Wat zijn zelflerende computersystemen?

# Deelvraag 1 inleiding

Elk jaar boekt de mens grootschalige vorderingen op het gebied van computers, zowel hardware als software. Iets waar wij echter nog niet in geslaagd zijn te maken is een ware **Artificial Intelligence**, al lukt de schijn van denken creëren steeds beter. Voorbeelden zijn de ‘persoonlijke assistenten’ die inmiddels in elke smartphone geïntegreerd zijn. *Siri*, *Google Now* en *Coratana* maken gebruik van spraakherkenning om de gebruiker de gevraagde informatie te tonen, maar denken zoals mensen doen ze hierbij niet. Hoe een computersysteem toch beter kan worden in het imiteren van menselijk gedrag en van zijn eigen fouten kan leren onderzoeken in deze deelvraag.

# Verschillende algoritmes

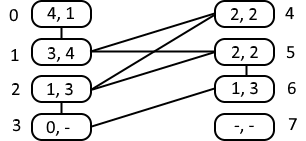
Computers hebben geen dus bewustzijn. Om deze reden kunnen ze niet zelf bepalen iets te doen. Waar computers wel in uitblinken, is het uitvoeren van taken die ze zijn opgelegd. Vaak komen deze taken in de vorm van code. Via code kan je computers opdrachten geven, bijvoorbeeld “laat een scherm zien”. De boodschap valt echter niet op deze manier over te brengen. Afhankelijk van de taal waarin je programmeert zijn er vaste commando’s waar de computer op zal reageren.  
Naarmate de opdracht die je een computer wil laten uitvoeren complexer worden, zal ook het gebruik in deze commando’s een verandering zien. Hier komen algoritmes in het spel. Een algoritme is een soort stappenplan voor de computer, waarin een complexere handeling in duidelijke opdrachten weergegeven wordt. De volgende definitie geeft een betekenis in de meest algemene zin: een algoritme is een eindige reeks instructies om vanaf een beginpunt een bepaald doel te bereiken.[a]  
Een toegankelijke vergelijking is koken. Er is een **input** van voedsel waar uiteindelijk een gerecht uit moet komen, de **output**. Voor het tot stand komen van dit gerecht gebruik je misschien een recept. Dit recept is als het ware het algoritme.  
Uit de gegeven definitie kan is af te leiden dat het aantal mogelijke algoritmes ontzettend groot is. Niet alleen is het een ruim begrip, ook kan het desbetreffende doel waarschijnlijk op meerdere manieren bereikt worden. In deze verschillende methodes kan de een echter beter zijn dan de andere, bijvoorbeeld door efficiënter te zijn.

Uiteraard zijn er ook vele algoritmes die gebruik maken van toepassingen, zoals een *queue* en een *stack*, die betrekking hebben tot ons onderwerp. Enkele hiervan zullen hier beschreven worden:

# Breadth-first search (BFS)

Dit algoritme is een zoekalgoritme voor datasets in de vorm van grafieken of *boom-structuren* zoals een **Binary Tree**. Het is bedacht in de jaren vijftig van de vorige eeuw door E.F. Moore, een Amerikaans professor in de wiskunde en computer sciences en een voortrekker in kunstmatig leven. [b] Bij een binary tree wordt één node als oorsprong benoemd. Dit wordt de **root** genoemd*.* Ook wordt een bepaalde uitkomst als doel gesteld. Vervolgens krijgt elke node drie waardes aangewezen:

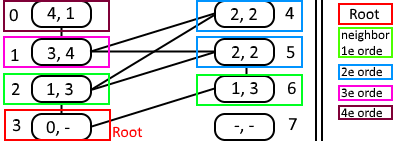
* De afstand van de huidige node naar de root. Dit is het aantal stappen dat gezet moet worden om bij de root te komen.
* De node die vóór de huidige node kwam, de **predecessor**. Anders gezegd: bij welke node je uitkomt als je een enkele stap terug zet.
* Een **state**. De state houdt bij of een node al gecontroleerd is.



