Yongfeng Liang
SID: 862128237

CS170 Project#1(Eight Puzzle) Report

Challenges

One of the challenges I faced is how to build the structure of the project. For example, the classes, functions, and variables that I need to initialize. The discussion session held by the TA and a little bit of online search helped me a lot.

Another big challenge is that I do not know how to trace the sequence of operation from initial state to goal state. A tree class described in the project specification will do the job, but I do not know how to implement a tree class. Instead, I took another approach by combining this part in the Node class and use a recursive function to print the sequence of operations.

Design

I used C++ for this project. I construct three classes along with some helper functions. These three classes are Node, Problem, and Comparator. Node class and Problem class are the core of this project and Comparator is just a minor class for the priority queue. I also have three function that corresponds different algorithms. The design of these functions is the same as mentioned in the project specification. The main function serves as an interface for user instructions.

The Problem class manages the initial state and goal state of the puzzle. It also provides operation for the search algorithm and modification for the initial state. As I mentioned above, the Node class has the component of Tree class which are parent node, child nodes, etc. This class stores the information of a state(node) including g(), h(), f(), and the functions to calculate them. Of course, it also has functions to print the operations, the puzzles of itself and its parents.

The program does not require any special command or flag, just put it in a C++ compiler and run it. First, the program asks the user to choose between a default initial puzzle state and a user specify initial puzzle state. When the user chooses the second option. It will ask the user to enter the value of initial state. Then, the program lists three choices of algorithm and prompts user to select one. After that, the program will use the selected algorithm to search for the goal state. In the meantime, it will print the information(g() and h()) and puzzle of each step.

When it finds the goal, it will print the trace of path as well as the time and space required. See example below.

Comparison of time and space between algorithms

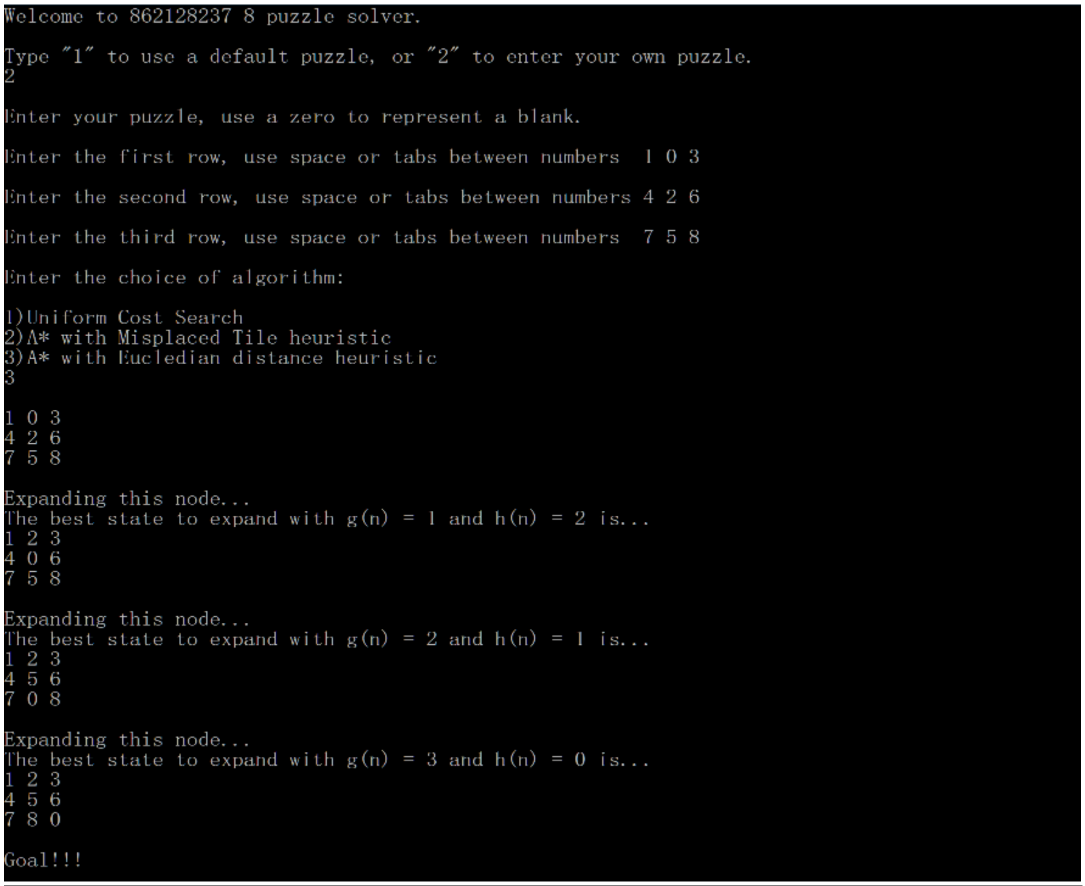
Trivial	Very Easy	Easy	Doable	Oh Boy	Impossible
[1 2 3]	[1 2 3]	[1 2 *]	[* 1 2]	[8 7 1]	[1 2 3]
[4 5 6]	[4 5 6]	[4 5 3]	[4 5 3]	[6 * 2]	[4 5 6]
[7 8 *]	[7 * 8]	[7 8 6]	[7 8 6]	[5 4 3]	[8 7 *]

