

Assignment 3

Code optimization and reporting

Mads Wulff Nielsen
Claus Kramath

21. april 2021

Indhold

1	Introduktion	3
2	Instruktion	3
3	Fremgangsmåde	4
3.1	Fravalg af forslag 1	5
3.2	Anvendelse af alternativ reader	5
3.3	Indlæsningstider	6
3.4	Indlæsning af filen i bidder vha. char array	6
4	Videre optimering	7

1 Introduktion

Programmet, vi vil forsøge at optimere, beregner frekvensen af forekomsten af et bogstav i en tekst. Programmet er klonet fra flg. github repository:

1. <https://github.com/CPHBusinessSoftUFO/letterfrequencies>

Det oprindelige program indlæser en tekstfil, gennemløber karaktererne i denne enkeltvis og akkumulerer antallet af hver karakters forekomster løbende i et HashMap. Til sidst udskrives antallet af hver karakters forekomst; frekvensen.

2 Instruktion

Herunder er instruktioner til at køre den optimerede kode på egen maskine:

1. Vores optimerede løsning kan downloades her:
<https://github.com/CK2800/UFO/tree/master/3>
2. Åbn projektet i IntelliJ eller tilsvarende IDE.
3. Kør `Main.main()` fra IDE'et, bemærk at der køres 500 loops. Vores benchmark indikerer, at det bør tage ca. 75 sek. på en i5, 2.2 GHz.
4. Observer mean og standard deviation udregningerne i konsollen. Disse udregnes for hvert 50. loop.
5. Kør nu Main i kommandoprompten ved at navigere til `/target/classes` og skrive flg. kommando:

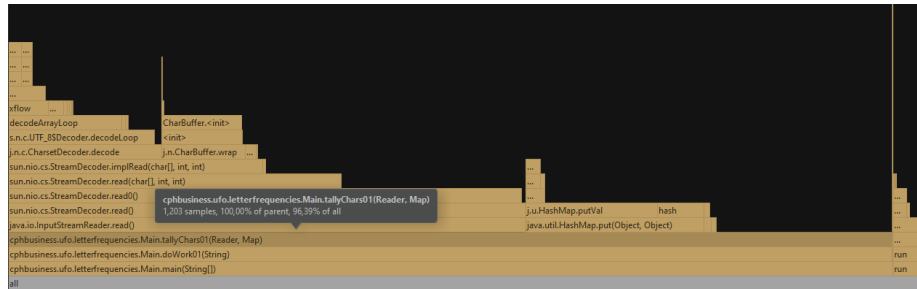
```
java cphbusiness.ufo.letterfrequencies.Main
```

6. Observer igen mean og standard deviation værdierne i konsollen.

3 Fremgangsmåde

For at identificere flaskehalse i programmet, har vi brugt Java flight recorder profilen, som findes i vores IDE; IntelliJ. Af billedet herunder fremgår det, at kørslen af programmet i sin helhed især bruger tid på

Main.tallyChars(Reader, Map)



Figur 1: Flammegraf for original tallyChars metode.

Billedet med flammegrafen viser også, at der er relativt mange skridt forbundet med den originale kode. Eksempelvis kaldes der ofte metoder på StreamDecoder- og CharSetDecoder-objekter, før karakteren indsættes i HashMap. Med identifikationen af den mulige flaskehals, granskede vi den originale kode for at forstå den, samt fremkomme med umiddelbare optimeringsskridt.

```
/**
 * Reads bytes from the file one at a time and increments occurrences in a hashmap.
 * @param reader
 * @param freq
 * @throws IOException
 */
private static void tallyChars01(Reader reader, Map<Integer, Long> freq) throws IOException {
    int b;
    while ((b = reader.read()) != -1) {
        try {
            freq.put(b, freq.get(b) + 1);
        } catch (NullPointerException np) {
            freq.put(b, 1L);
        }
    }
}
```

Figur 2: Original tallyChars metode.

Vores umiddelbare mistanke faldt på det faktum, at koden traverserer hele filen, 1 bogstav ad gangen. Dette resulterer i mange læsninger på disken, hvorfor vi drøftede følgende løsningsforslag:

1. Indlæsning af (dele af) filen til RAM.
2. Anvendelse af alternativ reader, eventuelt med indbygget buffer.
3. Læse filen i større bidder ved brug af byte arrays.

3.1 Fravalg af forslag 1

Vi indså hurtigt at indlæsning af filen inden kørslen af `Main.tallyChars` ville bryde med kontrakten, idet en ændring af indlæsningstidspunkt for filen vil medføre, at indlæsningsmetoden skal køres først. Dette vil medføre ændringer i kodebasen andre steder, idet en ny metode til indlæsning af filen i RAM ville skulle kaldes først.

