# Kortste pad algoritme van Dijkstra

[[1]](#footnote-1)

**Concepten:** Dijkstra’skortste-padalgoritme, graaf, gretig algoritme.

**Leerdoelen:**

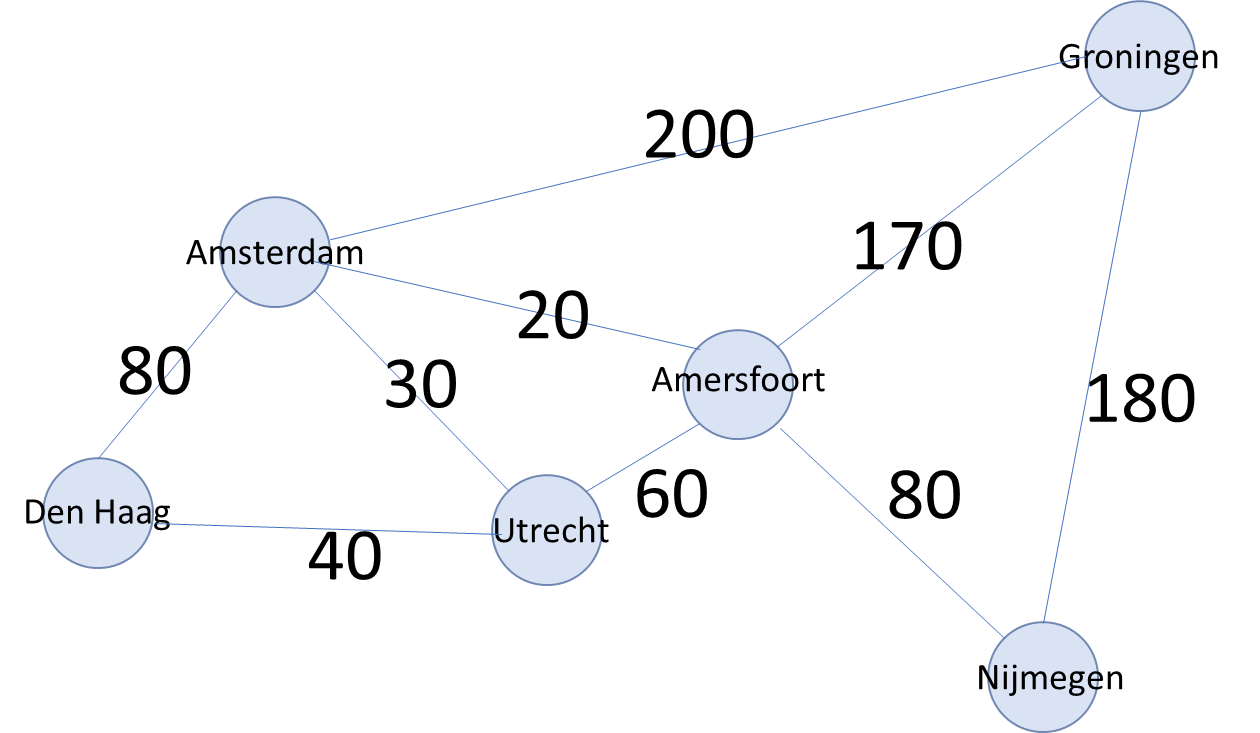
Na afloop van deze opdracht kun je:

* het kortste-pad algoritme van Dijkstra toepassen.
* een probleem analyseren en de relevante informatie (abstractie) in een geschikte representatie (graaf) weergeven.
* redeneren over de correctheid van het algoritme.

**Praktische context**: Het vinden van de kortste route tussen twee locaties (voor bijvoorbeeld een routeplanner). Dit wordt ook gebruikt bij robot navigatie, route planners, roosters maken, netwerk routering protocollen, en berekenen van optimale vrachtwagen-routering over drukke wegen.

# Uitdaging: De kortste pad tussen twee steden vinden

Je wilt per trein reizen van Den Haag naar Nijmegen. Hieronder zie je een spoorkaart met de afstanden (in kilometers) tussen ieder twee steden. Hoe bepaal wat de kortste route is van Den Haag naar Nijmegen?



