Bestand waar Thijs in typt (wegens GitHub)

* Hierin staat een tijdelijke bronnenlijst. Deze moet later in het grote verslag geïmplementeerd worden.
* Waar \*\*\*\* staat, moet later naar gekeken worden. Dit kan staan bij een twijfel in de formulering of bij de nummering van een bron, etc.

# Inleiding

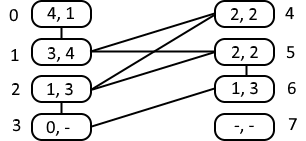
**Moet nog geschreven worden (einde project)**  
Awef (hoe persoonlijk willen we dit maken? We moeten er rekening mee houden dat dit verslag als het goed is de verwezenlijking van geweldigheid gaat worden. Misschien wordt het bij de universiteit bekeken en zou het raar zijn als we beginnen met “hallo, wij zijn Steven en Thijs van het Candea College in Duiven!”)

### Breadth-first search (BFS)

Dit algoritme, bedacht in de jaren vijftig van de vorige eeuw door E.F. Moore[b], een Amerikaans professor in de wiskunde en computer sciences en een voortrekker in kunstmatig leven, is een zoekalgoritme voor datasets in de vorm van grafieken of ‘boom’-structuren, zoals een **Binary Tree**. In deze data set wordt een \*\*\*\***node**\*\*\*\* als oorsprong benoemd, de **root***.* Ook wordt een bepaalde uitkomst als doel gesteld. Vervolgens krijgt elke node twee 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.

Bovendien heeft elke node een **state***.* Deze state houdt bij of een node al gecontroleerd is.



Figuur \*\*\*\*

Hierboven is een voorbeeld van een dataset te zien bestaande uit acht nodes, met node 3 als root. In elke node staat eerst de afstand tot node 3 aangegeven en daarnaast 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.)  
Node 3 wordt als root genomen. Daarna worden de nodes bekeken die aan 3 grenzen: 2 en 6 hebben een afstand van 1. De predecessor van deze nodes is 3. 2 en 6 worden de **neighbors** van 3 genoemd. Vervolgens worden de nodes die aan 2 grenzen bekeken en herhaald het proces zich voor al deze nodes. Daarna worden de nodes die aan 6 grenzen bekeken. De afstand tot de root is bij deze nodes dus 2, er moeten immers twee stappen gezet worden weer bij 3 te komen. Dit proces wordt herhaald totdat het doel gevonden is.  
De aangrenzende nodes worden echter niet tegelijkertijd gecontroleerd. In dit geval wordt eerst 2 en daarna 6 gecheckt, daardoor is de predecessor van 5 node 2 en niet node 6. Op het moment dat node 5 gecontroleerd wordt, wordt de state op ‘bezocht’ gezet. Als vervolgens de aanliggende nodes van 6 bekeken worden, wordt opgemerkt dat 5 al gecontroleerd is. Zo weet het systeem dat 5 niet nogmaals gecheckt hoeft te worden. Omdat ook de root al als bezocht staat, wordt 3 ook niet gecontroleerd.

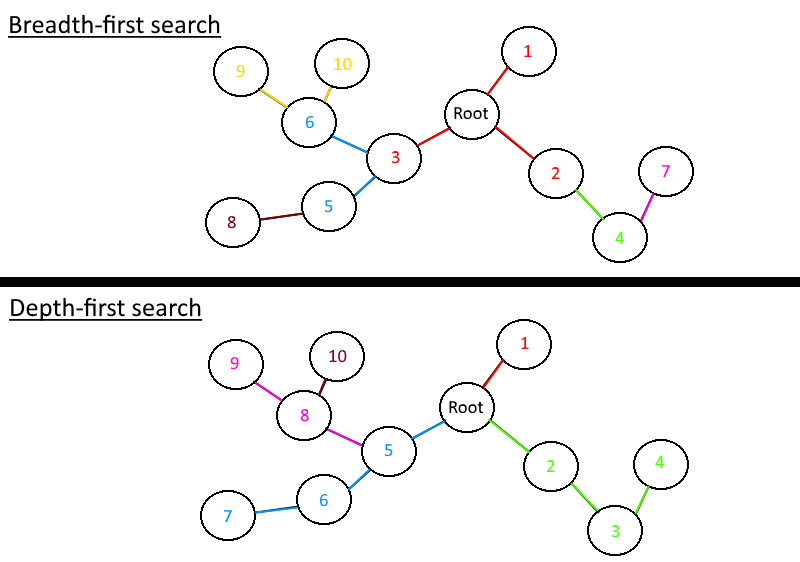
Om te zorgen dat het programma alle nodes afgaat, wordt een **queue** gebruikt. Dit is een wachtrij voor de nodes, er geldt: eerste erin, eerste eruit.  
In eerste instantie is node 3 de queue. Deze node wordt eruit gehaald en de aangrenzende nodes worden gecheckt en in de queue geplaatst, die is bestaat 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. Het programma kan nu laten zien want de kortste afstand is tot de root.

