// Chris Bleakley

als dit dan dat

// De algoritmen
// waar de digitale
// wereld op draait

Voor Eileen – dank je.

Veen Media
Oostenburgervoorstraat 166
1018 MR Amsterdam

Eerste druk, februari 2023

Titel
Als dit, dan dat
De algoritmen waar de digitale wereld op draait

Auteur
Chris Bleakley

Oorspronkelijke uitgave

Uitgever
Jim Jansen
Productiecoördinatie
Fenna van der Grient
Vertaling, eindredactie en index
Erick Vermeulen
Redactie
Serpenti tekstverzorging
Beeldresearch- en bewerking
Rikie Gorissen
Grafische vormgeving omslag
Villa Grafica
Ontwerp en opmaak
Sagor – grafische producties
Druk- en bindwerk
Wilco

ISBN: 9789085718079 / NUR: 980

Dit boek verscheen eerder als onderdeel van de serie Wetenschappelijke Bibliotheek, met als titel *Algoritmen*.

Inhoud

Inleiding

‘Een voor jou. Een voor mij. Een voor jou. Een voor mij.’ Je staat op het schoolplein. De zon schijnt. Je deelt een zak snoep met je beste vriend. ‘Een voor jou. Een voor mij.’ Wat je destijds niet besefte, was dat het delen van de snoepjes op die manier een uitvoering van een algoritme was.

Een algoritme is een reeks stappen die kan worden uitgevoerd om een informatieprobleem op te lossen. Op die zonnige dag gebruikte je een algoritme om de snoepjes eerlijk te delen. De invoer voor de algoritme was het aantal snoepjes in de zak. De uitvoer was het aantal snoepjes dat jij en je vriend elk ontving. Als het totale aantal snoepjes in de zak even was, zouden jullie allebei hetzelfde aantal snoepjes hebben ontvangen. Als het totale aantal oneven was, kreeg je vriend uiteindelijk een snoepje meer dan jij.

Een algoritme is als een recept. Het somt eenvoudige stappen op die, indien gevolgd, een verzameling van invoeren verandert naar een gewenste uitvoer. Het verschil is dat een algoritme informatie verwerkt, waar een recept voedsel bereidt. Doorgaans werkt een algoritme op fysische hoeveelheden die informatie voorstellen.

Vaak bestaan er alternatieve algoritmen voor het oplossen van een probleem. Je zou je snoepjes kunnen hebben gedeeld door ze te tellen, het totaal uit je hoofd in tweeën te delen en het juiste aantal snoepjes te geven. De uitkomst zou hetzelfde zijn geweest, maar de algoritme – de methode om de uitvoer te verkrijgen – zou anders zijn geweest.

Een algoritme wordt opgeschreven als een reeks instructies of opdrachten. Meestal worden die instructies in een bepaalde volgorde uitgevoerd, de een na de ander. Soms is de volgende uit te voeren instructie niet de volgende opeenvolgende stap maar een instructie verderop in de lijst. Het

kan bijvoorbeeld voorkomen dat een persoon die de algoritme uitvoert, moet teruggaan naar een eerdere stap en vandaar dan verder moet gaan. Dit terugspringen staat de herhaling van groepen van stappen toe – een krachtig kenmerk van veel algoritmen. De stappen 'Een voor jou. Een voor mij.' werden herhaald bij de snoepdelenalgoritme. De handeling van het herhalen van stappen staat bekend als *iteratie*.

Als het aantal snoepjes in het de zak even was, zou de volgende iteratieve algoritme hebben volstaan:

```
Herhaal de volgende stappen:
   Geef een snoepje aan je vriend.
   Geef een snoepje aan jezelf.
Stop met herhalen als de zak leeg is.
```

Bij de weergave van een algoritme zoals hier, worden de stappen voor de helderheid regel voor regel opgeschreven. Inspringen groepeert doorgaans onderling verwante stappen.

