Programming Assignment 2

This assignment is about anagrams, which are pairs of distinct words that contain the same letters. One example is the pair ‘ALLERGY’ and ‘LARGELY’. By reordering their letters, we can turn one word into the other. Note that the same letter can appear multiple times in each word, such as the letter ‘L’ appearing twice in each word here. It always has to be the same number for both words.

Now let us assume that we want to turn one word of an anagram into the other by a series of steps. In each step, we can remove the first letter in the word and insert it into any other position. For example, we can start with ALLERGY, remove its first letter, ‘A’, and insert it into the fourth position of the new word to get LLEARGY. Then we can take another step to modify the new word, and so on until we reach LARGELY. For example:

ALLERGY – move the first letter to position 4:
LLEARGY – move the first letter to position 6:
LEARGLY – move the first letter to position 2:
ELARGLY – move the first letter to position 5:
LARGELY

We want to use the A* algorithm to find an optimal (minimum number of steps) solution for a given pair of start and goal words. For this purpose, please start with the attached file “assignment1.py” which includes the base code.

a. (10 pts) Even before applying the A* algorithm, you can already decide whether a given problem is solvable or not. Please add code to anagram_solver to perform this check and terminate to avoid running A* on a problem that has no solution.

b. (50 pts) You should job is to fill in the body of the a_star function in the given assignment1.py file. The a_star function needs to know the problem’s start state, the desired goal state, and an expand function that expands a given node. The expand function receives as its input the current state, and the goal state, and returns a list of pairs of the form: (new_state, h-score). There is exactly one such pair for each of the possible moves in the current state, with new_state being the state resulting from that move, paired with its h-score. Note that this is not the f-score but the h-score, i.e., an admissible (optimistic) estimate of the number of moves needed to reach the goal from the current state. The expand function does not know g and therefore cannot compute f; this has to be done by the a_star function. Also, note that anagram_expand does not (and cannot) check whether it is creating a search cycle, i.e., whether it generates a state that is identical to one of its ancestors; this has to be done by a_star as well. The given anagram_expand function counts the number of mismatched tiles (excluding the empty tile) as a scoring function.

c. (40 pts) The code already includes a very simple expand function anagram_expand that, for a given state (i.e., the current word), returns a list of pairs, indicating all successor states and their h' scores. It is a complete function, i.e., the code is already able to solve some simple problems. The code also includes a counter variable to tell you how many A* iterations were performed to find the solution.

   However, the given scoring function is extremely simple: It returns 1 if the current state is not a solution and 0 otherwise. Because of this poor function, solving some problems is impossible within a reasonable amount of time. Therefore, your task is to replace the scoring function so that it can solve problems more efficiently.

Submission:

Submit the .py file on blackboard

Deadline: 12th October, 2023