University of West Bohemia

Faculty of Applied Sciences

Programming Techniques

Data structure for three dictionaries

Rafał Karwowski

10.11.2020

Problem analysis:

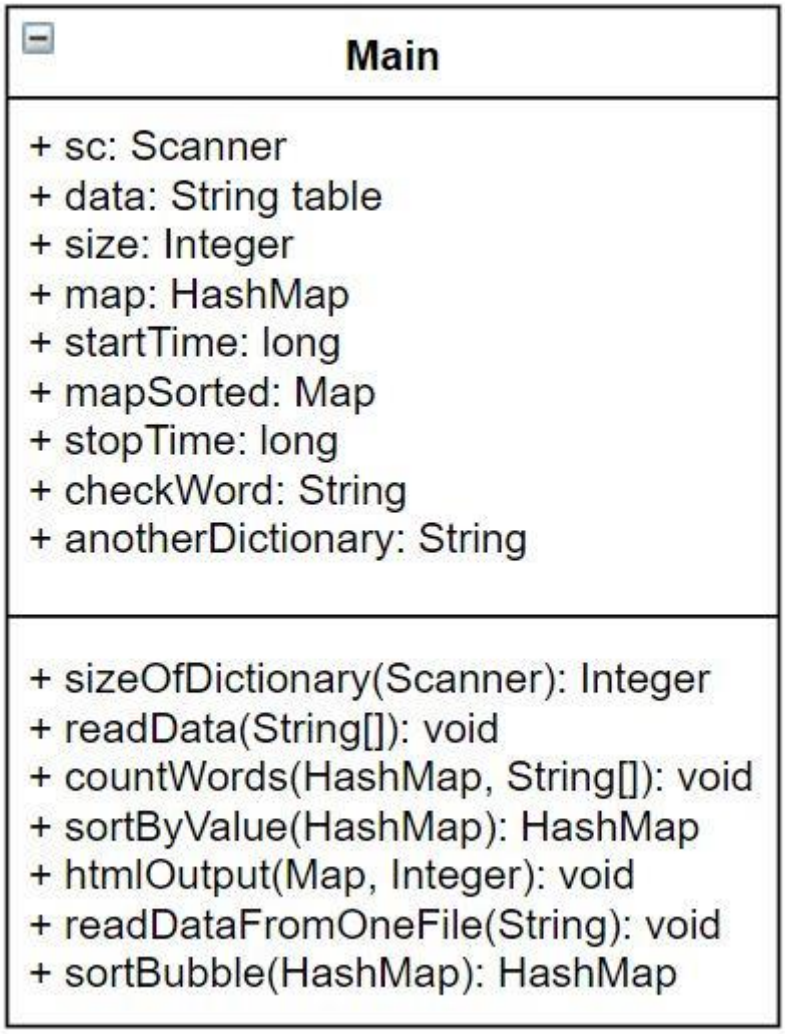
To create dictionaries, program has to start by reading data. Data are in small files, what increase time of execution. To decrease it, there are two main options: copy all data to one big file or use faster hard drive.

When data are ready, it is time to count occurrence of words. To do that it is necessary to have data structure in which program stores information. Data structure has to store only two values: word and number of occurrences. For this kind of problem the best option is using HashMap. It can access data really quickly by calculating index of key. In this way, it saves time, because it doesn't have to search for correct index.

Data are stored and program can sort them. From HashMap, data are copy to List. Next, method sort() from Collections class is called. It uses Merge Sort, which is fast, has time complexity of O(n log n), but is not as space-efficient as other sorting algorithms.

Finally, HTML file is created by putting sorted Map to table.

Software design:



Conclusion:

Usually, I would count words in the same function as reading words. In this way I would skip process of putting data to String array and reading it. Also program wouldn't have to go through data two times. I didn't do this because I wanted to count time of each operation, so I can see what improvement gives the best result.

As we can see the biggest problem is with reading data (from 7600 files):

```
Reading data from files: 43640 milliseconds.
Counting words: 254 milliseconds.
Sorting: 6 milliseconds.
Creating HTML file: 10 milliseconds.
```

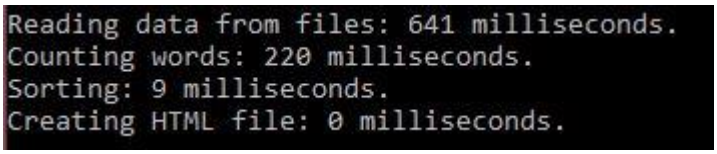
The easiest solution was to change the hard drive from hdd to sdd:

```
Reading data from files: 5900 milliseconds.
Counting words: 205 milliseconds.
Sorting: 5 milliseconds.
Creating HTML file: 0 milliseconds.
```

But, it turns out that storing data only in one file (I copied data from every file to one: "onefile.txt") is the most efficient way:

```
Reading data from files: 510 milliseconds.
Counting words: 305 milliseconds.
Sorting: 5 milliseconds.
Creating HTML file: 0 milliseconds.
```

What surprising, in this scenario, sdd is slightly slower than hdd:



I want to point out that every test was executed multiple times to make sure results are correct.

Of course there are more ways of making the process of reading faster (using BufferedReader, reading whole file into memory instead of line by line…), but I think it was not the point of the exercise. In the end the most important things are process of counting words and sorting them are really fast and they fulfil the assumptions.