**Brute force**

Een manier om dit probleem op te lossen is door alle mogelijke paden (zonder cykels) op te schrijven, en voor elk pad de afstand te bepalen. Je kiest dan na afloop het pad met de kortste afstand.

|  |  |
| --- | --- |
| **Route** | **Totale afstand** |
| Den Haag → Amsterdam → Utrecht → Amersfoort → Nijmegen | 80+30+60+80 = 250 |
| Den Haag → Amsterdam → Amersfoort → Nijmegen | 80+20+80 = 180 |
| Den Haag → Amsterdam → Groningen → Nijmegen | 80+200+180 = 460 |
| Den Haag → Utrecht → Amsterdam → Amersfoort → Nijmegen | 40+30+20+80 = 170 |
| … | … |

Alle mogelijkheden afgaan en dan de beste bepalen heet *brute force*. Voor een kleine kaart is dit te doen, maar als je op dezelfde manier een reis van Lissabon (Portugal) naar Wenen (Oostenrijk) zou willen plannen per trein, dan wordt dat een hele klus. Je krijgt erg veel mogelijkheden. Zou er een slimmere manier zijn om dat te doen?

|  |  |
| --- | --- |
| **Paden en Graven**  Een figuur zoals je bij het bovenstaande kaart ziet heet een **graaf**. Een graaf bestaat uit punten waarvan sommige verbonden zijn door lijnen. De punten in een graaf noemen we knopen en de verbindingslijnen heten kanten.  We noemen een graaf samenhangend als er vanuit elke knoop in de graaf een pad bestaat naar elke andere knoop. Een samenhangede graaf heeft dus geen onbereikbare kopen.  Samenhangende graaf  Het voorbeeld hiernaast is een niet samenhangende graaf omdat F niet verbonden is.  Wat is een pad eigenlijk? Een **pad** tussen twee knopen is een aaneenschakeling van kanten beginnend bij de ene knoop en eindigend in de andere. Hierbij mogen begin- en eindpunt hetzelfde zijn, maar dat hoeft zeker niet. In de graaf hiernaast is A-D-E-C een pad. Ook A-B-D-A is een pad, een met een cykel.  Onsamenhangende graaf |  |

**Brainstorm**

Bedenk zelf een slimmer manier om de kortste pad te vinden. Omschrijf hier jouw algoritme.

|  |
| --- |
| Antwoord: Een mogelijke oplossing is de beste-buur algoritme.  Het dichtste-buur algoritme, zonder terugkeren, maar met vermijden van  kringen.  we zoeken een pad van A naar Z; terwijl we het zoeken noteren we dat  pad op een papiertje; we beschrijven dat pad door een opeenvolging van  straten van de vorm X-Y waarbij X en Y punten zijn: in het begin is  het padpapiertje leeg  op een ander papiertje noteren we op welke plaats we zijn: in het  begin staat op het plaatspapiertje A  herhaal totdat Z op het plaatspapiertje staat:  stel X staat op het plaatspapiertje  neem vanuit X de kortste straat die leidt naar een  punt Y dat nog op het padpapier staat  schrijf de straat X-Y bij op je padpapier  veeg X uit van het plaatspapier en schrijf  er Y voor in de plaats  indien er geen zulke straat vanuit X bestaat  dan stop het algoritme met de melding "niet gevonden"  indien Z op het plaatspapiertje staat, dan staat het pad van A tot Z  op het padpapier  Caveat: niet alleen is het mogelijk dat er helemaal geen pad gevonden  wordt, maar ook dat het gevonden pad niet het kortste is.  Vraag: kan je "terugkeren" toevoegen in het algoritme, zodat je altijd  een pad vindt? |

*Laat je algoritme door een andere leerling uitvoeren. Zitten er fouten of onduidelijkheden in jouw algoritme?*

