Paul Curzon schreef dit verhaal in 2014, in het kader van Computer Science for fun (www.cs4fn.org). Hij werd daarin gesteund door het cs4fn-team, in het bijzonder door Jonathan Black. Curzon kreeg ook waardevolle opmerkingen van Nicola Plant en Zali Collymore-Hussain.
Bezoek de webpagina cs4fn.org/teachers voor meer cs4fn-leermiddelen voor scholen. Cursussen en leermiddelen voor leerkrachten, waaronder klasactiviteiten en dia's met een directe link naar dit verhaal, vind je op teachinglondoncomputing.org (een gezamenlijk project tussen Queen Mary University of London en King's College London, gefinancierd door de Greater London Assembly).

Vertaald en bijgewerkt door Bjarne Van de Velde en Natacha Gesquière van Dwengo vzw in het kader van het project AI in de Zorg (aipopschool.be/zorg) van AI Op School. Het past ook binnen Computationeel denken (aiopschool.be/computationeeldenken).

Zoektocht naar spraak

Hulp bieden aan mensen met het locked-in syndroom
Computationeel denken

Wat is computationeel denken?
Hoe vinden computers dingen?
Hoe kunnen we zien welk algoritme het beste is?

Op zoek naar een manier om te spreken

Het locked-in syndroom is een van de ergste medische aandoeningen. Je bent volledig verlamd, behalve dat je misschien nog kunt knipperen met een oog. Je intelligente geest zit opgesloten in een nutteloos lichaam: je kan alles voelen, maar niet communiceren.
Het kan iedereen overkomen, uit het niets, als gevolg van een beroerte.

Als je mensen met het locked-in syndroom zou willen helpen, word je dan best arts of verpleegkundige? Of kan je als computerwetenschapper ook helpen?

Er is geen remedie voor het locked-in syndroom. De medische wereld kan dus niet veel meer doen dan het de patiënten zo comfortabel mogelijk maken. Een grote uitdaging bestaat erin mensen met het locked-in syndroom te helpen 'praten'. Het lijkt voor de hand liggend hoe computerwetenschappers daarbij kunnen helpen: ze kunnen een of andere nieuwe technologie ontwerpen. Met wat computationeel denken kan echter een veel beter antwoord geboden worden dan enkel 'het voorzien van technologie'.

'Vlinders in een duikerpak' (origineel 'Le scaphandre et le papillon') is de autobiografie van Jean-Dominique Bauby, een boek dat hij schreef nadat hij, volledig verlamd, wakker werd in een ziekenhuisbed. Hij kon ook niet praten. Het enige wat hij nog kon, was knipperen met één oog.

In het boek beschrijft hij het leven met het locked-in syndroom. Dat betekent dat hij toch op een of andere manier kon communiceren waardoor hij het boek kon schrijven, en waardoor hij met de dokters en verpleegkundigen, met vrienden en familie kon praten. Hij gebruikte echter geen technologie; hij had enkel een helper die met pen en papier de woorden die hij 'spelde', opschreef. Hoe deed hij dat?

Beeld je in dat je zelf in zo'n toestand wakker wordt in een ziekenhuisbed.
Hoe zou je kunnen communiceren?
Hoe zou je een heel boek schrijven?
Je hebt alleen een helper die met pen en papier jouw 'woorden' opschrijft. En het enige wat je kunt doen, is met één oog knipperen.
Kun je een manier bedenken om te communiceren?

Zo eenvoudig als A, B, C

Je zal genoodzaakt zijn om een manier overeen te komen om het knipperen met een oog in letters om te zetten. Je eerste idee zou kunnen zijn dat één keer knipperen 'A' betekent, 2 keer knipperen 'B', enzovoort. De helper hoeft alleen maar het aantal keer dat geknipperd wordt te tellen en de bijbehorende letter op te schrijven.

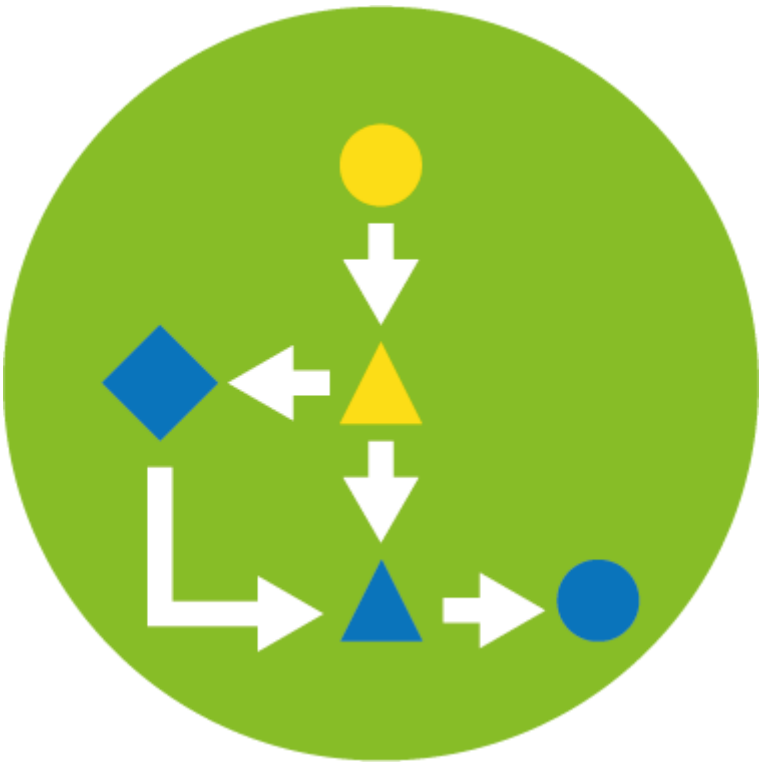
Bij het bedenken van deze oplossing denk je **computationeel**: het soort probleemoplossend denken dat computerwetenschappers gebruiken. Meer specifiek gaat het hier om **algoritmisch denken**. Een computerwetenschapper noemt de afgesproken manier om te communiceren een **algoritme**: een reeks stappen die in een bepaalde volgorde moeten worden uitgevoerd om een bepaald doel te bereiken (hier om letters en woorden te communiceren).
Algoritmisch denken gaat over het bedenken van algoritmes om problemen op te lossen.

