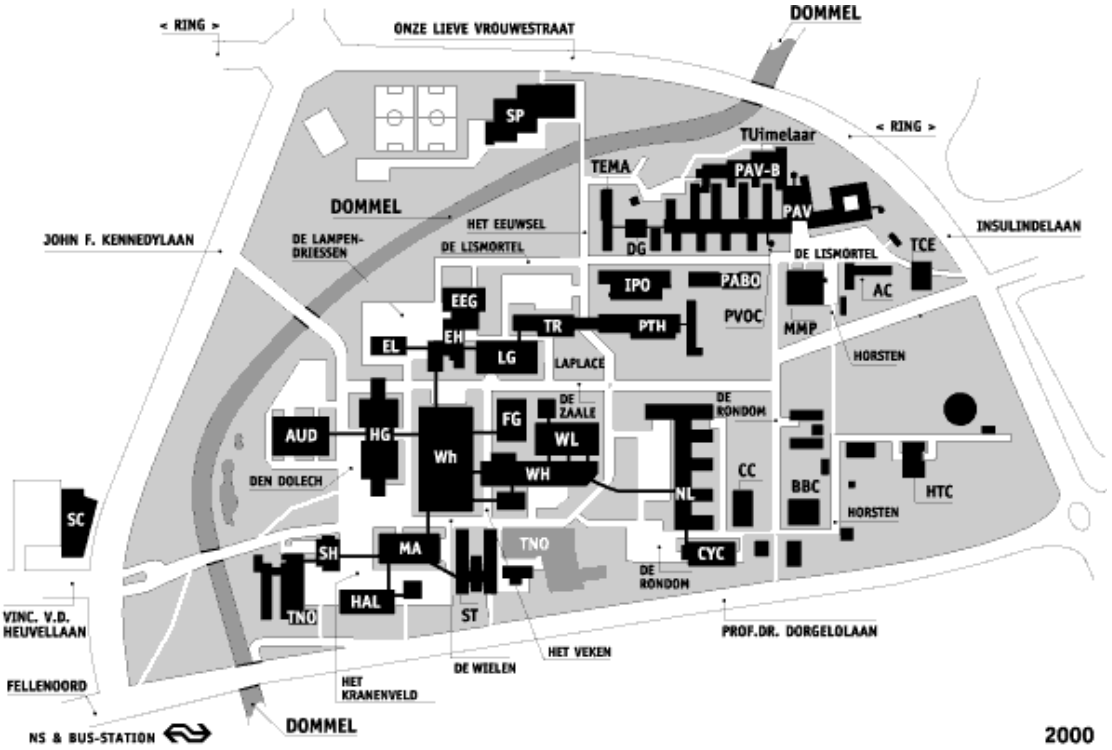
Kun je me de kortste weg vertellen?



Inhoudsopgave

1 Grafen **2**
- 1.1 Wat is een graaf? . . . 2
- 1.2 Opgaven . . . 4

2 Kortste bomen **6**
- 2.1 Het 'Greedy' Algoritme . . . 7
  - 2.1.1 Voorbeeld van het 'Greedy' Algoritme. . . . 8
- 2.2 Bewijs van het 'Greedy' Algoritme . . . 10
- 2.3 Opgaven . . . 11
- 2.4 Algoritme van Prim-Dijkstra . . . 12
  - 2.4.1 Voorbeeld van het Algoritme van Prim-Dijkstra . . . 13
- 2.5 Opgaven . . . 16

3 Kortste paden **18**
- 3.1 Een algoritme om een kortste pad te vinden . . . 18
  - 3.1.1 Voorbeeld van een kortste pad. . . . 22
- 3.2 Opgaven . . . 22

4 Kortste routes **24**
- 4.1 Handelsreizigers . . . 24
  - 4.1.1 Voorbeelden van het handelsreizigersprobleem. . . . 24
- 4.2 Moeilijkheden van het handelsreizigersprobleem . . . 29
- 4.3 Algoritmes voor het benaderen van het optimum voor het handelsreizigersprobleem . . . 30
  - 4.3.1 Het beste-buur-algoritme . . . 30
  - 4.3.2 Het invoegingsalgortime . . . 31
  - 4.3.3 Uitwisselingsalgoritmen . . . 34
- 4.4 Opgaven . . . 35