|  |
| --- |
|  |

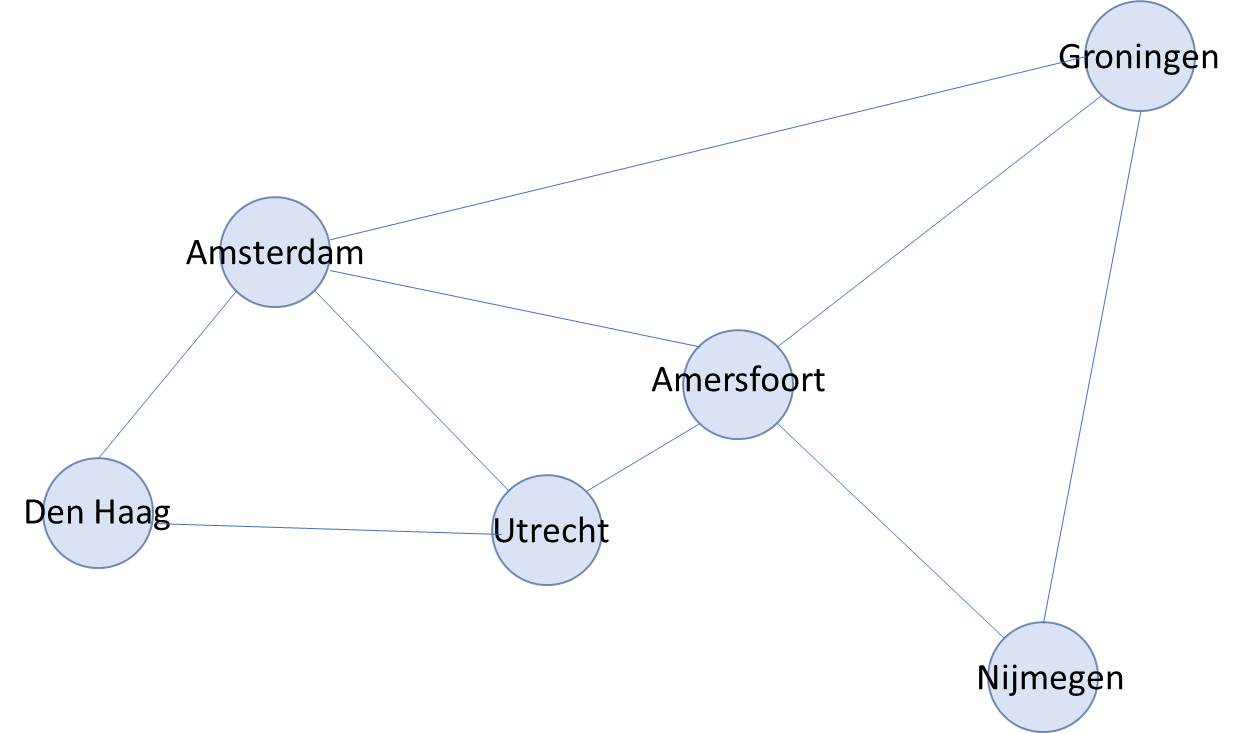
*Kun je en kaart (of graaf) verzinnen waarbij jouw algoritme niet het kortste-pad oplevert?*

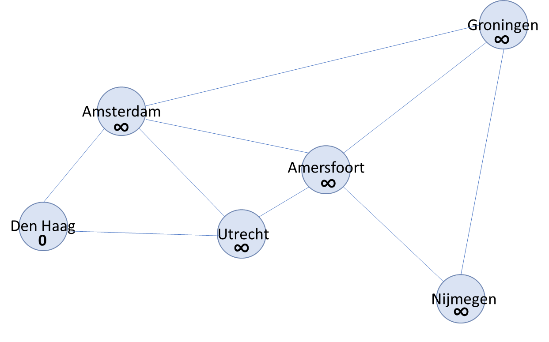
|  |
| --- |
|  |

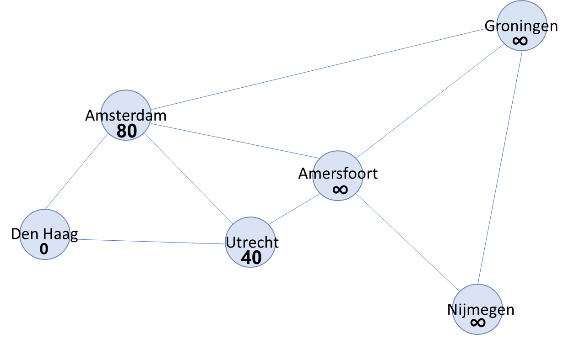
|  |
| --- |
| **Dichtste-buur algoritme**  Er zijn minstens twee naïeve methodes om de kortste pad te vinden. Naast alle mogelijkheden proberen (brute force) kun je ook de dichtste-buur gebruiken: vanuit een punt ja je simpelweg naar het dichtstbijzijnde punt. Hierbij let je op dat je niet terugkeert of in een kring loopt, dus je vermijd cykels.  De methode is niet perfect, want soms krijg je niet echt het kortste pad. |