Als het aantal snoepjes in de zak zowel even als oneven kan zijn, wordt de algoritme een beetje ingewikkelder. Dan moet er een beslissingsstap worden toegevoegd. De meeste algoritmen bevatten beslissingsstappen. Een beslissingsstap vereist dat de operator die de algoritme uitvoert, kiest tussen twee mogelijke handelwijzen. Welke handeling wordt uitgevoerd, hangt af van een *voorwaarde*. Een voorwaarde is een bewering die ofwel geldig ofwel ongeldig is. De meest algemeen voorkomende beslissing nemende constructie – als – dan – anders (*if – then – else*) – combineert een voorwaarde en twee mogelijke handelingen. 'Als' de bewering geldig is, 'dan' wordt de eerstvolgende handeling (of handelingen) uitgevoerd. 'Als' de bewering ongeldig is, wordt de stap (of de stappen) na 'anders' uitgevoerd.

Om rekening te houden met een oneven aantal snoepjes moeten de volgende beslissing nemende stappen in de algoritme worden ingebouwd:

```
Als dit het eerste snoepje is of je hebt net een snoepje
gekregen,
dan geef je dit snoepje aan je vriend,
anders geef je het snoepje aan jezelf.
```

De voorwaarde hier is samengesteld, wat inhoudt dat die bestaat uit twee (of meer) eenvoudige voorwaarden. De eenvoudige voorwaarden zijn 'dit is het eerste snoepje' samen met 'je hebt net een snoepje gekregen'. De twee eenvoudige voorwaarden zijn samengevoegd door een 'of'-operatie. De samengestelde voorwaarde is waar als een van de eenvoudige voorwaarden geldig is. In het geval dat de samengestelde voorwaarde geldig is, wordt de stap 'geef dit snoepje aan je vriend' uitgevoerd. In het andere geval treedt de stap 'geef dit snoepje aan jezelf' op.

De volledige algoritme luidt dan:

```
Neem een zak snoepjes als invoer.
Herhaal de volgende stappen:
   Neem een snoepje uit de zak.
Als dit het eerste snoepje is of je net een snoepje hebt
gekregen,
   dan geef je het snoepje aan je vriend,
   anders geef je het snoepje aan jezelf.
Stop met herhalen als de zak leeg is.
Plaats de lege zak in de afvalemmer.
De snoepjes zijn nu eerlijk gedeeld.
```

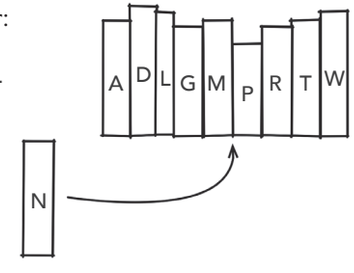
Zoals alle algoritmen is deze keurig en bereikt het doel op efficiënte wijze.

De bibliotheekstagiair

Informatieproblemen treden dagelijks op. Stel je een bibliotheekstagiair voor op de eerste werkdag. Duizend gloednieuwe boeken zijn net afgeleverd en bevinden zich in dozen op de vloer. Het hoofd van de bibliotheek wil dat de boeken alfabetisch op auteursnaam op de planken worden geplaatst, zo snel mogelijk. Dat is een informatieprobleem en er bestaan algoritmen om dat op te lossen.

De meeste mensen zouden intuïtief een algoritme gebruiken die Insertion Sort (invoegsortering) wordt genoemd.
Die algoritme werkt op de volgende manier:

INSERTION SORT
IN ACTIE



```
Neem een stapel ongesorteerde boeken als invoer.
Herhaal de volgende stappen:
      Pak een boek.
      Lees de naam van de auteur.
      Zoek langs de plank tot je het punt vindt waar het boek
      moet worden ingevoegd.
      Verschuif alle boeken voorbij dat punt een plek naar
      rechts.
      Voeg het nieuwe boek in.
Stop met herhalen als er op de vloer geen boeken meer zijn.
De boeken zijn nu gesorteerd.
```

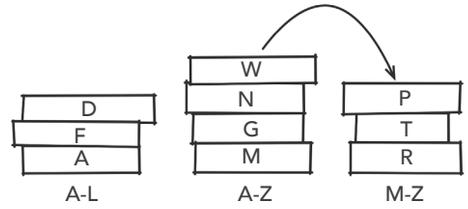
Op elk gegeven moment zijn de boeken op de vloer ongesorteerd. Het ene na het andere boek wordt naar de plank overgebracht. Elk boek wordt op de plank op alfabetische volgorde geplaatst. Het resultaat is dat de boeken op de plank altijd geordend zijn.

