**qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm**

|  |
| --- |
| Functioneel Programmeren  Minorvak  30-1-2017  Thom Wouda & Eden Cox |

Inhoud

[1. Inleiding 2](#_Toc474509293)

[2. Het algoritme 3](#_Toc474509294)

[3. De Talen 4](#_Toc474509295)

[3.1 C++ 4](#_Toc474509296)

[3.2 Haskell 4](#_Toc474509297)

[3.3 Scala 4](#_Toc474509298)

[4. De verschillen 5](#_Toc474509299)

[4.1 Snelheid 5](#_Toc474509300)

[4.2 Structuur 6](#_Toc474509301)

[4.3 Leesbaarheid 6](#_Toc474509302)

[4.4 Lengte 7](#_Toc474509303)

[5. Conclusie 7](#_Toc474509304)

[Bijlage A 8](#_Toc474509305)

[Graaf 1 8](#_Toc474509306)

[Graaf 2 9](#_Toc474509307)

[Bibliografie 10](#_Toc474509308)

# Inleiding

Voor het vak functioneel programmeren wordt een algoritme naar keuze uitgewerkt in drie verschillende talen. De eis is dat er een taal imperatief, functioneel en semi-functioneel is. De functionele taal is hierbij vastgesteld op Haskell. Tevens mag het algoritme niet te eenvoudig zijn. In dit verslag worden de drie verschillende codes naast elkaar gezet en worden de verschillen tussen functioneel en imperatief beoordeeld.

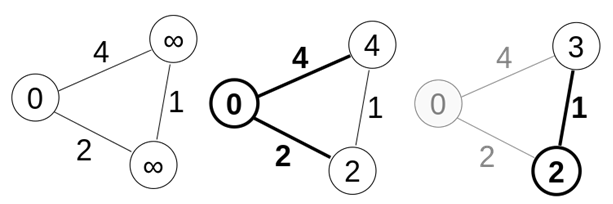
Deze opdracht is uitgevoerd door Eden Cox en Thom Wouda in de periode september 2016 – februari 2017.

# Het algoritme

Voor de realisatie van de opdracht dient er een algoritme gekozen worden. Dit moet een degelijk algoritme zijn, dus niet te eenvoudig. In deze uitwerking is er gekozen voor het Dijkstra algoritme, ook bekend als het kortste pad-algoritme.

Het Dijkstra algoritme is een algoritme dat gebruikt wordt om het kortste pad in een graaf te vinden. Hierbij heeft graaf heeft verschillende punten met een begin- en eindpunt. De punten hebben een bepaalde afstand. Het dijkstra algoritme loopt door deze graaf heen en telt de afstanden op, en bepaald aan de hand van de hoogste en laagste afstand welk pad het korst is.

*Een klein voorbeeld van het Dijkstra algoritme is te zien in Figuur 1.*



Figuur 1 - Dijkstra algoritme

# De Talen

In dit verslag wordt het algoritme in drie verschillende talen met elkaar vergeleken. De eis is dat er een imperatieve, functionele en semi-functionele taal wordt gebruikt. Waarbij semi-functioneel een mix is tussen imperatief en functioneel. De functionele taal staat hierbij vastgesteld op **Haskell**.

De imperatieve en semi-functionele talen zijn optioneel, zolang ze maar aan de eis voldoen. In dit verslag is er gekozen voor **C++**(imperatief) en **Scala**(semi-functioneel).

## C++

C++ is een imperatieve, objectgeoriënteerde programmeertaal dat gebaseerd is op C. Het is ontworpen door Bjarne Stroustrup als upgrade van C. De belangrijkste toevoegingen van C++ ten opzichte van C is de beschikbaarheid van onder ander: virtuele functies, abstracte klassen en overerving. Verschillende talen zijn gebaseerd op C++, zoals Java en C#.

## Haskell

Haskell is een functionele programmeertaal vernoemd naar de wiskundige Haskell Brooks Curry. De taal is in 1990 verschenen. Haskell is Multi-platform en wordt voornamelijk gecompileerd in de GHC-compiler. Haskell is puur functioneel, dit wil zeggen dat alle operaties beschrijvingen zijn van wat er gedaan moet worden in de gegenereerde code. Er zijn geen statements of instructies, alleen maar expressies.

