De kortste afstand tussen twee mensen is de glimlach.

Charlie Chaplin (1889–1977)

1 Grafen

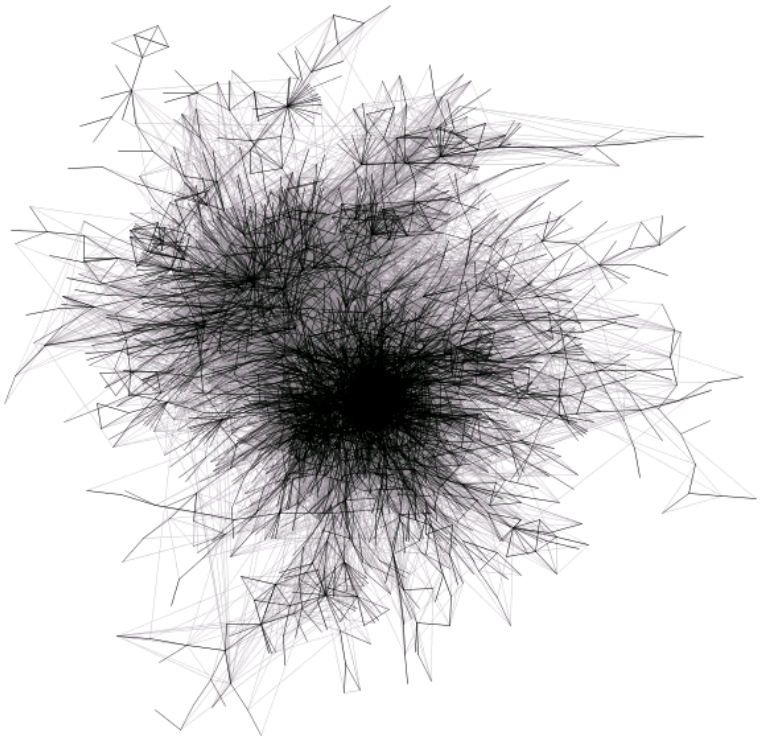
1.1 Inleiding

Een graaf is een verzameling knooppunten die met mekaar verbonden zijn. In de informatica- en communicatietechnologie kom je heel vaak problemen tegen die in de essentie terug te brengen zijn tot een aantal situaties of toestanden die al dan niet met mekaar verbonden zijn. Een typisch voorbeeld is het computernetwerk wereldwijd waarbij de verschillende elementen met mekaar verbonden zijn via hubs. Een ander voorbeeld is een wegenkaart waarbij kruispunten van straten met mekaar verbonden zijn. We vermelden ook het internet *an sich* waarbij de verschillende websites met mekaar verbonden zijn doordat ze links bevatten naar andere sites, of meer specifiek de 'blogosfeer' (verschillende weblogs verwijzen naar mekaar). In figuur 1.1 vind je de visualisatie van de blogosfeer. De knooppunten (de weblogs zelf) zijn weggelaten om overdaad te vermijden.

De verbindingen tussen de knooppunten hoeven niet fysiek te zijn. Een flow chart voor een complexe opdracht kan je ook bekijken als een graaf. De taak kan immers opgedeeld worden in deeltaken die deels na mekaar uitgevoerd moeten worden. De verbindingen geven aan welke opdracht volgt op een vorige opdracht.

Als je een systeem gemodelleerd hebt als graaf, kan je er dingen mee doen. Voor een wegennetwerk kan je de kortste of snelste route tussen punt A en punt B berekenen. Je kan ook routes zoeken die rekening houden met de capaciteit van de verbindingen. Voor een computernetwerk kan je simuleren hoe een worm zich in dat netwerk verspreidt. Er zijn ontzettend veel mogelijkheden.

Al in het begin van de vorige eeuw werden netwerken en de bijhorende theorieën gebruikt bij het bepalen van de capaciteit van telefoonkabels en schakelcentra. Later werd de verder ontwikkelde theorie over netwerken ook gebruikt bij problemen op het gebied van telecommunicatie, computernetwerken, fysieke distributie en projectplanning.



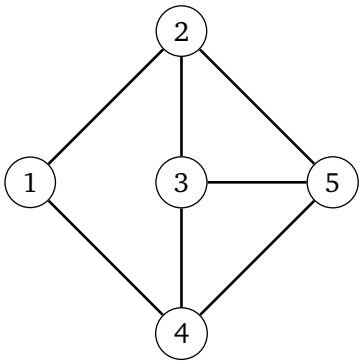
Figuur 1.1 Voorbeeld van een graaf: de blogosfeer (bron: http://datamining.typepad.com/gallery/blog-map-gallery.html

Netwerkproblemen zouden eventueel kunnen opgelost worden met de algemene methode van lineaire programmering, maar door hun speciale aard kunnen ze vaak toch efficiënter aangepakt worden met alternatieve methoden gebaseerd op grafen. Een dergelijke alternatieve methode is de methode van Dijkstra, waarmee men de kortste weg tussen twee punten in een netwerk kan berekenen. De methode van Floyd is een andere methode. Dit is echter stof voor het volgende hoofdstuk 3.

In voorliggend hoofdstuk definiëren we het begrip graaf. We laten zien hoe we matrices kunnen gebruiken om een graaf te modelleren. Ten slotte gebruiken we de matrices om te kijken welke routes er mogelijk zijn in de graaf en welke de kortst mogelijke route is om van plaats A naar plaats B te gaan.

1.2 Definities

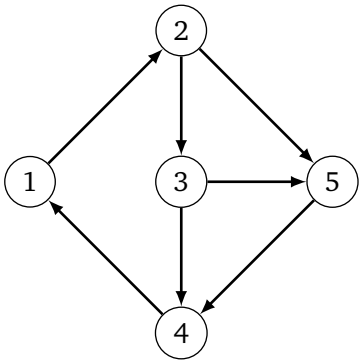
Een graaf is een verzameling van N *knooppunten* en een verzameling A van *takken* (lijnen, verbindingen) tussen sommige knooppunten. Figuur 1.2 toont een eenvoudig voorbeeld.



Figuur 1.2 Een niet-gerichte graaf

We beschrijven de graaf door de verzameling van knooppunten en takken te specifiëren. Hier is $N = \{1,2,3,4,5\}$ en $A = \{\{1,2\},\{1,4\},\{2,3\},\{2,5\},\{3,4\},\{3,5\},\{4,5\}\}$. Merk op dat A een *verzameling van verzamelingen* is. De takken bezitten geen richting en worden daarom volledig beschreven door de twee punten die verbonden worden door de takken. Er is geen begin- en eindpunt.

Als alle takken een richting bezitten (op de figuur is er dan een pijl te zien bij elke verbinding) wordt de graaf *gericht* genoemd. Als de graaf enkel takken bevat die niet voorzien zijn van een richting (de figuur van de graaf bevat enkel lijnstukken, geen pijlen), noemen we de graaf *niet-gericht*. Een *gemengde* graaf tenslotte bevat zowel takken met als zonder richting. De graaf van figuur 1.2 is een voorbeeld van een niet-gerichte graaf. Figuur 1.3 toont een gerichte graaf.



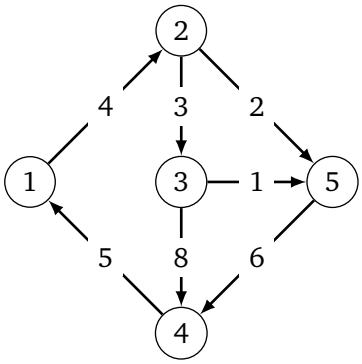
Figuur 1.3 Een voorbeeld van een gerichte graaf

Als een graaf gericht is, kan de verzameling van takken geen verzameling van verzamelingen meer zijn. De takken hebben immers een begin- en eindpunt. In zo'n geval schrijven we de verzameling takken als verzameling van koppels. De graaf voorgesteld in figuur 1.3 wordt beschreven door $N = \{1, 2, 3, 4, 5\}$ en $A = \{(1, 2), (2, 3), (2, 5), (3, 4), (3, 5), (4, 1), (5, 4)\}$.

Een *pad* is een opeenvolging van knooppunten. Tussen elke twee knooppunten van een pad moet vanzelfsprekend een tak bestaan. Zo is op figuur 1.2 de opeenvolging $1 \to 2 \to 3 \to 5 \to 4$ een (mogelijk) pad om van knooppunt 1 naar knooppunt 4 te gaan. Er zijn bij dit voorbeeld natuurlijk ook andere paden mogelijk.

Een graaf heet *samenhangend* als tussen elke twee knooppunten een pad bestaat. We maken de veronderstelling dat vanaf nu alle grafen die we bekijken samenhangend zijn.

Als in een graaf bij alle takken een getal wordt vermeld (afstand, tijdsduur, kostprijs,...) spreken we van een *netwerk* of *gewogen graaf*. Deze getallen noemt met de *gewichten* van de takken. Figuur 1.4 toont een netwerk.



Figuur 1.4 Een netwerk of gewogen graaf

1.3 Grafen voorstellen door matrices

De kenmerken van een graaf kunnen eenvoudig voorgesteld worden met behulp van een vierkante matrix. Het aantal rijen en kolommen is altijd gelijk aan het aantal knooppunten van de graaf. De verschillende knooppunten worden genummerd. Het element op rij i en kolom j zegt iets over de tak van knooppunt i naar punt j.