Het mooie van algoritmes is dat de stappen kunnen worden gevolgd zonder dat de betrokkenen enig begrip hebben van wat ze doen.
Met het hierboven voorgestelde algoritme zouden de helpers waarschijnlijk wel weten wat ze doen en waarom, maar zelfs als de helpers dit niet zouden weten, zou er nog steeds geschreven worden. Immers, het enige wat een helper hoeft te doen is het aantal keer knipperen tellen en de letters op te schrijven. Strikt genomen zou je de helpers zelfs een tabel kunnen geven waarin ze de letters kunnen opzoeken, zodat ze helemaal niet meer hoeven na te denken.
Het mooie van algoritmes is dat mensen op deze manier in staat zijn om dingen 'op automatisme' te doen, en dat betekent dus dat ook computers blindelings de instructies kunnen volgen.

Het voorgestelde algoritme om te communiceren bestaat eigenlijk uit twee delen. Er is het deel dat Bauby moet volgen (het juiste aantal keer knipperen) en er is het deel voor de helper (het aantal keer dat Bauby knippert tellen en de bijbehorende letter noteren op het moment dat het knipperen stopt).
Computerwetenschappers noemen dit soort 'afspraak' waarmee informatie tussen twee mensen (zoals hier) of tussen computers wordt doorgegeven, een **protocol**. Als beide personen hun deel van het protocol volgen, zullen de woorden die Bauby denkt, op het stuk papier terechtkomen. Als een van beiden een fout maakt - bijvoorbeeld de tel kwijtraakt en dus het protocol niet volgt - dan komt het bericht er niet correct door. Het mooie van computers is dat ze zulke dingen niet verkeerd doen: ze volgen de gegeven instructies exact op, telkens weer.

Algoritmisch denken is een soort van probleemoplossend denken waarbij je een manier bedenkt om tot een oplossing te komen in de vorm van stappen die iemand anders (en zeker ook een computer) kan volgen. Bovendien is het niet zo dat je het voorgestelde algoritme voor Bauby slechts één keer kunt gebruiken om uit te vissen wat Bauby probeert te zeggen, het is een manier waarop je telkens weer kunt uitzoeken wat hij wilt zeggen.

Het voorgestelde algoritme is vermoedelijk wel vrij traag. Zou er geen snellere manier zijn?
Nadenken over betere oplossingen is ook een onderdeel van algoritmisch denken. Een algoritme moet niet alleen **effectief** zijn (d.w.z. foutloos tot een oplossing leiden), maar liefst ook **efficiënt** (m.a.w. snel genoeg).



Hoe deed Bauby het?

Bauby had een betere manier, een sneller algoritme. Vergeet niet dat de helper wél kan spreken; daar heeft Bauby gebruik van gemaakt.

Bij het algoritme dat Bauby gebruikte, las de helper het alfabet hardop voor: "A ... B ... C ...". Als de letter waar Bauby aan dacht, werd uitgesproken, dan knipperde hij met zijn ogen. De helper schreef die letter op en begon toen opnieuw, letter voor letter.

Probeer deze methode eens uit met een vriend: communiceer je initialen op die manier. Beeld je in dat dit de enige manier is om met iemand te praten. Hopelijk heet je niet Zebedee Zacharius Zog of Zara Zootle!
Als je het algoritme een paar keer uitgetest hebt, dan heb je je misschien gerealiseerd dat er nog steeds problemen zijn die je moet aanpakken om het algoritme echt te laten werken.

Kan je manieren bedenken om het algoritme te verbeteren?

Je merkte misschien op dat er naast de 26 letters nog andere zaken zijn waarmee je moet werken, zoals spaties, cijfers, leestekens, en witruimtes. Je moet deze karakters toevoegen aan de lijst met letters die de helper telkens opsomt.
En wat als de patiënt per ongeluk knippert? Je hebt een manier nodig om te zeggen : "Negeer die laatste knipoog en som de letters opnieuw op van in het begin". Je zou bv. kunnen afspreken dat er in dat geval twee keer snel geknipperd wordt.