## Scala

Scala is een Multi-paradigma taal. Dat wil zeggen dat de taal beide imperatief en functioneel is. Dit maakt de taal dus semi-functioneel. De taal is verschenen in 2004 en ontworpen door Martin Odersky. Scala is puur object georiënteerd. Conceptueel is elke waarde een object en is iedere operatie een methode aanroep. Desondanks de syntaxis erg conventioneel is, is de taal ook volledig functioneel. Het heeft alles wat je kan verwachten van een functionele taal, zoals een library met immutable datastructuren of de voorkeur naar immutable over mutabele states.

Met Scala is het mogelijk om te programmeren in verschillende stijlen. Tevens is het mogelijk om stijlen naar voorkeur te combineren.



Figuur 2 - C++, Haskell, Scala

# De verschillen

In dit verslag worden de drie talen naast elkaar gelegd en worden verschillende eigenschappen en verschillen onderzocht. Deze zijn als volgt:

* De snelheid
* De structuur
* Leesbaarheid
* Lengte

## Snelheid

Om de snelheid te bepalen wordt er bij het uitvoeren van het programma een timer gestart. Na het berekenen en weergeven van het kortste pad wordt de timer stopgezet. Deze timing wordt gemeten met twee verschillende grafen. Hierbij is graaf 1 groter dan graaf 2. Alle benchmarks zijn uitgevoerd met een Intel Core i5 750 @ 4GHz processor met het besturingssysteem Microsoft Windows 8.1 Pro. Alle tijden zijn in milliseconden (ms) en zijn afgerond op twee decimalen.

*De betrokkene grafen zijn te vinden in Bijlage A.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taal | Paradigma | Algoritme | Graaf | Tijd in ms |
| C++ | Imperatief | Dijkstra | Graaf 1 | 0.54 |
| Haskell | Functioneel | Dijkstra | Graaf 1 | 0.30 |
| Scala | Semi-functioneel | Dijkstra | Graaf 1 | 8.89 |

Tabel 1 - Benchmark graaf 1

In Tabel 1 zien we dat tegen de verwachtingen in Haskell de snelste blijkt te zijn bij het uitrekenen van het Dijkstra algoritme, gevolgd door C++. De semi-functionele taal Scala doet her het langste over met bijna 9 milliseconden. Dit is op zich niet opmerkelijk, aangezien de taal op Java Virtual Machine(JVM) draait.

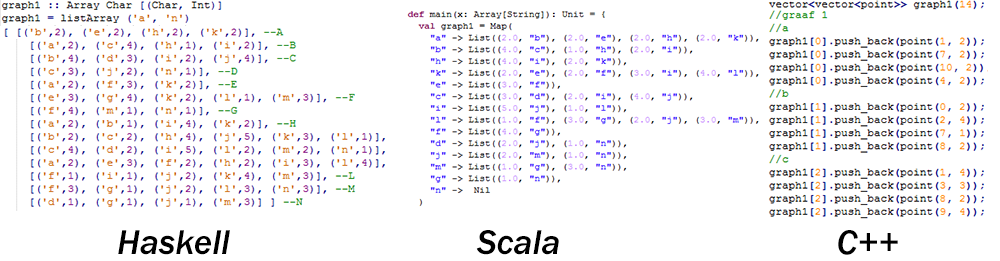
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taal | Paradigma | Algoritme | Graaf | Tijd in ms |
| C++ | Imperatief | Dijkstra | Graaf 2 | 0.24 |
| Haskell | Functioneel | Dijkstra | Graaf 2 | 0.20 |
| Scala | Semi-functioneel | Dijkstra | Graaf 2 | 0.60 |

Tabel 2 - Benchmark graaf 2

Tabel 2 laat de tijden van graaf 2 zien. De tijden liggen in deze tabel ook een stuk lager aangezien de graaf een stuk minder complex is, en dus wordt er minder rekenkracht vereist. Haskell heeft hierbij weer de laagste tijd, en wordt ook weer gevolgd door C++. De tijd van scala is in verhouding minder hoog bij deze graaf.