1.3.1 Verbindingsmatrix

De verbindingsmatrix (*adjacency matrix*) geeft aan welke knooppunten met mekaar in verbinding staan. Als er één of meerdere takken bestaan van knooppunt i naar knooppunt j, is het element op rij i, kolom j gelijk aan dat aantal takken. Anders is het gelijk aan nul. We tonen de verbindingsmatrix voor de grafen van figuur 1.2 (1.1) en figuur 1.3 (1.2). De woorden 'van' en 'naar' maken geen deel uit van de matrix, net zoals de labels van 1 tot 5. Die zijn enkel bijgevoegd om de opbouw van zo'n matrix duidelijker te maken.

$$\begin{array}{cc} & \begin{array}{ccccc} & & & \text{naar} & & \\ 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} \\ 1 \\ 2 \\ \text{van}\ \ 3 \\ 4 \\ 5 \end{array} & \left[\begin{array}{ccccc} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right] \end{array} \tag{1.1}$$

$$\begin{array}{cc} & \begin{array}{ccccc} & & & \text{naar} & & \\ 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} \\ 1 \\ 2 \\ \text{van}\ \ 3 \\ 4 \\ 5 \end{array} & \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \tag{1.2}$$

Merk op dat de verbindingsmatrix (1.1) symmetrisch is. Dat is altijd zo voor een niet-gerichte graaf.

Door de verbindingsmatrix met zichzelf te vermenigvuldigen, kan je zien hoeveel paden er bestaan van lengte twee tussen elk paar knooppunten van de graaf. We hernemen graaf 1.3 en bijhorende verbindingsmatrix (1.2). Een pad van lengte 2 van bijvoorbeeld knooppunt 2 naar knooppunt 4 bestaat enkel als er een tak bestaat van knooppunt 2 naar een willekeurig knooppunt x en van dat knooppunt x naar knooppunt 4. Als we de elementen van de tweede rij van de verbindingsmatrix vermenigvuldigen met de elementen van de vierde kolom, zien we of er zo een pad bestaat. Immers, de elementen van de tweede rij geven aan of er een

tak bestaat van knooppunt 2 naar een ander knooppunt, en de elementen van de vierde kolom geven aan of er takken aankomen in het vierde knooppunt. Als we deze producten met mekaar optellen, bekomen we het aantal paden van lengte 2 van knooppunt 2 naar knooppunt 4. In dit geval is die som gelijk aan

$$0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 2 \tag{1.3}$$

Er zijn dus twee paden van lengte 2 van knooppunt 2 naar knooppunt 4.

In plaats van altijd een rij te vermenigvuldigen met een kolom, kunnen we net zo goed de verbindingsmatrix met zichzelf vermenigvuldigen. Je krijgt dan onmiddellijk een overzicht van hoeveel paden er bestaan van lengte twee tussen elk paar knooppunten in deze graaf:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.4}$$

Er bestaat dus één pad van lengte 2 van knooppunt 1 naar knooppunt 3, van knooppunt 5 naar knooppunt 1 enz. Over welke knooppunten dit pad gaat, kan je niet afleiden.

Door het product (1.4) nogmaals met de verbindingsmatrix te vermenigvuldigen, krijg je een overzicht van de paden van lengte 3. Hoe vaak moet je de verbindingsmatrix met zichzelf vermenigvuldigen om de paden van alle lengte te kennen[1]?

1.3.2 Gewichtenmatrix

Uit de verbindingsmatrix kan je enkel afleiden of twee knooppunten al dan niet verbonden zijn met mekaar. Bij een netwerk zijn de gewichten van de takken evenzeer belangrijk. Daarom stelt men een netwerk voor door een *gewichtenmatrix*. Het element op rij i en kolom j geeft het gewicht aan van de tak van knooppunt i naar knooppunt j. Als er geen tak bestaat tussen twee knooppunten, wordt het gewicht gelijk gesteld aan oneindig (∞). In voorliggende tekst zetten we op de hoofddiagonaal 0. Dit is echter geen noodzaak en hangt af van de probleemstelling. Onderstaande matrix toont de gewichtenmatrix van de graaf voorgesteld in figuur 1.4.

$$\begin{array}{cc} & \text{naar} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} \\ 1 \\ 2 \\ \text{van} \quad 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 4 & \infty & \infty & \infty \\ \infty & 0 & 3 & \infty & 2 \\ \infty & \infty & 0 & 8 & 1 \\ 5 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array} \tag{1.5}$$

[1] Als het knooppunt n paden telt, moet je de verbindingsmatrix $n-1$ keer met zichzelf vermenigvuldigen.

Voor de volledigheid stellen we voor de graaf voorgesteld in figuur 1.4 ook de verbindingsmatrix op (matrix (1.6)). Merk op dat alle elementen op rij i en kolom j die in matrix (1.5) gelijk zijn aan oneindig, in matrix (1.6) gelijk zijn aan 0.

$$\begin{array}{cc} & \text{naar} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \text{van} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \tag{1.6}$$

Het is zinvol om voor een netwerk zowel verbindingsmatrix als gewichtenmatrix op te stellen. Met de eerste kan je bepalen of er al dan niet een pad bestaat van knooppunt i naar knooppunt j, zonder de gewichten van de takken in rekening te brengen. Dit werd geïllustreerd in sectie 1.3.1. Een ander voorbeeld vind je in sectie 1.4. Met de gewichtenmatrix kan je het pad tussen twee knooppunten bepalen dat een zo klein mogelijk totaal gewicht heeft. In hoofdstuk 3 worden algoritmes om dit uit te voeren uitgelegd.

We merken tenslotte op dat er in de literatuur geen eenduidige naamgeving bestaat wat betreft nabijheids- en gewichtenmatrix. Vaak wordt de verbindingsmatrix (*adjacency matrix*) ook *incidence matrix* genoemd en/of staat de verbindingsmatrix zowel voor de matrices die wij definiëren in sectie 1.3.1 als de gewichtenmatrix van sectie 1.3.2.

1.4 Toepassing: Breadth-first search

In sectie 1.3.1 toonden we aan dat we, door de verbindingsmatrix met zichzelf te vermenigvuldigen, kunnen bepalen of er al dan niet een pad bestaat tussen twee knooppunten. Met deze methode kan je echter niet achterhalen over welke knooppunten dat pad gaat. Het gevonden pad is bovendien niet noodzakelijk het pad met het minst aantal knooppunten. In deze sectie bespreken we een zoekalgoritme dat het pad met het minst aantal knooppunten bepaalt én dat toelaat dat pad te reconstrueren.

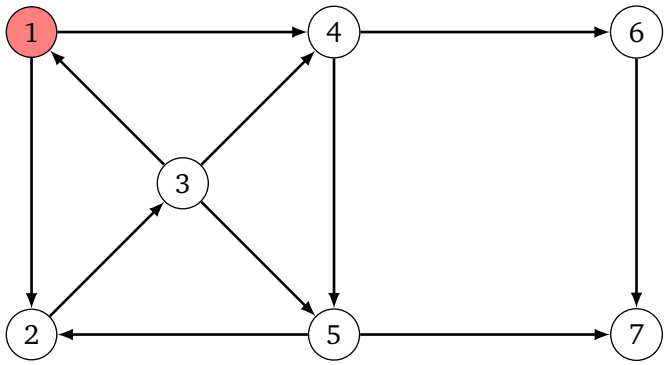
Het algoritme heet *breadth-first search*: het zoekt in de *breedte* naar mogelijke oplossingen. Vertrekkende van een startpunt, worden altijd knooppunten toegevoegd die *rechtstreeks* verbonden zijn met de al gevonden knooppunten. De lengte van de paden wordt systematisch opgevoerd: eerst worden alle paden van lengte 1 onderzocht, dan die van lengte 2 enz.

Er bestaat ook een *depth-first* algoritme. In tegenstelling tot *breadth-first*, begint een depth-first algorimte bij een knooppunt en voegt het telkens één knooppunt toe dat rechtstreeks verbonden is met het voorgaande knooppunt. Het algoritme probeert als het ware een pad samen te stellen dat hopelijk in de goede richting gaat. Als het pad doodloopt (omdat het enkel verder kan naar al gevonden knooppunten of omdat er geen knooppunten zijn die rechtstreeks

verbonden zijn), gaat het algoritme één of meer stappen in het pad terug totdat het pad opnieuw verder samengesteld kan worden. Voordeel van depth-first is dat de knooppunten van het gevonden pad onmiddellijk gekend zijn. Nadeel van dit algoritme is dat het gevonden pad niet noodzakelijk het minst aantal knooppunten telt.

