Building Programming Skills, 2021
Project: Jumble your letters

In this project, you will build an anagram finder.

1 The goal

Given a word/phrase, you need to find *anagrams* of the word. An anagram is a word that has exactly the same letters, like "code" and "deco". Your code will also need to compute anagram phrases. Given the word "program", the phrase "prom rag" is an anagram.

Your code should take two command line arguments. The word to be anagrammed, and the name of an output file where anagrams will be printed. Later on, we will discuss some extensions that can provided as additional command line arguments.

Now, you need a list of valid English words to find all anagrams. I have found a list here: `http://www.wordgamedictionary.com/twl06/download/twl06.txt`. This is a list of words used by Scrabble players, and is in the repo as the file `word-list.txt`.

Also, as an example of what your code should do, check out the online anagram finder at `https://www.crosswordsolver.com/anagram-solver`. You aim is to write up their backend algorithm.

2 Let me show you the way

Always start simple. As simple as possible. And then go even simpler.

Start by trying to just find a single anagram of the word (if it exists). This is much easier exercise, and should give you some good practice in recursion. It is based on another classic interview question. Simply construct all permutations of the word and see if any of them match a word in `word-list.txt`.

Now, let us try to up our game. See if you can anagram into two words, and then three. Ignore all one letter words (or maybe even two letter words).

The next step is to take as input a phrase that may be longer than a word. While it does not change the basic ideas, your input can be much larger. At this

point, your simple algorithm might fail. If the input phrase has 15 letters, there are 1.3 *trillion* permutations of the letters. There is no way your computer can compute that many permutations.

You have to think efficiency now. Observe that the vast majority of permutations are not anagrams. When your code builds a permutation, you can probably check quickly if there is any possibility that it will lead to an anagram. For example, if I'm trying to anagram "anagram", there is no point in finding all permutations that start with "mrg". There is no word that starts with these letters.

So for every initial portion of the permutation, you will need to check the word file to see if this portion extends to some word. Now think: what data structure should you store the words in to determine this quickly?

This project will make you understand the interplay of algorithms and data structures through efficiency. Often, to make your code faster, you will need to use the right data structure. Conversely, your data structures will often dictate how your algorithm should be structured.

This is a challenging project, so don't forget to have fun with the anagrams you discover!