**Dijkstra’s kortste-pad algoritme**

De Nederlander Edsger Dijkstra heeft in 1959 een kortste-pad algoritme beschreven, wereldwijd bekend als Dijkstra’s algoritme. Hoe het werkt doen we nu samen met de volgende kaart:

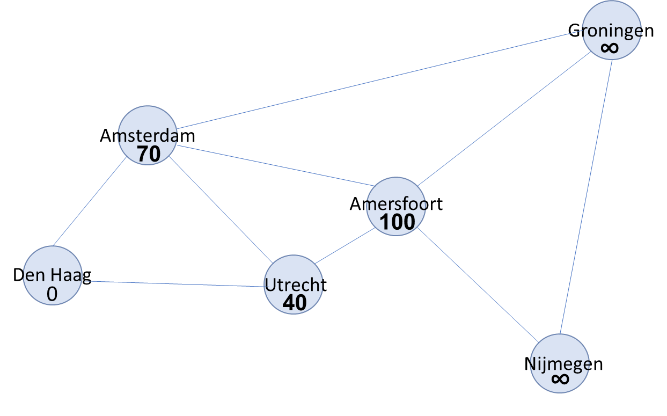


**Stap 1:** Geef je beginstad de waarde 0 en de andere steden de waarde oneindig (dat geven we aan met een ∞-teken). In dit voorbeeld nemen we Den Haag als beginstad. Je weet nu nog niet of de afstand naar een ander plek 10km of 1000 km zal zijn (die hebben we namelijk nog niet bekeken), dus daarom kenen we er ∞ (oneindig)-aan.



**Stap 2:** Vul de afstanden in voor de steden die je vanuit je beginstad met 1 stap kunt bereiken. Dus bij het bolletje Utrecht haal je een streep door ∞ en schrijf je de afstand van Den Haag naar Utrecht op. Dit doe je voor alle steden die vanuit Den Haag bereikbaar zijn, dus ook Amsterdam.

**Stap 3:**

1. Uit de nog niet eerdere gekozen steden pak je de stad met de laagste waarde (omdat Amersfoort, Groningen en Nijmegen allemaal ∞ zijn, Amsterdam 80 is en Utrecht 40, kies je Utrecht).
2. Voor alle steden die vanuit Utrecht in 1 stap bereikbaar zijn, bepaal hun afstand tot Den Haag. Dus bijvoorbeeld, Amersfoort wordt 60+40 = 100.
3. Als de berekende afstand kleiner is dan wat er al staat, vervang de afstand dan met de kleinere waarde. Dus Amsterdam via Utrecht is 40+30=70. 70 is kleiner dan de 80die er al stond, dus vervang je de 80 door 70.

**Stap 4:** Blijf zo doorgaan tot elk stad aan de beurt is geweest. De waarde die het eindpunt krijgt is de kortste afstand waarin het te bereiken is vanuit de beginpunt.

|  |
| --- |
| **Dijkstra’s kortste pad algoritme**  Het kortstepad-algoritme, ook bekend als Dijkstra's algoritme, is een graaf-algoritme beschreven door Edsger Dijkstra in 1959.  Uitgaande van een graaf, rekent het algoritme voor een bepaalde beginknoop de kortste afstand uit tot alle punten in de graaf. De graaf mag gewogen zijn, dat wil zeggen dat er afstanden of gewichten op de zijden aangegeven zijn.  Het algoritme wordt onder andere gebruikt bij verkeersmodellen, route-navigatiesystemen en telecommunicatie. |

