Read Me

CS 131

hw2

Neal Zhao

Assume: the position of pancakes is 1 indexed. (start with 1)

Important Warning:

This code must be execute with python version >= 3.10

The code uses python module `bisect` for binary search, in which a argument `key` is used, and this property is added in python 3.10

However, the **eecs terminal's python version is 3.9.2** (at least mine terminal is this case), so running on the eecs terminal would crash .

How to use

Execute the `hw2.py` file.

It will prompt requests for input.

- The cost function indicate which searching algorithm you want to use. Please input `astar` for A* and `ucs` for uniform-cost-search. Other input will not be accepted.
- The initial stack requires you to input integers separate by **Space**. It must be a valid stack: the input should be a permutation of successive integers starting 1. For example, `1 2 3 7 8 6 5` is a valid input, where as `1 2 2 3 4` or `2 3 4 5 6` are not.

Invalid input will immediately terminate the program.

Question responses

1. Define the problem
    - The initial state is the initial order of the pancake, given by successive integers start at 1. In the code, this is stored as a tuple.
    - Possible action is to flip some pancakes on the top.
    - Successor function: in the code, the action will reverse some number of element of at the end of the pancake stack tuple.
    - Goal: the pancakes are stacked with the larger at the bottom. In the code, the integers in the stack tuple is in descending order.
    - Path cost function: each flip cost 1. Total cost is the number of flips needed.
2. backward cost
    - The number of flips taken that made the initial stack to the current stack.
3. forward cost

    - As represented in the paper, the forward cost is determined by a heuristic function that counts the number of pancakes that is not below a adjacent size pancake.
4. See the code.
5. Yes, UCS could be done since the backward cost is always positive. Implementing this by making the heuristic function always return 0. Also because the cost of each step is fixed, UCS is equivalent to a bread-first-search in this case.

Implementation choices

Each node in the frontier is stored in class `node` and they form a tree-structure that can only trace from bottom to top. In the searching functions, the node instances will refer to a node in this tree. All the node in this tree also presents in a dictionary that takes the pancake stack tuple as the key, and the node as its value. The pancake stack is represented as a tuple because it is ordered and hashable. This decrease the time it takes to check if the node has already shows up. When a node is removed from the frontier, its stack tuple will be sent to a set called `visited`.