Report Title

Optimering af program

Author Name

Alex Langhoff og Andreas Due Jørgensen

En rapport om undersøgelserne i forbindelse med optimering af et program

Gennemgang af det orginal program

Dette vil være en gennemgang af det orginale program, og nogle tal på performance.

Der var ingen måder i programmet at måle på sig selv, så der er blevet introduceret en ny klasse, til at kunne fortage målinger.

```
1  public class Timer {
2      private long start, spent = 0;
3      private double st= 0.0,sst=0.0;
4      int n = 0;
5      public Timer() { play(); }
6      public double check() {
7          double time = (System.nanoTime() - start+spent);
8          st += time;
9          sst += time * time ;
10         n++;
11         return (System.nanoTime() - start+spent)/1e6;
12     }
13     public void play() { start = System.nanoTime(); }
14     public double getMean(){
15         return (st/n)/1e6;
16     }
17     public double getSDev(){
18         return (Math.sqrt((sst -getRealMean()*getRealMean()-n)/(n
               -1)))/1e6;
19     }
20     private double getRealMean(){
21         return st/n;
22     }
23 }
```

Denne klasse kan returnere tiden i milliesekunder fra play() til check(). Den kan også returnere empirical mean med getMean() og standard deviation med getSDev().

performance

Med klassen fra sidste kapitel, er der fortaget målinger af det orginale program.

De første målinger er taget ved at køre programmet 50 gange og tage en måling fra hver gang.

```
1  113 ms
2  73 ms
3  80 ms
4  84 ms
5  68 ms
6  42 ms
7  42 ms
8  40 ms
9  44 ms
10 46 ms
11 50 ms
12 43 ms
13 48 ms
```

```
45 ms
44 ms
41 ms
53 ms
38 ms
39 ms
40 ms
39 ms
39 ms
41 ms
39 ms
38 ms
39 ms
43 ms
71 ms
65 ms
40 ms
38 ms
79 ms
59 ms
38 ms
38 ms
40 ms
38 ms
37 ms
38 ms
39 ms
40 ms
38 ms
38 ms
40 ms
43 ms
41 ms
38 ms
39 ms
38 ms
40 ms
```

Dette er sat i en boksplot for at give et intryk af programmets performancen.

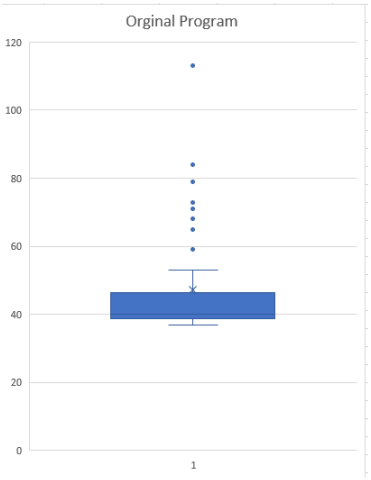

Figure 1: Boksplot af det orginale program

Dernæste er de 50 gange blevet kørt 50 gange for at skabe mean og standard deviation. så hver linie i følgnede er 50 kørsler af programmet.

```
1  mean is : 45,04 MS +/- 47,17 MS
2  mean is : 41,95 MS +/- 43,24 MS
3  mean is : 40,82 MS +/- 41,75 MS
4  mean is : 40,50 MS +/- 41,26 MS
5  mean is : 40,09 MS +/- 40,71 MS
6  mean is : 39,86 MS +/- 40,39 MS
7  mean is : 39,67 MS +/- 40,13 MS
8  mean is : 39,57 MS +/- 40,01 MS
9  mean is : 39,44 MS +/- 39,84 MS
10 mean is : 39,35 MS +/- 39,72 MS
11 mean is : 39,26 MS +/- 39,60 MS
12 mean is : 39,26 MS +/- 39,59 MS
13 mean is : 39,24 MS +/- 39,55 MS
14 mean is : 39,22 MS +/- 39,53 MS
15 mean is : 39,15 MS +/- 39,44 MS
16 mean is : 39,12 MS +/- 39,39 MS
17 mean is : 39,18 MS +/- 39,48 MS
18 mean is : 39,15 MS +/- 39,43 MS
19 mean is : 39,16 MS +/- 39,44 MS
20 mean is : 39,21 MS +/- 39,49 MS
21 mean is : 39,19 MS +/- 39,46 MS
22 mean is : 39,18 MS +/- 39,45 MS
23 mean is : 39,18 MS +/- 39,44 MS
24 mean is : 39,25 MS +/- 39,51 MS
25 mean is : 39,21 MS +/- 39,46 MS
26 mean is : 39,19 MS +/- 39,43 MS
27 mean is : 39,21 MS +/- 39,45 MS
28 mean is : 39,20 MS +/- 39,43 MS
29 mean is : 39,19 MS +/- 39,41 MS
30 mean is : 39,18 MS +/- 39,40 MS
31 mean is : 39,16 MS +/- 39,38 MS
```

```
mean is : 39,16 MS +/- 39,37 MS
mean is : 39,19 MS +/- 39,39 MS
mean is : 39,19 MS +/- 39,40 MS
mean is : 39,22 MS +/- 39,42 MS
mean is : 39,23 MS +/- 39,43 MS
mean is : 39,21 MS +/- 39,41 MS
mean is : 39,23 MS +/- 39,43 MS
mean is : 39,23 MS +/- 39,42 MS
mean is : 39,25 MS +/- 39,44 MS
mean is : 39,24 MS +/- 39,44 MS
mean is : 39,23 MS +/- 39,41 MS
mean is : 39,23 MS +/- 39,42 MS
mean is : 39,21 MS +/- 39,40 MS
mean is : 39,21 MS +/- 39,39 MS
mean is : 39,21 MS +/- 39,39 MS
mean is : 39,20 MS +/- 39,38 MS
mean is : 39,20 MS +/- 39,38 MS
mean is : 39,20 MS +/- 39,37 MS
mean is : 39,20 MS +/- 39,37 MS
```