Figuur \*\*\*\*

Hierboven is een voorbeeld van een dataset te zien bestaande uit acht nodes. Voor het gemak staat in elke node de afstand tot de root, node nummer 3, aangegeven, gevolgd door het nummer van de predecessor. Een liggend streepje betekent dat er geen waarde is. (De afstand van 3 tot 3 is niets, tussen 3 en 7 is geen verbinding, enz.) Het Breadth-first search algoritme werkt als volgt:

* Er wordt één node als root benoemd. In dit geval is dat node 3. Noteer de root als bezocht.
* De nodes die aan 3 grenzen worden bekeken: 2 en 6 hebben een afstand van 1. De predecessor van deze nodes is 3. 2 en 6 worden de **neighbors** van 3 genoemd en dat 3 is de predecessor van 2 en 6. In figuur \*\*\*\* is aangegeven in welke volgorde alle nodes bezocht worden. Noteer 2 en 6 als bezocht.
* Vervolgens worden de nodes die aan 2 grenzen bekeken, mits deze nog niet eerder bezocht zijn. Noteer de nieuw gecontroleerde nodes als bezocht.
* Herhaal deze stap voor alle andere neighbors van de root, in dit geval node 6
* Daarna worden de nodes die aan de eerste orde neighbors grenzen bekeken. Ook dit gebeurt niet in één keer maar stapsgewijs: eerst de neighbors van node 2, dan die van node 6. Hierdoor is de predecessor van 5 node 2 en niet node 6.  
  De afstand tot de root is bij deze neighbors van tweede orde dus 2, er moeten immers twee stappen gezet worden weer bij 3 te komen.
* Herhaal voor elke nieuwe orde aan nodes de voorgaande stappen totdat het doel bereikt is of alle nodes gecontroleerd zijn.

e 

Figuur \*\*\*\*: de volgorde waarin de nodes bezocht wroden is 3 – 2 – 6 – 4 – 5 – 1 – 0

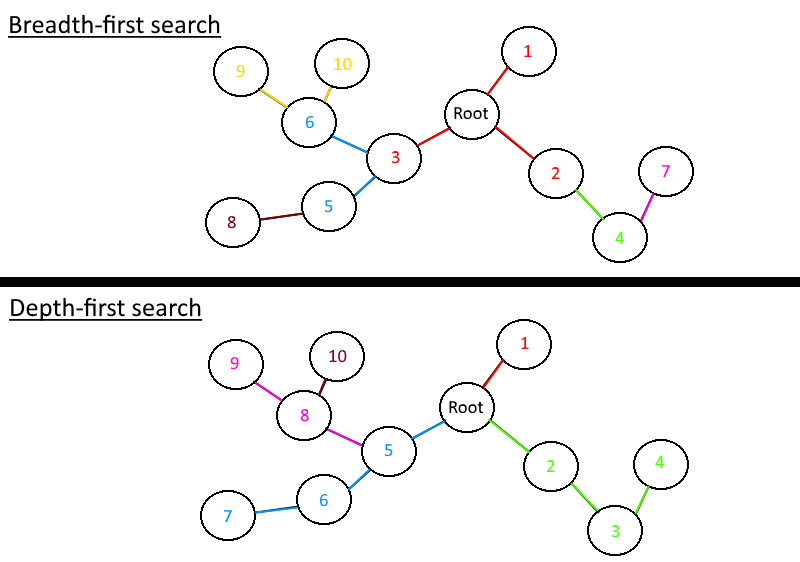
De aangranzende nodes worden dus één voor één afgegaan. Om te zorgen dat het programma weet welke neighbors nog aan de beurt moeten komen en alle nodes afgaat, wordt een **queue** gebruikt. Dit is een wachtrij voor de nodes, er geldt: eerste erin, eerste eruit.  
In eerste instantie is node 3 de queue. Deze node wordt eruit gehaald en de aangrenzende nodes worden gecheckt en in de queue geplaatst, die is bestaat nu uit 2 op de eerste en 6 op de tweede plek. Dan wordt 2 eruit gehaald en sluiten 4 en 5 achteraan in de queue. 3 is al bezocht en hoeft dus niet toegevoegd te worden. Telkens wordt nu de eerste node uit de queue gehaald en worden de aangrenzende, niet eerder gecontroleerde nodes achteraan geplaatst. Hieruit valt op te maken dat prioriteit in de queue afhankelijk is van de afstand tot de root.