Algoritmisch denken gaat over het nadenken over al die details en het vinden van oplossingen. Het gaat erom dat je beseft dat er verschillende manieren kunnen zijn om dingen te doen, en dat je dan de beste manier voor de situatie kiest.

Merk ook op dat een van de problemen gaat over wat mensen doen. In theorie werkt de voorgestelde oplossing: knipper gewoon op het juiste moment!

Je zou er kunnen van uitgaan dat de mensen gewoon het juiste moeten doen en dat het hun fout is als het verkeerd loopt. Maar in de praktijk is het nu eenmaal zo dat mensen soms op het verkeerde moment knipperen.
Het is beter als je een oplossing voor het probleem bedenkt die voor mensen werkt. Je probeert tenslotte een persoon te helpen!
Computationeel denken gaat ook over het **begrijpen van mensen**.

Het beter doen

Soms kan je halverwege een woord al raden wat het is. Als je 'a-n-t-i-l' hebt, zou het het woord 'antiloop' een goede gok zijn. Als je de regels aanpast zodat de helper zulke gissingen kan maken, dan kan dat de dingen versnellen. Je hebt dan wel een manier nodig waarop de patiënt kan aangeven dat het een foute gok is. Misschien kan de regel zijn: knipperen als het woord juist geraden is en niets doen als het fout is.

Denk nu eens aan 'woordsuggestie' op je smartphone. Het algoritme dat je telefoon daarvoor gebruikt, werkt eigenlijk op dezelfde manier als de helpers van Bauby. Misschien had je zelf dat verband al gelegd.
Als je dat deed, dan gebruikte je een andere computationele denkvaardigheid, namelijk ***patroonherkenning****.*

Vaak blijkt een probleem in wezen hetzelfde te zijn als een probleem dat je al in een andere situatie hebt gezien. Als je al een oplossing hebt voor dat andere probleem, dan kan je die oplossing in de nieuwe situatie gebruiken, al dan niet met een kleine aanpassing. Eigenlijk beschikte je over een soort algemene oplossing, die je in meerdere situaties kunt aanwenden. Met computationeel denken probeer je tot dit soort algemene oplossingen te komen.
Het probleem dat een smartphone heeft om uit te zoeken welke woorden worden getypt, is hetzelfde als dat van een helper die het woord moet vinden waar iemand met het locked-in-syndroom aan denkt. Zodra je dit beseft, kan je elke oplossing die je voor het ene bedenkt ook voor het andere gebruiken.
Bauby's helpers gebruikten dus eigenlijk een vorm van woordsuggestie.

Bauby besefte dat er bovendien nog een manier was om het ABC-algoritme te verbeteren. Vóór hij in het ziekenhuisbed was beland, was hij hoofdredacteur geweest van het Franse vrouwenblad *Elle*. Hij wist dus veel over taal. Het was hem dan ook bekend dat sommige letters vaker voorkomen dan andere. 'E' is bv. de meest voorkomende letter (zowel in het Nederlands, het Engels, als het Frans).
Hij liet daarom de helper de letters voorlezen in volgorde van dalende frequentie. In het Nederlands is de volgorde 'E, N, A, T, ..., Q', in het Engels 'E, T, A, O, ..., Z' en in het Frans is het 'E, S, A, R, ..., W'. Bauby sprak Frans, dus hij gebruikte de Franse volgorde. Op die manier bereikte de helper sneller de meer gebruikte letters.

Door de eeuwen heen werd een soortgelijke truc gebruikt om geheime codes te kraken. Het algoritme om letterfrequenties te gebruiken werd meer dan 1000 jaar geleden uitgevonden door moslim- geleerden. Mary Queen of Scots werd eigenlijk onthoofd omdat Sir Francis Walsingham, de hoofdspion van Koningin Elizabeth I, beter was in dit dit soort computationeel denken dan

zij, maar dat is een ander verhaal.

Bauby's idee om frequentieanalyse te gebruiken is een ander voorbeeld van patroonherkenning. Zodra je hebt begrepen dat het kraken van codes en het raden van letters vergelijkbare problemen zijn, besef je dat de oplossing die voor het ene probleem is bedacht, nl. frequentieanalyse, ook bruikbaar is voor het andere probleem.

Hoe snel is het algoritme?

Een voor de hand liggende vraag is hoe snel Bauby's algoritme nu eigenlijk is. Hoelang duurde het om dat boek te schrijven? Is er geen sneller algoritme dat het hem gemakkelijker had kunnen maken om het boek te schrijven?

Op welke manier kan je meten hoe snel een algoritme is? Je zou het experimenteel kunnen doen door te timen hoelang het duurt om een bepaalde passage te communiceren. Je kan dat dan een groot aantal keer laten doen door verschillende mensen en de gemiddelde tijd die het vergde, bepalen.
Als je die gemiddelde tijd kent voor verschillende algoritmes, weet je welk algoritme het snelste en dus het meest efficiënte is.