## Structuur

Bij het Dijkstra algoritme wordt er begonnen met het vullen van een graaf. De punten worden aangemaakt en toegevoegd. Vervolgens worden er connecties gemaakt tussen deze punten. Daarbij wordt een bepaalde afstand bij toegevoegd. Dit concept is bij alle drie codes bijna identiek. In C++ wordt er een tweedimensionale vector gevuld met als type een point struct. Deze point beschikt over een eindbestemmingen punt en een afstand. De positie van deze punt in de vector bepaalt het vertrekpunt. In Haskell wordt de graaf gerepresenteerd door een arraylist van tuples. Deze tuples bevatten een node en een afstand. In Scala is de graaf een map met als inhoud een string key die de bron representeert, en een list met tuples met de eindbestemming letter en afstand.



Figuur 3 - De grafen in 3 vershillende talen

## Leesbaarheid

De leesbaarheid van iedere programmeertaal is soortgelijk aan talen, het hangt sterk af van de persoonlijke kennis, preferenties en ervaring van de persoon. De leesbaarheid is beoordeeld vanuit het perspectief van een programmeur die voornamelijk object georiënteerd geprogrammeerd heeft.

Een belangrijke factoor van leesbaarheid is statische typering. Haskell maakt hier gebruik van. Dit houdt in dat alle types bij het compilen worden achterhaald. Het voordeel hiervan is dat de types niet hoeven worden gedeclareerd bij het schrijven van Haskell code. Het nadeel is dat bij het teruglezen en het interpreteren van Haskell code het lastig kan zijn om te achterhalen welke types er gebruikt worden. Bij Scala is dit optioneel, de programmeur mag zelf de kiezen of hij statische typering wil gebruiken.

Een ander belangrijk verschil dat men opmerkt tussen de codes is bij het gebruik van lussen. Imperatieve code zal gebruik maken van een iterator. Een iterator is de teller die wordt gebruikt om door een datacontainer te lopen. Bij functionele code wordt dit gedaan met lijstcomprehensies en recursieve functies. lijstcomprehensies zijn simpele acties die op een datacontainer uitgevoerd kan worden. Bij recursie roept de functie zichzelf opnieuw aan. Recursieve functies hebben een stopconditie om uiteindelijk te stoppen.

## Lengte

Bij het vergelijken van de lengte van de code is er alleen naar de algoritmes en functies gekeken. Het aantal woorden en tekens zijn opgeteld en worden weergegeven in Tabel 3.

|  |  |
| --- | --- |
| Taal | Woorden en tekens |
| C++ | 114 |
| Haskell | 204 |
| Scala | 97 |

Tabel 3 - lengte code

Tot verbazing heeft Haskell het dubbele aantal woorden en tekens ten opzichte van C++ en Scala. Tevens is het opmerkelijk dat Scala in staat is om het algoritme zó compact te representeren in vergelijking met de andere talen.

# Conclusie

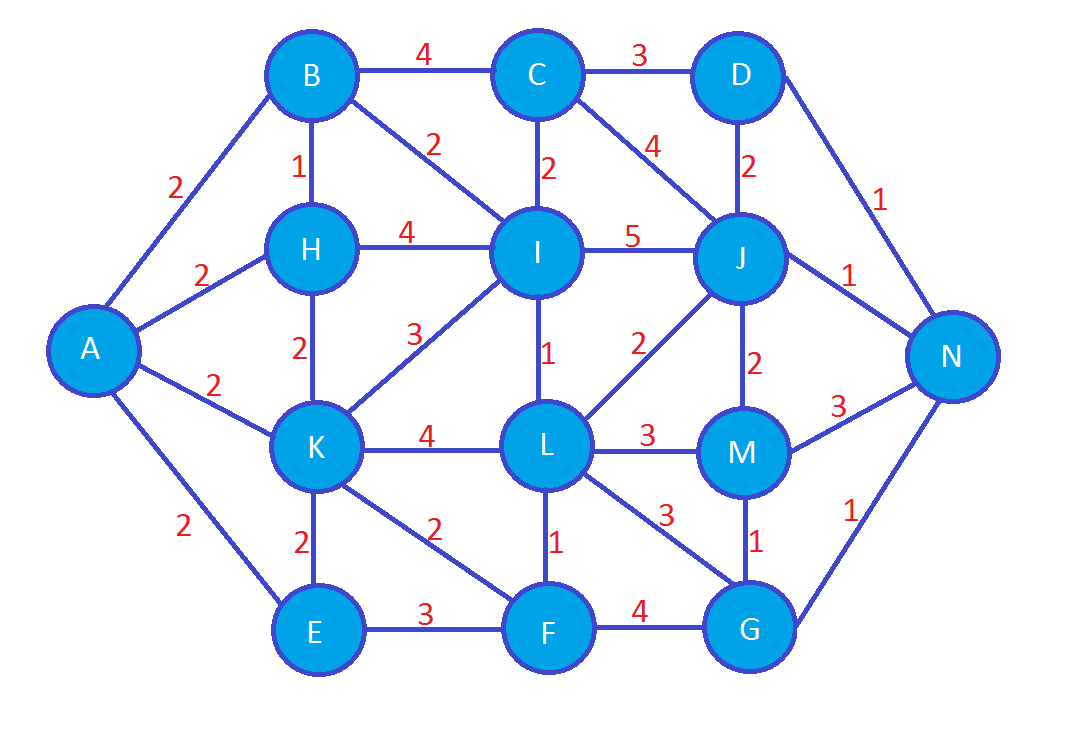
Bij het vergelijken van imperatieve, functionele en semi-functionele code blijkt het dat er aanzienlijk grote verschillen zijn. In prestaties is dit met de functionele en imperatieve taal niet heel groot. Toch zien we dat de semi-functionele taal waarneembaar trager is. Dit zal te maken hebben met het gebruik van de Java Virtual Machine. Het grootste verschil ligt eigenlijk bij de definitie van de code. Ondanks het feit dat de functionaliteit van de code identiek is verschild de syntaxis compleet. Imperatieve talen staan dichter bij natuurlijke talen dan functionele talen, terwijl functionele talen dichterbij de wiskundige kant van het taal spectrum zitten.