**Opgave 1.**

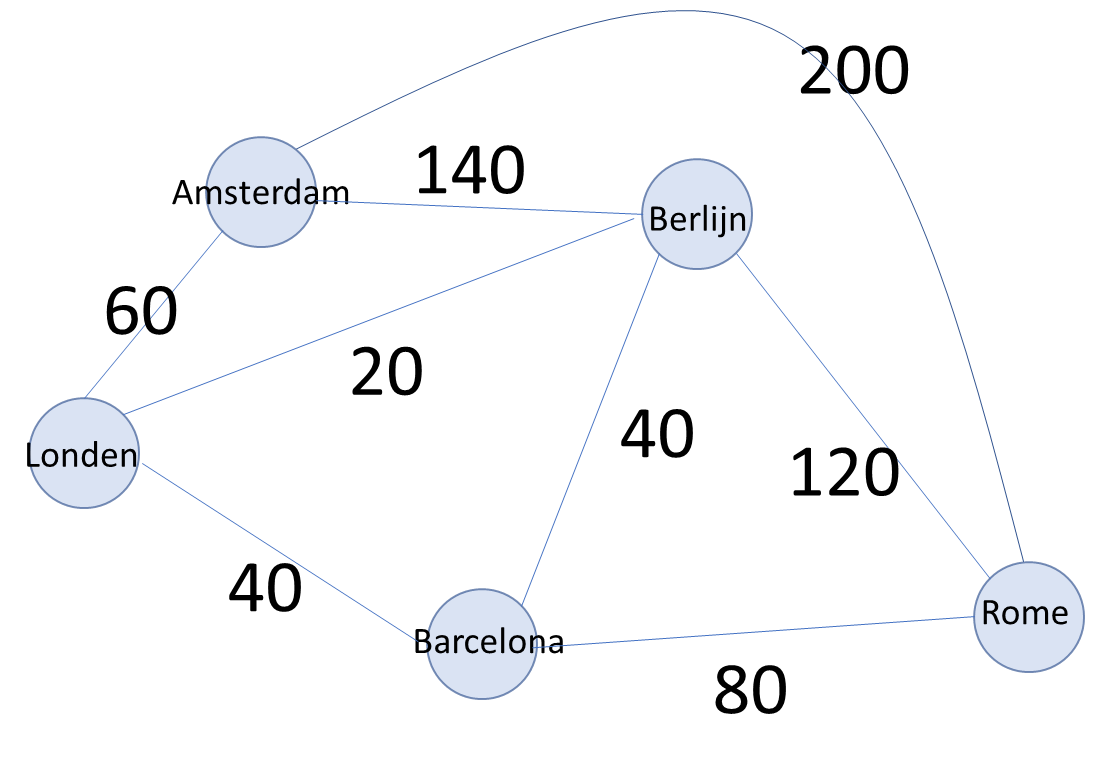
Op de spoorkaart staan de afstanden (in kilometers) tussen ieder twee steden aangegeven. Pas Dijkstra’s kortste-pad algoritme, zoals hierboven is beschreven om de kortste route van **Groningen** naar **Den Haag** te vinden.

*Geef aan wat de kortste route is van Groningen naar Den Haag*.

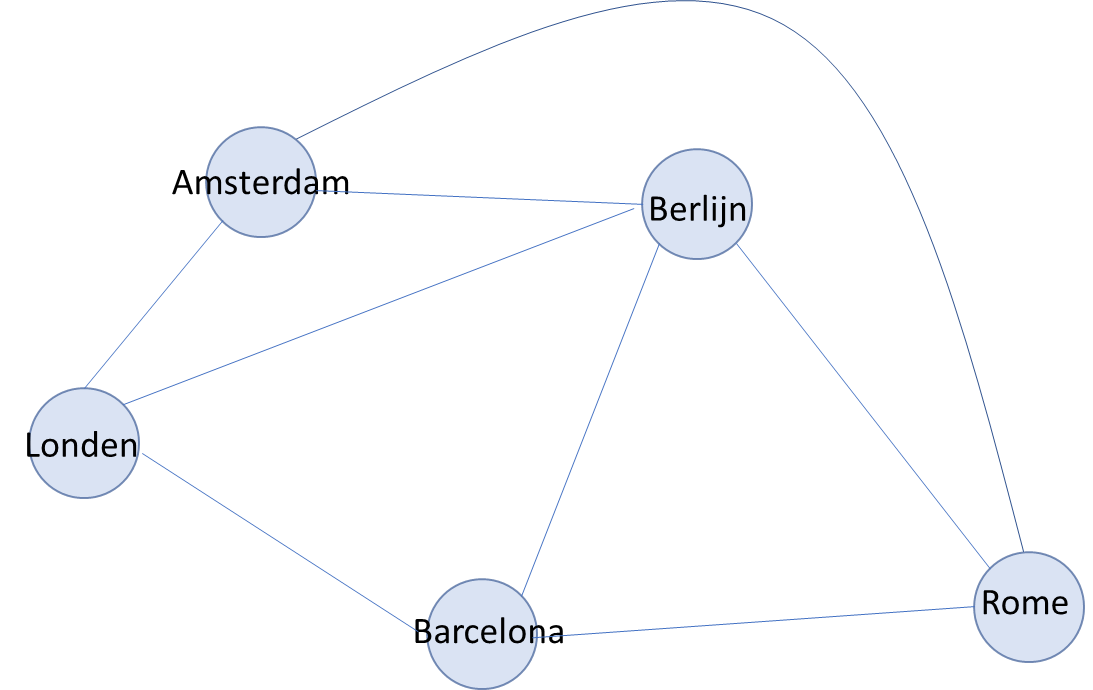
|  |
| --- |
| *Antwoord:*  *270km. Volg dan het traject Groningen → Amersfoort → Utrecht → Den Haag* |