Maar zo'n experiment zou veel tijd en moeite kosten. Er is een betere manier: **logisch denken**. Je kan immers, aan de hand van een aantal eenvoudige berekeningen, de tijd inschatten die nodig is om het boek te schrijven. Eens je weet hoe lang het duurt om één letter te zeggen en je weet hoeveel letters van het alfabet de helper moet opsommen om het boek neer te schrijven, dan kan je daarmee de nodige tijd berekenen.

De eigenlijke berekening hoeft zelfs niet te gebeuren om algoritmes met elkaar te vergelijken. De algoritmes verschillen immers in nodige tijd omdat het aantal letters dat de helper moet opsommen per algoritme verschilt. Dus om te weten wat het snelste algoritme is, moet je enkel de hoeveelheid letters die de helper moet overlopen, bepalen.
Je hebt een ander concept van computationeel denken gebruikt: **abstractie**. Door te abstraheren kan je een probleem vereenvoudigen. Je verbergt of negeert sommige details die niet relevant zijn om het probleem op te lossen. Hier gebruik je 'aantal gezegde letters' als een abstractie van 'nodige tijd'.

Hoe kom je er nu achter hoeveel letters er opgesomd moeten worden? Er zijn verschillende vragen die je kan stellen. Wat is het beste geval? M.a.w. wat is het kleinste aantal letters dat de helper zou moeten zeggen om het boek te schrijven? Hoeveel letters zou de helper moeten opsommen in het slechtste geval? Tot slot kunnen we kijken naar het gemiddeld aantal letters; dat geeft ons een realistische schatting van hoeveel werk het eigenlijk zou kosten.

Het beste en slechtste geval

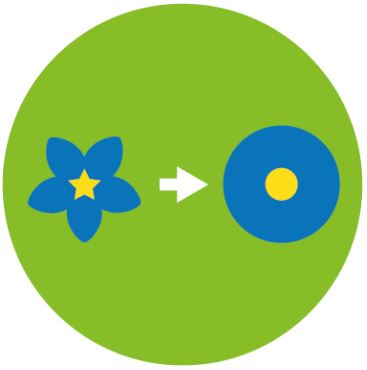
We beperken ons, bij wijze van redenering, tot het communiceren van de letters van het alfabet, dus zonder cijfers en leestekens. We zullen het eenvoudige algoritme van de helper die A, B, C zegt, analyseren ...

In het beste geval zou het hele boek enkel A's bevatten: "AAAAAAAA" (mogelijks een uiting van de pijn die hij ondervindt). Om een enkele letter 'A' te communiceren, zeggen we slechts één letter, nl. 'A', dus na één opgesomde letter hebben we het antwoord. Vermenigvuldig dat met het aantal letters in het boek en we hebben de beste tijd voor het schrijven van het hele boek.
In het slechtste geval, misschien het vertellen van een verhaal waarin iemand de hele tijd snurkt, "ZZZZZZ", duurt het telkens zesentwintig opgesomde letters om een letter te verkrijgen.
Dat geeft ons de grenzen van het communiceren: minder dan 1 kan niet voorkomen en meer dan 26 ook niet.

Een betere schatting zou het gemiddelde aantal opgesomde letters per letter zijn. Gemiddeld over het hele boek worden er 13 à 14 vragen per gedicteerde letter gesteld. Vermenigvuldig het aantal letters in het boek met 13,5 en je hebt een schatting van hoeveel werk er is gedaan om het te schrijven.
Vermenigvuldig dat met de gemiddelde tijd die de helper nodig heeft om een letter te zeggen en je hebt de tijd die het vergt om het boek te schrijven.

Bauby's aanpassing om eerst naar de courante letters te vragen, verbetert de dingen een beetje; misschien komt het gemiddeld neer op 9 of 10 gesproken letters. Het is dus een verbetering, maar er zal nog steeds een letter zijn waarvoor er 26 letters opgesomd moeten worden.
Zoals elke computerwetenschapper weet, kunnen we het veel beter doen. Het is gegarandeerd mogelijk om elke letter te vinden na hoogstens 5 letters op te sommen!

Kan je uitzoeken welke 5 vragen je moet stellen?



Doe het in 5

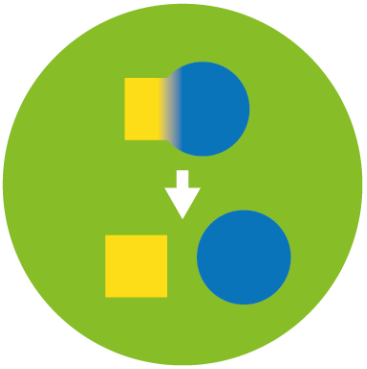
Of je nu tot het antwoord bent gekomen of niet, ik garandeer je dat je weet wat de juiste vraag is. Laat ons eerst naar een ander probleem kijken.
In het spel 'Wie Ben Ik' denkt iemand aan een beroemd persoon en iemand anders moet proberen te raden aan wie door vragen te stellen. Op die vragen mag alleen maar met ja of nee geantwoord worden.

Speel het spel met een vriend en denk na over het soort vragen dat je stelt terwijl je dat doet.

Laten we eens kijken hoe zo'n wedstrijd zou kunnen verlopen.

