

# Exploration and Presentation - Assignment 3

April 19, 2021

benjamin kongshaug `cph-bk131@cphbusiness.dk`

Amanda Hansen `cph-ah433@cphbusiness.dk`

Amalie Landt `cph-sl307@cphbusiness.dk`

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Task 1</b>	<b>4</b>
2.1	Find a point in your program that can be optimized (for speed), for example by using a profiler . . . . .	4
2.2	A hypothesis of what causes the problem . . . . .	4
2.3	A changed program with better performance . . . . .	5
2.4	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) . . . . .	6
<b>3</b>	<b>Conclusion</b>	<b>7</b>

# 1 Introduction

Vi har fået udleveret projektet LetterFrequencies. Programmet læser indholdet af et dokumentet kaldet FoundationSeries.txt og tæller antallet af hver bogstav i dokumentet. Vi har i dette projekt optimeret programmet og dokumenteret analyse af baseline samt de ændringer vi har lavet og hvordan det har forbedret programmet.

## 2 Task 1

### 2.1 Find a point in your program that can be optimized (for speed), for example by using a profiler

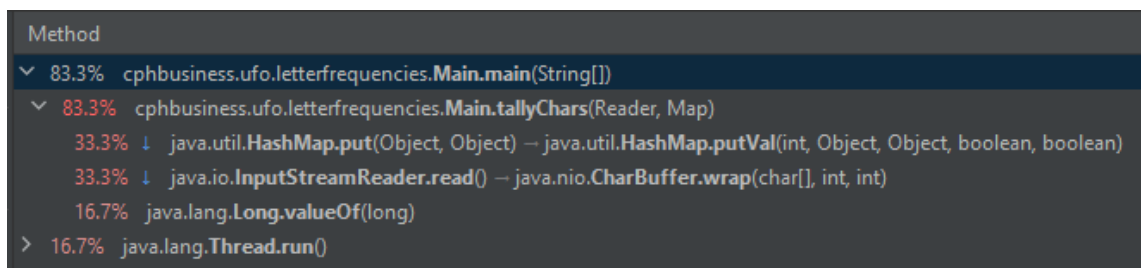
Før optimering tager programmet: Time elapsed (millisec): 389

Først og fremmest kan vi se 83 procent af tiden befinder programmet sig i metoden tallychars og at det er her vi skal optimere

Sekundært ser vi en ligelig fordeling på 33,3 procent af tiden brugt på at lægge input i hashMap og på at læse fra input reader filen

Vi kan optimere ved at finde en alternativ måde at indlæse fra filen og ved at gemme mængden af hver bogstav på en mere effektiv måde

Figure 1: Output fra profiler



Method	
83.3%	cphbusiness.ufo.letterfrequencies.Main.main(String[])
83.3%	cphbusiness.ufo.letterfrequencies.Main.tallyChars(Reader, Map)
33.3%	↓ java.util.HashMap.put(Object, Object) → java.util.HashMap.putVal(int, Object, Object, boolean, boolean)
33.3%	↓ java.io.InputStreamReader.read() → java.nio.CharBuffer.wrap(char[], int, int)
16.7%	java.lang.Long.valueOf(long)
16.7%	java.lang.Thread.run()

### 2.2 A hypothesis of what causes the problem

Reader reader = new FileReader(fileName); En reader læser data fra en fil, en linje af gangen når man bruger reader.read() hvilket ikke er særlig effektivt når man har af gøre med en stor fil

## 2.3 A changed program with better performance

ved at anvende en `BufferedInputStream` istedet for en `reader` kan vi gøre indlæsningen af dataen hurtigere, da vores `.read` metode tager mod data fra bufferen istedet for filen og det er hurtigere at indlæse data fra en buffer.

Vi forbedre programmet ved at anvende en `Buffered input stream` som illustreret nedenfor:

```
1      String fileName = "/Users/benja/Documents/Downloads/  
    letterfrequencies/src/main/resources/FoundationSeries.txt"  
    ;  
2      //Reader reader = new FileReader(fileName);  
3  
4      File f = new File(fileName);  
5      FileInputStream fis = new FileInputStream(f);  
6      BufferedInputStream reader = new BufferedInputStream(  
    fis);
```

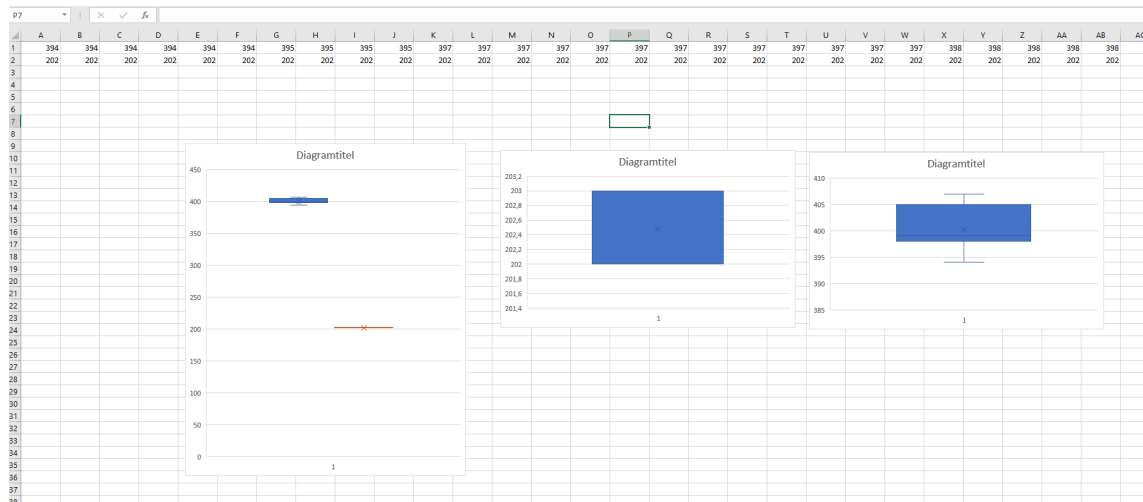
Code Listing 1: optimised tallyChars

Tid for metoden at køre for optimering i millisec over 100 kørsler :

Tid for metoden at køre efter optimering i millisek over 100 kørsler:

Vi kan se ud fra vores box plot at vores mean og standard diviacion på ingen måde overlapper og at der tydeligt er et proformance boost i vores program.

Figure 2: box plots af proformance



### 3 Conclusion

Vi har ved at anvende en profiler fundet frem til at programmet bruger 33 procent af runtime på at læse fra filen, som anvender en FileReader. Vi har optimeret ved at udskifte FileReaderen med en BufferedInputStream. Før optimering var vores average performance omkring 400 milisek og den ligger nu omkring 200 milisek. Vi har derfor opnået en 50 procents forbedring.