**Opgave 2.**

Met een low-budget luchtvaartmaatschappij kun je goedkoop door Europa reizen. Per vlucht staan hieronder de prijzen in Euro’s aangegeven. Pas Dijkstra’s algoritme toe om de goedkoopste route te vinden van ***Amsterdam*** naar ***Rome***.



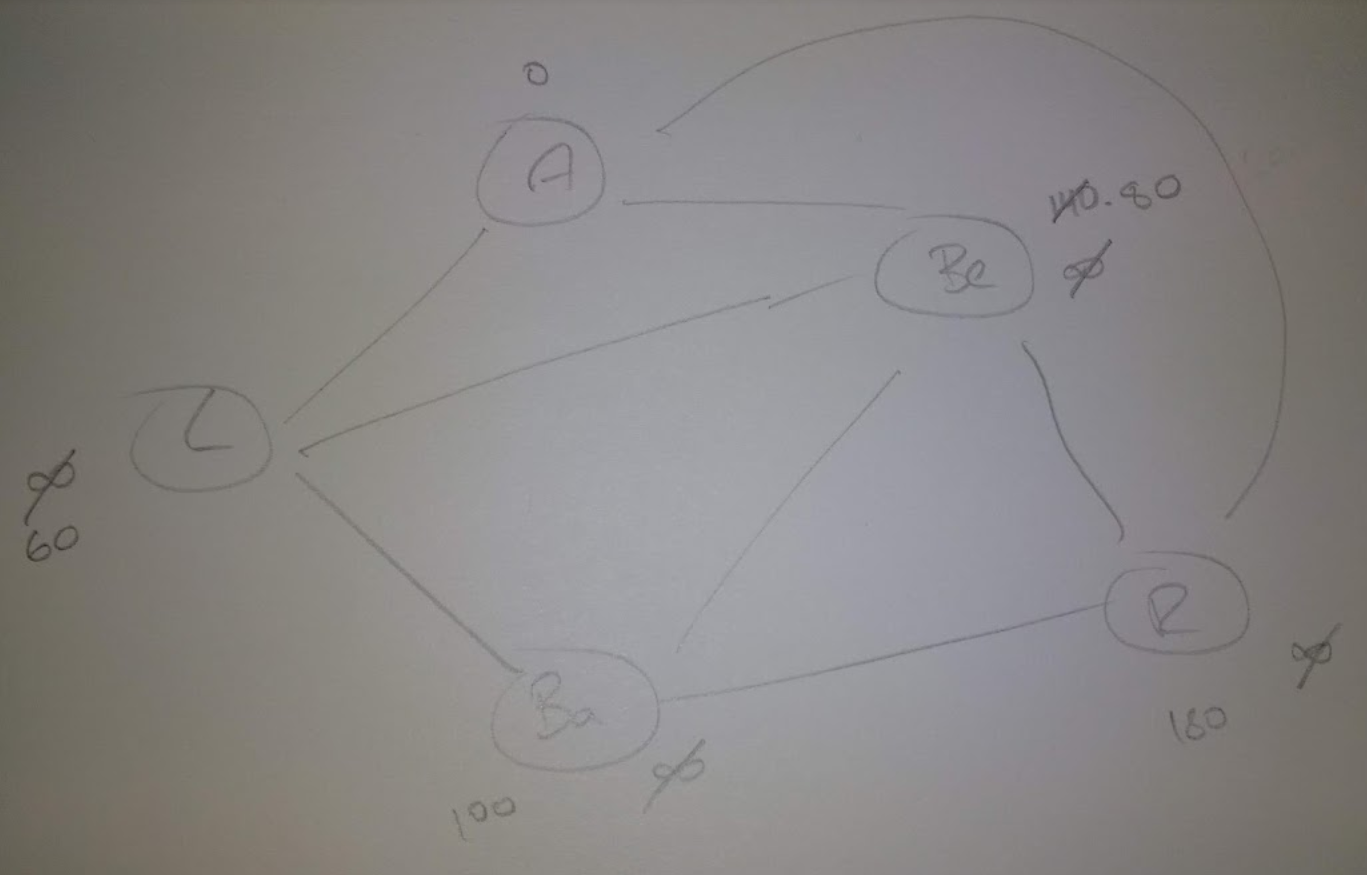
Voor het uitvoeren van het algoritme kun je deze graaf gebruiken:



