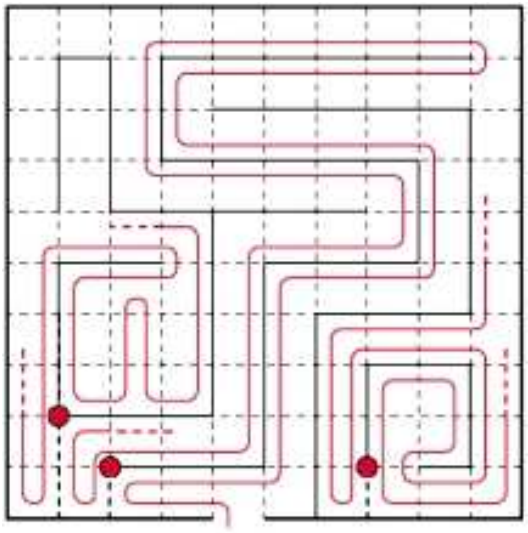
# Klein Duimpje

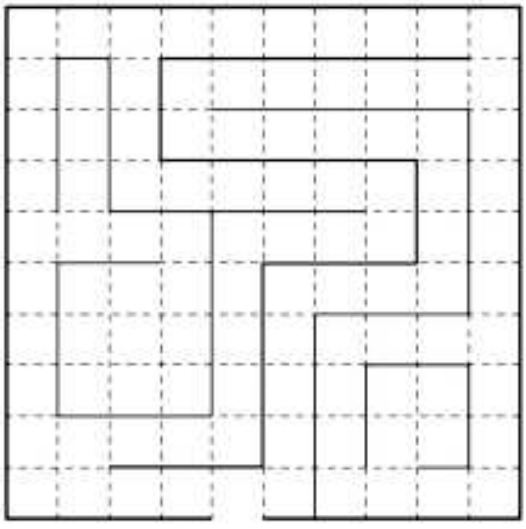
Bij de colleges Algoritmiek en Datastructuren leer je hoe je een doolhof kunt doorzoeken. De meest gebruikte methode heet *depth-first search*: je loopt zover mogelijk door, willekeurig gangen inlopend, en wanneer het pad doodloopt, keer je op je schreden terug tot het laatste punt waar je afgeslagen bent, om daar een andere, en hopelijk nieuwe, gang te kiezen. In de praktijk betekent dit dat je een krijtje mee moet nemen om de plekken te markeren waar je al geweest bent, om niet nodeloos rond te lopen.



*Figuur 2: Wandeling in doolhof met linker onderhoeken; Denkbeeldige heggen bijgeplaatst.*

**Door Hendrik Jan Hoogeboom**

Stel je nu voor dat je een robot moet programmeren die een doolhof moet doorzoeken om daar een voorwerp te ontdekken, een geldprijs of een pakje semtex bijvoorbeeld. Voor het gemak stellen we ons voor dat een doolhof volgens een rooster is opgebouwd met heggen (scheidingswanden) tussen de hokjes, zie figuur 1.



*Figuur 1: Doolhof*

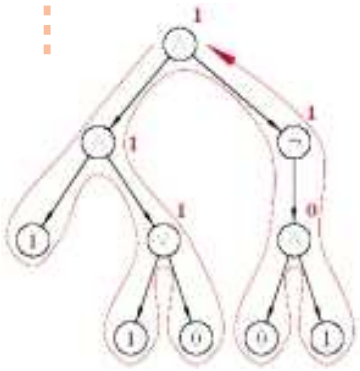
Als je de robot mag uitrusten met een fatsoenlijk geheugen kan hij de vorm van de doolhof leren door rond te lopen en kan hij bijhouden waar hij geweest is. Moeilijker wordt het als we slechts een beperkt geheugen hebben en eisen dat het programma moet werken voor élk doolhof, hoe groot ook. De robot heet dan een *eindige automaat* en moet het verder stellen met zo min mogelijk hulpmiddelen. Als de robot weer in de doolhof mag schrijven waar hij is geweest, voldoet het programmeren van een eenvoudige *depth-first search*. Als dat niet kan (of mag) zijn we aangewezen op andere trucs. Een wel eens gehoorde techniek is om met één hand de heg die de hokjes scheidt te volgen tot je bij de uitgang bent. Dat werkt redelijk, maar je bezoekt geen hokjes die niet aan een heg grenzen en je bezoekt ook geen 'eilanden' binnen de doolhof, zoals rechtsonder in het voorbeeld in figuur 2. In 1978 werd deze techniek op ingenieuze wijze verbeterd: geef de robot één teller in het geheugen en hij kan van elke heg (ook als die los staat van de buitenrand) de linker onderhoek herkennen. Met behulp van deze posities kan de robot denkbeeldige heggen bijplaatsen (recht onder zo'n positie) waardoor in principe gewoon de heg gevolgd kan worden om alle plekken te bezoeken. Zie hiervoor figuur 2. Vind je een teller geen eindig geheugen? In dat geval kan de teller vervangen worden door precies twee kiezelstenen die de robot in de doolhof neerlegt en steeds verplaatst om zijn weg te bepalen.