Uiteindelijk wordt ook de node die als doel werd gesteld gecontroleerd. Op dat moment zou het zoekproces stilgezet kunnen worden, als je slechts geïnteresseerd bent in het doel. Als je het programma laat doorlopen kan het laten zien want de kortste afstand is van de root tot het doel.

# Depth-first search (DFS)

Evenals breadth-first search is depth-first search een algoritme voor het doorlopen van datasets in grafieken of trees. DFS verschilt echter op twee manieren van BFS:

* Depth-first search gebruikt een **stack** in plaats van een queue. Waar nodes in een BFS systeem in een wachtrij werden geplaats met een “Als eerst erin, als eerst eruit” principe, handhaaft een DFS systeem een wachtrij meer vergelijkbaar met een stapel papieren. Telkens pak je het bovenste element van de stapel om mee te werken, maar als je iets in de wachtrij stopt, komt dit ook weer bovenop de stapel te liggen. De meest recente toevoeging zal dus als eerste weer eruit gehaald worden.
* Breadth-first search begon bij een root. Vervolgens werd gekeken naar alle neighbors. Als de gewenste uitkomst niet tussen deze neighbors zit, worden de neighbors van deze neighbors gecontroleerd. Dit proces herhaalt zich totdat het doel gevonden is.  
  Depth-first search begint ook bij een root, maar kijkt direct naar een weg tot een node bereikt is die geen neighbors meer heeft. Als het doel dan niet bereikt is wordt een andere weg geprobeerd. Hiervoor wordt gebruik gemaakt van **recursive backtracking**.



Figuur \*\*\*\*

In figuur \*\*\*\* is de werking van BFS en DFS weergegeven. Het getal in elke node geeft aan als hoeveelste het bereikt wordt. De kleuren representeren de gebieden die het algoritme per stap doorloopt.

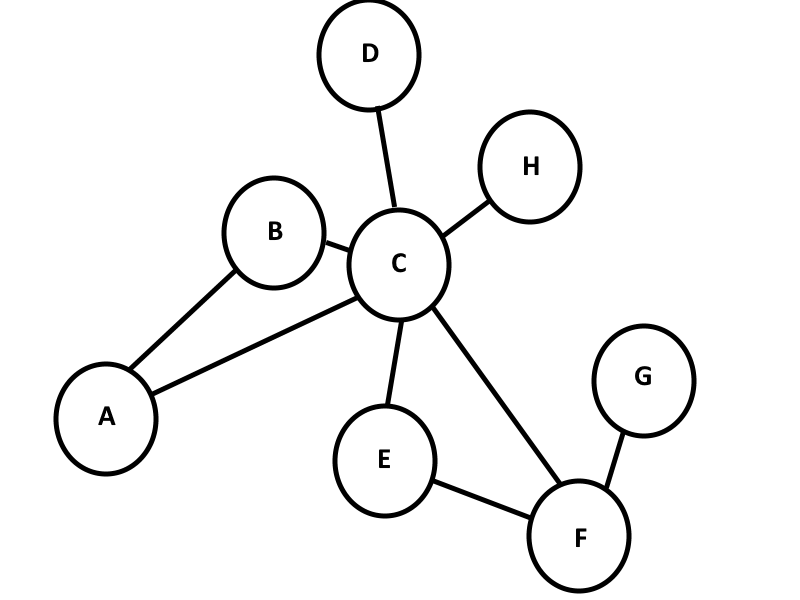
Ook bij DFS hebben de nodes een state: bezocht of niet bezocht.  
Ten eerste wordt de root gekozen en wordt deze als bezocht opgeslagen. Zoals te zien wordt er vanaf de root één (willekeurige) neighbor gekozen om te onderzoeken. Elke bezochte neighbor wordt als bezocht genoteerd. De root wordt in de stack geplaats. Als de gekozen neighbor niet het doel is, wordt de meest recentelijk toegevoegde node, de root, gehaald, de eerst bezochte had immers geen aanliggende nodes om te onderzoeken. Deze tweede neighbor, cirkel 2 in de figuur, heeft wel een neighbor. Deze wordt gecontroleerd, evenals diens neighbors. Telkens wordt de huidige node toegevoegd aan de stack.  
Als het programma de laatste in een reeks nodes bereikt heeft, wordt de bovenste node uit de stack gepakt en gekeken of daar nog niet bezochte nodes aan grenzen. Dit wordt backtracking genoemd.  
Dit proces wordt herhaald totdat het doel gevonden is of totdat alle nodes geweest zijn.