We leggen het breadth-search algoritme uit aan de hand van de graaf voorgesteld in figuur 1.5. We zoeken het pad van knooppunt 1 naar knooppunt 7 dat zo weinig mogelijk knooppunten telt. Het algoritme stelt eerst knooppunt 1 (startpunt) voor als oplossing. Wellicht is dit knooppunt niet het gewenste eindpunt. Daarom zegt het algoritme om alle paden van lengte 1 die vertrekken uit knooppunt 1 te controleren. Als het gezochte pad daar niet bij is, moeten alle paden van lengte 2 die een 'verlenging' zijn van de paden die gevonden werden in de vorige stap, gecontroleerd worden. Dan bekijken we de 'verlengingen' van die paden (met lengte 3) enz. De knooppunten die al bezocht werden, worden in elke stap gemarkeerd. Dit doen we om te vermijden dat we een pad met een 'lus' zouden creëren.

We kleuren in de eerste stap alvast knooppunt 1. Dit knooppunt is niet het gezochte knooppunt. We moeten dus een volgende stap zetten.

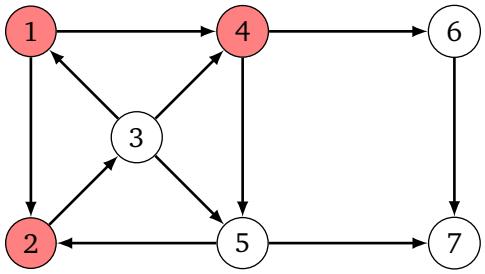


Figuur 1.5 Graaf waarvoor het breadth-first algoritme toegepast wordt.

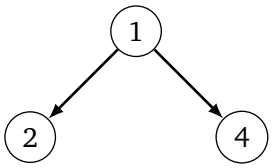
In de tweede stap kijken we welke knooppunten rechtstreeks verbonden zijn met knooppunt 1. Dat zijn knooppunt 2 en 4. Beide zijn niet het gezochte knooppunt 7. We moeten dus verder zoeken. We kleuren knooppunt 2 en 4 (figuur 1.6a). Om later het gevolgde pad te bepalen, houden we in een schema bij welke knooppunten op mekaar volgen (figuur 1.6b).

In de derde stap zoeken we voor de laatst gevonden knooppunten (2 en 4) weer buren, d.w.z. knooppunten die rechtstreeks verbonden zijn met knooppunt 2 en 4. Als die buren nog niet gekleurd zijn, kleuren we ze (figuur 1.7a) en voegen we ze toe aan het boomdiagram (figuur **??**). Als één van de buren het gezochte knooppunt 7 is, moeten we stoppen met zoeken. Knooppunt 2 is rechtstreeks verbonden met 3; knooppunt 4 is rechtstreeks verbonden met 5 en 6.

In de vierde stap moeten we de takken die vertrekken in knooppunt 3, 5 en 6 onderzoeken. Vanuit knooppunt 3 vertrekken drie takken: naar knooppunt 1, 4 en 5. Al deze knooppunten zijn al gekleurd. Als we ze zouden toevoegen aan het pad, zou de lengte onnodig verlengd

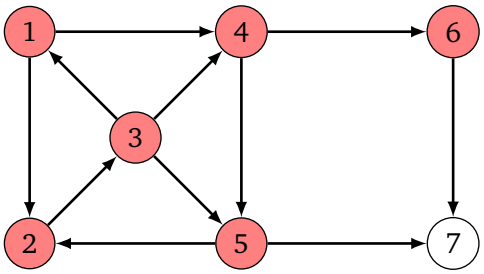


(a) Bezochte knooppunten 2 en 4 worden gekleurd

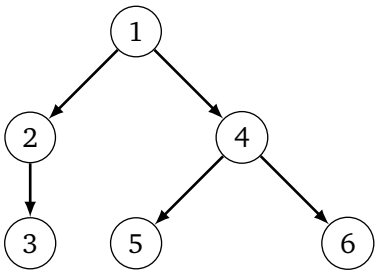


(b) Overzicht van gevonden knooppunten en hun onderlinge relatie: stap 2

Figuur 1.6 Breadth-first: tweede stap.



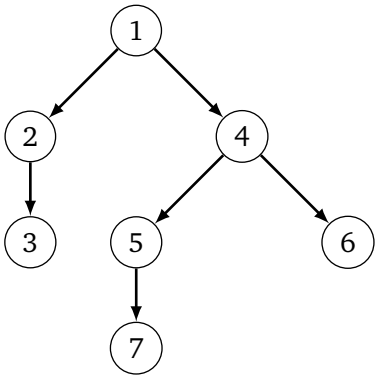
(a) Bezochte knooppunten 3, 5 en 6 worden gekleurd



(b) Overzicht van gevonden knooppunten en hun onderlinge relatie: stap 3

Figuur 1.7 Breadth-first: derde stap.

worden. We nemen ze dus niet op. Uit knooppunt 5 vertrekken twee takken: knooppunt 2 is al gekleurd en mogen we niet nemen; knooppunt 7 is het gezochte knooppunt. We hebben ons doel bereikt. We voegen knooppunt 7 toe aan de boomdiagram (figuur 1.8) en lezen het gevonden pad af van deze figuur. Het pad van knooppunt 1 naar knooppunt 7 met het kleinst aantal knooppunten loopt langs knooppunten 4 en 5.



Figuur 1.8 Breadth-first: vierde stap. Het gezochte knooppunt 7 is bereikt.

De gevonden oplossing $1 \rightarrow 4 \rightarrow 5 \rightarrow 7$ is niet uniek: het pad $1 \rightarrow 4 \rightarrow 6 \rightarrow 7$ is ook een oplossing van het probleem.

Het spreekt voor zich dat je bij het programmeren van breadth-first search gebruik maakt van de verbindingsmatrix van de graaf. Er stellen zich twee problemen:

- Hoe houd je bij welk knooppunt je moet onderzoeken?
- Hoe vind je het optimale pad terug?

In antwoord op het eerste probleem, gebruikt het breadth-first algoritme het principe van een *queue* (wachtrij) of *first-in-first-out-list* (FIFO). Zo'n queue is een lijst waarvan de lengte niet vastligt. De knooppunten die rechtstreeks verbonden zijn met een knooppunt en die je nog moet onderzoeken, voeg je langs rechts toe. Als je een knooppunt gaat onderzoeken, haal je het langs links uit de queue.

Om het optimale pad terug te vinden, moet je voor elk knooppunt dat je kleurt zijn voorganger onthouden. Dan kan gemakkelijk in een vector. In wat volgt, tonen we stap voor stap de queue Q en voorgangers-vector A (*ancestor*) van het voorbeeld hierboven.

stap 1: initialiseren Q

1

A

1	2	3	4	5	6	7

stap 2: knooppunt 1 controleren Q

~~1~~	2	4

A

	1		1			
1	2	3	4	5	6	7

stap 3: knooppunt 2 controleren Q

~~1~~	~~2~~	4	3

A

	1	2	1			
1	2	3	4	5	6	7

stap 3: knooppunt 4 controleren Q

~~1~~	~~2~~	~~4~~	3	5	6

A

	1	2	1	4	4	
1	2	3	4	5	6	7

stap 4: knooppunt 3 controleren Q

~~1~~	~~2~~	~~4~~	~~3~~	5	6

A

	1	2	1	4	4	
1	2	3	4	5	6	7

stap 4: knooppunt 5 controleren Q

~~1~~	~~2~~	~~4~~	~~3~~	~~5~~	6	7

A

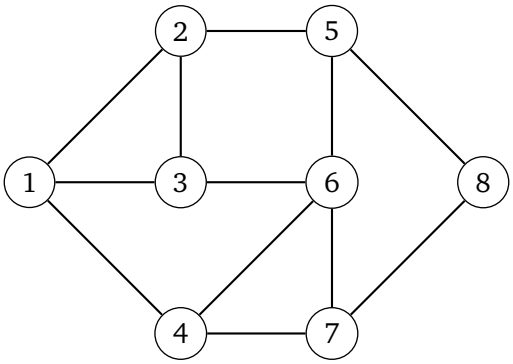
	1	2	1	4	4	5
1	2	3	4	5	6	7

1.5 Oefeningen

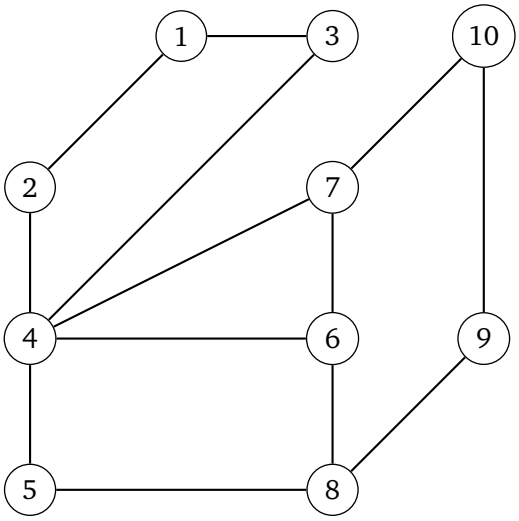
Zelfde vraag voor elke oefening:

1. Stel de verbindingsmatrix op van de getekende graaf.
2. Zoek het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.