In recent onderzoek kwamen we een vergelijkbaar probleem tegen, maar dan bij het evalueren van expressies.Expressies vormen een onderdeel van elke programmeertaal en we kennen ze in

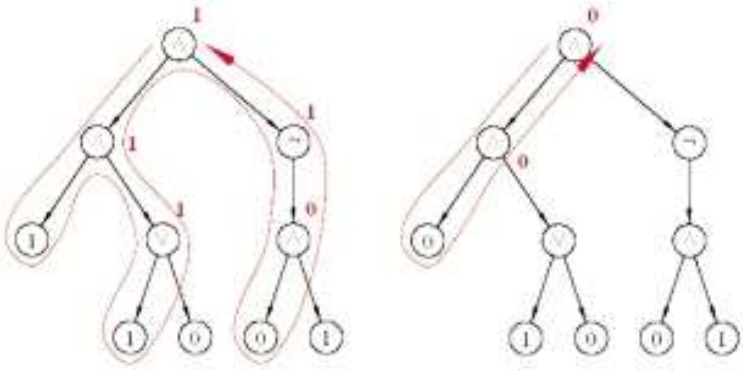
vele varianten: bijvoorbeeld rekenkundige expressies (voor getalswaarden), Boolese expressies (true/false of 1/0) of reguliere expressies (voor verzamelingen strings). Bijvoorbeeld, voor de Boolese uitdrukking (1∧(1∨0))∧ÿ(1∧0) vinden we een boom. Zie figuur 3. Als we de waarde van die expressie willen uitrekenen kunnen we de bijbehorende boom *bottom-up* evalueren door (recursief) de waarde in elke knoop te bepalen op grond van de waarde van de kinderen. De waarde van de hele expressie komt dan in de wortel te staan.

Deze evaluatie kan ook als een sequentieel proces gezien worden. We maken dan een post-orde wandeling door de boom en evalueren de knopen tijdens die wandeling. Deze wandeling is te programmeren als eindige automaat die in twee richtingen langs de takken van de boom mag lopen. Die boomwandelautomaat moet dan wel de knopen van de boom gebruiken om de deelantwoorden te bewaren.
Maar wat als we de antwoorden niet in de knopen van de boom mogen schrijven, omdat de datastructuur die we gebruiken daar bijvoorbeeld geen plaats voor heeft gereserveerd? We zien daar de robot in terug die geen aantekeningen in de doolhof mag maken. Bij Boolese expressies is daar een slimme oplossing voor te verzinnen. De automaat (robot) evalueert geen deelbomen wanneer de waarde daarvan niet bijdraagt tot het eindresultaat. Bijvoorbeeld in een knoop corresponderend met ∧ (logische 'en') hoeven we de rechterdeelboom niet te bezoeken als we links al 0 (false) gevonden hebben, de totale waarde is dan toch 0. Daarom, als we in een ∧ knoop vanuit de rechterkant terugkomen, hoeven we niet terug te zoeken wat de waarde links is geweest. Die was 1 (true) omdat we anders überhaupt niet rechtsaf waren gegaan.
Op die manier zijn Boolese expressies te evalueren met een eindige boomwandelautomaat zonder verder geheugengebruik. De boomwandeling hangt af van de onderweg gevonden waarden. Zie figuur 4 en 5.

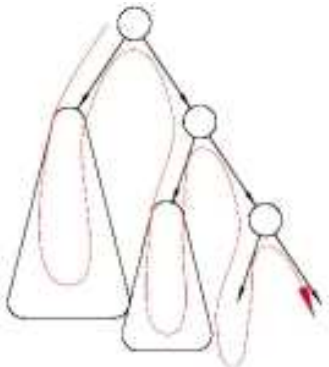
Helaas zijn er eigenschappen van bomen waarvoor niemand nog een eindige boomwandelautomaat heeft weten te programmeren. Stel dat we willen controleren dat langs het pad dat rechtsaf een binaire boom afzakt steeds de linkersubboom ten minste één blad met argument 1 bevat. Dit klinkt eenvoudig: zoek voor elke linkersubboom met behulp van een wandeling naar zo'n blad. Helaas is het niet zo eenvoudig. Zonder verdere hulpmiddelen kan de boomwandelautomaat niet zien of hij al terug is op het rechterpad en hij weet dus niet wanneer hij klaar is met zoeken. De oplossing: we geven de automaat een (denkbeeldige) kiezel mee die een knoop op het rechterpad kan markeren.Het is echter nog helemaal niet duidelijk in welke gevallen zo'n kiezel nodig is om de weg niet kwijt te raken. We hebben het gebruik van kiezels in boomwandelingen verder onderzocht. We lieten zien dat zoekacties gegeven in een bepaalde specificatietaal omgezet kunnen worden in wandelingen van een robot met kiezels. 'Is dat nuttig?', hoor ik de decaan al vragen. Dat blijkt inderdaad (een beetje tot onze verbazing) toepassingen te hebben. In de theorie van databases probeert men zoekacties (*queries*) op gegevens bestanden zo efficiënt mogelijk te implementeren. Tegenwoordig hebben die gegevens vaak een boomstructuur (bijvoorbeeld in een XML-formaat). Voor zo'n *query* moet elk gegeven op de gewenste boomstructuur getest worden. Dat laatste wordt dan gemodelleerd met boomwandelautomaten. Met kiezels om de weg niet kwijt te raken. ■



*Figuur 3: Post-orde evaluatie van een expressie gegeven door de boomstructuur.*



*Figuur 4: Implementatie van een evaluatie van logische expressies met een eindige boomwandelautomaat; De wandeling hangt van de argumenten af.*



*Figuur 5: Denkbeeldig pad van een wandeling die steeds bomen aan de linkerzijde moet controleren.*