Als vuistregel kan het volgende gehanteerd worden: depth-first search wordt gebruikt als je weet dat er maar één uitkomst is, breadth-first search als je de kortste weg wil weten.

# Voorbeelden algoritmes

Algoritmes hebben meestal vele toepassingen. Hier zijn enkele voorbeelden van de eerder genoemde algoritmes.

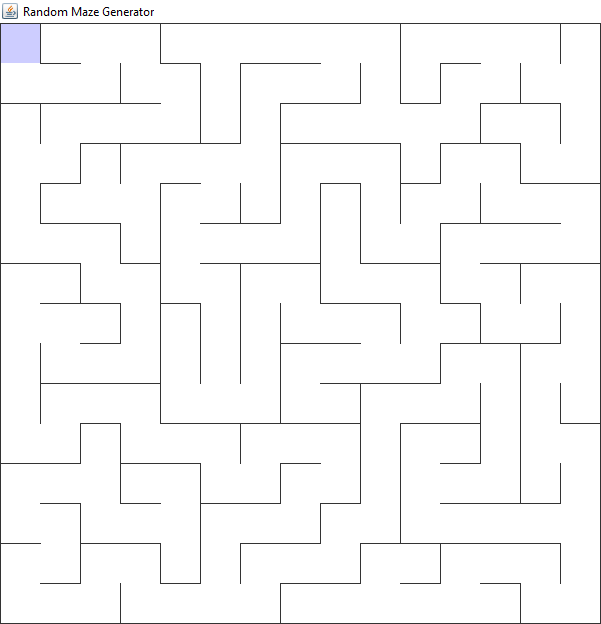
## Breadth-first search



Figuur \*\*\*\*

In figuur \*\*\*\* is een dataset te zien, bijvoorbeeld een telefoonboom. Elke cirkel representeert een persoon. Zo kan persoon A de personen B en C bellen, maar A bezit geen andere telefoonnummers. Toch zou hij een boodschap naar H kunnen sturen: via C.   
Stel, persoon A wil nu iets tegen F zeggen. In een kleine dataset als deze is makkelijk met het oog te zien dat de snelste manier hiervoor A – C – F is en dat A – B – C – E – F veel langer is. Bij grotere datasets is dit echter al snel moeilijk met zekerheid te zeggen. Hiervoor kan breadth-first search ingezet worden.

