**qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm**

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

Inhoud

[1. Inleiding 2](#_Toc474354365)

[2. Het algoritme 3](#_Toc474354366)

[3. De Talen 4](#_Toc474354367)

[3.1 C++ 4](#_Toc474354368)

[3.2 Haskell 4](#_Toc474354369)

[3.3 Scala 4](#_Toc474354370)

[4. De verschillen 5](#_Toc474354371)

[4.1 De snelheid 5](#_Toc474354372)

[4.2 De code 5](#_Toc474354373)

[Bijlage A 6](#_Toc474354374)

[Graaf 1 6](#_Toc474354375)

[Graaf 2 7](#_Toc474354376)

# Inleiding

Voor het vak functioneel programmeren wordt een algoritme naar keuze uitgewerkt in drie verschillende talen. De eis 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.

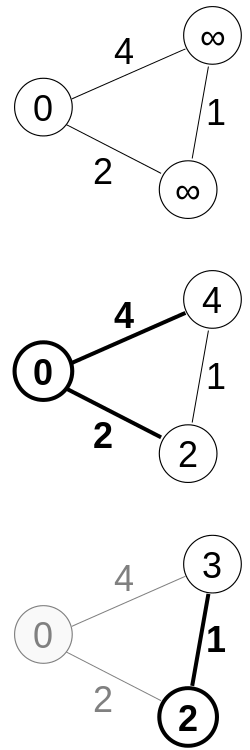
Deze opdracht is uitgevoerd door Eden Cox en Thom Wouda in de periode september 2016 – februari 2017.

# 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 korstepad-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**(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.

## 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.

# 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

## De snelheid

Om de snelheid te bepalen wordt er bij het uitvoeren van het programma een timer gestart. Na het berekenen en weergeven van het korste 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).

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taal | Paradigma | Algoritme | Graaf | Tijd in ms |
| C++ | Imperatief | Dijkstra | Graaf 1 | 1.91 |
| 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, gevolgt door C++. De semi-functionele taal Scala doet her het langste over met bijna 9 milliseconden. Dit is opzich niet opmerkelijk, aangezien de taal op Java Virtual Machine draait.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taal | Paradigma | Algoritme | Graaf | Tijd in ms |
| C++ | Imperatief | Dijkstra | Graaf 2 | 1.34 |
| Haskell | Functioneel | Dijkstra | Graaf 2 | 0.20 |
| Scala | Semi-functioneel | Dijkstra | Graaf 2 | 0.60 |

Tabel 2 - Benchmark graaf 2

Tabel 2 laat de uitvoering van graaf 2 zien. De timings 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 toch weer de laagste tijd, maar dit keer gevolgd door Scala. C++ is niet veel sneller dan bij de vorige graaf, vermoedelijk verliest C++ aanzienlijk veel tijd bij het aanmaken van de graaf, dit gebeurd namelijk in een 2D vector.

## De 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 is bij alle 3 codes bijna identiek. In C++ wordt er een tweedimensionale vector gevuld met als type een point. Deze point beschikt over een eindbestemmings punt en een afstand. De positie van deze punt in de vector bepaald het vertrekpunt.

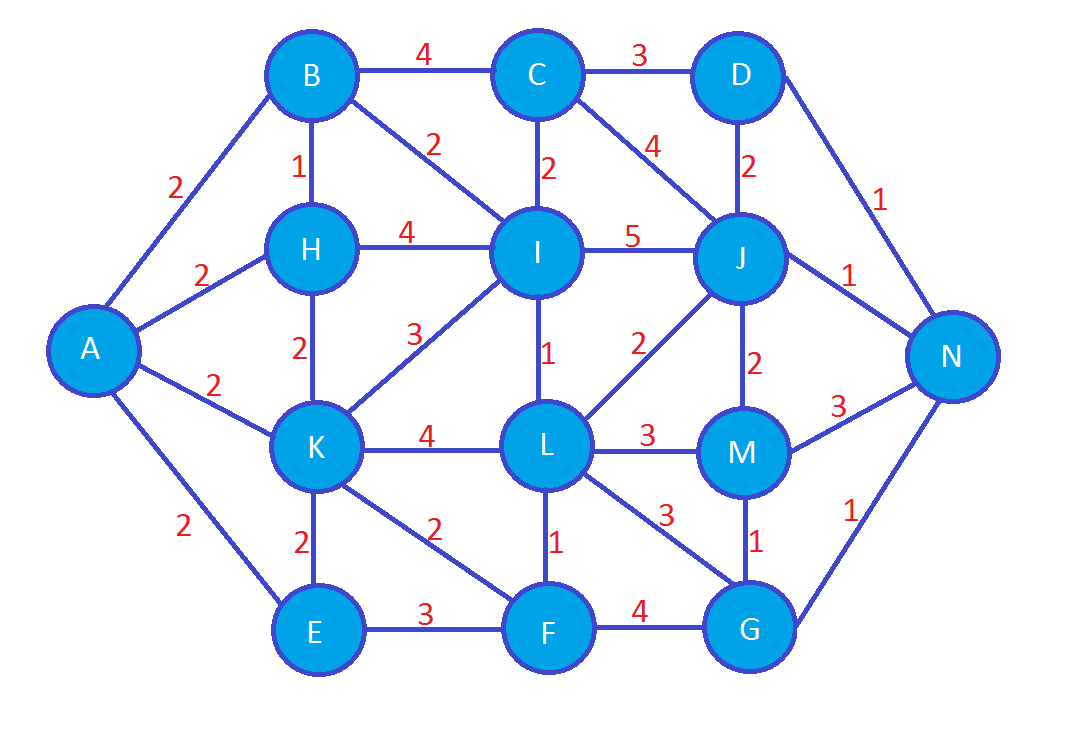
## Leesbaarheid

## Lengte

# Bijlage A

## Graaf 1

Graaf met 14 knopen



## Graaf 2

Graaf met 6 knopen