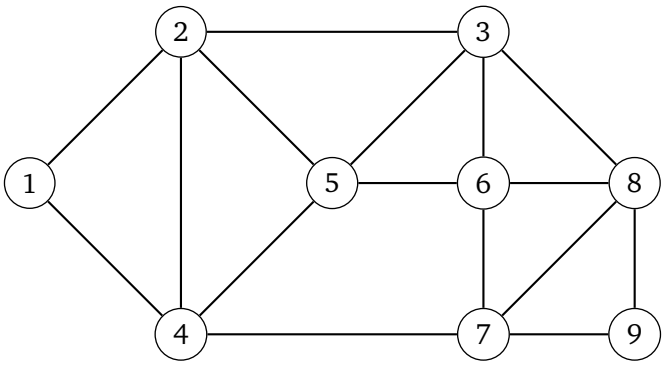
Oefening 1.1



Oefening 1.2



Oefening 1.3



The real danger is not that computers will begin to think like men, but that men will begin to think like computers.

Sydney J. Harris

2

Het algoritme van Floyd

2.1 Inleiding

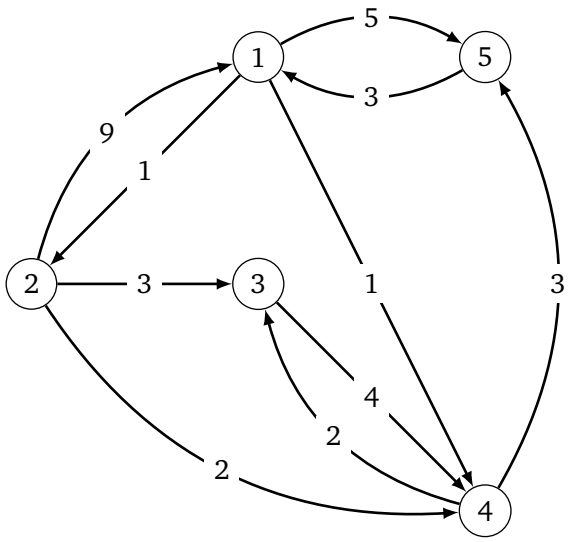
In hoofdstuk 1 definieerden we grafen als een verzameling van knooppunten die met mekaar verbonden zijn door takken. We bespraken hoe we een graaf kunnen voorstellen door een matrix en zochten hoe je kan berekenen of er een pad bestaat tussen twee gegeven knooppunten. Het algoritme breadth-first berekent het pad met het kleinst aantal knooppunten.

In dit hoofdstuk focussen we op netwerken. Een netwerk is een graaf waar de takken niet alleen een richting hebben, maar ook een gewicht. Een voorbeeld van zo een netwerk is een wegenkaart waarbij de gewichten staan voor de afstanden tussen verschillende steden, of de reistijd om van één plaats naar een ander te gaan. In dit en het volgende hoofdstuk bespreken we twee algoritmes die het pad bepalen tussen twee knooppunten zodat het *totale* gewicht zo klein mogelijk is.

2.2 Inleiding

Het algoritme van Floyd staat ook bekend onder de benaming 'Floyd–Warshall' of 'All pairs shortest path'. Het berekent het de kortste afstand *van elk punt naar elk ander punt*. Het algoritme van Dijkstra (zie volgende hoofdstuk) daarentegen zoekt enkel de kortste afstand van één gegeven punt naar elk ander punt.

We illustreren het algoritme aan de hand van de graaf van figuur 2.1. Matrix (2.1) toont de



Figuur 2.1 Graaf met 5 knooppunten

gewichtenmatrix van deze graaf.

$$
\begin{array}{cc} & \text{naar} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \text{van} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \left[\begin{array}{ccccc} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{array}\right] \end{array} \tag{2.1}
$$

2.3 Algoritme

Het algoritme van Floyd werkt als volgt. We beginnen met de gewichtenmatrix. Deze geeft alle rechtstreekse verbindingen tussen twee punten. Als er geen rechtstreekse verbinding is, dan noteren we die afstand als ∞ (*oneindig*, in een praktische implementatie eenvoudig te vervangen door een groot getal).

We vragen ons nu af of we een *kortere verbinding* kunnen maken tussen twee punten door tussenliggende knooppunten toe te laten. Deze aanpak moet natuurlijk gestructureerd gebeuren. Daarom gaan we alle punten af, te beginnen met punt 1 enz.

Formeel uitgedrukt: $D^{(k)}(i, j) =$ lengte van het kortste pad van punt i naar punt j, waarbij je als tussenliggende knooppunten enkel gebruik mag maken van punten 1 tot en met k. Vanzelfsprekend is $D^{(0)}$ de startmatrix (gewichtenmatrix), want die geeft enkel de rechtstreekse

verbindingen (zonder dus via tussenliggende punten te gaan). We streven ernaar om $D^{(n)}$ te vinden (n is het aantal knooppunten).

In de volgende secties werken we enkele stappen van het algoritme met de hand uit, als voorbeeld. De pseudocode van het uiteindelijke algoritme probeer je zelf op te stellen.

2.3.1 Kortere verbindingen via punt 1

We beschouwen nu knooppunt 1 en vragen ons af of er een kortere route mogelijk is van elk punt i naar elk ander punt j door *punt 1* als tussenstop te gebruiken.

Concreet merken we bvb. dat 1 voor een kortere verbinding zorgt om van 2 naar 5 te gaan. De huidige kortste afstand van 2 naar 5 is nu immers ∞, aangezien er geen rechtstreekse verbinding is van punt 2 naar punt 5. Als we echter via punt 1 gaan, kan het wel degelijk korter. De route van 2 naar 1 is 9 lang en die van 1 naar 5 is 5 lang. Vermits $9+5<\infty$ vervangen we in onze nieuwe matrix $D^{(1)}(2,5)$ de oorspronkelijke waarde ∞ door 14.

$$D^{(1)} = \begin{bmatrix} 0 & 1 & \infty & 1 & \boxed{5} \\ \boxed{9} & 0 & 3 & 2 & \mathbf{14} \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \tag{2.2}$$

Deze methode is eenvoudig grafisch voor te stellen. Overloop alle elementen van de gegeven gewichtenmatrix $D^{(0)}$. Maak de som van het element op deze rij maar in de eerste kolom met het element op deze kolom maar in de eerste rij. Als deze som kleiner is dan het huidig element vervang je de huidige waarde door deze som. Formeel uitgedrukt komt het hierop neer:

$$D^{(1)}(i,j) = \min\{D^{(0)}(i,j), D^{(0)}(i,1) + D^{(0)}(1,j)\} \tag{2.3}$$

Als we de matrix verder aflopen, merken we dat de route van punt 5 naar punt 2 ook sneller kan via punt 1, want $3+1<\infty$. We passen dus onze $D(1)$-matrix opnieuw aan:

$$D^{(1)} = \begin{bmatrix} 0 & \boxed{1} & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ \boxed{3} & \mathbf{4} & \infty & \infty & 0 \end{bmatrix} \tag{2.4}$$

Ook element $D^{(1)}(5,4)$ moet aangepast worden want $3+1<\infty$:

$$D^{(1)} = \begin{bmatrix} 0 & 1 & \infty & \boxed{1} & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ \boxed{3} & 4 & \infty & \mathbf{4} & 0 \end{bmatrix} \tag{2.5}$$

2.3.2 Kortere verbindingen via punt 2

Ook hier zijn er twee elementen die moeten aangepast worden, omdat de route via tussenliggend knooppunt 2 korter is. We gegeven hieronder beide opeenvolgende versies van de matrix $D^{(2)}$. Ga na of je de redenering snapt.

$$D^{(2)} = \begin{bmatrix} 0 & \boxed{1} & \mathbf{4} & 1 & 5 \\ 9 & 0 & \boxed{3} & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & \infty & 4 & 0 \end{bmatrix} \tag{2.6}$$

$$D^{(2)} = \begin{bmatrix} 0 & 1 & 4 & 1 & 5 \\ 9 & 0 & \boxed{3} & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \boxed{4} & \mathbf{7} & 4 & 0 \end{bmatrix} \tag{2.7}$$

2.3.3 Kortere verbindingen via punt 3

Als je alle elementen van $D^{(2)}$ afloopt, merk je dat je door via punt 3 te gaan nergens verbetering kan brengen. Daarom is $D^{(3)} = D^{(2)}$:

$$D^{(3)} = \begin{bmatrix} 0 & 1 & 4 & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & 7 & 4 & 0 \end{bmatrix} \tag{2.8}$$

2.3.4 Kortere verbindingen via punt 4

Maak zelf de oefening. Je zou volgende matrix moeten bekomen. De 5 veranderde elementen staan in het vet.

$$D^{(4)} = \begin{bmatrix} 0 & 1 & \mathbf{3} & 1 & \mathbf{4} \\ 9 & 0 & 3 & 2 & \mathbf{5} \\ \infty & \infty & 0 & 4 & \mathbf{7} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & \mathbf{6} & 4 & 0 \end{bmatrix} \tag{2.9}$$

2.3.5 Kortere verbindingen via punt 5

Als we ook punt 5 betrekken in de verbindingen, bekomen we ten slotte volgende matrix (veranderde waarden staan terug in het vet). Elk getal uit deze matrix geeft de lengte van de kortste afstand tussen twee punten.

$$D^{(5)} = \begin{bmatrix} 0 & 1 & 3 & 1 & 4 \\ \mathbf{8} & 0 & 3 & 2 & 5 \\ \mathbf{10} & \mathbf{11} & 0 & 4 & 7 \\ \mathbf{6} & \mathbf{7} & 2 & 0 & 3 \\ 3 & 4 & 6 & 4 & 0 \end{bmatrix} \tag{2.10}$$

2.4 De pointer-matrix

In de eindmatrix $D^{(5)}$ lees je bvb. af dat de korste afstand tussen punt 3 en punt 1 gelijk is aan 10 (nl. element $D^{(5)}(3,1)$). Wat je in deze matrix niet kan aflezen is *wat* deze route juist is. Je kan op de tekening het pad proberen terugvinden, maar dat is natuurlijk niet de bedoeling. Voor een klein netwerk zal dat nog wel lukken, maar je wil een algoritme dat je ook het antwoord kan geven voor een netwerk met 5000 knooppunten.