## Depth-first search

****

Figuur \*\*\*\*

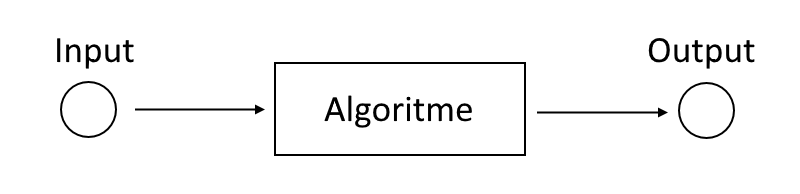
Depth-first search kan gebruikt worden voor zowel het maken als oplossen van doolhoven. In figuur \*\*\*\* is een doolhof te zien dat gemaakt is met behulp van DFS. (\*\*\*\*programma gemaakt in kader van deze deelvraag\*\*\*\*). Het algoritme in de vorm van een stappenplan is als volgt:

1. Maak de start cel current en markeer deze als bezocht.
2. Terwijl er nog niet bezochte cellen aanwezig zijn
   1. Als current neighbors heeft die nog niet bezocht zijn
      1. Kies willekeurig een van de neighbors
      2. Voeg current toe aan de stack
      3. Verwijder de muur tussen de huidige cel en de gekozen cel
      4. Benoem de gekozen cel als current en zet de state op bezocht
   2. Anders, als de stack niet leeg is
      1. Haal de laatst toegevoegde cel uit de stack en verwijder deze hieruit
      2. Maak deze cel current

# Zelflerend?

Breadth-first search en Depth-first search zijn beide algoritmes met vele toepassingen. Toch kunnen beide algoritmes niet als zelflerend worden beschouwd, ze verbeteren hun manier van zoeken namelijk niet. Hoe zit een zelflerend systeem dan wel in elkaar? Hoe kan een algoritme zichzelf verbeteren?

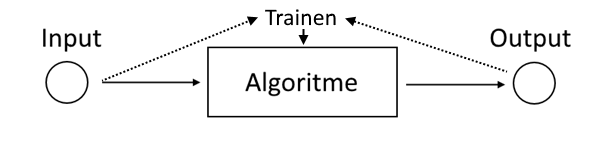
# Machine learning

Een zelflerend systeem is een algoritme gebaseerd op machine learning. Machine learning werd door Arthur Samuel, een pionier op dit gebied, gedefinieerd als: "A field of study that gives computers the ability to learn without being explicitly programmed.” [1]. In tegenstelling tot de eerder genoemde algoritmes is een zelflerend systeem in staat zichzelf te verbeteren. Hierdoor kan het taken uitvoeren waarbij reguliere algoritmes tekort schieten. Welke taken dit betreft, zullen we in de derde deelvraag behandelen.

Figuur 2 Schematische weergave van een zelflerend systeem

In figuur \*\*\*\* is een schematische weergave van een zelflerend systeem afgebeeld. Bepaalde input data gaat het systeem in en bepaalde output data komt het systeem uit. De input en output data bestaat uit één of meerdere getallen. Als de input simpelweg een reeks getallen betreft, zal dit direct als input gebruikt kunnen worden. In het geval dat de input uit een ander datatype bestaat, zoals een plaatje, zal dit omgezet moeten worden in een reeks getallen voordat het in een zelflerend systeem gebruikt kan worden. Het algoritme zal deze getallen bewerken tot de gewenste output. Deze output wordt eveneens in getallen gegeven. Waar nodig zullen deze getallen dus weer moeten worden omgezet tot het gewenste datatype.

Er zijn veel verschillende algoritmes die gebruikt kunnen worden voor een zelflerend systeem. Elk algoritme heeft voor- en nadelen en is geschikt voor andere doeleinden. Een aantal van deze algoritmes zullen we in de tweede deelvraag behandelen.