"Ben ik een vrouw?"
Nee
"Leef ik nog?"
Nee
"Ben ik een filmster?"
Nee
"Kom ik uit Groot-Brittannië?"
Nee
"Kom ik uit Amerika?"
Nee
"Kom ik uit Azië?"
Ja
"Kom ik uit India?"
Ja
"Ben ik een politicus?"
Ja
"Ben ik Ghandi?"
Ja

Naar alle waarschijnlijkheid stel jij soortgelijke vragen wanneer je het spel speelt. Je begint bijna zeker niet met vragen als "Ben ik Adele," of "Ben ik Usain Bolt", of "Ben ik de koningin?" Met zo'n vragen zou je het antwoord bijna nooit kunnen vinden. Je stelt dat soort specifieke vragen pas aan het einde als je er vrij zeker van bent dat je weet wie het is (zoals in het bovenstaande voorbeeld).
In plaats daarvan heb je waarschijnlijk een vraag gesteld als: "Ben ik een man?".
Waarom is dat een goede eerste vraag? Omdat het quasi de helft van de mogelijkheden uitsluit, wat het antwoord ook is. Als je vraagt "Is het Adele?", dan heb je geluk als je gelijk hebt. Maar als je het mis hebt - wat waarschijnlijker is - sluit je slechts één persoon uit. Je zou al heel veel geluk moeten hebben om het op die manier goed te doen.
Dus het geheim van het spelen van 'Wie Ben Ik' is om vragen te stellen die de helft van de mensen elke keer uitsluiten.



Hoe goed is dat?

Hoe goed is dat? Laten we aannemen dat ik aan het begin aan een miljoen mensen denk. Als ik bij elke vraag de helft van de mensen uitsluit, hoeveel vragen zijn er dan nodig? Na één vraag zijn we gezakt tot 500 000 mensen, na twee vragen tot 250 000, daarna tot 125 000 mensen, ongeveer 64 000 mensen (een beetje vereenvoudigen om de aantallen gemakkelijker te maken), 32 000 mensen, 16 000, 8000 mensen, 4000, 2000, 1000 ... Na tien vragen zijn er nog maar 1000 mensen over van het oorspronkelijke miljoen. Ga door ... Er zijn er 500 over na een andere vraag; vervolgens 250, 125, ongeveer 64, 32, 16, 8, 4, 2 en na de twintigste vraag is er nog maar één persoon over. Als je perfecte halveringsvragen stelt, dan win je gegarandeerd.

Met de juiste vragen zijn er in het slechtste geval slechts 20 vragen nodig om de persoon waar ik aan denk, te vinden uit een miljoen mogelijkheden. Vergelijk dat met de stelling dat het ons 13 vragen (en in het slechtste geval 26) kost om één letter uit de 26 letters van het alfabet te vinden. Ja/nee is niet anders dan knipperen/niet-knipperen. Een vraag zoals “Is het A?” of “Is het B?” is een soortgelijke vraag als “Ben ik Nelson Mandela?”, of “Ben ik Mickey Mouse?”. Het komt erop neer dat je uit veel dingen dat ene ding probeert te raden waar ik aan denk. Het is eigenlijk hetzelfde probleem!

Het idee van '**het transformeren van problemen**' komt hier weer om de hoek kijken. Aangezien het hetzelfde probleem is, zal de strategie van de halveringsoplossing ons zeker een betere oplossing geven dan degene die we tot nu toe hebben bedacht. Wat is het equivalent van de halveringsoplossing voor letters van het alfabet? We moeten het alfabet bij elke gestelde vraag halveren. De voor de hand liggende eerste vraag is: “Is het vóór N?”. De volgende vraag hangt af van het antwoord op de eerste. Als het antwoord “Ja” was, dan vragen we vervolgens: “Is het vóór F?”. Als het antwoord “Nee” was, vragen we “Is het vóór T?, enz. Op die manier zijn we er zeker van dat we elke letter waar de persoon aan denkt, achterhalen in slechts 5 vragen.

We kunnen de dingen zelfs nog verbeteren m.b.v. de frequentie-analysetruc. Met slechts 26 letters zouden we er bijvoorbeeld kunnen voor zorgen dat we de letter ‘E’ in slechts 3 vragen verkrijgen. We kunnen ook nog steeds de woordsuggestie-truc gebruiken om woorden te raden die al gedeeltelijk zijn voltooid. Al die oplossingen zijn hier nog steeds van toepassing.

Zoekalgoritmes

*Onze oplossing kon worden overgedragen omdat het probleem in wezen hetzelfde was. Het is een 'zoekprobleem': zoek, gegeven een reeks dingen, een bepaald ding dat we zoeken. De oplossingen voor dit probleem zijn 'zoekalgoritmes'. Het zijn manieren om gegarandeerd dingen te vinden. Onze eerste benadering, nl. het één voor één controleren van elke mogelijkheid ("Is het A? Is het B? ... Is het Adèle? Is het James Bond? …"), is een algoritme dat '**lineair zoeken**' wordt genoemd.*