*Geef aan wat de goedkoopste route is van* ***Amsterdam*** *naar* ***Rome***.

*Antwoord:*

*Goedkoopste route kost 180Euro: Amsterdam -> London -> Barcelona -> Rome*



|  |
| --- |
| **Gretige algoritme**  Dijkstra’s algoritme heet **gretig** omdat het bij elke stap steeds de kortste afstand kiest, zonder vooruit te kijken naar de gevolgen van die beslissing. |

*Zul je met het kortste-pad algoritme van Dijkstra altijd de meest optimale route vinden? Ook als bijvoorbeeld de kortste route over meer knopen gaat dan een langere?*

*Antwoord: Het kortste-pad algoritme van Dijkstra zal altijd de meest optimale route vinden. Het maakt daarbij niet uit of de kortste pad over meer knopen gaat dan een langere. (De enige voorwaarde is dat de getallen op de lijnen positief zijn.)*

**Opgave 3.**

Dijkstra’s algoritme kun je voor veel meer problemen gebruiken dan alleen om de kortste route te bepalen. Als je in plaats van afstanden bijvoorbeeld kosten in je graaf vermeld, dan kun je het gebruiken om geld te besparen. Maar je kan er ook geld mee verdienen!

Als je geld wisselt van de ene valuta naar een ander, dan kan dat soms gunstiger uitpakken dan anders. Jouw opdracht is om de bepalen hoe je het meeste goud kan kopen met je euro’s.

**Stap 1**: Hieronder staan de wisselkoersen. Maak een graaf van de mogelijke transacties. Let er op dat je op de verbindingslijnen pijltjes zet, want het maakt uit of je Dollars naar Pond omwisselt of andersom.

|  |  |
| --- | --- |
| van | naar |
| Euro | Britse Pond |
| 0,90 | |

|  |  |
| --- | --- |
| van | naar |
| Euro | U.S. Dollar |
| 1,10 | |

|  |  |
| --- | --- |
| van | naar |
| U.S. Dollar | Britse Pond |
| 0,80 | |

|  |  |
| --- | --- |
| van | naar |
| Britse Pond | U.S. Dollar |
| 1,30 | |

|  |  |
| --- | --- |
| van | naar |
| Britse Pond | Japanse Yen |
| 143 | |

|  |  |
| --- | --- |
| van | naar |
| U.S. Dollar | Japanse Yen |
| 111 | |

|  |  |
| --- | --- |
| van | naar |
| Euro | Goud |
| 0,0007 | |

|  |  |
| --- | --- |
| van | naar |
| U.S. Dollar | Goud |
| 0,0008 | |

|  |  |
| --- | --- |
| van | naar |
| Japanse Yen | Goud |
| 0,00001 | |

**Stap 2**: Schrijf bij je Euro-knoop een 1, want je begint met 1 Euro. Bij alle andere knopen zet je de waarde ∞.

**Stap 3**: Bepaal wat je voor 1 Euro krijgt als je deze inwisselt tegen een ander valuta. Om dit te berekenen vermenigvuldig je 1 Euro met de waarde in de bijbehorende tabel. Zet die waarde bij de bijbehorende knoop in je graaf.

**Stap 4**: Doorloop Dijkstra’s algoritme. Maar let op: je wilt natuurlijk zo veel mogelijk goud. Daarom onthoud je niet telkens de kleinste waarde, maar juist de grootste waarde. Houd ook bij hoe je daar gekomen bent (welke transacties je hebt gedaan om tot dat bedrag te komen).

