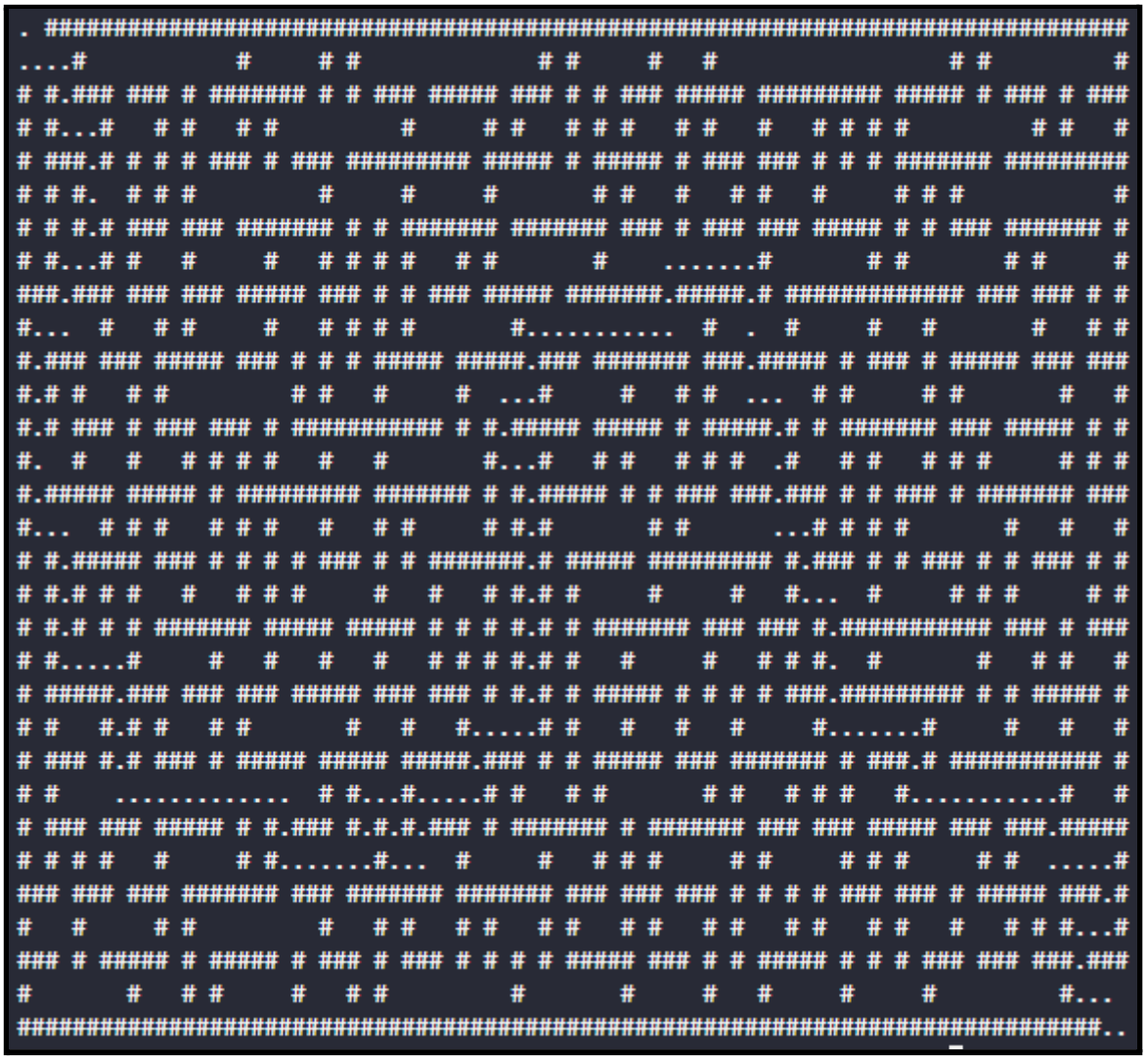
Functioneel programmeren 2

Dijkstra algoritme in C#, Go en Haskell



Leerling	Merlijn Stokhorst
E-mail	merlijn.stokhorst@student.nhlstenden.com
Begeleider	Jos Foppele
Datum	03-05-2021

Inleiding

Voor het vak Functioneel programmeren twee was het de bedoeling dat er een algoritme in drie verschillende talen werd uitgewerkt waarvan een Haskell. De andere twee talen en het algoritme mochten door de student worden uitgezocht. Vervolgens moesten er drie verschillende uitwerkingen met elkaar worden vergeleken. In dit verslag lees je de uitwerking van de opdracht.

Talen en motivatie

De twee talen die ik heb gekozen voor deze opdracht zijn C# en Go.

Ik heb C# gekozen omdat ik hier al vrij bekend mee ben. Daardoor kon ik het algoritme eerst makkelijk in deze taal uitwerken en dit als basis gebruiken voor de uitwerking in de andere talen.

Dit vak leek mij ook een goede gelegenheid om een nieuwe taal te leren. Met Go had ik nog niet eerder gewerkt. Wel had ik gehoord dat het een erg snelle en makkelijk te lezen taal is. En dat het erg snel groeit in populariteit. Daarom wou ik deze ook graag verwerken in de opdracht.

