|  |
| --- |
| Functioneel Paradigma\_  Algoritmes, Paradigma’s en Programmeertalen |

Mike Doornenbal - 1628421

[kies een datum]

INHOUDSOPGAVE

[1 Onderzoek 3](#_Toc209173246)

[1.1 Zuiverheid 3](#_Toc209173247)

[1.2 Lazy Evaluation 3](#_Toc209173248)

[1.3 3](#_Toc209173249)

[2 Challenge 4](#_Toc209173250)

[3 Implementatie 5](#_Toc209173251)

[4 Reflectie 6](#_Toc209173252)

[5 Conclusie 7](#_Toc209173253)

[6 Bronvermelding 8](#_Toc209173254)

[9](#_Toc209173255)

# Onderzoek

Om te beginnen ben ik online op zoek gegaan naar een kant en klare uitwerking van RLE-compressie in Haskell. Deze had ik snel gevonden, wat betekent dat ik daarvan kan leren hoe RLE-compressie in Haskell precies werkt wanneer ik er zelf niet uitkom. Ik hoop hier geen gebruik van te hoeven maken, maar nood breekt wet en als ik begrijp wat de uitwerking doet en hoe deze tot stand is gekomen, kan ik altijd proberen deze zelf op een andere manier te schrijven.

Een zekerheid is wel dat ik op dit moment geen compressiealgoritme in Haskell ga kunnen schrijven, omdat ik simpelweg de syntax van Haskell niet goed weet. Daarom heb ik besloten om een eerste uitwerking te maken in Java. Die probeer ik dan zo functioneel mogelijk te maken, zodat ik deze direct kan vertalen naar Haskell.

Allereerst ben ik begonnen met het uitschrijven van de stappen die ik moet zetten om tot een werkend algoritme te komen. Dit heb ik gedaan in codecommentaar, zodat ik meteen kan zien wat de volgende stap in het proces moet zijn mocht ik vastlopen. Daarna ben ik begonnen met het schrijven van een implementatie in Java op een non-functionele wijze. Het resultaat daarvan is [hier](https://github.com/MikTheMilkMan/APP-Functioneel-Paradigma/blob/f1dd7cba50ce26fadf72d17c0f7e6a094210a221/Programma/java_tests/src/main/java/nl/han/APP/Challenge/Challenge.java) te zien. Deze implementatie werkte niet 100% goed, maar het was goed genoeg voor een begin.

Vervolgens heb ik geprobeerd deze implementatie te herschrijven naar eentje die functioneel is. Het resultaat daarvan is [hier](https://github.com/MikTheMilkMan/APP-Functioneel-Paradigma/blob/32d8a9fdf4a85e4003695e6a7a7ada894d84c1de/Programma/java_tests/src/main/java/nl/han/APP/Challenge/Challenge.java) te zien. Ook hier ben ik niet geheel tevreden mee, maar wederom is dit een goede tweede stap. Deze implementatie lijkt wel 100% goed te werken (gebaseerd op mijn gelimiteerde testcases)

# Challenge

Voor deze opdracht heb ik gekozen om RLE-compressie te implementeren. Van de vooraf gegeven opties, leken degene die met compressie te maken hebben de meest interessante. Uiteindelijk heb ik RLE-compressie gekozen over LZW-compressie, omdat ik (na het bestuderen van de Wikipedia-pagina’s voor beide compressie-algoritmen) bij RLE-compressie al snel een idee kreeg bij hoe het algoritme comprimeert.

Ondanks het initiële begrip, denk ik dat deze opdracht mij alsnog een flinke uitdaging zal bezorgen. Het begrijpen van een een puzzel betekent immers niet dat je direct de oplossing hebt, of überhaupt de weg naar de oplossing ziet. Compressie boeit mij al langer omdat het een beetje voelt als magie: je slaat exact dezelfde data op maar het neemt minder ruimte in beslag en door creatief om te gaan met de resulterende data kan je de originele data weer terugkrijgen. Natuurlijk bestaat er ook lossy compressie, waarbij data verloren gaat omwille van ruimtebesparing. Dit voelt minder speciaal en uitdagend, omdat je feitelijk strategisch data verwijdert zonder de verwachting deze later terug te krijgen. Van de andere kant is deze vorm van compressie weer bijzonder door andere redenen, zoals het feit dat het dataverlies soms niet makkelijk te merken is.

Ik heb nog nooit eerder geprobeerd data te comprimeren (en zeker niet in een functionele taal) en daarom lijkt dit me tegelijkertijd een leuke maar ook zeker een uitdagende challenge.

# Implementatie

# Reflectie

# Conclusie

Samenvatting van de belangrijkste (leer)punten

# Bronvermelding

Bronvermelding

<https://www.seas.upenn.edu/~cis1940/spring13/lectures.html>