Insertion Sort is gemakkelijk te begrijpen, maar werkt langzaam. Het is langzaam omdat bij elk boek dat van de vloer wordt opgepakt de bibliotheekstagiair elk boek op de plank moet bekijken of opschuiven. In het begin zijn er maar weinig boeken op de plank, en dan gaat het bekijken en opschuiven snel. Aan het eind heeft onze bibliotheekstagiair bijna duizend boeken op de plank staan. Gemiddeld vereist het plaatsen van een boek op de juiste plek 500 operaties (handelingen), waarbij een operatie bestaat uit het vergelijken van auteursnamen of het verschuiven van een boek. Het sorteren van alle boeken vereist dan gemiddeld 500.000 (1000 × 500) operaties. Stel dat een enkele operatie een seconde vergt. In dat geval zal het sorteren van de boeken met Insertion Sort ongeveer zeventien werkdagen vereisen. Het hoofd zal daar niet blij mee zijn.

Een sneller alternatief algoritme – Quicksort – werd in 1962 uitgevonden door de computerwetenschapper Tony Hoare. Hij werd in 1938 geboren in Sri Lanka, als zoon van Britse ouders. Hij volgde onderwijs in Engeland, studeerde aan de Universiteit van Oxford en werkte bij een computerbedrijf voordat hij in 1968 hoogleraar computerwetenschap werd. Zijn methode voor sorteren is een verdeel-en-heers-algoritme. Die is veel ingewikkelder dan Insertion Sort, maar zoals de naam aangeeft, werkt hij veel sneller.

Quicksort verdeelt de stapel boeken in tweeën. Dat verdelingspunt, de spil, is de *pivotletter*. Boeken waarvan de auteursnaam begint met een letter voor de pivotletter worden op een nieuwe stapel links van de ongesorteerde boeken gelegd. Boeken met een auteursnaam beginnend met de letter na de pivotletter komen op een stapel rechts terecht. De ontstane stapels worden dan verdeeld op basis van nieuwe pivotletters. Daarbij worden de stapels in de juiste volgorde geplaatst. De meest linkse stapel bevat de boeken waarvan de auteursnamen het eerst in het alfabet voorkomen. De volgende stapel bevat de boeken die daarna komen enzovoort. Dit stapels splitsende proces wordt voor de grootste stapel herhaald totdat de grootste stapel slechts vijf boeken bevat. Dan worden de stapels afzonderlijk gesorteerd met Insertion Sort. Ten slotte worden de stapels, in de juiste volgorde, overgebracht naar de plank.