Search Algorithms	Trivial	Very Easy	Easy	Doable	Oh Boy	Impossible
Uniform Cost Search	Time: 0 Space: 0	Time: 3 Space: 5	Time: 3 Space: 4	Time: 28 Space: 18	Time: 85524 Space: 24969	Time: ∞ Space: ∞
A* with the Misplaced Tile heuristic	Time: 0 Space: 0	Time: 1 Space: 3	Time: 2 Space: 3	Time: 4 Space: 4	Time: 6717 Space: 3846	Time: ∞ Space: ∞
A* with the Euclidean Distance heuristic	Time: 0 Space: 0	Time: 1 Space: 3	Time: 2 Space: 3	Time: 4 Space: 4	Time: 1493 Space: 857	Time: ∞ Space: ∞

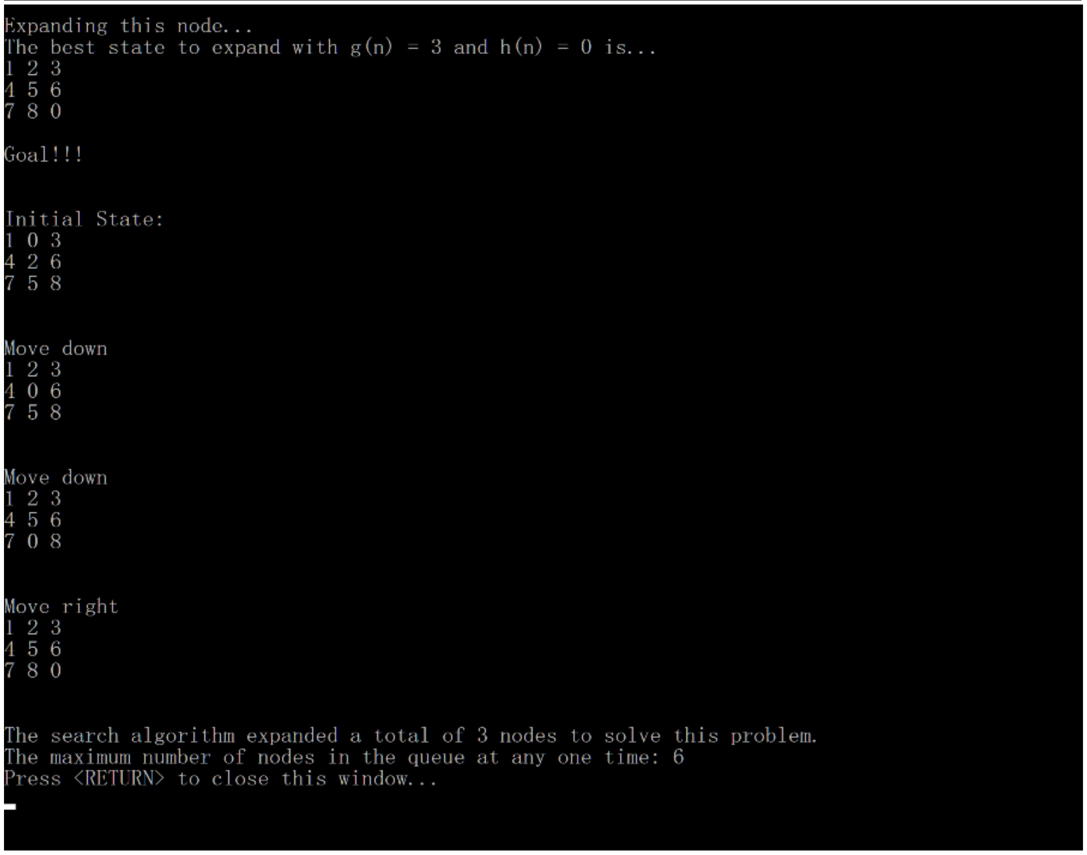
The time and space required between three algorithms is Uniform Cost Search > A* with the Misplaced Tile heuristic > A* with the Euclidean Distance heuristic. A* with the Euclidean Distance heuristic search algorithm is most efficient algorithm among them.

Test Cases: A trace of the Euclidean distance A* on the puzzle

[1 * 3]
[4 2 6]
[7 5 8]



```
Welcome to 862128237 8 puzzle solver.

Type "1" to use a default puzzle, or "2" to enter your own puzzle.
2

Enter your puzzle, use a zero to represent a blank.

Enter the first row, use space or tabs between numbers  1 0 3

Enter the second row, use space or tabs between numbers 4 2 6

Enter the third row, use space or tabs between numbers  7 5 8

Enter the choice of algorithm:

1)Uniform Cost Search
2)A* with Misplaced Tile heuristic
3)A* with Eucledian distance heuristic
3

1 0 3
4 2 6
7 5 8

Expanding this node...
The best state to expand with g(n) = 1 and h(n) = 2 is...
1 2 3
4 0 6
7 5 8

Expanding this node...
The best state to expand with g(n) = 2 and h(n) = 1 is...
1 2 3
4 5 6
7 0 8

Expanding this node...
The best state to expand with g(n) = 3 and h(n) = 0 is...
1 2 3
4 5 6
7 8 0

Goal!!!
```



```
Expanding this node...
The best state to expand with g(n) = 3 and h(n) = 0 is...
1 2 3
4 5 6
7 8 0

Goal!!!


Initial State:
1 0 3
4 2 6
7 5 8


Move down
1 2 3
4 0 6
7 5 8


Move down
1 2 3
4 5 6
7 0 8


Move right
1 2 3
4 5 6
7 8 0


The search algorithm expanded a total of 3 nodes to solve this problem.
The maximum number of nodes in the queue at any one time: 6
Press <RETURN> to close this window...
```

References

https://stackoverflow.com/questions/20670882/priority-queue-of-pointers-c

https://www.youtube.com/watch?v=nPYRG8vwYe4

https://gist.github.com/Hossam-Elbahrawy/391f060242e7203702da0843fd523d4f