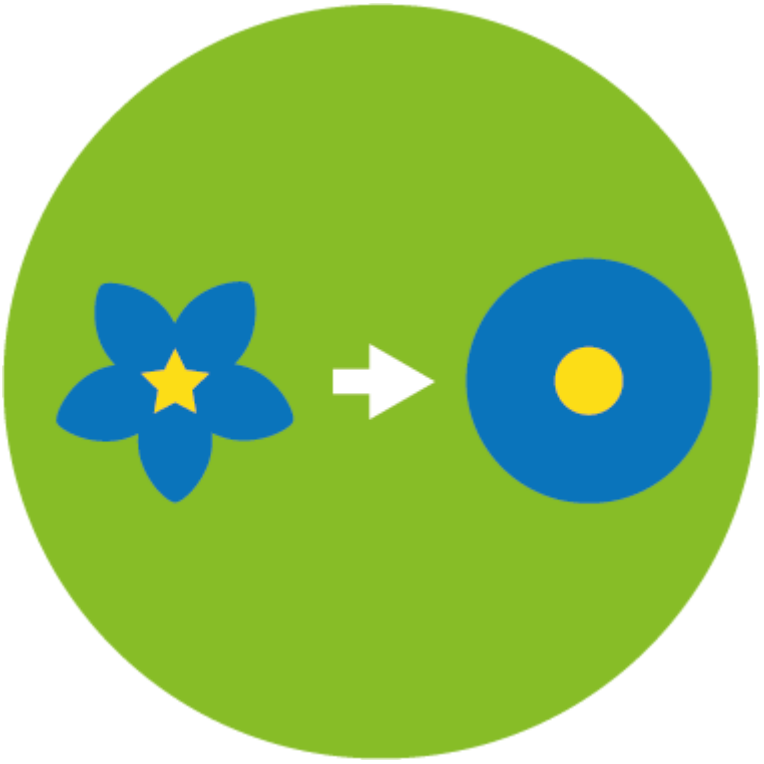
Soms is dat het beste wat je kunt doen. Als je bijvoorbeeld getuige was van een overval en de politie laat alle verdachten op een rij staan, dan kan je niets beters doen dan lineair zoeken: controleer een voor een elk gezicht, totdat je de persoon in de rij ziet die het heeft gedaan! Lineair zoeken werkt goed als er geen volgorde is voor de dingen die je doorzoekt. Als je op zoek bent naar een trui die eender waar in je ladekast kan liggen, begin dan bovenaan en controleer ze één voor één.

Het andere algoritme betrof het vinden van halveringsvragen: "Is het vóór de N? Is het een vrouw?". Het vinden van halveringsvragen is een algemene probleemoplossende strategie genaamd '**Verdeel en heers**'. Als je een verdeel-en-heersoplossing voor een probleem kunt bedenken, zal je de oplossing snel vinden, want herhaalde halvering brengt je heel snel tot één antwoord, veel sneller dan via '*lineair zoeken*'.

'**Binair zoeken**' is het meest eenvoudige verdeel-en-heerszoek-algoritme is. Stel je voor dat je alle dingen waar je doorheen zoekt in volgorde bekijkt. Binair zoeken houdt in dat je naar het midden gaat en controleert of het ding dat je zoekt ervoor of erna komt. Je gooit dan de overbodige helft weg en doet hetzelfde opnieuw op wat er overblijft. Dat blijf je doen totdat er maar één ding overblijft, het ding dat je zocht. Dat leunt waarschijnlijk aan bij wat je doet als je een bepaalde naam wilt opzoeken in een grote papieren telefoongids. Je zou zeker niet beginnen op de eerste pagina en elke naam achtereenvolgens controleren totdat je de naam vindt die je zoekt!

Er zijn veel meer zoekalgoritmes dan alleen deze twee. Hoe zoekt Google bijvoorbeeld in fracties van een seconde door elke webpagina op de planeet? Google heeft daarvoor toch een meer geschikt algoritme nodig!

Zoekalgoritmes maken gebruik van een andere vorm van ***abstractie***. Door een probleem slechts te bekijken als een zoekprobleem en de details die specifiek zijn voor het probleem eruit te abstraheren, kan je een zoekalgoritme als kant-en-klare oplossing gebruiken.
Als we op een andere manier nadenken over het opsommen van de letters en inzien dat het eenzelfde probleem is als een dat ze met de strategie van de 20 vragen kunnen winnen, dan kunnen we die oplossing onder de vorm van verdeel-en-heers ***veralgemenen***. We hebben een algemene strategie die ook voor andere problemen werkt.



Algoritmisch denken eerst

Had Bauby de helper dan niet moeten aanzetten tot het stellen van halveringsvragen? Denk er eens over na: in het slechtste geval 5 vragen in plaats van gemiddeld 13 of 14, en dat voor alle letters in zijn boek. Dat was niet alleen handig geweest voor het boek, maar ook om te praten met zijn vrienden en familie, de artsen en verpleegkundigen. Als hij maar wat computationeel denken had gekend, zou zijn leven veel makkelijker zijn geweest!

Het is eigenlijk wel opmerkelijk dat we nog helemaal niet naar technologie hebben gekeken, het ging enkel over het opsommen van letters en over twee mensen die met elkaar 'praten'.