QUICKSORT
IN ACTIE

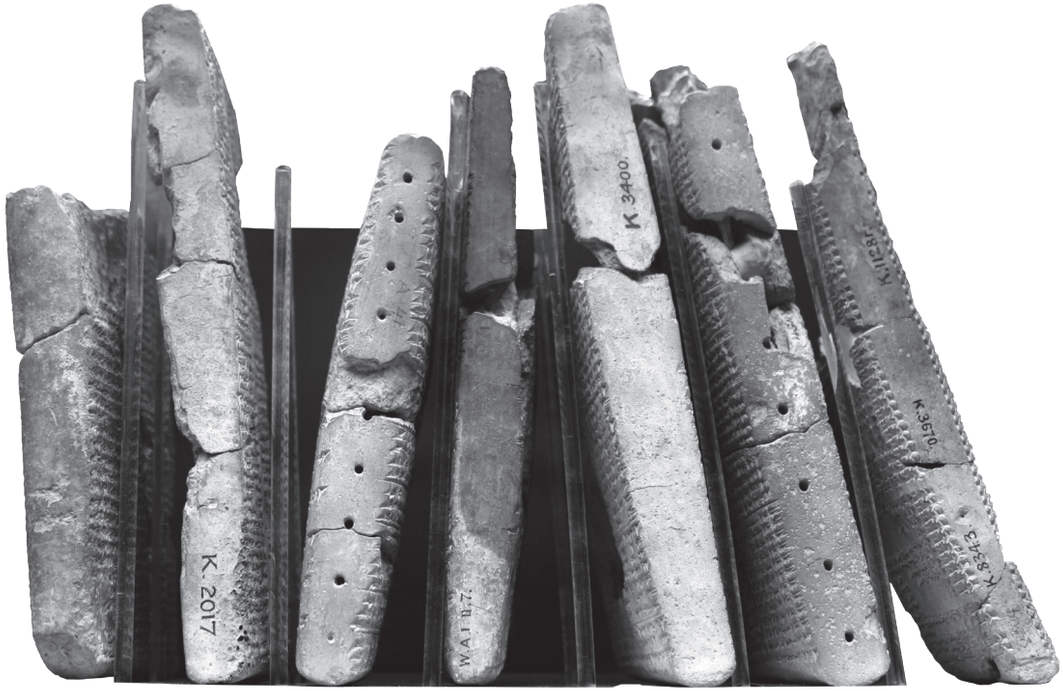


Voor een maximale snelheid moeten de pivotletters de stapels in twee helften verdelen.

Laten we aannemen dat de oorspronkelijke stapel boeken van A tot Z bevat. Een goede keus voor de eerste spil is dan waarschijnlijk de M. Dat levert twee nieuwe stapels op: A-L en M-Z. Als de A-L-stapel groter is, dan wordt die daarna gesplitst. Een goede spil voor A-L kan bijvoorbeeld de F zijn. Daarna zijn er drie stapels: A-E, F-L en M-Z. Vervolgens wordt M-Z gesplitst en ga zo maar door. Bij twintig boeken kunnen de uiteindelijke stapels dan A-C, D-E, F-L, M-R en S-Z zijn. Die stapels worden afzonderlijk geordend met Insertion Short waarna de boeken stapel na stapel naar de plank worden overgebracht.

De volledige Quicksort-algoritme kan dan als volgt worden genoteerd:

```
Neem een stapel ongesorteerde boeken als invoer.
Herhaal de volgende stappen:
   Kies de grootste stapel.
   Maak aan weerszijden ruimte voor stapels vrij.
   Kies een pivotletter.
   Herhaal de volgende stappen:
```



Soemerische kleitabletten.

1

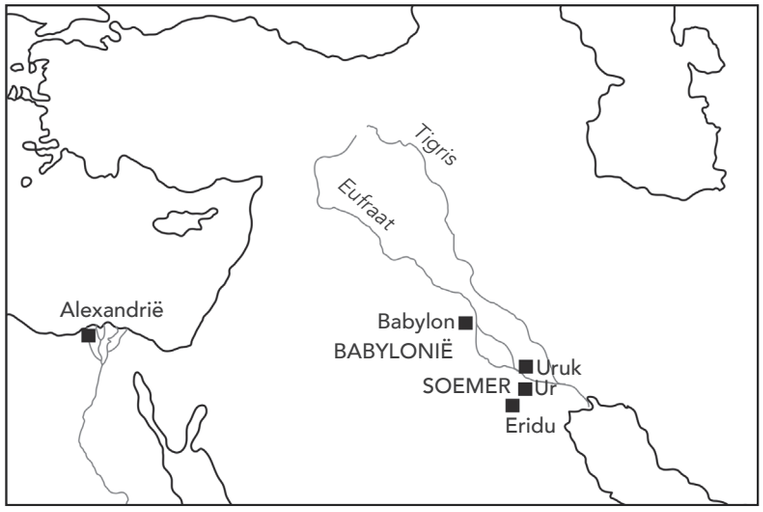
Algoritmen uit de oudheid

Beklim Uruks muur, Oersjanabi! Loop erop!
Bezie de fundamenten! Keur het metselwerk!
Zijn het geen harde in het vuur gebakken stenen?
Is die grondslag niet het werk van de zeven wijzen!
Een vierkante mijl is de stad, een vierkante mijl de boomgaarden,
een vierkante mijl zijn de kleiputten,
evenals het open terrein rond Isjtars tempel.
Drie vierkante mijlen en het open terrein vormen Uruk.

Auteur onbekend.
Het Gilgamesj-epos, circa 2000 v.Chr.

De woestijn heeft nagenoeg heel Uruk heroverd. De grote gebouwen liggen vrijwel geheel begraven onder zandophopingen, hun balken uiteengevallen. Hier en daar liggen gebakken kleistenen, door wind of archeologen blootgelegd. De verlaten ruïnes lijken onbelangrijk, vergeten, futiel. Niets wijst erop dat zevenduizend jaar geleden dit land de allerbelangrijkste plek op aarde was. Uruk, in het land van Soemer, was een van de eerste steden. Het was hier in Soemer, dat de beschaving werd geboren.

