Name: Carter Huang

## **Objective:**

Given a dictionary, the program solves crossword puzzles with all the vowels provided already.

Here is an example for the crossword puzzle:





Initial

Solved

By filling in the blanks the words in rows and columns should be valid ones from the dictionary.

## Usage:

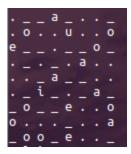
To run the program, just type:

\$ python solver.py <test\_case\_file>

For example, to solve the puzzle above, go to "crossword" directory and type:

\$ python solver.py test\_cases/test3

Each puzzle board is represented as chars separated by space. The above puzzle can be represented as:





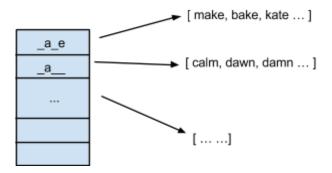
Dots represent black tiles in the board while underscores represent blank tiles. To do a complete run of all five test cases, just execute "\$ ./run.sh".

"solver.py" serves as the driver module of the program. "crossword.py" covers the main logic and "utility" includes all the helper subroutines. "corpus" is the word dictionary that has 10,0000 words. Note: all the characters in the "corpus" should be lower-cased.

If you are creating new test cases, make sure all the words are covered by "corpus". If not, please manually add them into "corpus". "fruitninja" and "aspca" are not automatically covered.

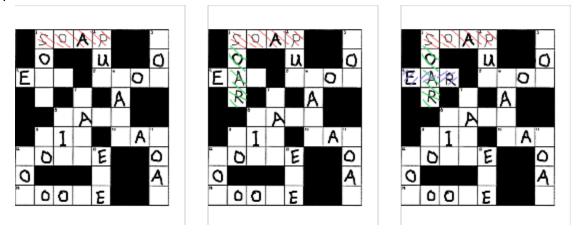
## Algorithm:

The first part of the algorithm is preprocessing the dictionary and grouping of words with the same pattern. A pattern represents relative positions of vowels in a word. For example, The pattern for both "make" and "bake" is "\_a\_e". Therefore, words with the same patterns can be grouped together and stored in the same hash slot of a hash table. Here is a graphical demonstration:



Before processing the dictionary, scan the board and find all the patterns. Then find all the words that match those patterns. Assume the dictionary is of size O(N), then by doing this the seach space for each word is shrinked to  $O(\frac{N}{\alpha})$ , while  $\alpha > 1$ .

The second part is to recursively try all different combinations as well as backtracking. The algorithm follows a DFS pattern to move from one incomplete word to another. Here is a graphical demonstration:



Assume that a board contains M incomplete words, then the backtracking algorithm yields  $O((\frac{N}{\alpha})^M)$  runtime complexity as well as O(N) space complexity for storing the hashmap.

Most of crossword puzzles are solved within one minute using this program.