Het punt is dat welke technologie we ook gebruiken, er onderliggend altijd een zoekalgoritme nodig is. Kies het verkeerde algoritme en de communicatie zal nog steeds traag zijn, ongeacht hoe goed de technologie ook is.
Als we ons niet eerst op de algoritmes hadden gefocust, dan hadden we een frustrerend traag systeem kunnen bedenken.

Computerwetenschappen gaat niet alleen over de technologie, het gaat om het computationeel denken dat gepaard gaat met het bedenken van goede oplossingen.

Mensen begrijpen komt eerst

Dus een beetje meer computationeel denken had Bauby's leven kunnen verbeteren. Maar wacht eens even. Is dat wel zeker?
Misschien hadden we er met onze oplossing voor gezorgd dat zijn boek nooit af was geraakt en dat zijn leven nog meer een hel was. We zijn niet vertrokken van technologie, wel van computerwetenschappen. Maar misschien hadden we met de persoon moeten beginnen. Hebben we wel met de juiste aspecten van het probleem rekening gehouden?

Wij zijn op zoek gegaan naar het snelste algoritme. Als maatstaf voor die snelheid gebruikten we het aantal vragen die de helper moet stellen, de gebruikte 'abstractie'. De taak van de helper is misschien een vervelende taak, maar het is geen moeilijke.

Maar wat als knipperen een grote inspanning was voor Bauby? Zijn oplossing hield in dat hij slechts één keer per letter knipperde. Ons verdeel-en-heersalgoritme vereist dat hij 5 keer knippert. En dat voor een heel boek. We hadden het 5 keer moeilijker kunnen maken voor hem.

Het is echter mogelijk dat knipperen gemakkelijk is en ons algoritme dus beter. We weten het antwoord echter niet, omdat we de vraag niet hebben gesteld. We hadden dat eerst moeten achterhalen. We hadden met de persoon moeten beginnen.

Bovendien is zijn oplossing gemakkelijk te begrijpen voor iedereen die er binnenloopt. De onze is complexer en heeft misschien wat uitleg nodig voordat de bezoeker het begrijpt, en Bauby zal niet degene zijn die de uitleg doet. Denken aan mensen is belangrijk!

Het werkte voor hem

Eén ding is zeker over Bauby's oplossing: ze werkte voor hem. Hij schreef zo immers een heel boek.
Nu we een goede manier, een goed algoritme, hebben uitgewerkt, kunnen we nadenken over hoe we het kunnen automatiseren met geschikte technologie. We zouden een oogvolgsysteem kunnen bouwen dat het aantal keer knipperen detecteert, of we zouden misschien kunnen werken met een elektrode-dop in de hersenen die kan oppikken of hij nu 'ja' of 'nee' denkt. Er zou geen enkele helper meer nodig zijn in de kamer.

Misschien deden de helpers ook meer dan alleen zijn woorden opschrijven. Misschien openden ze de gordijnen, spraken ze met hem over de wereld buiten het ziekenhuis, of zorgden ze gewoon voor wat dagelijkse menselijke warmte. Misschien ging het eigenlijk niet om het boek, maar gaf het schrijven van het boek hem een excuus om een persoon bij zich te hebben om de hele tijd mee te communiceren.

Het communicatie-algoritme zou dan niet gaan over het schrijven van het boek, maar over een diepe behoefte aan directe communicatie met een persoon. Vervang de mens door technologie en misschien heb je het ding vervangen dat hem in leven hield.

Anderzijds, als hij eenmaal in staat is om met een computer te praten, kan hij vanuit zijn ziekenhuisbed in de virtuele wereld komen, vrienden e-mailen, tweeten, een Facebook-pagina bijhouden, een avatar besturen. Misschien zou technologie het toch beter gemaakt hebben. Nogmaals, we moeten uitzoeken wat hij echt wil.

In een situatie zoals deze die heel extreem is voor de gebruiker, is het belangrijk dat die gebruiker echt overal bij betrokken is. In feite is dit zogenaamd '*user-centered design*' altijd beter, bij het ontwerpen van eender welk systeem voor mensen, en niet alleen in extreme situaties. Uiteindelijk zijn het de gebruikers die wat er beschikbaar is, voor hen laten werken, niet alleen technisch, maar ook emotioneel en sociaal. Anders zouden we een 'oplossing' kunnen bedenken die in theorie prachtig is, maar in praktijk 'een hel op aarde'.
Computerwetenschappers moeten aan veel denken, niet enkel aan computers.

De computerwetenschap

Zoekalgoritmes

Wanneer een ‘zoekalgoritme’ iets heeft om te zoeken (bekend als de 'sleutel'), garandeert het dat het dat zal vinden als het er is. De sleutel kan een letter zijn waar iemand aan denkt, een nummer in een lijst, de webpagina van een filmster of een personeelsdossier.

Lineair zoeken

Een eenvoudig zoekalgoritme wordt 'lineair zoeken' genoemd. Het gaat over het op een rij zetten van alle dingen die je doorzoekt en ze één voor één van het ene uiteinde van de rij naar het andere uiteinde te controleren.

Als je het ding vindt dat je zoekt, kan je stoppen. Als je de positie ervan markeert, kan je er direct naar teruggaan. Als je aan het einde komt zonder de sleutel te vinden, weet je zeker dat het gezochte ding er helemaal niet is.