Gelukkig is dit niet zo moeilijk. We voeren een nieuwe matrix P (*pointermatrix* genaamd) in. In ons voorbeeld is dat ook een matrix met 5 rijen en 5 kolommen. Deze matrix wordt altijd geïnitialiseerd als een *nulmatrix*. Het principe is eenvoudig. Elke keer als je in het algoritme in vorige sectie een kortere afstand vindt door een knooppunt k te gebruiken, bewaar je deze k op de overeenkomstige plaats in de P-matrix.

Als voorbeeld kijken we terug naar matrix (2.2). De route van 2 naar 5 was oorspronkelijk oneindig lang (d.w.z. geen directe verbinding). Door knooppunt 1 als tussenpunt te gebruiken, kunnen we echter van punt 2 naar punt 5 geraken met een afstand van 14. We noteerden in onze D-matrix daarom een 14 in de plaats van de oorspronkelijke ∞.

In de P-matrix noteren we nu op dezelfde plaats (rij 2, kolom 5) het gebruikte tussenknooppunt, nl. 1. P wordt bijgevolg gelijk aan:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.11}$$

Herhaal dit voor elke stap. Je bekomt uiteindelijk volgende matrix P:

$$P = \begin{bmatrix} 0 & 0 & 4 & 0 & 4 \\ 5 & 0 & 0 & 0 & 4 \\ 5 & 5 & 0 & 0 & 4 \\ 5 & 5 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \end{bmatrix} \tag{2.12}$$

Het aflezen van het volledige pad is een eenvoudige *recursieve procedure*. We geven een voorbeeld. Zoek het pad van punt 3 naar punt 1. Op $D^{(5)}(3,1)$ lezen we de lengte van het pad af: 10. Het pad zelf stellen we zelf samen via de pointermatrix. Op $P(3,1)$ lezen we af dat het pad via knooppunt 5 gaat. Anders geformuleerd: het kortste pad van 3 naar 1 bestaat uit twee deelpaden: van 3 naar 5 en vervolgens van 5 naar 1. $P(3,5) = 4$ en dus bestaat het pad van 3 naar 5 op zijn beurt uit twee stukken: van 3 naar 4 en van 4 naar 5. Je blijft elk van die stukken op hun beurt opzoeken in de pointermatrix. *De recursie stopt als het overeenkomstig element in de pointermatrix gelijk is aan 0*.

$P(3,4) = 0$ en dus is er een rechtstreekse verbinding van 3 naar 4. Analoog wijst $P(4,5) = 0$ op een rechtstreekse verbinding van 4 naar 5. Hiermee is het deel van 3 naar 5 al afgewerkt. Blijft nog het deelpad van 5 naar 1 over. $P(5,1) = 0$ en dus loopt de snelste route over de directe verbinding van 5 naar 1.

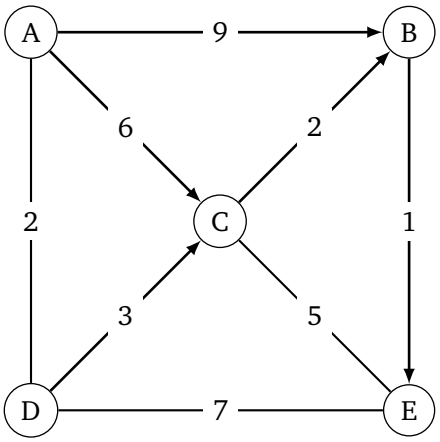
2.5 Verfijning: één matrix

Je hoeft niet alle D-matrices te bewaren. Eigenlijk heb je genoeg aan één zo'n matrix, waarin je telkens alle wijzigingen bewaart. Op het einde zal deze matrix dan gelijk zijn aan $D^{(n)}$.

2.6 Oefeningen

Oefening 2.1

Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 2.2 te berekenen.



Figuur 2.2 Netwerk bij oefening 2.1

Oefening 2.2

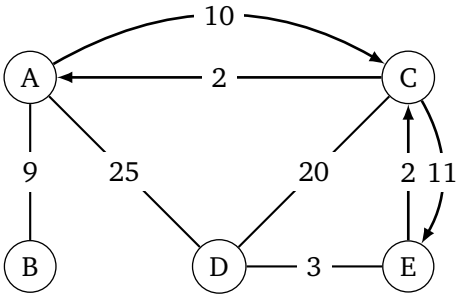
Een graaf bestaat uit 6 knooppunten en wordt gegeven door zijn gewichtenmatrix

$$D^{(0)} = \begin{bmatrix} 0 & 7 & 3 & \infty & 5 & \infty \\ 2 & 0 & \infty & 12 & \infty & \infty \\ \infty & 3 & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix}$$

1. Teken het netwerk.
2. Gebruik de methode van Floyd om de kortste paden tussen de verschillende punten van het netwerk te berekenen.

Oefening 2.3

Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 2.3 te berekenen.



Figuur 2.3 Netwerk bij oefening 2.3

3

Het algoritme van Dijkstra

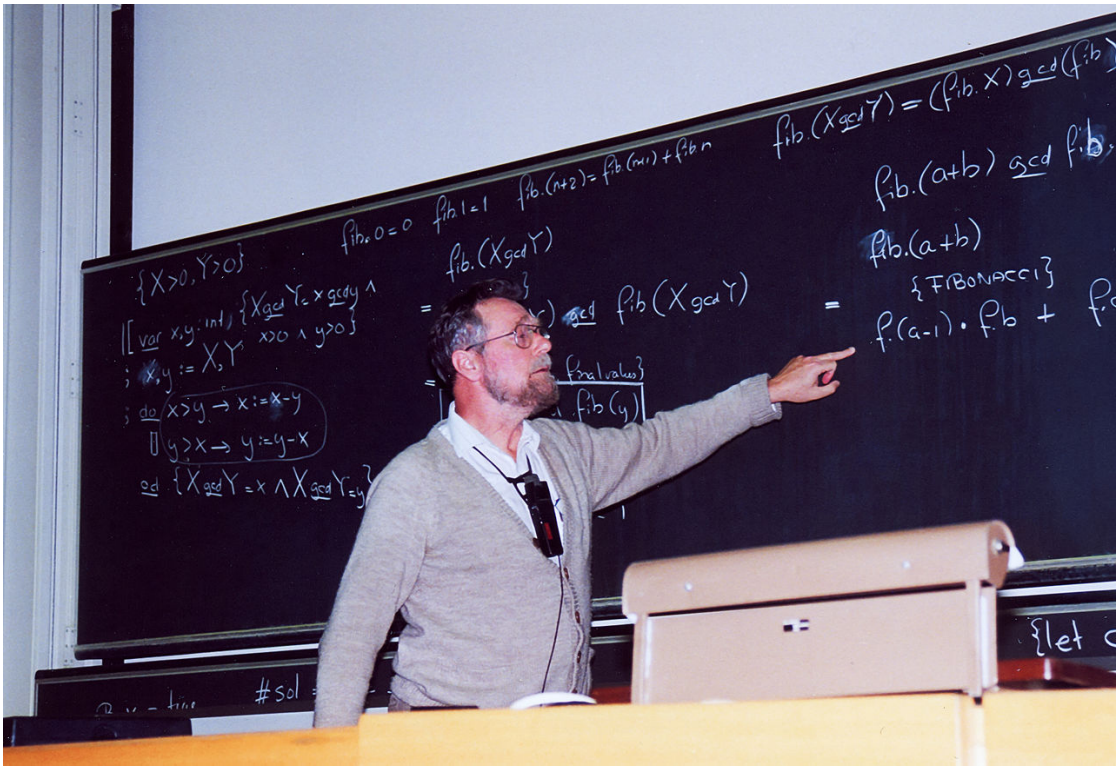
3.1 Inleiding

Het algoritme van Dijkstra is, in tegenstelling tot het algoritme van Floyd, een zgn. 'single-source shortest path'. Het berekent het kortste pad vanuit één gegeven knoop naar elke willekeurige andere knoop. In het voorgaande hoofdstuk hebben we het kortste pad algoritme uitgelegd aan de hand van bewerkingen op de gewichtenmatrix. In voorliggend hoofdstuk leggen we het algoritme visueel uit aan de hand van de tabelmethode. Hierdoor is de implementatie in Java een wat grotere uitdaging. In het practicum zal je hierin evenwel begeleid worden.

