1 2 3
4 5 0
6 7 8
```

```
-------Menu-------
1. BFS
2. DFS
3. UCS
4. A*
5. Exit

2
Match found!
Num. of visited nodes: 13
Depth of graph: 12
Elapsed time: 0.0010941028594970703 secs.
```

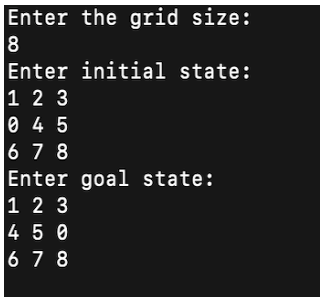
c. **Uniform Cost Search:** The Uniform Cost Search (UCS) is a state space search algorithm in which it finds the goal state based on the cost of the node. That is, the nodes are expanded with a minimum cost path. To calculate cost of every node, consider this equation, c(m) = c(n) + c(n, m), where c(m) is the cost of the current node, c(n) is the cost of the previous node, and C (n, m) is the weight of the edge.

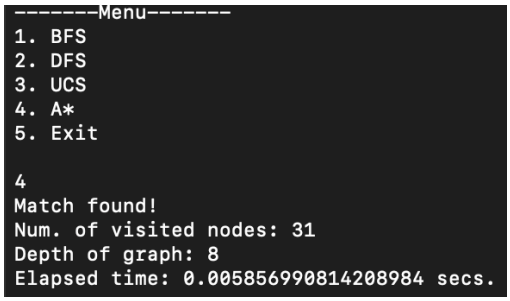




- **Heuristics Search Methods:**

a. **A* Search:** A* is an extension of Dijkstra's algorithm with some characteristics of Breadth First Search. A* search implemented with 2 different heuristics. The first is not really a heuristic at all like other blind search methods, it simply returns true if the root is in the goal position. The second uses only the Manhattan Distance heuristic which Look at your cost function and finds the minimum cost for moving from one space to an adjacent space.





3. Conclusion

Even though all of these search methods can find an optimal solution, they each take different amounts of time, different depth size and different visited nodes to do so. The more complicated the heuristic search gives the faster the results among other search algorithms.