Anagrams

A sample application, which reads a text file and retrieves all the possible anagram sets.

How to get the application

The application is available for download or cloning at the public github repository

https://github.com/falagna/anagrams/tree/master

How to run the application

To run the application, either:

- Go to https://travis-ci.org/falagna/anagrams and click on the left on "Restart build";
- From any development environment (Eclipse, Intellij), run class AnagramServiceIntegrationTest as a junit test;
- From a shell, go to the project folder and run "mvn test" (requires Maven).

The source code

To implement the algorithm, an **AnagramClassModel** has been used. It represents a collection of words which are all anagrams among themselves. The collection is identified by a key, which can be any of the words of the class.

The core application logic is the **AnagramService** interface, implemented in two different ways in the classes **BasicAnagramService** and **DirectPickingAnagramService**.

BasicAnagramService is a first rough implementation. For each word in the source file, it loops among all the already existing **AnagramClassModel** objects to find a corresponding match. If a match is found, the word is added to the anagram class; if not, a new anagram class is created, containing the current word.

The algorithm has a computational complexity of O(n^2), where n is the number of words in the source file; hence, it is very inefficient for medium - high loads (n > 1000).

DirectPickingAnagramService is a more efficient algorithm, with a computational complexity of O(n). Instead of looping through a list of **AnagramClassModel** objects to find a maching one, the algorithm picks his anagram class (when it exists) directly through a Map.get(key). This is possible because the key used to identify the class itself is the sorted version of the word, which is unique for any anagram class.

Test cases

Three different test cases have been considered:

1. Sample dataset (7 words)
2. Small dictionary (80368 words)
3. Medium dictionary (349900 words)

While the first dataset can be easily processed by both the basic and the direct picking algorithm, the inefficiency of the basic algorithm is clearly shown when processing dataset 2 and 3. In fact, while the basic algorithm takes some minutes to process dictionaries 2 and 3, the direct picking one just takes less than a second to process both.

Scalability considerations

In terms of scalability, the direct picking algorithm performs well for $n < 10000000$. Assuming an average of 10 characters per word and one byte per character (ASCII encoding), this means about 100Mb of heap memory consumed to store the whole anagram map.

For higher volumes of data, a different solution would be preferable. One possible solution would be storing the anagram mapping on a database. Although this may be less efficient of an in-memory storing, it would allow an higher load and parallel computing.