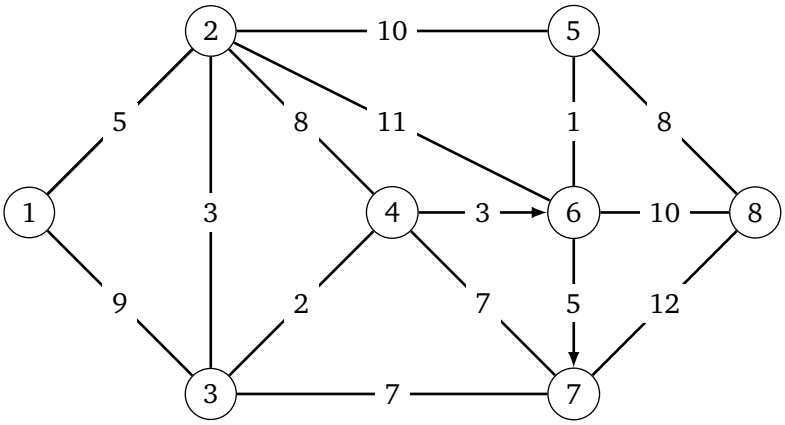
Het algoritme is genoemd naar E. Dijkstra (1930–2002). Dijkstra is gekend als de 'vader van het gestructureerd programmeren'. Hij ontwierp o.a. besturingssystemen en programmeertalen. Hij speelde een belangrijke rol in de vervanging van het commando 'GOTO' door functies en lussen. Tenslotte gaf hij richting aan de informatica van 'aanmodderen' tot een echte wetenschap: hij ontwikkelde methodes om op een gestructureerde manier een algoritme te definiëren waarvan *bewezen* kon worden dat het werkte.

Dijkstra beet zich ook vast in het in die tijd reeds gekende 'handelsreizigersprobleem'. Een handelsreiziger moet een aantal steden bezoeken. Hij wil dit op een zo snel mogelijke manier doen, dus zonder al te veel kilometers te maken. Een mogelijke oplossing bestaat erin om alle mogelijke combinaties op te schrijven en de kortste eruit te kiezen. Dit is echter voor grotere netwerken niet haalbaar omwille van te grote rekentijd. Dijkstra loste het handelsreizigersprobleem zelf niet op, maar focuste op een deelprobleem. Hij ontwikkelde een efficiënt algoritme om te berekenen wat het kortste pad is tussen twee gegeven knooppunten.

Bekijk het netwerk in figuur 3.2. Merk op dat het een gemengd samenhangend netwerk is. Twee van de takken zijn gericht. De knooppunten zouden bvb. steden kunnen zijn, en de getallen bij de takken afstanden in km. of tijden in minuten. We zoeken een antwoord op de vraag wat de kortste weg is om van knooppunt 1 naar knooppunt 8 te gaan, en hoe lang die weg dan wel is, rekening houdend met de gewichten van de takken.



Figuur 3.1 Edsger Wybe Dijkstra (1930–2002)



Figuur 3.2 Een netwerk

3.2 Grafische methode

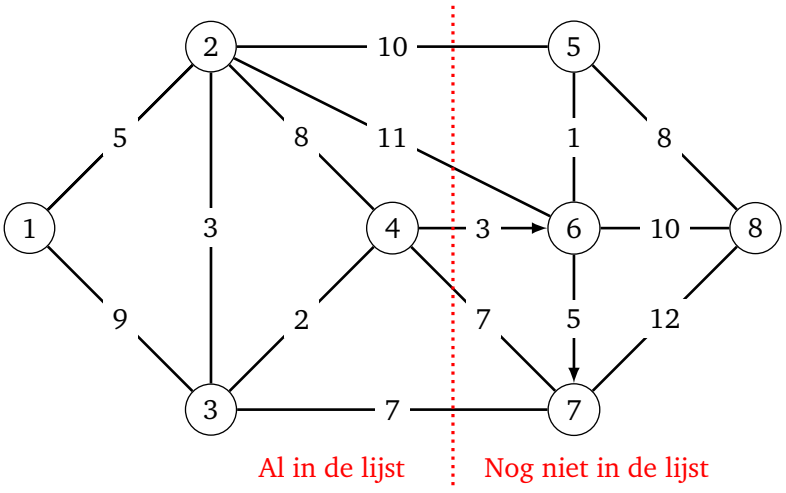
Het algoritme van Dijkstra werkt als volgt. Zoek het knooppunt dat het dichtst bij het beginpunt ligt. Noteer dit knooppunt, de afstand en de bijhorende route. Spoor vervolgens het knooppunt op dat na het zojuist gevonden punt het dichtst bij het beginpunt ligt. Noteer opnieuw alle gegevens. Zoek vervolgens het knooppunt dat na de twee al gevonden punten het dichtst ligt, enz. . . Op deze manier ontstaat een lijst van knooppunten die steeds verder verwijderd liggen van het beginpunt. Bemerk dat deze methode meer doet dan gevraagd. We waren enkel geïnteresseerd in de kortste afstand tussen punt 1 en punt 8. Deze methode stelt een geordende lijst op van *alle* afstanden van punt 1 tot elk ander punt.

We leggen nu op het vorig voorbeeldje (netwerk in figuur 3.2) dit principe uit. We pikken ergens in het midden van de berekeningen in. Stel: Na een aantal stappen heb je al het lijstje in tabel 3.1 gevonden.

Tabel 3.1 Al gevonden punten en afstanden

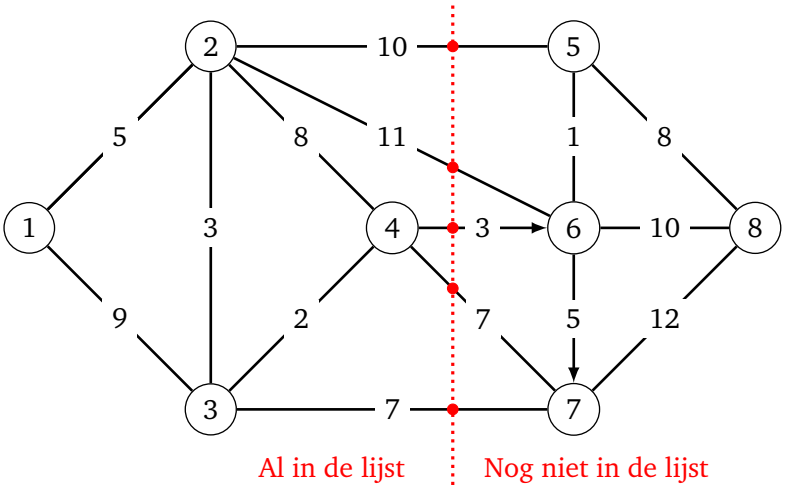
Stad	Afstand van stad 1	Route
1	0	$1 \rightarrow 1$
2	5	$1 \rightarrow 2$
3	8	$1 \rightarrow 2 \rightarrow 3$
4	10	$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

Op figuur 3.3 maakt een scheidingslijn het onderscheid tussen de al gevonden knooppunten, en de knooppunten die nog niet op de lijst staan.



Figuur 3.3 Toestand na enkele stappen van het algoritme

De volgende stad in het lijstje zal in elk geval *rechtstreeks* moeten verbonden zijn met één van de al gevonden steden. We zoeken m.a.w. enkel die knooppunten die met één tak verbonden zijn met één van de steden 1, 2, 3 of 4. De rechtstreekse verbindingen lees je af in een graaf als de takken die gesneden worden door de 'scheidingslijn'. Op figuur 3.4 lees je volgende verbindingen af: $2 \rightarrow 5$, $2 \rightarrow 6$, $4 \rightarrow 6$, $4 \rightarrow 7$ en $3 \rightarrow 7$. Knooppunt 8 kan dus niet rechtstreeks bereikt worden vanuit de al gevonden knooppunten 1 tot en met 4. Bemerk ook dat vanuit 1 geen enkele rechtstreekse verbinding is met de knooppunten rechts van de scheidingslijn.



Figuur 3.4 Rechtstreekse verbindingen

We moeten dus enkel volgende sommaties uitvoeren en vergelijken:

Afstand $1 \rightarrow 2$ + lengte van tak $2 \rightarrow 5$	=	$5+10$	=	15
Afstand $1 \rightarrow 2$ + lengte van tak $2 \rightarrow 6$	=	$5+11$	=	16
Afstand $1 \rightarrow 4$ + lengte van tak $4 \rightarrow 6$	=	$10+3$	=	**13**
Afstand $1 \rightarrow 4$ + lengte van tak $4 \rightarrow 7$	=	$10+7$	=	17
Afstand $1 \rightarrow 3$ + lengte van tak $3 \rightarrow 7$	=	$8+7$	=	15

De kortste van deze afstanden is 13. Het bijhorende nieuwe knooppunt is punt 6. We breiden tabel 3.1 uit met een extra rij tot tabel 3.2. Tenzij we het laatste knooppunt bereikt hebben, wordt het hele proces nu herhaald.