Training

Een zelflerend systeem begint in de meeste gevallen zonder enige kennis van de data. Om de gewenste output te kunnen produceren is het dus nodig om het systeem eerst input data te geven zodat het kan leren. Dit proces wordt het ***trainen*** genoemd. Voor het trainen van een zelflerend systeem is training data nodig. Deze data moet gelijk of gelijkwaardig zijn aan de *echte* data. De training data kan in veel verschillende vormen voorkomen en de manier van trainen is afhankelijk van de vorm van de (training) data. In figuur \*\*\*\* is te zien dat het trainen los staat van het algoritme. Dit verschil zullen we in de volgende deelvraag wat duidelijker maken.   
Er zijn drie prominente manieren waarop een zelflerend systeem getraind kan worden: **supervised**, **unsupervised** en **reinforcement learning**.

Figuur 3 Schematische weergave van een zelflerend systeem

Supervised Learning

In het geval van supervised learning heb je te maken met **labeled** training data. Anders gezegd: van een bepaalde input is de gewenste output al bekend. Een klassiek voorbeeld van een labeled dataset is een dataset van huisprijzen en huiseigenschappen (zie figuur xx)

|  |  |  |  |
| --- | --- | --- | --- |
| Huisprijs (output) | Huiseigenschappen (input) | | |
|  | Woonoppervlakte | Perceeloppervlakte | Aantal kamers |
| € 519.000 | 124 m² | 311 m² | 4 |
| € 569.000 | 133 m² | 309 m² | 5 |
| € 569.500 | 170 m² | 310 m² | 6 |

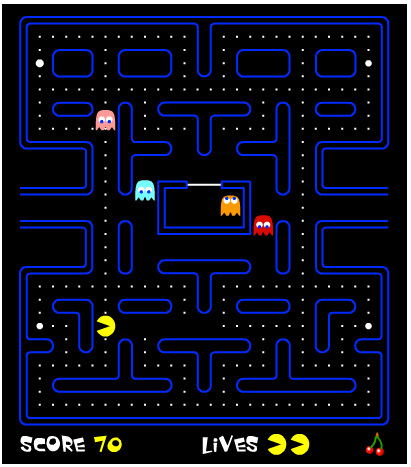
Figuur xx Labeled dataset   
Bron: http://www.funda.nl/koop/huizen/   
  
Bij de training dataset van figuur \*\*\*\* is de gegeven input de huiseigenschappen en de gewenste output de huisprijs. Het systeem wordt met deze dataset getraind. Hierdoor leert het een output te produceren die steeds dichter bij de gewenste output ligt. Als er een verband bestaat tussen de huiseigenschappen en de huisprijs, wat waarschijnlijk het geval is, zal het zelflerende systeem na genoeg trainen in staat zijn zelf bij nieuwe huiseigenschappen een huisprijs te voorspellen. [2]

## Unsupervised Learning

Unsupervised learning kan gebruikt worden bij een **unlabeled** dataset ofwel, een dataset waarbij de data niet geclassificeerd is en er geen gewenste output bekend is. Als je een dataset hebt van heel veel niet-geordende foto’s is het niet mogelijk om dit te classificeren. Als een deel van de dataset gelabeld wordt, zal met behulp van supervised learning de rest van de dataset geclassificeerd kunnen worden. Dit is echter in veel gevallen niet mogelijk, bijvoorbeeld doordat de dataset enorm groot is of er zodanig veel verschillende groepen bestaan dat het menselijk niet mogelijk is ook maar een deel te labelen. Ook kan het zo zijn dat men niet weet of er een verband aanwezig is.   
Kortom: unsupervised learning wordt gebruikt voor het classificeren van data, zonder dat er groepen vooraf gedefinieerd zijn. Met behulp van deze vorm van training zullen in een grote dataset verbanden kunnen worden ontdekt, die men misschien niet zonder hulp had kunnen achterhalen.[3]

## Reinforcement Learning

Reinforcement learning is een zeer specifieke soort van leren. Er is bij deze vorm van learning geen dataset met input data, maar is er een bepaalde **context**. In deze context bevindt zich een ***agent****.* Een agent is een object dat bepaalde opdrachten kan uitvoeren. De context is een *wereld* waarin deze agent zich bevindt. Door de agent bij bepaalde acties pluspunten of minpunten te geven kun je bepaald gedrag bevorderen.



