Exploration and Presentation - Assignment 3

Anders Jacobsen, Dima Karaush

April 26, 2021

Contents

1 Intro

We are using the project Letter Frequencies downloaded from `https://github.com/CPHBusinessSoftUFO/letterfrequencies` in this paper. The entire project with all files can be found here `https://github.com/Cosby1992/CPH-business-assingments/tree/master/Data%20Science/Assignment_3_optimization`.

2 System Information

OS Name	Microsoft Windows 10 MO
OS Version	10.0.19042 N/A Build 19042
System Type	x64-based PC
Processor(s)	Intel® Core™ i7-10700KF Processor, 16M Cache, up to 5.10 GHz
BIOS Version	American Megatrends Inc. 1.10, 21-05-2020
Total Physical Memory	32.688 MB
Disc(s)	Force Series™ MP510 980GB M.2 SSD (up to 3480MB/sec read)

3 Enviroment

IDE	Visual Studio Code
IDE version	1.55.2 x64
Language	Java
Language Version	15

4 Analysis - Task 1

1. Find a point in your program that can be optimized (for speed), for example by using a profiler
2. Make a measurement of the point to optimize, for example by running a number of times, and calculating the mean and standard deviation (see the paper from Sestoft)
3. If you work on the letter frequencies program, make it at least 50% faster

4.1 Find a point in your program that can be optimized

We did not use a profiler. The reason is that we could not get the profiler to run in Visual Studio Code. We have however located multiple points in the program that can be optimized. Since we are reading a relatively big text-file, we believe it is here we can optimize the most. We took multiple steps to make the program faster as shown in the list below:

1. Upgrading the FileReader to a BufferedReader.
2. Replacing the “HashMap<Integer, Long>“ with a “int[]“

3. Limiting the while loop to only saving letters A-Z

We've decided to measure the two methods in the program "tallyChars()" and "print_tally()". We measure them together as a sequence to see the difference execution time on all our optimizations.

4.2 Bottlenecks

1. A bottleneck in this program was the use of a FileReader instead of a BufferedReader. A buffer when reading a file is a huge advantage and can significantly improve reading times of the file.

2. Another bottleneck in this program was that there were no checks on weather it was needed to count the frequency of the data. It should be limited to only count characters a-z.

3. The use of a HashMap and LinkedHashMap instead of primitive data type array. It is much faster to write a number to a now location in an Array than to add to values to a HashMap or LinkedHashMap.

A bottleneck in this program was the use of a FileReader instead of a BufferedReader. A buffer when reading a file is a huge advantage and can significantly improve reading times of the file.

4.3 Make a measurement of the point to optimize

To make measurements we first had to create a timer that supports our measurements. The timer class can be found in "letterfrequencies/src/main/java/cphbusiness/ufo/letterfrequencies/Timer.java".

Procedure

We've decided to make multiple measurements in order to create a realistic image of the run-times of both the optimized and non-optimized classes. The way we do this is by creating a loop and then run the two methods for a specified amount of iterations and writing each iterations run-time to a CSV file for the analysis.

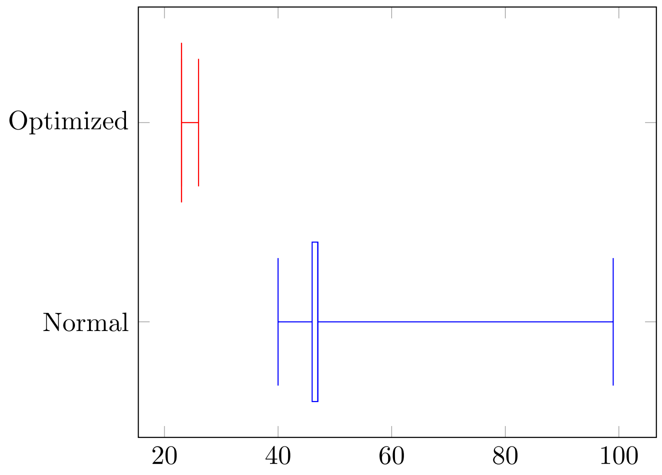
We've chosen to use 500 continuous measurements to calculate the statistics afterwords.

4.4 Make it at least 50% faster

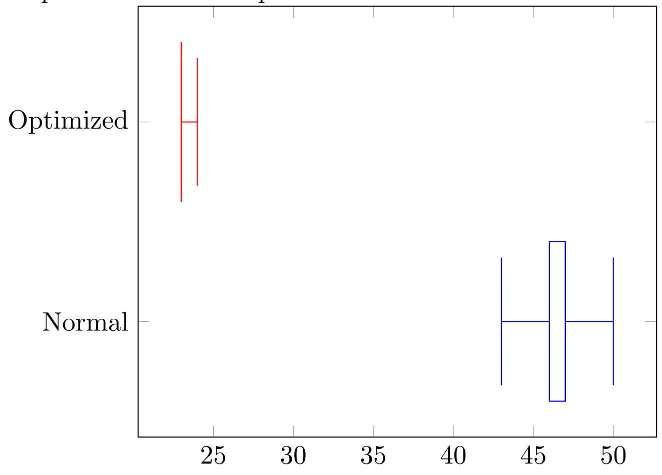
The task was to make the program 50% faster, and that has been achieved. Below are some tables and statistics along with a box-plot descriping our results.

	Normal	Optimized
Mean	46,562ms	23,232ms
Std. dev	3,351510293	0,571690178
Min	40ms	23ms
1. Quatile	46ms	23ms
Median	47ms	23ms
3. Quartile	47ms	23ms
Max	99ms	26ms

From this data it is possible to create a box-plot to visualize the difference.



As it can be seen in the box-plot, there is a few outliers that could be removed. Esspecially from the Normal plot. With the ouliers removed, the new boxplot looks like the plot below:



It is now possible to see that not only did we improve a whole lot on the execution-time, but we have also stabilized the system a whole lot. This is visible from observing the whiskers on each entry in the box-plot. As it is visible from the results table, the results have improved by a whole lot, but how much exactly? This depends on what you are measuring, we've decided that two of the numbers we can use to illustrate the improvements are the Mean and the Median. Therefore we have calculated how much the run-times have improved in percent. The results can be seen below:

Mean improvement	50,11%
Median improvement	51,06%

Results

In conclution we have improved the program by approx. 50%-51%. But what is also worth noticing is the stabilization of the execution-times with our optimized solution.

5 Steps to reproduce

Go to `https://github.com/Cosby1992/CPH-business-assingments/tree/master/Data%20Science/Assignment_3_optimization` and follow the instructions from the README.md