Binair zoeken

Een snellere manier van zoeken wordt 'binair zoeken' genoemd. Het gaat over zaken die op een rij opgesteld zijn in een bekende volgorde, zoals numerieke volgorde of alfabetische volgorde. Dat stelt ons in staat om controles uit te voeren die de helft van de lijst bij elke stap uitsluiten. We controleren het middelste element of de middelste twee elementen in de rij. Als de sleutel zich, gezien de volgorde, vóór die middelste elementen bevindt, moet de sleutel zich in de eerste helft van de lijst bevinden (omdat deze geordend zijn). Als de sleutel zich, gezien de volgorde, na de middelste elementen bevindt, dan moet deze in de tweede helft zitten.
We sluiten de overbodige helft van de lijst uit en doen hetzelfde aan de kant die overblijft. We herhalen dit telkens opnieuw totdat er nog maar één ding over is, nl. de sleutel, of de sleutel is er niet.

Verdeel en heers

De strategie die bij binair zoeken wordt gebruikt om het probleem te halveren en dan op het resterende deel dezelfde techniek toe te passen, kan in veel situaties toegepast worden, niet enkel om te zoeken. Het is een strategie om problemen op te lossen die 'verdeel en heers' wordt genoemd. Het is een manier om heel snelle algoritmes te ontwerpen voor veel verschillende dingen.

Efficiëntieanalyse

Er zijn veel verschillende zoekalgoritmes. Hoe kunnen we er één uit kiezen? We kunnen ons daarvoor baseren op hun efficiëntie. We kunnen een bepaalde 'kritische[1]' bewerking kiezen die een goed idee geeft van hoeveel werk er wordt gedaan, zoals het aantal gestelde vragen of het aantal keer knipperen dat nodig is. We kunnen dan uitzoeken hoe vaak die bewerking in het beste geval, in het slechtste geval en gemiddeld plaatsvindt.

[1] Een kritische bewerking is een bewerking waarop het onderscheid kan gemaakt worden.

Computationeel denken

Mensen begrijpen

Computationeel denken gaat vaak over het oplossen van problemen voor mensen. Zet de mens dan op de eerste plaats. Je moet het probleem dat je oplost, begrijpen vanuit hun oogpunt, voordat je oplossingen bedenkt.
Anders is je geweldige technische oplossing misschien nutteloos.

Algoritmisch denken

Algoritmisch denken gaat over het bedenken van een precieze manier om een taak uit te voeren; hierbij wordt met alles rekening gehouden. Bij een gegeven algoritmische oplossing kunnen andere mensen of computers de instructies vervolgens mechanisch volgen. Ze hoeven het probleem niet zelf op te lossen om antwoorden te krijgen. Volg een zoekalgoritme en je vindt wat je zoekt.

Patroonherkenning

Patroonherkenning gaat over een manier om tot oplossingen te komen, door te zien wanneer het ene probleem hetzelfde is als het andere. Als we erin slagen om problemen om te zetten in problemen die we eerder hebben opgelost, dan kunnen we de oplossing hergebruiken. Zodra we een goed zoekalgoritme hebben, kunnen we het aanpassen voor gebruik in veel verschillende zoek-problemen.

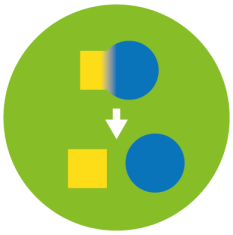
Generaliseren

Een andere manier om erover na te denken is dat, als we eenmaal een strategie hebben bedacht om een bepaald nieuw probleem op te lossen, we die oplossing dan kunnen veralgemenen tot een strategie die ook voor andere problemen werkt.
Het stellen van vragen in een spel 'Wie ben ik?' veralgemeent naar de verdeel-en-heersstrategie. Op dezelfde manier kunnen we een algoritme *generaliseren* voor een specifiek probleem om een zoekalgoritme te bekomen. Het generaliseren van het idee om "Is het A," of "Is het B?" te vragen, geeft ons het lineaire zoekalgoritme dat van toepassing is op elk zoekprobleem.

Logisch denken

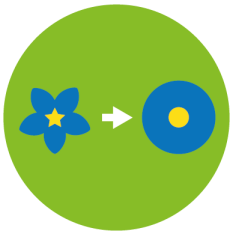
Met logisch denken kunnen we op een solide manier verschillende algoritmes vergelijken. Door abstractie te gebruiken, richten we ons enkel op de details die ertoe doen (we moeten er wel voor opletten dat we geen belangrijke details verliezen). De methode die het meest geschikt is voor ons doel is misschien de snelste, maar andere eigenschappen zoals geheugencapaciteit die nodig is, kunnen er ook toe doen.

Concepten van computationeel denken

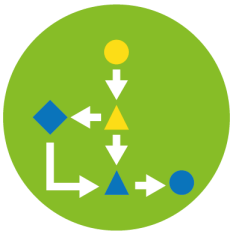


Decompositie

Patroonherkenning



Abstractie



Algoritme

Logica

Samenwerken

Debuggen

Exploreren

dwengo