Uiteindelijk vonden wij Dijkstra het makkelijkst uit te programmeren in C++ vanwege de complexiteit van het probleem en en het gemak om dit te verwoorden in C++. Haskell presteert beter en maakt het mogelijk om problemen met een wiskundige oplossing te benaderen.

# Bijlage A

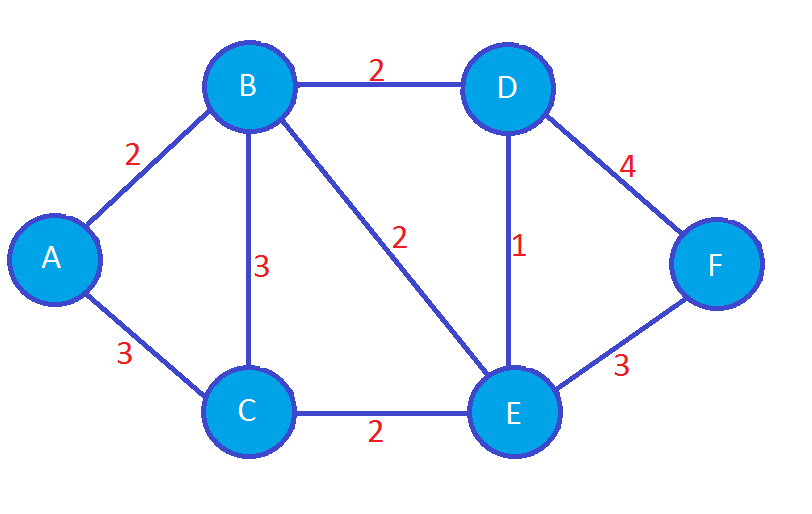
## Graaf 1

Graaf met 14 knopen



## Graaf 2

Graaf met 6 knopen



# Bibliografie

Albatross. (2016). *A Brief Description*. Retrieved 01 30, 2017, from cplusplus.com: http://www.cplusplus.com/info/description/

*C++*. (2016, 12 8). Retrieved 01 30, 2017, from Wikipedia: https://nl.wikipedia.org/wiki/C%2B%2B

*Haskell (programmeertaal)*. (2015, 03 7). Retrieved 2 7, 2017, from Wikipedia: https://nl.wikipedia.org/wiki/Haskell\_(programmeertaal)

*Haskell Language*. (2017). Retrieved 02 03, 2017, from Haskell: https://www.haskell.org/

*Scala (programmeertaal)*. (2016, 12 7). Retrieved 02 8, 2017, from Wikipedia: https://nl.wikipedia.org/wiki/Scala\_(programmeertaal)

*What is Scala?* (2017). Retrieved 02 09, 2017, from Scala: https://www.scala-lang.org/what-is-scala.html