Anagrams

In this assignment, you will solve the combinatorial problem of finding all the anagrams of a sentence.[1]

An anagram of a word is a rearrangement of its letters such that a word with a different meaning is formed. For example, if we rearrange the letters of the word "Elvis" we can obtain the word "lives", which is one of its anagrams. In a similar way, an anagram of a sentence is a rearrangement of all the characters in the sentence such that a new sentence is formed. The new sentence consists of meaningful words, the number of which may or may not be the same as the number of words in the original sentence. For example, tan anagram of the sentence "I love you" is "You olive". In this exercise, we will consider permutations of words to be anagrams of the sentence. In the above example, "You I love" is considered to be a separate anagram. When producing anagrams we will ignore the character casing and the punctuation symbols.

Your goal is to implement a program which takes a sentence and lists all its anagrams of that sentence. You will be given a dictionary that contains all meaningful words.

The general idea is to examine the list of characters. To find the anagrams of a word, we will find all the words from the dictionary which have the same character list. To finding anagrams of a sentence, we will extract subsets of characters to see if we can form meaningful words. For the remaining characters we will solve the problem recursively and then combine the meaningful words we have found.

For example, consider the sentence "You olive". The list of characters in this sentence is "eiloouvy". We start by selecting some subset of the characters, say "i". We are left with the characters "eloouvy". Checking the dictionary we see that "i" corresponds to the word "I", so we've found one meaningful word. We now solve the problem recursively for the rest of the characters "eloouvy". This will give us a list of solutions: [["love", "you"], ["you, "love"]]. We then combine "I" with that list to obtain the sentences "I love you" and "I you love", which are both valid anagrams.

The types are as follows:

- A "word" is a string that contains lowercase and uppercase characters, but no whitespace, punctuation or other special characters.
- A "sentence" is a list of words.
- A "character count" is a mapping from characters to integer numbers. It can be represented using a list of `(Char,Int)` pairs, or the `Map` type in `Data.Map`. Examine especially the `fromList` and `fromListWith` functions. Whether the order of the entries in the mapping is significant or not is up to your design.

The steps of the assignment are outlined below:

1. Write a function `wordCharCounts` that takes a word and returns its character counts. For example, if the word is "mississippi" the result should report that there are 1 of 'm', 4 of 'i', 4 of 's', and 2 of 'p'. Try to express this function as some combination of higher-order list functions like `map` and `filter`. *Hint*: You can use prelude functions like `toLower` or `nub`.
2. Write a function `sentenceCharCounts` that takes a list and returns its character counts. Try to express it as a composition of functions.
3. Write a function `dictCharCounts` that takes a list of dictionary words and returns a mapping from words to their character counts. For example, if the dictionary contains the words "eat", "all", and "tea", this should report that the word "eat" has 1 of 'e', 1 of 'a', 1 of 't'; the word "all" contains 1 of 'a', 2 of 'l'; the word "tea" contains 1 of 't', 1 of 'e', 1 of 'a'.
4. Write a function `dictWordsByCharCounts` which takes in the result of the previous function and returns a map of words which have the same character counts. For the example above the result should map the character counts 1 of 'a', 2 of 'l' to the word list ["all"], and the character counts 1 of 'a', 1 of 'e', 1 of 't' to the word list ["tea", "eat"].
5. Write a function `wordAnagrams` which takes a word and the result of the previous function and returns a list of anagrams for the given word. For example, if the parameters are `"ate"` and the result given above, it should return the words "tea" and "eat".

1 This exercise and is text is taken from an online course by Prof. Martin Odersky with some minor changes.

6. Write a function `charCountsSubsets` that takes character counts and returns all possible subsets of these counts. For example, the character counts of the word "all" is 1 of 'a', 2 of 'l', and the subsets of the character counts should be (listed in no specific order):
    - empty
    - 1 of 'a'
    - 2 of 'l'
    - 1 of 'l'
    - 1 of 'a', 1 of 'l'
    - 1 of 'a', 2 of 'l'
7. Write a function `subtractCounts` that takes two mappings of character counts and returns a new mapping where counts in the second map are subtracted from the counts in the first map. For example, if your first map is 1 of 'a', 3 of 'e', 2 of 'l', and your second map is 1 of 'a', 1 of 'l', the result should be 3 of 'e', 1 of 'l'. You can assume that the second map is a subset of the first map.
8. Write a function `sentenceAnagrams` that takes a sentence and returns a list of sentences which are anagrams of the given sentence. Test your function on short sentences, no more than 10 characters. The search space gets huge very quickly as your sentence gets longer, so the program may run for a very long time. However, for short sentences such as "Linux rulez", "I love you", or "Mickey Mouse" the program should end fairly quickly.
9. Write a `main` function that takes a short sentence as a command line parameter and prints its anagrams. The program should generate its dictionary by reading the given dictionary text file (extracted from the zip file).

Name your module *Main.hs* and make sure that the following command creates an executable:

```
ghc Main.hs -o bonus3
```

Then it should be runnable as explained above:

```
./bonus3 "i love you"
```