Algemeen geldt: stel dat men al een ranglijst van afstanden tot en routes naar p knooppunten (inclusief knooppunt 1, het knooppunt waaruit wordt vertrokken) bezit. Het knooppunt dat na deze p knooppunten het dichtst bij het beginpunt ligt, is dan een knooppunt dat *rechtstreeks* verbonden is met één van de al gevonden p knooppunten. Om dit knooppunt te vinden, bekijkt men alle nog niet op de lijst voorkomende knooppunten die rechtstreeks met een knooppunt van de lijst verbonden zijn. Voor deze knooppunten bepaalt men de som van de lengte van de rechtstreekse verbinding en de al gevonden afstand van knooppunt 1 tot het knooppunt van de lijst. Als een nog niet op de lijst opgenomen knooppunt rechtstreeks

Tabel 3.2 Nieuw knooppunt toegevoegd

Stad	Afstand van stad 1	Route
1	0	$1 \rightarrow 1$
2	5	$1 \rightarrow 2$
3	8	$1 \rightarrow 2 \rightarrow 3$
4	10	$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$
6	13	$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6$

verbonden is met meer dan één knooppunt van de lijst, dienen alle bijbehorende sommen beschouwd te worden.

Vervolgens dient men uit deze collectie sommaties die met de kleinste uitkomst te kiezen. Het knooppunt, behorend bij de kleinste som, is dan het knooppunt dat na de genoemde p knooppunten het dichtst bij knooppunt 1 ligt. Het procédé moet nu met de $p+1$ knooppunten van de lijst herhaald worden.

Men dient deze methode te starten met een lijst met daarop alleen knooppunt 1, afstand 0 en route $1 \rightarrow 1$, dus met $p = 1$.

3.3 Tabelmethode

Meestal is het handiger om – eens je de principes door hebt – gebruik te maken van een tabel. Tabel 3.3 stelt hetzelfde netwerk voor als figuur 3.2. Elk knooppunt wordt voorgesteld door een kolom. In elke kolom komt een opsomming van alle verbindingen vanuit dit knooppunt, met de respectievelijke lengte van de verbinding. Zo betekent '43 2' in de vierde kolom dat er een weg is van punt 4 naar punt 3, met een lengte van 2. Waaruit blijkt in deze tabel dat het opgegeven netwerk eenrichtingsverkeer bevat? De tabel bevat verder nog een extra eerste rij. Deze rij zal in de loop van het algoritme de totale afstand van het knooppunt tot het beginpunt (1) bevatten.

Tabel 3.3 Gegeven netwerk in een tabel gegoten

1	2	3	4	5	6	7	8
12 5	21 5	31 9	42 8	52 10	62 11	73 7	85 8
13 9	23 3	32 3	43 2	56 1	65 1	74 7	86 10
	24 8	34 2	46 3	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	26 11						

We beginnen in elk geval met boven stad 1 een 0 te zetten. Verder worden alle verbindingen die eindigen met een 1 geschrapt, omdat de kortste weg van 1 naar 1 van deze verbindingen

zeker geen gebruik zal maken. Het resultaat vind je in tabel 3.4. Op de ranglijst van steden staat op dit moment dan enkel stad 1.

Tabel 3.4 Beginstap

0							
1	2	3	4	5	6	7	8
12 5	~~21 5~~	~~31 9~~	42 8	52 10	62 11	73 7	85 8
13 9	23 3	32 3	43 2	56 1	65 1	74 7	86 10
	24 8	34 2	46 3	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	26 11						

Zoals al in de grafische vorm van de methode van Dijkstra beschreven zoeken we nu alle steden die rechtstreeks verbonden zijn met de al 'gevonden' steden, t.t.z. met stad 1. We kijken dus in de eerste kolom en merken op dat weg $1 \to 2$ de kortste is. We merken deze tak door er een kadertje rond te tekenen. De totale afstand van 1 naar 2 is bijgevolg gelijk aan 0 (afstand van $1 \to 1$) + 5 (afstand van $1 \to 2$). In kolom 2 zetten we bijgevolg de afstand '5'. Tevens schrappen we alle wegen die op 2 eindigen. Tabel 3.5 toont het resultaat na de eerste stap van het algoritme.

Tabel 3.5 Na de eerste iteratie.

0	5						
1	2	3	4	5	6	7	8
12 5	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	73 7	85 8
13 9	23 3	~~32 3~~	43 2	56 1	65 1	74 7	86 10
	24 8	34 2	46 3	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	26 11						

Op de ranglijst van steden staan nu stad 1 (afstand 0) en stad 2 (afstand 5). We zoeken nu alle steden die rechtstreeks in verbinding staan met één van beide al gevonden steden. Dit zijn de steden 3 (via twee routes!), 4, 5 en 6. We moeten bijgevolg volgende sommen vergelijken:

$$\begin{aligned}
&\text{Afstand } 1 \to 1 + \text{lengte van tak } 1 \to 3 = 0+9 = 9\\
&\text{Afstand } 1 \to 2 + \text{lengte van tak } 2 \to 3 = 5+3 = 8\\
&\text{Afstand } 1 \to 2 + \text{lengte van tak } 2 \to 4 = 5+8 = 13\\
&\text{Afstand } 1 \to 2 + \text{lengte van tak } 2 \to 5 = 5+10 = 15\\
&\text{Afstand } 1 \to 2 + \text{lengte van tak } 2 \to 6 = 5+11 = 16
\end{aligned}$$

De kleinste som is 8. Stad 3 is bijgevolg het dichtst gelegen bij 1 (na de al gevonden steden 1 zelf en 2). De kortste route is $1 \to 2 \to 3$. We plaatsen een kadertje rond '23 3' in tabel 3.6, plaatsen een '8' boven de derde kolom, en schrappen alle wegen die op een 3 eindigen.

Tabel 3.6 Na de tweede iteratie.

0	5	8					
1	2	3	4	5	6	7	8
12 5	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	85 8
~~13 9~~	23 3	~~32 3~~	~~43 2~~	56 1	65 1	74 7	86 10
	24 8	34 2	46 3	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	26 11						

Een belangrijke opmerking vooraleer we verder rekenen. Eigenlijk hadden we onszelf heel wat werk kunnen besparen bij het zoeken van de kleinste som. Voor de overgang van tabel 3.5 naar tabel 3.6 zochten we *alle* sommen gebruik makend van kolommen 1 en 2. We hadden echter kunnen volstaan met *één som per kolom*, nl. het totaal van het getal in de eerste rij boven de stad en de lengte van de kortste route in die kolom. Voor de volgende stappen zullen we hiervan gebruik maken.

De lijst met gevonden steden bevat nu 1, 2 en 3. We zoeken opnieuw alle rechtstreeks verbonden steden, en kijken dus in de respectievelijke kolommen. Kolom 1 hoeven we niet meer te bekijken: alle wegen zijn al omkaderd of geschrapt. We bekijken kolom 2. De kleinste nog niet geschrapte weg heeft lengte 8 (nl. weg $2 \to 4$). We onthouden $5 + 8 = 13$. In de derde kolom blijkt de kortste weg 2 lang te zijn. De som die we hier maken is bijgevolg $8+2 = 10$. Hieruit volgt ons besluit: de volgende stad op ons lijstje wordt stad 4 op een afstand van 10 van stad 1 verwijderd. We maken alle noodzakelijke aanvullingen, schrappingen en omkaderingen. . . en bekomen zo tabel 3.7.

Tabel 3.7 Na de derde iteratie.

0	5	8	10				
1	2	3	4	5	6	7	8
12 5	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	85 8
~~13 9~~	23 3	~~32 3~~	~~43 2~~	56 1	65 1	~~74 7~~	86 10
	~~24 8~~	34 2	46 3	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	26 11						

De volgende stappen worden zonder bijkomende uitleg getoond in tabellen 3.8 tot en met tabel 3.11. In deze laatste tabel is alles ingevuld, omkaderd of geschrapt, zodat het algoritme beëindigd kan worden.

Tot slot antwoorden we op het gevraagde. De kortste afstand van stad 1 tot stad 8 heeft lengte 22. De route vinden we als volgt. Zoek een kadertje dat eindigt op 8. Je vindt 58 8.

Tabel 3.8 Na de vierde iteratie.

0	5	8	10		13		
1	2	3	4	5	6	7	8
[12 5]	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	85 8
~~13 9~~	[23 3]	~~32 3~~	~~43 2~~	~~56 1~~	65 1	~~74 7~~	~~86 10~~
	~~24 8~~	[34 2]	[46 3]	58 8	67 5	78 12	87 12
	25 10	37 7	47 7		68 10		
	~~26 11~~						

Tabel 3.9 Na de vijfde iteratie.

0	5	8	10	14	13		
1	2	3	4	5	6	7	8
[12 5]	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	~~85 8~~
~~13 9~~	[23 3]	~~32 3~~	~~43 2~~	~~56 1~~	[65 1]	~~74 7~~	~~86 10~~
	~~24 8~~	[34 2]	[46 3]	58 8	67 5	78 12	87 12
	~~25 10~~	37 7	47 7		68 10		
	~~26 11~~						

Tabel 3.10 Na de zesde iteratie.