5 Gemengde Opgaven **36**

Combinatorische Optimalisering

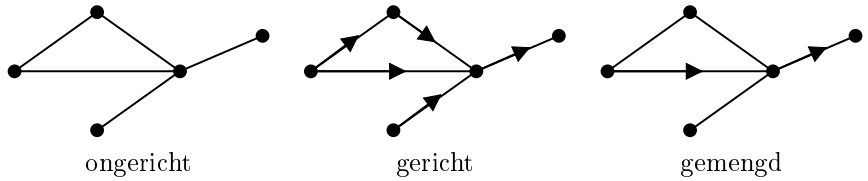
28 november 2000

1 Grafen

Dit boekje is een bewerking van de masterclass combinatorische optimalisering: "Kun je me de kortste weg vertellen?", gegeven op 5 maart 1999 door Prof.dr.J.K. Lenstra in samenwerking met Dr.ir. C. Hurkens.

1.1 Wat is een graaf?

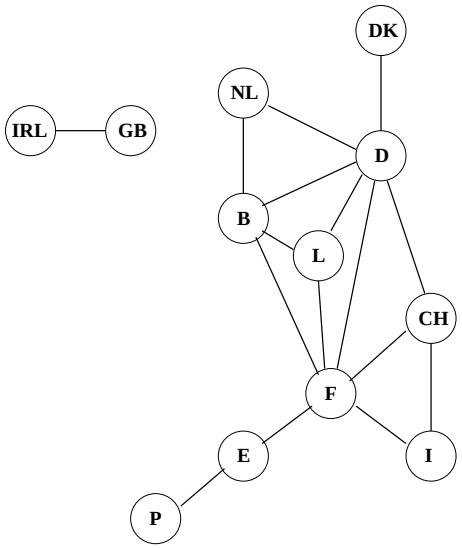
Een graaf is een tekening die alleen bestaat uit punten en verbindingslijnen. De verbindingslijnen noemen we kanten. (Andere benamingen die je wel eens tegenkomt zijn knooppunten en wegen.) Een kant verbindt twee punten en een kant tussen de punten i en j geven we aan met ij. Soms heeft een kant een richting en loopt van i naar j. Dit schrijven we als $i \to j$. In figuur 1 zie je een **ongerichte** graaf, een **gerichte** graaf en een **gemengde** graaf. Een **gerichte** graaf is een graaf met pijlen in de kanten. Deze pijlen geven een verband tussen de (knoop)punten aan, b.v. 'Wie is de afstammeling van wie?' of 'Wie heeft gewonnen van wie?' Een **gemengde** graaf is een combinatie van een gerichte en een ongerichte graaf.



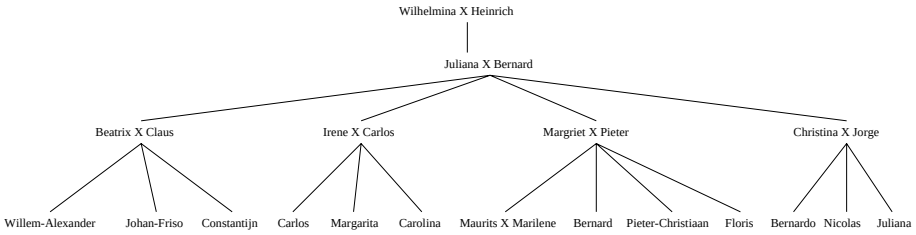
Figuur 1: Grafen

Met grafen kun je allerlei situaties beschrijven. Het gaat dan om situaties waarin er tussen **objecten**, voorgesteld door de punten, **paarsgewijze relaties** bestaan, voorgesteld door de kanten. In figuur 2 zie je bijvoorbeeld een graaf waarvan de punten twaalf landen in West-Europa zijn. Een kant ij geeft aan dat de landen i en j aan elkaar grenzen. In figuur 3 zie je een gerichte graaf die de afstamming van een persoon P tot in het derde voorgeslacht uitbeeldt. Een

kant $i \to j$ geeft hier aan dat j een kind is van i.



Figuur 2: Grenzen in West-Europa



Figuur 3: Stamboom

Een graaf die een bepaalde situatie beschrijft noemen we een **model** van die situatie. In een model neem je alleen de gegevens van een situatie op die belangrijk zijn voor het probleem dat je wilt oplossen.