Algoritme en motivatie

Tijdens het zoeken naar een leuk algoritme om te programmeren kwam ik op een site terecht die verschillende pathfinding algoritmes visualiseert (Pathfinding Visualizer, n.d.). Dat bracht mij op het idee om Dijkstra's algoritme in een grid based map uit te voeren omdat dit ook in elke taal makkelijk naar de console te printen is. En je kan direct zien of je resultaat goed is.

Om een input te genereren waar doorheen kan worden genavigeerd heb ik een site gevonden die een doolhof in ascii kan genereren (Pathfinding Visualizer, n.d.). Dit sla ik op in een text file die in in alle drie de talen uit lees.

Uitwerkingen

In dit hoofdstuk bespreek ik alle drie die uitwerkingen. In alle drie de uitwerkingen ga ik hetzelfde te werk volgens dit stappenplan.;

- Genereer een tweedimensionale array met nodes vanuit de textfile.
- Geef elke node een waarde, hoeveel nodes hij van het startpunt ligt. Totdat het eindpunt is gevonden of als hij niet meer verder kan.
- Kijk wat de waarde van het eindpunt is geworden, navigeer vanaf daar weer terug naar het startpunt.
- Print het resultaat.

C#

Tijdens het maken van deze uitwerking ben ik niet tegen problemen aangelopen. Het programmeren van deze oplossing ging vrij vlot omdat ik de taal erg makkelijk te lezen vind.

Go

Dit was de eerste keer dat ik in Go programmeerde ik was zelf dus nog niet bekend met de syntax, naar een paar tutorials kreeg ik het al snel in de hand. Ik was van plan dit deze uitwerking over te nemen van de C# uitwerking. Maar ik liep steeds tegen fouten aan. Waar variabelen geen waardes kregen.

Na wat onderzoek bleek dat je in Go pointers en references moet gebruiken omdat je anders steeds een kopie van een object meegeeft. Hierdoor moest ik het programma toch even herschrijven.

Ook vond ik het jammer dat er in Go dingen missen die C# standaard wel heeft. Je kan bijvoorbeeld niet makkelijk een query op een lijst uitvoeren. Het heeft ook geen ternary operators die ik vaak gebruik.

Deze uitwerking was voor mij iets lastiger. Maar ik ben blij dat ik hiervoor heb gekozen omdat ik er wel veel van heb geleerd.

Haskell

De haskell uitwerking had ik onderschat. Bij zowel de Go en C# opdrachten veranderde ik steeds waardes in objecten. Maar in haskell is dit eigenlijk niet geoorloofd. Dit was heel irritant voor het genereren van de maze. omdat ik steeds mijn maze opnieuw moest genereren als ik een paar nodes wou aanpassen.

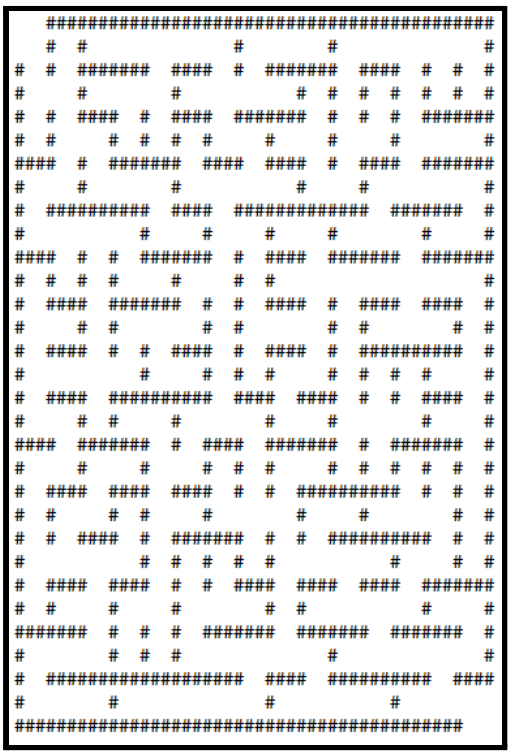
Wat wel weer een voordeel was, dat ik bijna alle methodes in een regel kon programmeren waardoor het programma niet zo groot werd. Het aantal regels was dan ook een stuk minder dan bij de andere twee oplossingen. Ik vond dit voordeel alleen niet opwegen tegen de leesbaarheid van de andere talen.

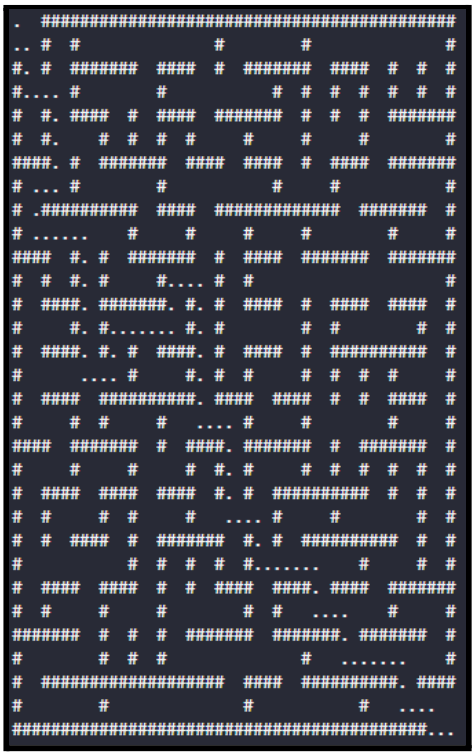
Resultaten.