Soemer ligt in het zuidelijke deel van Mesopotamië. Dit gebied wordt begrensd door de rivieren Eufraat en Tigris, die vanuit de bergen van Turkije



Kaart van het Oude Mesopotamië en
de latere havenstad Alexandrië.

in het noorden stromen naar de Perzische Golf in het zuiden. Tegenwoordig grenzen hier Iran en Irak aan elkaar. Het klimaat is heet en droog en het land onherbergzaam, afgezien van de regelmatige bevloeiing van de vlakten door de rivier. Geholpen door irrigatie gedijde de vroege landbouw in het 'land tussen de rivieren'. Dankzij de daaruit voortvloeiende voedselrijkdom kon de beschaving hier voet aan de grond krijgen en opbloeien.

De koningen van Soemer bouwden grote steden – Eridu, Uruk, Kisj en Ur. Op het hoogtepunt woonden er in Uruk 60.000 mensen. Alle aspecten van het leven kwamen er voor – gezin en vrienden, handel en religie, politiek en oorlog. We weten dat omdat het schrift in Soemer werd uitgevonden, ongeveer vijfduizend jaar geleden.

Gegrift in klei

Het lijkt erop dat het schrift zich ontwikkelde vanuit eenvoudige markeringen afgedrukt in natte kleitabletten. Oorspronkelijk werden die gebruikt voor de boekhouding en het ruilen. Een tablet kon overeenkomen met een hoeveelheid graan of het aantal koppen van de veestapel. Na verloop van tijd begonnen de Soemeriërs meer ingewikkelde patronen op grotere stukken klei te schrijven. Naarmate de eeuwen voorbijgingen, veranderden eenvoudige pictogrammen naar een volledig ontwikkeld schriftstelsel. Dat stelsel noemen we nu spijkerschrift. De naam is afgeleid van de kenmerkende wigvormige markeringen die worden gevormd door het indrukken van een rietstengel in de natte klei. Symbolen bestonden uit geometrische rangschikkingen van die wigvormige indrukken. Deze inschriften werden bewaard door het drogen van de natte tabletten in de zon. Als je ze nu ziet, zijn de tabletten esthetisch aantrekkelijk – de indrukken dun en elegant, de symbolen regelmatig, de tekst keurig georganiseerd in rijen en kolommen.

De uitvinding van het schrijven moet die gemeenschappen hebben veranderd. De tabletten maakten communicatie over tijd en ruimte mogelijk. Brieven konden worden verstuurd. Afspraken konden voor toekomstige raadpleging worden vastgelegd. Schrijven maakte de soepele werking en uitbreiding van een civiele samenleving mogelijk.

Gedurende een millennium legde het spijkerschrift de Soemerische taal vast. In de 24e eeuw voor Christus werd Soemer binnengevallen door de legers van het Akkadische Rijk. De veroveraars pasten de Soemerische schriftmethoden aan hun eigen taal aan. Gedurende enige tijd werden beide talen op de kleitabletten gebruikt. Geleidelijk, naarmate de politieke macht verschoof, werd Akkadisch echter de exclusieve taal op de tabletten.

Het Akkadische Rijk duurde drie eeuwen voort. Toen het ten onder ging, vond er een heropleving van de bezette stadstaten plaats, totdat die later versmolten tot Assyrië in het noorden en Babylonië in het zuiden. In de achttiende eeuw voor Christus verenigde Hammoerabi, de koning van Babylon, de steden van Mesopotamië. De stad Babylon werd het onbetwiste centrum van de Mesopotamische cultuur. Onder toezicht van de koning breidde de stad zich uit en werden er indrukwekkende monumenten en fraaie tempels gebouwd. Babylonië werd een regionale supermacht. De Akkadische taal, en het spijkerschrift waarin die werd geschreven, werd de lingua franca voor internationale diplomatie binnen het Midden-Oosten.