Flaskehalsen

For at finde flaskehalsen i programmet, er der startet i main, og tids målingen er flyttet rundt til at finde det præcise sted hvor tiden bliver brugt.

```
Timer t = new Timer();
                String fileName = "src/main/resources/
                    FoundationSeries.txt";
                Reader reader = new FileReader(fileName);
                Map<Integer, Long> freq = new HashMap<>();
                t.play();
                tallyChars(reader, freq);
                timeSets[x] = t.check();
                print_tally(freq);
```

Da tidsmålingen målte på tallyChars(), viste målingen at det var her, 99

Opdeling af tallyChars()

en analyse af tallyChars() viser at der er 4 dele der kan undersøge:

- Reader
- While loop
- Try / catch
- Map

Første forsøg

Første forsøg var at kigge på try catch, da denne kan give et unødvendigt overhead. En ikke så pæn løsning ser således ud:

```
1           int b; // will be slower, only check for lowercase chars
2           boolean check[] = new boolean[60];
3           while ((b = reader.read()) != -1) { // Big O = N
4               if((b>96 && b<123)) {
5                int c=b-97; // a = 0
6                   if(check[c]) {
7                       freq.put(b, freq.get(b) + 1);
8                   }
9                   else {
10                      freq.put(b, 1L);
11                      check[c]= true;
12                  }
13              }
14          }
```

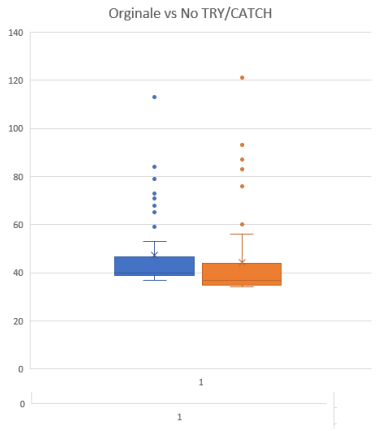

Figure 2: Boksplot af det orginale program mod en no Try/catch

Ved hurtigt øjekast ser no try/catch lidt hurtigere ud, men i løsninger bliver kun små bogstaver talt, og at lave et program der også tager de store, vil kun lægge tid til, og ikke blive hurtigere. Så denne løsningen der ikke arbejdet videre med.

Andet forsøg

Dette forsøg er der kigget på Map funktionen, for at lave en bedre Map funktion er følgende kode lagt prøvet:

```
1 int b;
2   while ((b = reader.read()) != -1) {
3     freq.merge(b,1L,Long::sum);
4         }
```

Dette gør at der ikke er brug for en Try/catch, eller en speciel måde at håndtere om et map er tomt eller ej.

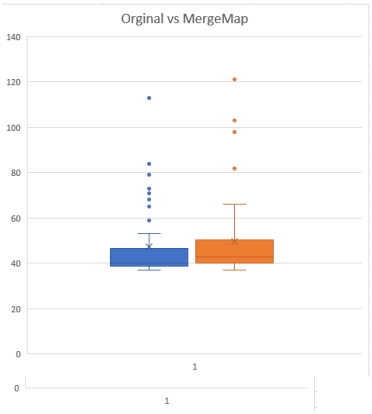

Figure 3: Boksplot af det orginale program mod en MergeMap

denne løsning, selve om den ser langt pænere ud, tog faktisk længere tid at udføre end det orginale program. Derfor er denne linie af forsøg stoppet.

