Meetrapport Snelheid en Memorygebruik

*26 maart 2020*

Door Tobias van den Hoogen en Jippe Heijnen

Inhoud

[Doel 2](#_Toc36222548)

[Hypothese 2](#_Toc36222549)

[Werkwijze 2](#_Toc36222550)

[Resultaten 3](#_Toc36222551)

[Verwerking 4](#_Toc36222554)

[Conclusie 5](#_Toc36222555)

[Bronvermelding 5](#_Toc36222556)

# Doel

Ons doel is om te onderzoeken of de snelheid en memorygebruik van onze implementatie significant beter is ten opzichte van de default implementatie.

# Hypothese

We denken dat onze implementatie sneller is dan de standaardimplementatie omdat we in onze implementatie veel variabelen op de heap zetten, efficiënt omgaan met het geheugen doormiddel van deconstructors en omdat we zo min mogelijk variabelen hebben aangemaakt. Ook hebben we voor opslag gebruik gemaakt van een map en denken we dat deze datastructuur het meest geschikt is voor het opslaan van pixels.

# Werkwijze

Voor dit experiment hebben wij ongeveer 28 afbeeldingen gebruikt(zie working directory in meetrapporten). Per afbeelding hebben we de default code én onze code uitgevoerd en hebben we via de interne diagnostiek van Visual Studio de resultaten bekeken. Hierbij letten we op de runtime en het geheugengebruik. Deze zaken hebben we vervolgens in een Excel sheet gezet om te kunnen vergelijken.

# Resultaten

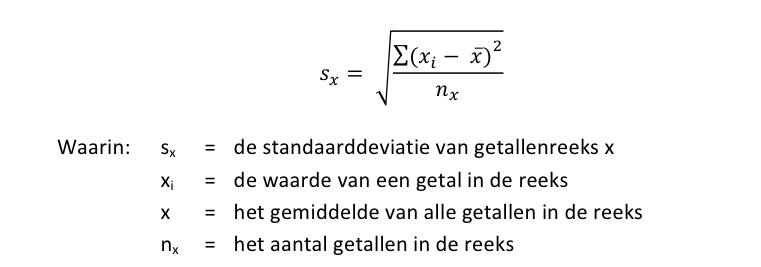
Nadat we de resultaten hadden genoteerd in het Excel sheet besloten we ze om te zetten in een staafdiagram(zie hieronder). Je kan hierbij zien dat er twee staafdiagrammen zijn. Op de eerste staafdiagram wordt er de snelheid van de runtime per afbeelding getoond. Van de tweede staafdiagram wordt er de memorygebruik per afbeelding getoond.

# 

# 

# Verwerking

Van de onderzoeksresultaten hebben wij eerst per implementatie een gemiddelde genomen. Daarvan hebben wij de correcte resultaten bij elkaar opgeteld en gedeeld door het aantal afbeeldingen.



Tabel : De formule voor het berekenen van de standaarddeviatie (van der Zee, 2017)

Voor onze resultaten hebben wij ook gebruik gemaakt van de standaarddeviatie. Deze hebben wij berekend door eerst het gemiddelde te berekenen van onze waardes. En aan de hand daarvan kunnen we met de bovenstaande formule en het aantal foto’s in onze populatie, de standaard afwijking berekenen. Dit hebben we doormiddel van Excel in tabellen berekend.

# Conclusie

Tabel 1: Gemiddelde en Standaarddeviatie van de memorygebruik

|  |  |  |
| --- | --- | --- |
| **Formule** | **Default** | **Onze** |
| Gemiddelde | 6,0357143mb | 21,82142857mb |
| Standaarddeviatie | 0,1889822mb | 2,93198923mb |

Uit bovenstaande tabel kunnen we van het gemiddelde opmaken dat de default implementatie een minder intensief gebruik van geheugen heeft dan onze eigen implementatie. Daarnaast kunnen we ook zien dat onze code een hogere standaardafwijking heeft. Dit geeft o.a. aan dat onze implementatie minder stabiel is dan de default qua geheugenverbruik.

Tabel 2: Gemiddelde en Standdaardeviatie van de snelheid

|  |  |  |
| --- | --- | --- |
| **Formule** | **Default** | **Onze** |
| Gemiddelde | 2,928571s | 4,321429s |
| Standaarddeviatie | 0,377964s | 1,156418s |

Aan deze bovenstaande tabel kunnen we van het gemiddelde opmaken dat onze implementatie trager is dan de default implementatie. Ook is onze code instabieler op het gebied van snelheid omdat de standaardafwijking groter is dan bij de default implementatie.

Hiermee kunnen we concluderen dat onze implementatie dus niet significant beter is op het gebied van snelheid en memorygebruik vergeleken met de default implementatie.

# Bronvermelding

van der Zee, F. (2017). *Standaarddeviatie of spreiding*. Geraadpleegd op 13 maart 2020, van <https://hulpbijonderzoek.nl/online-woordenboek/standaarddeviatie/>