0	5	8	10	14	13	15	
1	2	3	4	5	6	7	8
[12 5]	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	~~85 8~~
~~13 9~~	[23 3]	~~32 3~~	~~43 2~~	~~56 1~~	[65 1]	~~74 7~~	~~86 10~~
	~~24 8~~	[34 2]	[46 3]	58 8	~~67 5~~	78 12	~~87 12~~
	~~25 10~~	[37 7]	~~47 7~~		68 10		
	~~26 11~~						

Tabel 3.11 Na de zevende iteratie.

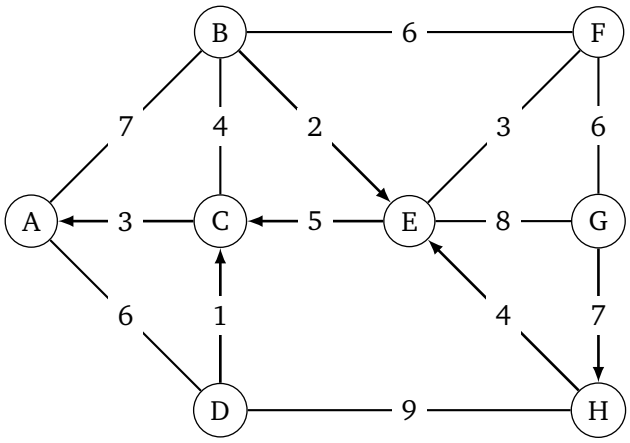
0	5	8	10	14	13	15	22
1	2	3	4	5	6	7	8
[12 5]	~~21 5~~	~~31 9~~	~~42 8~~	~~52 10~~	~~62 11~~	~~73 7~~	~~85 8~~
~~13 9~~	[23 3]	~~32 3~~	~~43 2~~	~~56 1~~	[65 1]	~~74 7~~	~~86 10~~
	~~24 8~~	[34 2]	[46 3]	[58 8]	~~67 5~~	~~78 12~~	~~87 12~~
	~~25 10~~	[37 7]	~~47 7~~		~~68 10~~		
	~~26 11~~						

We zoeken nu een kadertje dat eindigt op 5. Er is er maar ééntje, met name $\boxed{65\ 1}$. Ga na dat we zo onze weg verder achterwaarts kunnen opbouwen via $\boxed{46\ 3}$, $\boxed{34\ 2}$, $\boxed{23\ 3}$ en $\boxed{12\ 5}$. De kortste route is dus $1 \to 2 \to 3 \to 4 \to 6 \to 5 \to 8$. Analoog kan je uit deze tabel de afstand en het pad zoeken van 1 naar eender welke stad.

3.4 Oefeningen

Oefening 3.1

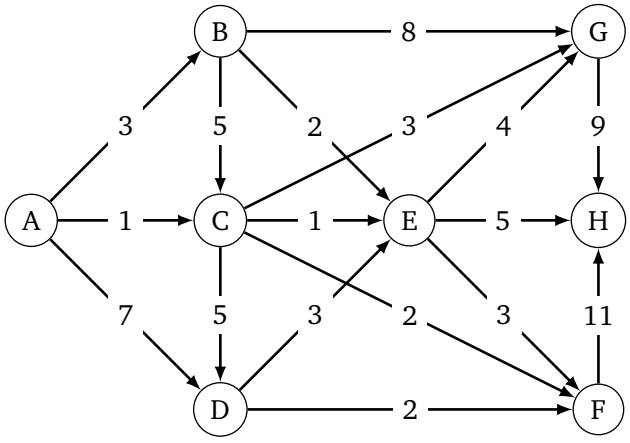
Figuur 3.5 toont een netwerk. De aangegeven getallen bij elke verbinding zijn het aantal km tussen beide knooppunten. Stel een lijst op met de kortste afstanden vanuit *het punt C* naar elk ander punt. Geef ook telkens de kortste route aan. Gebruik de methode van Dijkstra.



Figuur 3.5 Netwerk bij oefening 3.1

Oefening 3.2

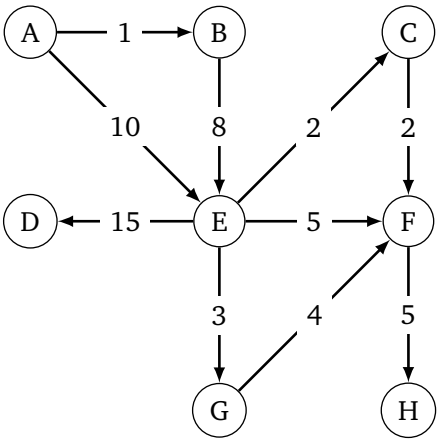
Zelfde vraag als vorige oefening voor figuur 3.6.



Figuur 3.6 Netwerk bij oefening 3.2

Oefening 3.3

Gebruik de methode van Dijkstra om het kortste pad te bepalen van knooppunt A naar de andere knooppunten van het netwerk in figuur 3.7. Geef ook aan over welke knooppunten dat pad loopt.

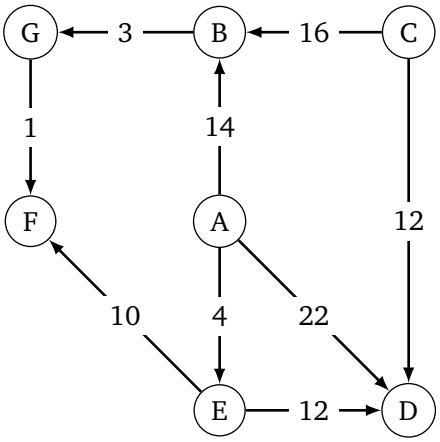


Figuur 3.7 Netwerk bij oefening 3.3

.

Oefening 3.4

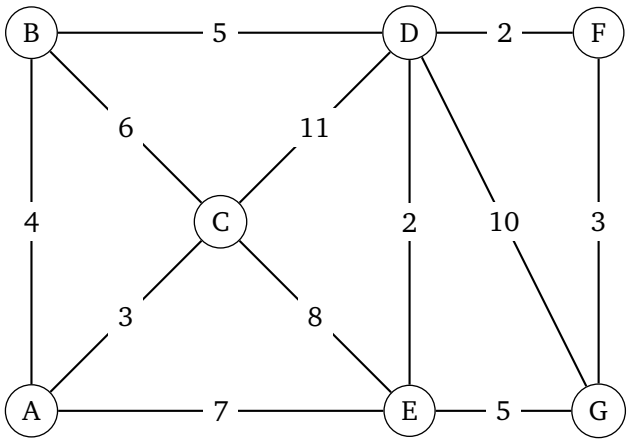
Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 3.8 te berekenen.



Figuur 3.8 Netwerk bij oefening 3.4

Oefening 3.5

Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 3.9 te berekenen.



Figuur 3.9 Netwerk bij oefening 3.5

Oplossingen

Oplossing 1.1 $1 \to 2 \to 5 \to 5$ of $1 \to 4 \to 7 \to 8$

Oplossing 1.2 $1 \to 2 \to 4 \to 7 \to 10$ of $1 \to 3 \to 4 \to 7 \to 10$

Oplossing 1.3 $1 \to 4 \to 7 \to 9$

Oplossing 2.1 Enkele paden: $A \to D \to C \to B \to E$; $C \to B \to E \to D \to A$; $D \to C \to B \to E$

Oplossing 2.2 Gewichtenmatrix $D^{(6)}$ en bijhorende pointermatrix P:

$$D^{(6)} = \begin{bmatrix} 0 & 6 & 3 & 7 & 4 & 6 \\ 2 & 0 & 5 & 9 & 6 & 8 \\ 5 & 3 & 0 & 4 & 1 & 3 \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & 11 & 8 & 3 & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 3 & 0 & 6 & 3 & 5 \\ 0 & 0 & 1 & 6 & 3 & 5 \\ 2 & 0 & 0 & 6 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden: $2 \to 1 \to 3 \to 5 \to 6$ met lengte 8; $5 \to 1 \to 3 \to 2 \to 4$ met lengte 3

Oplossing 2.3

$$D^{(5)} = \begin{bmatrix} 0 & 9 & 10 & 24 & 21 \\ 9 & 0 & 19 & 33 & 30 \\ 2 & 11 & 0 & 14 & 11 \\ 7 & 16 & 5 & 0 & 3 \\ 4 & 13 & 2 & 3 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 1 & 5 & 3 \\ 0 & 1 & 0 & 5 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden: $4 \to 5 \to 3 \to 1 \to 2$ met lengte 16; $3 \to 5 \to 4$ met lengte 14

Oplossing 3.1

Stad	Kortste afstand vanuit C	Route
A	3	$C \to A$
B	4	$C \to B$
C	0	
D	9	$C \to A \to D$
E	6	$C \to B \to E$
F	9	$C \to B \to E \to F$
G	14	$C \to B \to E \to G$
H	18	$C \to A \to D \to H$

Oplossing 3.2

Stad	Kortste afstand vanuit C	Route
A	∞	
B	∞	
C	0	
D	5	$C \to D$
E	1	$C \to E$
F	2	$C \to F$
G	3	$C \to G$
H	6	$C \to E \to H$