Vragen die je bijvoorbeeld met grafen kunt oplossen zijn:

Als de punten van de graaf plaatsen zijn en de kanten zijn wegen tussen die plaatsen,
(1) hoe vind je dan een kortste netwerk dat alle plaatsen met elkaar verbindt?
(2) hoe vind je dan een kortste weg tussen twee gegeven plaatsen?

(3) hoe vind je een kortste route die langs alle plaatsen gaat?

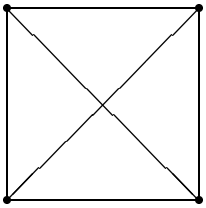
Je zult zien dat je, als je deze vragen kunt beantwoorden, ook problemen kunt oplossen die op het eerste gezicht niets met wegennetwerken te maken hebben. Je kunt namelijk veel andere problemen voorstellen als een wegennetwerk. Deze kun je dan ook weer met behulp van een graaf proberen op te lossen. Het is zelfs zo dat verschillende situaties tot hetzelfde model kunnen leiden.

1.2 Opgaven

1. Verzin zelf 3 voorbeelden van situaties die je als graaf kunt weergeven en teken die grafen.

2. Kun je situaties verzinnen die tot hetzelfde model leiden als de grafen die je bij 1. hebt getekent?

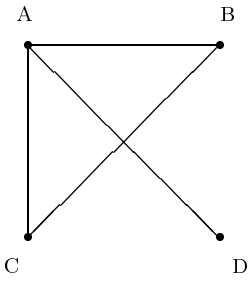
3. Een graaf heet **volledig** als elk punt in de graaf verbonden is met alle andere punten in de graaf. Hieronder zie je de volledige graaf met 4 punten.

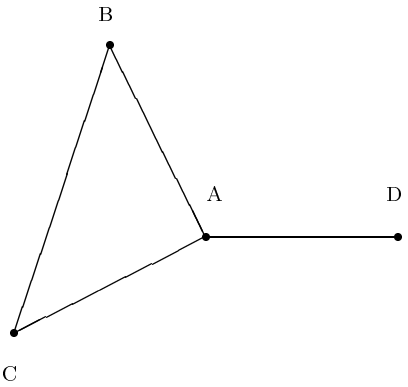


   (a) Teken de volledige graaf met 6 punten.

   (b) Hoeveel kanten heeft een volledige graaf met n punten?

4. Hieronder zie je twee gelijke grafen.

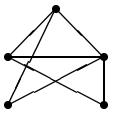




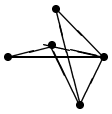
   Twee grafen zijn gelijk als ze dezelfde (knoop)punten hebben en als tussen dezelfde punten evenveel kanten lopen die, indien we het over een gerichte

graaf hebben, dezelfde richting hebben.

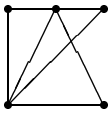
Welke van de volgende grafen komen met elkaar overeen?



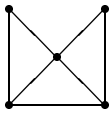
A



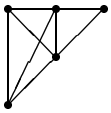
B



C



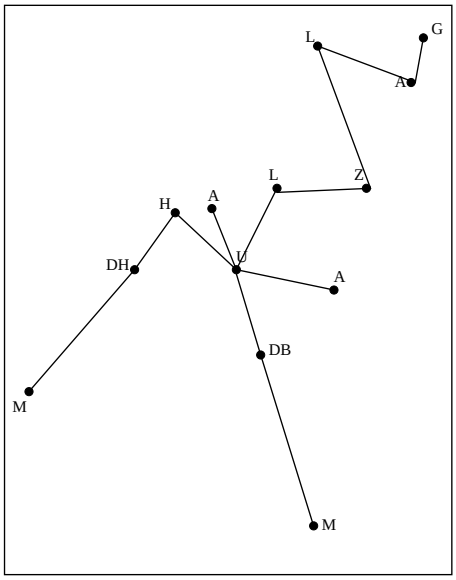
D



E

2 Kortste bomen

Hoe vind je een kortste wegennetwerk dat alle plaatsen met elkaar verbindt? Verbinden wil zeggen dat je vanuit elk punt ieder ander punt(eventueel via andere punten) kunt bereiken. Als voorbeeld nemen we een probleem met dertien plaatsen: Amsterdam en de twaalf Nederlandse provinciehoofdsteden. Een mogelijk netwerk zie je in figuur 4.



