Gaming fan's nadir

Two strings are anagrams of one another if, ignoring capitalisation, punctuation and whitespace they contain the same characters. For instance `Finding Anagrams` and `Gaming fan's nadir` are anagrams. Utilities such as I rearrangement servant can help find anagrams which arise most commonly in cryptic crossword puzzles but also in other sorts of wordplay. The intent of this étude is to ask you to reproduce some of the functionality of such utilities.

Task

Write a program that, using a user-supplied dictionary, finds all anagrams of a given string using at most a certain number of words from the dictionary. Assuming Java as the programming language the program (say `Anagrams`) should be invoked as follows:

```
java Anagrams "*Finding Anagrams!*" 3 < dictionary.txt
```

This would output to `stdout` all anagrams of `*Finding Anagrams!*` consisting of at most three words from the supplied dictionary. The dictionary will contain up to 100,000 strings consisting only of lower case characters from `a` to `z` inclusive, one per line. Note that the input string may well require some modification (as in this example — the output would be the same if the input were `findinganagrams`).

The anagrams you find must be output to `stdout` in the following order:

- Individual anagrams should be listed on a single line in descending order of word length, with words of equal length in alphabetical order.
- Anagrams using fewer words precede ones using more words.
- If two anagrams use the same number of words, they should first be compared by word length — if the longest words differ in length, then the one with a longer first word should be listed first. If the longest words are equal in length but the second longest words differ in length then the one with a longer second longest word should be listed first. And so on.

- If two anagrams have the same number of words, and corresponding words have the same length, then they should be listed in alphabetical order.

(Pair 2)

Notes

This étude requires you to juggle three things at once.

- you have to find all ways of decomposing something into pieces from a dictionary
- you are given character *sequences* but have to work with character *bags*
- you have to produce the output in a particular way.

Concerning the last point, there are in general two ways to produce output in a particular order:

- Generate the output in that order, or
- generate the output in any convenient order, storing items as you go, then sort the stored items, then produce the output in that order.

You are *not* being told which way to use or which way is better in this specific case; this is just a reminder to think about the question.

Another question: if an anagram is a sequence of words, is it a String or is it something else?

Testing

As always, start by teting the very simplest cases.

- What should happen for an empty string?
- What should happen for an empty dictionary?
- What should happen if the string is “a” and the dictionary is just the word “aa”?

- What should happen for a string of n copies of the letter 'a' if the dictionary contains just the word "a"?
- What should happen for a string of n copies of the letter 'a' if the dictionary contains the two words "a" and "aa"?
- What should happen if the dictionary contains the 26 words "a", "b", ..., "z"?

Each pair of you should write your own code. That doesn't mean you can't use another pair's code as an *oracle*. If program X and program Y give different answers, at least one of them is wrong.