Na meer dan een millennium te hebben geheerst, viel Babylonië, inmiddels geregeerd door de Meden, vrijwel zonder weerstand ten prooi aan Cyrus II de Grote, de koning van Perzië. Met zijn hoofdstad, Pasargadae, in het huidige Iran verzwolg het Perzische Rijk het Midden-Oosten. Het rijk van Cyrus strekte zich uit van de straat van Bosporus tot Midden-Pakistan en van de Zwarte Zee tot de Perzische Golf. Het Perzische spijkerschrift ging de administratie overheersen. Op het eerste gezicht vergelijkbaar met de Akkadische kleitabletten, hanteerden de nieuwe tabletten de Perzische taal en een volledig andere verzameling symbolen. Het gebruik van het

Frankfurt, Hamburg, Berlijn, München – een handelsreiziger kan steden in vele volgorden bezoeken. Algoritmen helpen de kortste route te vinden.

6

Spelden in hooibergen

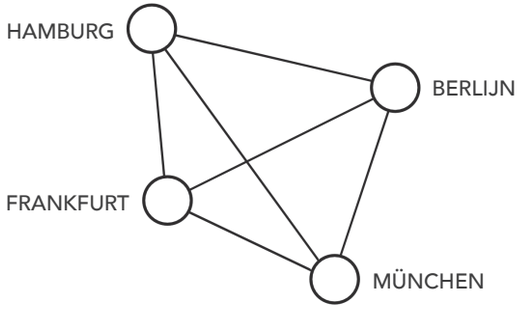
Die Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur.

Titel van een handleiding voor de succesvolle handelsreiziger uit 1832

In de jaren zeventig werd een van de grootste raadselen van de wiskunde geopenbaard tijdens een bijeenkomst van wiskundigen die de eigenschappen van algoritmen onderzochten. Ondanks een prijs van een miljoen dollar voor de oplossing is het raadsel nog steeds niet opgelost. In de kern van de zaak schuilt een schijnbaar onschuldig probleem.

Het handelsreizigersprobleem

Het handelsreizigersprobleem vraagt om de vaststelling van de kortste route langs een verzameling steden. De namen van de steden en de onderlinge afstanden zijn gegeven. Alle steden mogen slechts eenmaal worden bezocht. De steden kunnen in elke volgorde worden bezocht, zolang de reis maar begint en eindigt in de woonplaats van de handelsreiziger. De uitdaging is het vinden van de rondreis waarin de reiziger de kortste totale afstand aflegt.



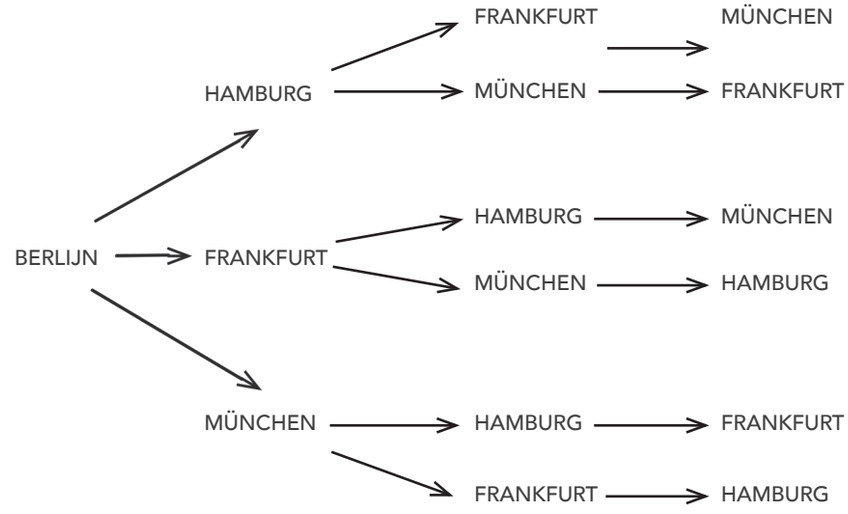
HET HANDELSREIZIGERSPROBLEEM:
vind de kortste route die elke stad eenmaal aandoet en weer thuiskomt.

Het handelsreizigersprobleem werd voor het eerst in de negentiende eeuw vermeld. Destijds was het een praktische zorg voor commerciële reizigers die tussen de steden van het continentale Europa reisden. Later werd het probleem geherformuleerd als een wiskundig speeltje door William Hamilton en Thomas Kirkman.