Figuur 4: Een wegennetwerk in Nederland

Hoe bepaal je nu het kortste netwerk? We maken hiervoor een model waarbij we een ongerichte graaf gebruiken. De steden stellen we als punten voor en de wegen tussen die steden als kanten. De afstanden tussen twee steden i en j noemen we dan c_{ij}. Kant ij heeft dus een **lengte** c_{ij}.

Voorbeeld
Als i = Amsterdam en j = Den Haag, dan is c_{ij} de afstand tussen Amsterdam en Den Haag.

Ook als we het over andersoortige problemen hebben, b.v. leeftijdsverschillen, hebben we het nog steeds over **lengte** c_{ij}. We nemen aan dat $c_{ij} \geq 0$ voor alle ij: de lengtes zijn niet-negatief.

Een reeks kanten ij, jk, kl, lm, ... die op elkaar aansluiten noemen we een **pad**. Een gesloten pad, waarvan het beginpunt en het eindpunt samenvallen,

heet een **circuit**.

Voorbeeld
Als i = Amsterdam, j = Den Haag, k = Utrecht en l = Lelystad, dan is ij, jk, kl, li het circuit Amsterdam-Den Haag-Utrecht-Lelystad-Amsterdam.

Je kunt meerdere circuits binnen een graaf hebben.

Een netwerk dat alle punten met elkaar verbindt en geen circuits bevat heet een **boom**. In een boom is er tussen elk tweetal punten precies één pad. En een boom op 13 punten bestaat uit 12 kanten.

Het netwerk waar je naar op zoek bent bevat natuurlijk geen circuits.

Vraag:
Waarom kan zo'n netwerk geen circuits hebben?

2.1 Het 'Greedy' Algoritme

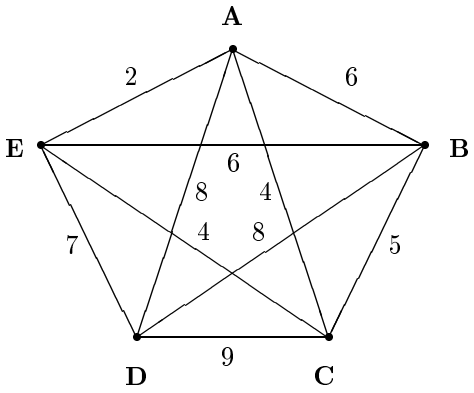
Je zoekt dus naar een boom die zo is opgebouwd dat als je alle lengtes van zijn kanten bij elkaar optelt, je een zo klein mogelijk getal krijgt. We noemen dit een kortste boom. Hiervoor bestaat een **algoritme**. (Een algoritme is een soort recept van hoe je iets moet doen). Dit algoritme ziet er als volgt uit:

1. Kies een willekeurig punt i in je graaf als begin van je netwerk.

2. Als je netwerk nog niet alle punten bevat, bepaal dan twee punten k en j, met j in het netwerk en k niet, waarvoor c_{jk} minimaal is. Voeg de kant kj aan het netwerk toe. Als er meerder mogelijkheden zijn dan kies je er een willekeurig.

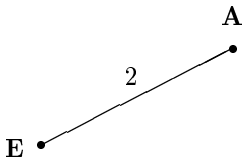
Dit algoritme noemen we het 'Greedy' Algoritme.

2.1.1 Voorbeeld van het 'Greedy' Algoritme.

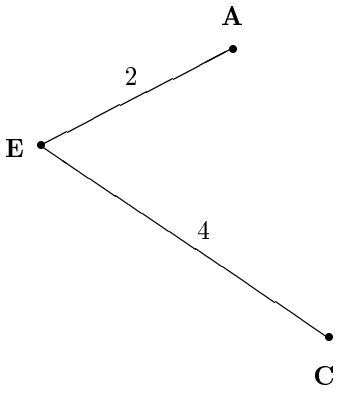


Stap 1.
Kies het "willekeurige punt" A.