Figuur 4 Pacman   
Bron: https://www.flickr.com/photos/methodshop/4865516413

In figuur \*\*\*\* is het spel pacman te zien. De agent is hierbij pacman, deze kan namelijk een object dat bepaalde opdrachten kan uitvoeren zoals: loop naar links, loop naar rechts. De context is hierbij het level, ofwel: de positie van de muren (de blauwe obstakels), de posities van de *ghosts (de geleurde vijanden)*, de posities van de pac-dots (de kleine stipjes, ofwel punten) en de posities van de power-pellets (de grotere stipjes). [4] Het eten van de pac-dots is positief, het geraakt worden door de ghosts is negatief. Door reinforcement learing toe te passen op het spel zal de agent steeds beter worden in het spelen van het spel.

# Conclusie

Zelflerende computersystemen zijn algoritmes gebaseerd op machine learning. Een zelflerend systeem verschilt van reguliere algoritmes zoals breadth-first search en depth-first search doordat ze in staat zijn zichzelf te verbeteren.

# Bronnen

[1] Bron: <https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf>Geraadpleegd op: 21-5-2017  
Laatst gewijzigd op: niet bekend

[2] Bron: <https://www.coursera.org/learn/machine-learning/lecture/1VkCb/supervised-learning>  
Geraadpleegd op: 21-5-2017  
Laatst gewijzigd op: niet bekend

[3] Bron: <https://www.coursera.org/learn/machine-learning/lecture/olRZo/unsupervised-learning>  
Geraadpleegd op: 21-5-2017  
Laatst gewijzigd op: niet bekend

[4] Bron: <http://pacman.wikia.com>  
Geraadpleegd op: 27-5-2017  
Laatst gewijzigd op: niet bekend

**Gebruikte bronnen (temporary)**

[a] Link: <http://www.woorden.org/woord/algoritme>   
Geraadpleegd op: 22- 05 – 2017   
Laatst gewijzigd op: niet bekend

Link: <https://beebom.com/examples-of-artificial-intelligence/>   
Geraadpleegd op: 22- 05 – 2017   
Laatst gewijzigd op: 26 – 09 – 2017

Link: <http://study.com/academy/lesson/what-is-an-algorithm-in-programming-definition-examples-analysis.html>  
Geraadpleegd op: 22- 05 – 2017   
Laatst gewijzigd op: 2017

[b] Link: <http://www.secfac.wisc.edu/senate/2003/0929/1727(mem_res).pdf>   
Geraadpleegd op: 23- 05 – 2017   
Laatst gewijzigd op: 23 – 09 – 2003

Link: <https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm>   
Geraadpleegd op: 22- 05 – 2017   
Laatst gewijzigd op: 26 – 09 – 2017

Link: <http://www.graph500.org/specifications#sec-5>   
Geraadpleegd op: 22- 05 – 2017   
Laatst gewijzigd op: 26 – 09 – 2017

Link: <https://www.kirupa.com/developer/actionscript/depth_breadth_search.htm>   
Geraadpleegd op: 27- 05 – 2017   
Laatst gewijzigd op: 13 – 01 – 2006

Link: <http://www.algolist.net/Algorithms/Graph/Undirected/Depth-first_search>   
Geraadpleegd op: 27- 05 – 2017   
Laatst gewijzigd op: niet bekend

Link: <https://en.wikipedia.org/wiki/Maze_generation_algorithm#Recursive_backtracker>   
Geraadpleegd op: 27- 05 – 2017   
Laatst gewijzigd op: 19 – 05 – 2017

Link: <https://www.khanacademy.org/computer-programming/depth-first-traversals-of-binary-trees/934024358>   
Geraadpleegd op: 27- 05 – 2017   
Laatst gewijzigd op: 2012