*Geef aan hoe je valuta moet wisselen om het meeste goud voor je euro’s te krijgen*.

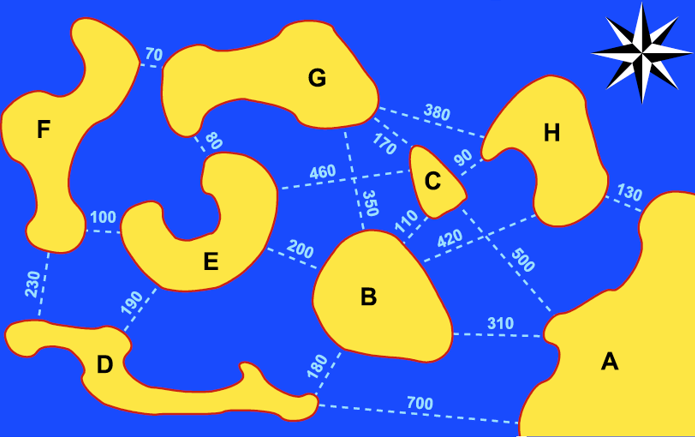
*Antwoord:*

*Euro -> Britse Pond -> U.S. Dollar -> JapanseYen -> Goud*

*Je krijgt dan 0,0012987 oz Goud per euro.*

**Opgave 4.**

Hieronder zie je een kaart van de eilandenstelsel van Algos. Om te reizen tussen de eilanden of met het vaste land wordt gebruik gemaakt van veerponden. Omdat het verkeer tussen het vasteland A en eiland D toeneemt, wil Algos bruggen gaan bouwen. Op de kaart staan de afstanden (in meters) aangegeven.

[[2]](#footnote-2)[[3]](#footnote-3)

*Wat is de goedkoopste verbinding tussen A en D?*

Antwoord: De goedkoopste verbinding tussen A en D is een verbinding van A naar B en B naar D. Dit kost 490.

*Gebruik een van de gretig algoritmes om A met D te verbinden. Hoeveel kost dat?*

Antwoord: Een gretig algoritme kiest telkens de goedkoopste mogelijkheid zonder vooruit te kijken. In dit geval is dat eerst A met H, dan H met C, dan C met B, en dan B met D. Dat is 510.

Vergelijk de twee bovenstaande antwoorden. Wat kan je zeggen over de oplossing van gretige algoritmes?

Antwoord: Soms loont een gretig algoritme. Het zal in ieder geval altijd een oplossing geven. Echter, omdat deze niet vooruit kijkt kan het zijn dat je niet tot de allerbeste oplossing komt.

**Waar gaat dit eigenlijk over?**

Een **graaf** is een handig schema om de gegevens in weer te geven. Een graaf heeft **knopen** (bijvoorbeeld steden) en **verbindingslijnen** (bijvoorbeeld wegen). Hebben de verbindingslijnen een waarde (zoals bijvoorbeeld een afstand, reistijd of benzinekosten), dan spreken we van een **gewogen** graaf. Hebben de verbindingslijnen een richting, dan heet zo’n graaf een **gerichte** graaf.

Dijkstra’s kortste-pad algoritme berekent de kortste route tussen een begin- en eindpunt. Het algoritme heet **gretig** omdat het bij elke stap steeds de kortste afstand kiest zonder vooruit te kijken naar de gevolgen van die beslissing. Zoals je hebt gezien, loont het soms om gretig te zijn. Zo’n gretig algoritme zal in ieder geval altijd een oplossing geven. Maar, zo’n gretige aanpak levert niet altijd de meest optimale oplossing op.

In de informatica kennen we veel problemen waarbij de hoeveelheid werk om een goede oplossing te vinden enorm toeneemt als de hoeveelheid gegevens stijgt. Hiervoor moeten slimme en efficiënte algoritmen worden uitgedacht.

Dijkstra’s algoritme kan voor veel meer problemen worden gebruikt. Je kunt het ook inzetten om de snelste of goedkoopste route te vinden.

1. Bron: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993. [↑](#footnote-ref-1)
2. Bron: Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., & Wagner, D. (Eds.). (2010). *Algorithms unplugged*. Springer Science & Business Media. [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)