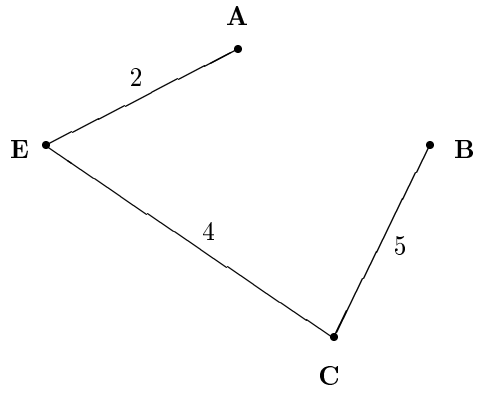
Stap 2.
De kant AE heeft de kleinste **lengte**. Deze kant moet je dus toevoegen.



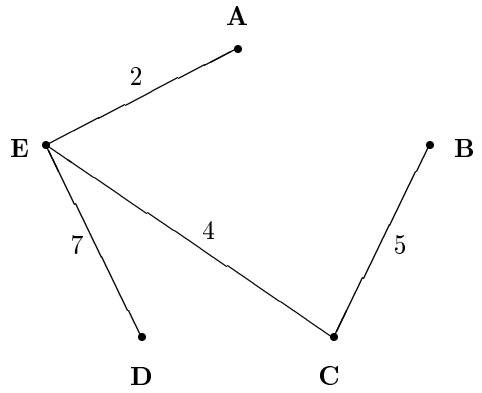
Stap 3.
Vervolgens pakken we de kant met de op één na kleinste **lengte**. We hebben er twee, namelijk AC of CE met **lengte** 4. Kies bijvoorbeeld CE.



Stap 4.
Nu kunnen we AC niet meer in onze boom opnemen, want dat zou een circuit opleveren. De daaropvolgende kant met de kleinste **lengte** is BC met **lengte** 5.

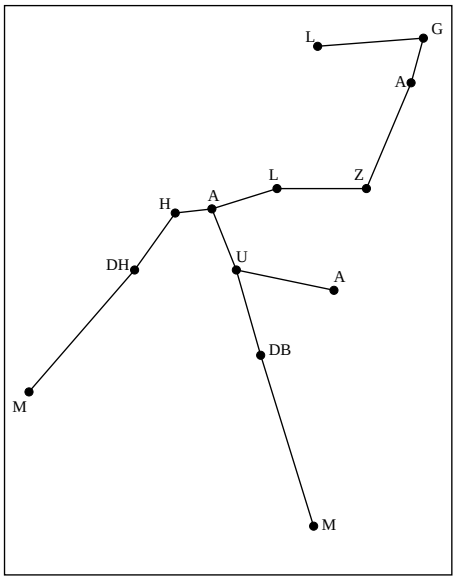


Stap 5.
Hierna zouden de kanten AB en BE in aanmerking komen, maar deze leveren ook beide een circuit, dus kiezen we kant DE met **lengte** 7. Hiermee hebben we een kortste opspannende boom gecreëerd met **lengte** 18.



Opmerking:
Als we bij de tweede Stap AC hadden genomen in plaats van CE dan hadden we een andere kortste opspannende boom gekregen, maar de **lengte** is dan nog steeds 18.

Als je dit algoritme op de steden van Nederland toepast dan krijg je de oplossing die je hieronder ziet.



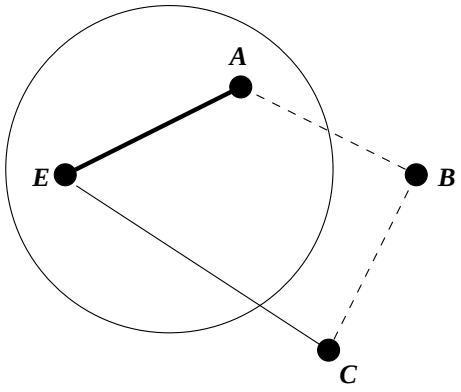
Figuur 5: Kortste wegennetwerk in Nederland