In dit hoofdstuk bespreek ik verschillende resultaten van de programma's, aan het eind trek ik een conclusie welke taal het beste het probleem oploste.

input / output

Ik heb het programma op meerdere doolhoven getest. de output is bij elke programma altijd hetzelfde al bij de andere programma's.





Regels

Voor het aantal regels in Haskell duidelijk de winnaar. Zelfs als je bijna geen haskell ervaring hebt hoef je niet je best te doen om je programma zo kort mogelijk te maken. Dit wil niet zeggen dat minder regels altijd beter is. omdat het de leesbaarheid ook kan verslechteren.

Taal	Regels
C#	145
Go	149
Haskell	56

Snelheid

Om de snelheid te meten, meet ik de tijd voordat het algoritme begint en de tijd als het algoritme klaar is. Daarna kijk ik wat het verschil hier tussen is. Het wordt op dezelfde computer gemeten. Voor het meten van de haskell code gebruik ik `:set +s`. Het doolhof dat hiervoor wordt gebruikt is doolhof 1 in “/input/1.txt”

Taal	Tijd in ms (met print)	Tijd in ms (zonder print)
C#	30	18 tot 20
GO	37	1 tot 2
Haskell	26	12

Er wordt vooral gekeken naar de resultaten zonder print. Omdat de ene print wel veel zwaarder kan zijn dan de ander en hier heel veel tijd in zit.

Er is goed te zien dat Go bijna meteen klaar is. Haskell duurt iets langer en daarna komt C#.

Leesbaarheid

Deze beoordeling is erg subjectief maar voor elke beoordeling is een toelichting. De beoordeling wordt weergegeven in een cijfer waarbij 1 erg slecht is en 10 heel goed.

Taal	Leesbaarheid
C#	9
Go	7
Haskell	5

C# is erg goed te lezen omdat types goed sterk gedefinieerd zijn. Namen van standaard methodes zijn erg goed beschrijvend. Het is ook allemaal stapsgewijs waardoor je goed ziet in welke volgorde alles gebeurd.

Go is bijna hetzelfde qua syntax as C#, alleen door de toevoeging van pointers is het wat lastiger te lezen, maar ik neem aan als je er wat meer ervaring mee hebt dat dit niet meer opgaat. Wel ontbreken er veel basis methodes waardoor je soms wat extra code moet schrijven waardoor het allemaal wat complexer lijkt.

Voor iemand die vooral bekend is met OOP is haskell erg lastig te lezen. Ook zit er weinig ruimte tussen code omdat alles bijna altijd op een regel moet. Om bijna alles wat bij elkaar hoort moeten haakjes omdat het anders niet goed kan worden begrepen door de compiler.

Ik heb het gevoel als je een groot haskell project open dat je overwelmt wordt met code. wat ik bij de andere talen niet heb,

Complexiteit

Taal	Complexiteit
C#	9
Go	8
Haskell	6

Haskell vond ik erg complex. omdat je bij de meeste methodes alles in een regel typt, Elke methode in deze uitwerking is een lijstcomprehensie, De manier waarop je constant oplopende expressies achter elkaar typt maakt het ook zeker niet minder complex. Bijvoorbeeld voor het geven van waardes aan nodes moest ik een recursieve methode maken waarin ik 4 parameters heb, bij 2 parameters voer ik dan weer een lijstcomprehensie uit. Dit is zoveel werk in super weinig code dat het allemaal super complex lijkt.

Bij Go en C# is het erg fijn dat niet al het werk zo dicht op elkaar staat. Je hebt ook niet constant een expressie achter een expressie staan waardoor het minder complex is.

Conclusie

Elke taal heeft zo zijn voor en nadelen. Uiteindelijk vind ik dat Go dit specifieke probleem het beste oplost. Omdat het erg snel en goed leesbaar is. Bij de andere 2 talen ontbreekt een van deze voordelen. Zelf hecht ik ook niet erg veel waarde aan het aantal regels van de code.

Uiteindelijk zijn de stappen in alle talen hetzelfde gebleven, alleen is er vooral een groot verschil te zien in de oplossing van de individuele stappen tussen de imperatieve en de functionele talen.

Zelf denk ik dat het algoritme dat ik heb gekozen misschien niet volledig de kracht van haskell liet zien omdat er geen gebruik wordt gemaakt van de luie functionaliteit.

Bronnen

Maze Generator (Perfect) - Online Software Tool. (n.d.). Dcode.Fr. Retrieved May 3, 2021, from https://www.dcode.fr/maze-generator

Pathfinding Visualizer. (n.d.). Clementmihailescu.Github.Io. Retrieved May 3, 2021, from https://clementmihailescu.github.io/Pathfinding-Visualizer/