Tredje forsøg

Dette ville have dækket over en while loop, men dette forsøg er udskudt, da det ikke virker fornuftigt at problemet skulle være her.

0.0.1 Fjerde forsøg

Sidste forsøg er på Reader, her er der forsøgt at udskift en BufferedReader, således at der ikke hvergang skal hentes et tegn fra filen, men derimod fra en buffer.

Koden er som følger:

```
1     private static void tallyChars(BufferedReader reader, Map<
          Integer, Long> freq) throws IOException {
2         int b;
3         while ((b = reader.read()) != -1) {
4             freq.merge(b,1L,Long::sum);
5         }
6     }
```

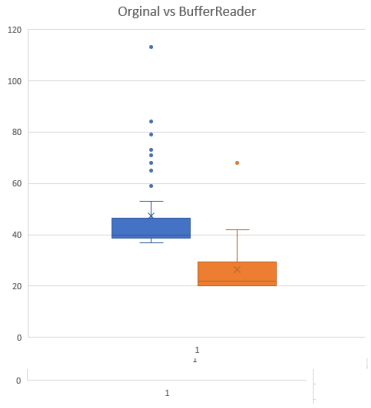

Figure 4: Boksplot af det orginale program mod en BufferReader

Denne løsning, bliver det hele markant hurtigere som kan ses af boksplotten. Forskelle skyldes at programmet ikke skal tilgå filen for hver karakter, men derimod load det ind i en buffer.

Kigger vi på mean og standard deviation kan vi ses en næsten 100 procent forbedring af koden.

```
1  mean is : 26,13 ms +/- 27,51 ms
2  mean is : 23,85 ms +/- 24,78 ms
3  mean is : 23,48 ms +/- 24,40 ms
4  mean is : 22,99 ms +/- 23,75 ms
5  mean is : 22,69 ms +/- 23,32 ms
6  mean is : 22,44 ms +/- 22,99 ms
7  mean is : 22,39 ms +/- 22,89 ms
8  mean is : 22,20 ms +/- 22,65 ms
9  mean is : 22,13 ms +/- 22,54 ms
10 mean is : 22,04 ms +/- 22,42 ms
11 mean is : 21,98 ms +/- 22,34 ms
12 mean is : 21,98 ms +/- 22,33 ms
13 mean is : 21,90 ms +/- 22,23 ms
14 mean is : 21,84 ms +/- 22,15 ms
15 mean is : 21,80 ms +/- 22,09 ms
16 mean is : 21,74 ms +/- 22,02 ms
17 mean is : 21,70 ms +/- 21,97 ms
18 mean is : 21,67 ms +/- 21,91 ms
19 mean is : 21,63 ms +/- 21,87 ms
20 mean is : 21,62 ms +/- 21,86 ms
21 mean is : 21,62 ms +/- 21,86 ms
22 mean is : 21,61 ms +/- 21,84 ms
23 mean is : 21,62 ms +/- 21,85 ms
24 mean is : 21,63 ms +/- 21,86 ms
25 mean is : 21,63 ms +/- 21,85 ms
26 mean is : 21,78 ms +/- 22,26 ms
27 mean is : 21,75 ms +/- 22,21 ms
28 mean is : 21,72 ms +/- 22,17 ms
29 mean is : 21,70 ms +/- 22,14 ms
30 mean is : 21,68 ms +/- 22,10 ms
31 mean is : 21,65 ms +/- 22,06 ms
32 mean is : 21,63 ms +/- 22,02 ms
33 mean is : 21,61 ms +/- 22,00 ms
34 mean is : 21,59 ms +/- 21,96 ms
35 mean is : 21,57 ms +/- 21,94 ms
36 mean is : 21,55 ms +/- 21,91 ms
37 mean is : 21,54 ms +/- 21,89 ms
38 mean is : 21,53 ms +/- 21,86 ms
39 mean is : 21,54 ms +/- 21,88 ms
40 mean is : 21,52 ms +/- 21,86 ms
41 mean is : 21,53 ms +/- 21,86 ms
42 mean is : 21,53 ms +/- 21,86 ms
43 mean is : 21,51 ms +/- 21,84 ms
44 mean is : 21,51 ms +/- 21,83 ms
45 mean is : 21,50 ms +/- 21,81 ms
46 mean is : 21,49 ms +/- 21,79 ms
47 mean is : 21,47 ms +/- 21,77 ms
48 mean is : 21,46 ms +/- 21,76 ms
49 mean is : 21,45 ms +/- 21,74 ms
50 mean is : 21,44 ms +/- 21,73 ms
```

Koden og latex filer er delt på følgende link
https://github.com/Janudanie/UFO/tree/main/Opgave%202