2.2 Bewijs van het 'Greedy' Algoritme

Geeft dit algoritme, dat het "Greedy" Algoritme genoemd wordt, nu echt de kortste boom? Ja, en dit kunnen we ook laten zien. Laten we nog eens bij stap (1) beginnen. Stap 1 is het kiezen van een punt, i. Dit punt zit zeker in de kortste boom, want alle punten van de graaf moeten in de kortste boom zitten. Nu kies ik een punt j uit de graaf dat de kortste afstand heeft tot punt i van alle punten in de graaf. De boom bestaande uit die twee punten is de kortst mogelijke boom tussen die twee punten binnen onze graaf. Vervolgens kiezen we een punt k dat de kortste afstand heeft tot punt i of punt j van alle punten in de graaf, uitgezonderd i en j natuurlijk, want die zitten al in je boom. Na toevoegen van punt k met de bijbehorende kortste kant is de boom die uit drie punten bestaat nog steeds de kortste boom tussen die drie punten. Zo kunnen we verder gaan, totdat we alle punten hebben gehad. De enige voorwaarde waar je aan moet houden is dat je geen circuits mag maken.(zie voorbeeld 2.1.1.)

Stel eens dat je kijkt naar het stukje boom dat in stap 2 van je algoritme af is. Dit is een netwerk op een gedeelte van alle punten in de graaf (een **deelverzameling** van de verzameling van alle punten in je graaf). In het voorbeeld zijn dit de punten A en E met kant AE. Dit netwerk zit ook in een kortste boom op je hele graaf. We kiezen nu op grond van het algoritme de twee punten C en E; C zit nog niet in je netwerk, E wel. Veronderstel nu eens dat de kant CE niet in je kortste boom zit. Dan moet C via een ander pad met je netwerk verbonden zijn. In dit pad zit een kant, bijvoorbeeld AB, met A in je netwerk

en B niet, waarvan we zeker weten dat $c_{AB} \geq c_{CE}$, anders konden we E en C niet zo kiezen in stap 2, dus mogen we c_{AB} door c_{CE} vervangen.



Figuur 6: Waarom de boom niet korter kan

2.3 Opgaven

1. Teken alle verschillende kortste bomen bij voorbeeld 2.1.

2. In de volgende tabel zie je de afstanden (in mijlen) tussen zes plaatsen in Ierland. Teken een graaf en gebruik het "Greedy" Algoritme om een kortste boom te vinden die deze plaatsen verbindt.

	Athlone	Dublin	Galway	Limerick	Sligo	Wexford
Athlone	-	78	56	73	71	114
Dublin	78	-	132	121	135	96
Galway	56	132	-	64	85	154
Limerick	73	121	64	-	144	116
Sligo	71	135	85	144	-	185
Wexford	114	96	154	116	185	-

3. Een inbraakalarm is weergegeven in de vorm van een graaf waarvan de kanten gemaakt zijn van zeer kostbaar koperdraad. Elke kant heeft een verschillende waarde (=**lengte**). Het alarm gaat af als de graaf niet verbonden is. Een inbreker wil zoveel mogelijk van het kostbare koperdraad stelen. Welke kanten moet hij weghalen om een maximale buit binnen te krijgen?

4. Laat zien hoe je het "Greedy" Algoritme moet aanpassen om een langste opspannende boom te creëren.

5. Maak bij de volgende tabellen een graaf en zoek hierbij een kortste en een langste opspannende boom.

	Berlijn	Londen	Madrid	Moscou	Parijs	Rome
Berlijn	-	7	15	11	7	10
Londen	7	-	11	18	3	12
Madrid	15	11	-	27	8	13
Moscou	11	18	27	-	18	20
Parijs	7	3	8	18	-	9
Rome	10	12	13	20	9	-

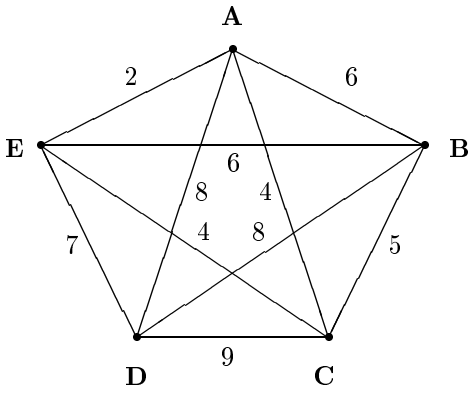
	Aberd.	Edinb.	F.W.	Glasg.	Inv.	Perth
Aberdeen	-	120	147	142	104	81
Edinburgh	120	-	132	42	157	45
Fort William	147	132	-	102	66	105
Glasgow	142	42	102	-	168	61
Inverness	104	157	66	168	-	112
Perth	81	45	105	61	112	-

