1 Environments and instructions

- Python 3.8
- To run test, simply run `python pancake_sort.py`. You can check the result by the clear visualization of the search process
- Notice that we do not test the legitimacy of the input. You can input the number `1~10` in any order and seperate with `,`
- The correctness of UCS is checked but only shows message if it goes wrong

2 Define the problem as a search problem

A search problem is defined by 5 components, in this pancake problem we have:

1. **Initial state**: A list (permutation) of 10 numbers, which reprensent the size of the pancakes. *The first position in the list represents the first pancake of the stack.*
2. **Possible actions available**: The stack of pancakes with size 10 can be flipped in 9 different positions, from the very bottom to the bottom of the first pancake. We will consider the very bottom of the stack as the 9th position and the bottom of the first as the 1st position.
3. **Successor function**(Transition model): The $flip(k)$ operation will reverse the first $k+1$ number(s) in the list.
4. **Goal test**: Check if any candidates has reached the goal state `[1,2,3,4,5,6,7,8,9,10]`
5. **Path cost function**: we define a backward cost and a forward cost for this problem.

3 Backward cost

The cost function $g(s)$of this problem is defined as the number of $flip()$ operations that have been performed. Notice that it is reasonable to assign a cost of 1 to each $flip()$, regardless of its location. This is because any problems with $flip(1)$ in the solution cannot be replaced by $flip(2)$ or $flip(9)$.

4 Forward cost

In this problem, the heuristic value of a state is the number of stack positions for which the pancake at that position is not of adjacent size to the pancake below plus one if largest pancake is not on the bottom position. More formally, Let $s = (s_1, s_2, \ldots, s_{10})$ be an 10-pancake state, the forward cost is defined as:

$$h(s)\begin{cases} := |\{i|i \in \{1,\ldots,9\}, |s_i - s_{i+1}| > 1\}|, s_{10} = 10 \\ := |\{i|i \in \{1,\ldots,9\}, |s_i - s_{i+1}| > 1\}|, s_{10} \neq 10 \end{cases}$$

5 Implementation of an A* algorithm

In each iteration of search in the A* algorithm, we pop the state with the lowest total cost from the priority queue. If the target state is found in the frontier, then the algorithm terminates; otherwise we need to fetch all possible states and add them to the priority queue by $flip()$ operation.

6 Implementation of UCS algorithm

It is basically the same as A* algorithm except that we remove the $h(s)$ cost in A* algrothm.