Laten we veronderstellen dat de handelsreiziger uit de titel in Berlijn woont en dat hij een bezoek moet afleggen aan Hamburg, Frankfurt en München. De eenvoudigste manier om de kortste route te vinden is door middel van een *uitputtende* zoektocht. Die rechtstreekse zoektocht, ofwel met brute kracht, berekent de lengte van elke mogelijke rondreis en kiest daarna de kortste. Een uitputtende-zoektochtalgoritme gaat als volgt:

```
Neem een verzameling stadsnamen als invoer.
Als er slechts één stad in de verzameling is,
dan lever een rondreis die alleen die stad bevat,
anders:
   Maak een lege lijst.
   Herhaal het volgende voor elke stad in de verzameling:
      Maak een kopie van de verzameling zonder de
         gekozen stad.
      Pas deze algoritme toe op deze kleinere verzameling.
      Plaats de gekozen stad aan het begin van alle
         geleverde rondreizen.
      Voeg de rondreizen toe aan de lijst.
   Lever alle gevonden rondreizen.
```


BOOMDIAGRAM VAN ALLE MOGELIJKE RONDREIZEN
Alle rondreizen eindigen in Berlijn (niet aangegeven).



Het starten met de verzameling van alle steden met uitzondering van de thuisstad vormt de invoer van de algoritme. De thuisstad is het bekende begin- en eindpunt van elke rondreis dus hoeft die niet in de zoektocht worden meegenomen. De algoritme maakt een boom van stadsbezoeken uit de invoerverzameling. De algoritme berust op twee mechanismen. Allereerst gebruikt die herhaling – de algoritme kiest elke stad in de invoerverzameling, de een na de ander, als de volgende die wordt bezocht. Ten tweede gebruikt hij recursie (zie hoofdstuk 1). Voor elke stad roept de algoritme een *instantie*, een kopie van zichzelf, op. Een instantie van een algoritme is een andere, afzonderlijke uitvoering van de algoritme die werkt op zijn eigen gegevens. In dit geval levert elke instantie een nieuwe sub-boom in het diagram. Nadat elke stad is bezocht, is de verzameling van steden ingevoerd naar de volgende instantie van de algoritme gereduceerd. Dus moeten de instanties steeds minder steden verwerken totdat er slechts eentje in de verzameling achterblijft. Als dat gebeurt, eindigt de instantie met een boomblad, een knoop zonder verdere vertakkingen, en levert die een rondreis die slechts een enkele stad bevat. De vorige instanties van de algoritme nemen die uitvoer en voegen daar de steden in omgekeerde volgorde aan toe. Op die manier ontrolt de algoritme zich, rondreizen

Algoritmen regeren steeds meer ons leven. Ze bepalen online welk nieuws we zien en beïnvloeden welke producten we kopen. Ze doen suggesties voor potentiële datingpartners. Ze creëren en vernietigen hele bedrijfstakken en beïnvloeden verkiezingen. Hoewel algoritmen overal om ons heen worden gebruikt, weten maar weinigen hoe ze precies werken, of wie ze heeft ontwikkeld.

In *Als dit, dan dat* beschrijft informaticus Chris Bleakley op een heldere manier hoe een reeks simpele stappen complexe problemen kan oplossen. Hij duikt in de werking en het ontstaan van een groot scala aan algoritmen: van eenvoudige sorteeralgoritmen tot algoritmen voor weersvoorspellingen, routeplanningen, cryptovaluta, zoekmachines en kunstmatige intelligentie.

Dit boek neemt je mee naar de eerste algoritmen van de oude Grieken en de opkomst van de computer. Het beschrijft hoe de Britse wiskundige Alan Turing in de Tweede Wereldoorlog met zijn algoritme de Enigmacode van de nazi's kraakte, en hoe de Nederlandse informaticus Edsger Dijkstra een algoritme ontwikkelde dat aan de basis staat van routeplanners.

Chris Bleakley studeerde informatica aan Queen's University in Belfast en promoveerde in de elektrotechniek aan Dublin City University. Na het bekleden van diverse rollen in technologische bedrijven is hij tegenwoordig hoofd van de School of Computer Science aan University College Dublin. Hier houdt hij zich bezig met het ontwikkelen van nieuwe algoritmen voor het analyseren van sensorgegevens.

In samenwerking met NewScientist

9 789085 718079