3.2 Anvendelse af alternativ reader

Med vores mistanke om flaskehalsens årsag in mente, tog vi en `BufferedInputStream` i brug, for at teste, om læsning af buffer fra memory ville gøre nogen forskel. Som det kan ses af flg. graf, var det en forbedring i kørselstiderne. Brugen af `BufferedInputStream` er imidlertid også et brud på kontrakten, idet der fordres en ny signatur på metoden:

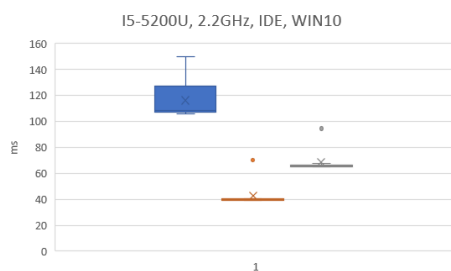
```
private static void tallyChars(BufferedInputStream stream,
                               Map<Integer, Long> freq)
```

Anvendelsen af `BufferedInputStream` skal således ses som en afprøvning af konceptet med en buffer at læse fra. Senere kunne vi formentlig have brugt `BufferedReader` for overholdelse af kontrakten;

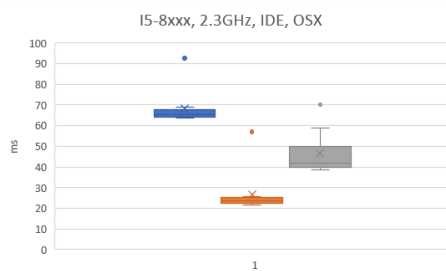
```
private static void tallyChars(Reader reader, Map<Integer,
                               Long> freq)
```

Som det fremgår af figurerne 3 - 6, er indlæsning i større bidder en plausibel løsning, idet den blå kasse med hale viser den originale `tallyChars`-metode. Den grå kasse med hale viser tider med indlæsning vha. `BufferedInputStream`:

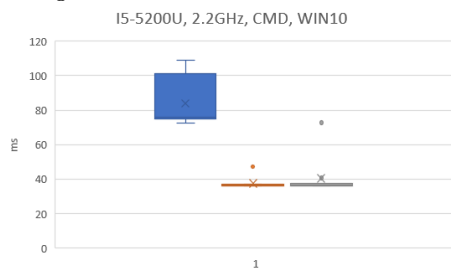
3.3 Indlæsningstider



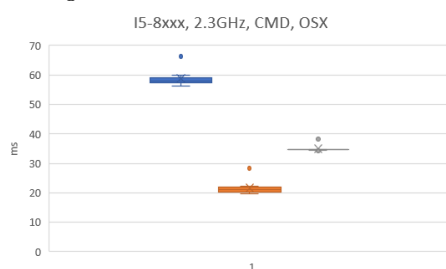
Figur 3: Kørselstider, IDE, Win10.



Figur 4: Kørselstider, IDE, OSX.



Figur 5: Kørselstider, CMD, Win10.



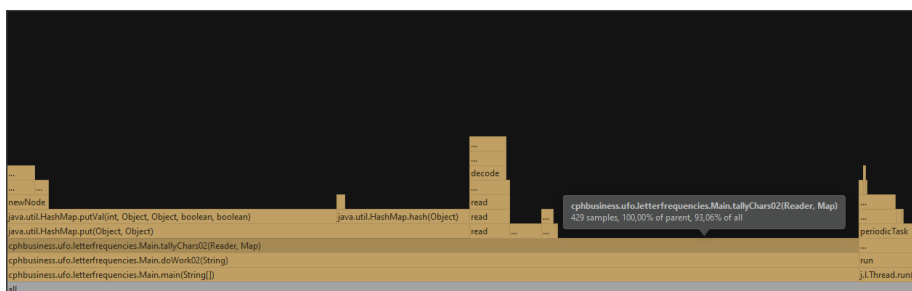
Figur 6: Kørselstider, CMD, OSX.

3.4 Indlæsning af filen i bidder vha. char array

Som det videre fremgår af figurerne 3-6 ovenfor, er indlæsning vha. et char-array en hurtig løsning. Den orange kasse med hale er, i alle 4 figurer, den hurtigste og har samtidig ganske stabile kørselstider, uanset om den afvikles fra ide eller kommandoprompt. Vi har forsøgt at justere på længden af char-array'et, den bedste performance opnås dog ved enten 128 eller 256 pladser. Vores tese er, at det lagrede array, ved mange elementer, fylder mere end der er plads til i RAM, derfor skrives dele af array'et måske til disken.

4 Videre optimering

Vores optimerede kodes hurtigere kørselstid kan også ses af nedenstående flammegraf, der viser en markant mindre dybde i nestede kald fra tallyChars-metoden.



Figur 7: Flammegraf med færre nestede kald fra tallyChars.

Koden er således klar til videre forædling. Vi opstiller her et par forslag som vil give yderligere forbedringer i kørselstiderne.

1. Fjernelse af try-catch sætningen og istedet indsætte vha. `getOrDefault`-metoden på Map således:

```
freq.put((int)b, freq.getOrDefault((int)b, 0L) + 1)
```

2. Tråde til at indlæse parallelt.
3. Alternativ datastruktur til HashMap (dog brud med kontrakt).

```

/**
 * Reads bytes from the file in chunks of 256 bytes and increments occurences in a hashmap.
 * @param reader
 * @param freq
 * @throws IOException
 */
private static void tallyChars02(Reader reader, Map<Integer, Long> freq) throws IOException {
    char[] chars = new char[256];
    while((reader.read(chars)) != -1){
        for(char b : chars)
            try {
                freq.put((int)b, freq.get((int)b) + 1);
            } catch (NullPointerException np) {
                freq.put((int)b, 1L);
            }
    }
}
}

```

Figur 8: Optimeret tallyChars metode.