CS205 Project1: The eight-puzzle

NAME Zijian Li
SID 862545559

File Structure

There are four python files in total, their functions are as follows:
main.py: The main file handles user input and the preliminary preparations for running the code (such as initial state, final state, algorithm used, etc.)
uniform_Cost_Search.py: Here are the functions needed by the *Uniform Cost Search algorithm*. First, I define the nodes in the search tree, and the rest are the functions needed for the code to run (see the code introduction section for details).
misplaced_Tile_heuristic.py: Here are the functions needed for the *A* with the Misplaced Tile heuristic algorithm*. I define the nodes in the search tree the same way as above, and the rest are functions needed for the code to run.
manhattan_Distance_heuristic.py: Here are the functions needed for the *A* with the Manhattan Distance heuristic algorithm*. I define the nodes in the search tree the same way as above, and the rest are functions needed for the code to run.

How to run

1. First, make sure that all four files are in the same directory path.
2. Then run the *main.py* file
3. Follow the instructions to fill in the parameters you want (such as matrix size, initial state, algorithm used, etc.). If you want to use the default parameters, just press the *enter* key to skip. Note that the default initial matrix is a 3x3 matrix, you cannot use the default matrix after entering a different matrix size.
4. Wait for the output result. Generally speaking, if the matrix is small, the result can be obtained immediately. If the initial matrix has a solution, the output result is a complete search path; otherwise, *No solution found* is output.

Algorithms

A* algorithm

The A* algorithm is a heuristic graph search algorithm used to find the path with the minimum total cost from the starting point to the end point. It uses the following evaluation function:

$$f(n)=g(n)+h(n)$$

g(n): The actual cost from the starting point to the current node n.
h(n): The estimated cost from the current node n to the target (heuristic function).

Uniform Cost Search

Uniform Cost Search is a variant of breadth-first search, which expands nodes from small to large according to the actual cost of the path g(n). This process is very similar to Dijkstra's algorithm. Therefore, Uniform Cost Search always chooses the path with the smallest cumulative cost to expand. Since only the distance from the current node to the starting state is considered, not the distance to the final state, the time complexity of this algorithm is higher than the other two algorithms.

A* with the Misplaced Tile heuristic algorithm

Misplaced Tile heuristic means how many numbers are not in the position they should be in the current state compared to the target state. For example, in the following state:

The target state is:

1 2 3
4 5 6
7 8 0

The current state is:

1 2 3
4 5 6
8 7 0

Then 8 and 7 are not in their target positions, so in this step h(n) = 2

A* with the Manhattan Distance heuristic algorithm

Manhattan distance refers to the sum of the differences between the horizontal and vertical coordinates of each position and its target position (in this question, it is simply the sum of the differences between the horizontal and vertical coordinates). By counting the Manhattan distance of each number from its final position, we can know the search distance from the current node to the target node. For example, in the following state:

The target state is:

1 2 3
4 5 6
7 8 0

The current state is:

1 2 3
4 0 6
7 5 8

Then for position 5, the Manhattan distance is: 1; For position 8, the Manhattan distance is: 1.

Code Introduction

uniform_Cost_Search.py

First, I defined the nodes in the search tree, which record the current state of the node (the position of each number), the parent node of the node, and the distance from the node to the initial node (g(n)). I used the heap algorithm (*heapq*) in Python to build and maintain the search tree. Since we need to find the node with the lowest cost each time among a bunch of candidate nodes to expand, we need a priority queue sorted by cost. At the same time, I used a recursive method to iterate each round of possible candidate nodes. When the recursion reaches the final state or the heap is empty (the final state cannot be reached), the algorithm ends and the result is output. In the function, *set_parameter()* is used to receive parameters from the *main.py* file, and *general_search()* is the main function used to initialize the algorithm (the same as in the assignment requirements).

misplaced_Tile_heuristic.py

Same as uniform cost search, I first define the nodes in the search tree. Different from uniform cost search, I use *misplaced_tiles()* function to calculate the h(n) value of the current node and add it to the total cost. I also use the heap algorithm to find the node with the smallest cost in each iteration. The rest of the algorithm is the same as uniform cost search, so I won't go into details here.

manhattan_Distance_heuristic.py

Same as the previous two algorithms, I first define the nodes in the search tree. Here I use *manhattan_distance()* function to calculate the h(n) value of the current node, that is, the distance from each number to its final result position, and add it to the total cost. I also use the heap algorithm to find the node with the smallest cost in each iteration.

Results

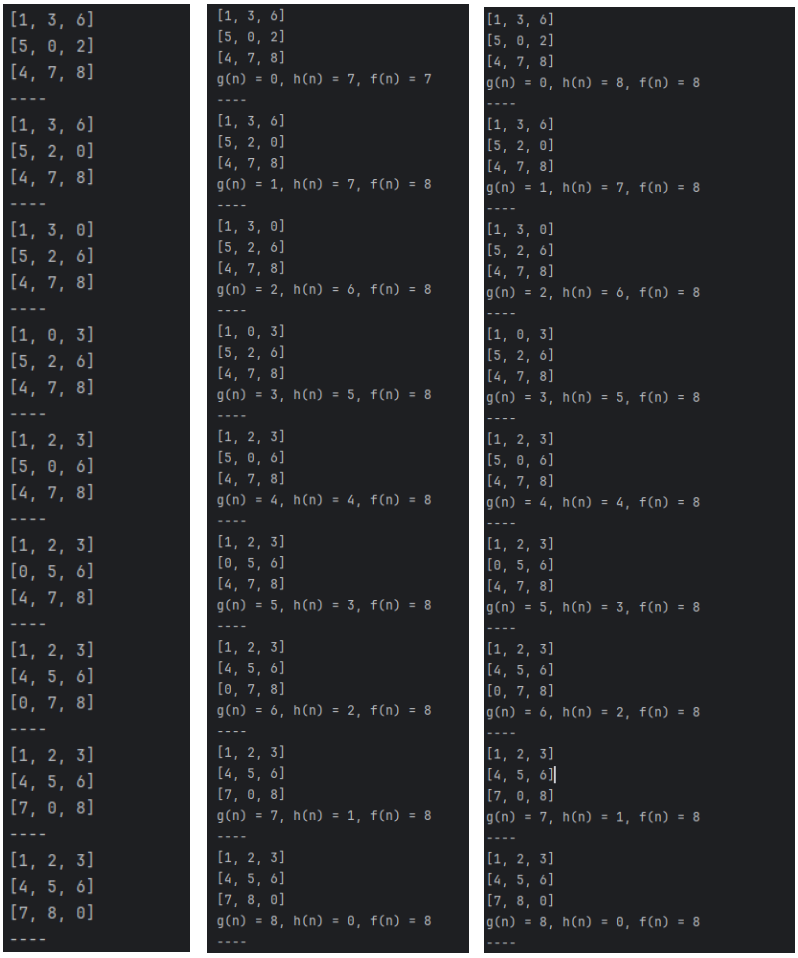
Here we take the default initial state and goal state in the code as an example.

initial state:

1 3 6
5 0 2
4 7 8

final state:

1 2 3
4 5 6
7 8 0



All the results are shown in the figure above. To save space, I put three figures together. The first figure is the result of *Uniform Cost Search*, the second is the result of *A* with the Misplaced Tile heuristic algorithm*, and the third is the result of *A* with the Manhattan Distance heuristic algorithm*.

References

https://github.com/sevdeawesome/A-Star-Search

https://www.datacamp.com/tutorial/manhattan-distance