## 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 de bovenste van de stapel om mee te werken, maar als je iets in de wachtrij stopt, komt deze 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.

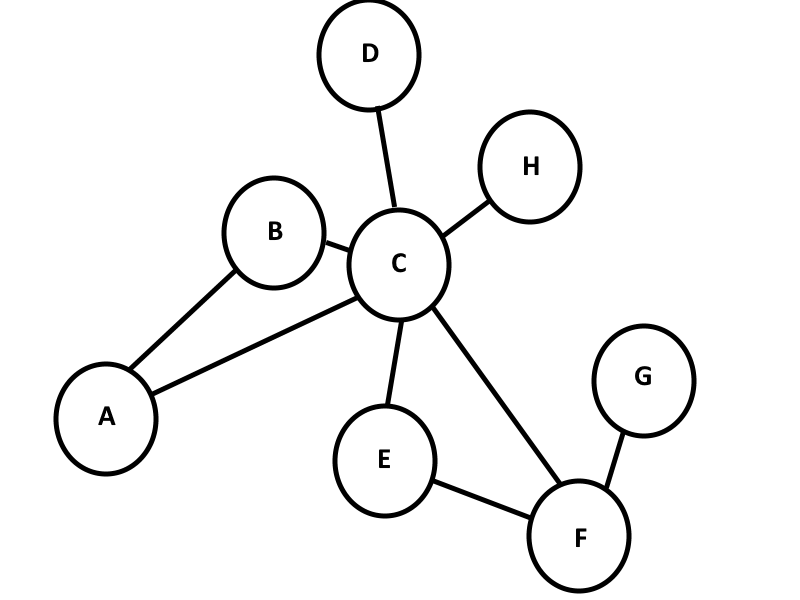
Ook bij DFS hebben de nodes een state: bezocht of niet bezocht.  
Ten eerste wordt de root gekozen en wordt deze als bezocht opgeslagen. Zoals te zien wordt er vanaf de root één (willekeurige) neighbor gekozen om te onderzoeken. Elke bezochte neighbor wordt als bezocht genoteerd. De root wordt in de 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.

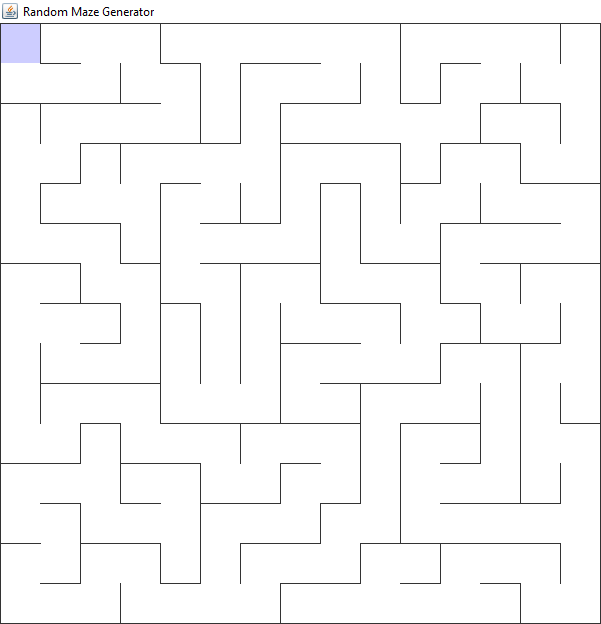
**Breadth-first search**



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

**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