2.4 Algoritme van Prim-Dijkstra

Het 'Greedy' Algoritme is makkelijk met de hand toe te passen als je een kleine graaf hebt. Om het door een computer te laten doen is echter moeilijker. Je wilt steeds de kanten ordenen op aflopende lengte en je moet steeds kijken of er geen circuit ontstaat. Door een kleine aanpassing in het "Greedy" Algoritme is dit op te lossen. Het algoritme dat dan overblijft is beter bekend als het "Algoritme van Prim-Dijkstra" en loopt als volgt:

1. Je maakt eerst een tabel van alle kanten tussen de knopen in je graaf.
2. Neem nu een willekeurige knoop in je graaf. Deze komt in de boom die je wilt creëren. Stel je kiest B.
3. Verwijder nu rij B uit je tabel en zoek in kolom B de kleinste waarde op.
4. Kijk welke knoop bij deze kleinste waarde hoort. Stel dat is knoop C.
5. Voeg dan kant BC aan je boom toe.
6. Verwijder rij C uit je tabel en zoek nu in de kolommen B en C naar de kleinste waarde.
7. Herhaal nu vanaf stap 4, waarbij je steeds bij één kolom meer naar de kleinste waarde moet zoeken.
8. Als je geen rijen meer over hebt in je tabel heb je een kortste boom gevonden.

2.4.1 Voorbeeld van het Algoritme van Prim-Dijkstra



Stap 1.
Bij de graaf in de bovenstaande figuur hoort de volgende tabel:

	A	B	C	D	E
A	-	6	4	8	2
B	6	-	5	8	6
C	4	5	-	9	4
D	8	8	9	-	7
E	2	6	4	7	-

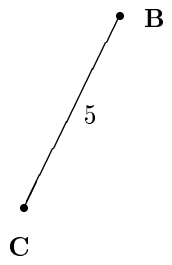
Stap 2.
Kies B om je boom mee te beginnen.

- **B**

Stap 3.
Ik moet nu dus **rij** B uit de tabel verwijderen en de kleinste waarde in **kolom** B opzoeken.

	A	B	C	D	E
A	-	6	4	8	2
C	4	[5]	-	9	4
D	8	8	9	-	7
E	2	6	4	7	-

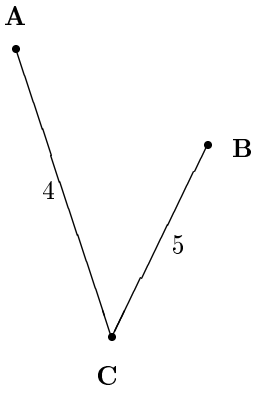
Stap 4 en 5.
Uit de tabel blijkt nu dat BC de kant is met de kleinste **lengte**, dus voeg je kant BC en knoop C aan je boom toe.



Stap 6.
Nu moet je **rij** C uit de tabel halen en de kleinste waarde in **kolom** B of **kolom** C opzoeken.

	A	B	C	D	E
A	-	6	[4]	8	2
D	8	8	9	-	7
E	2	6	[4]	7	-

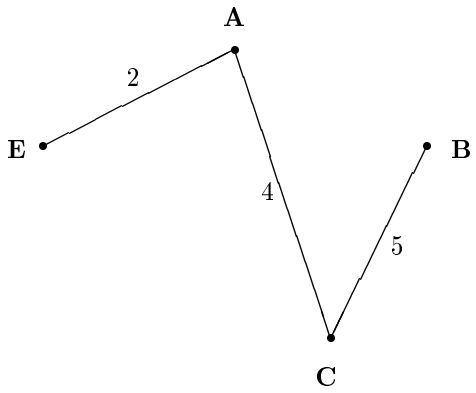
Stap 7
CA en CE zijn de volgende kanten met de kleinste **lengte**, waardoor je boom groter kan worden. Kies daar ééntje van, CA. De boom wordt uitgebreid met kant CA en knoop A.



Stap 8.
Rij A haal je uit de tabel en de kleinste waarde die je vindt onder de **kolommen** B, C en A is die in rij E, nl.: 2.

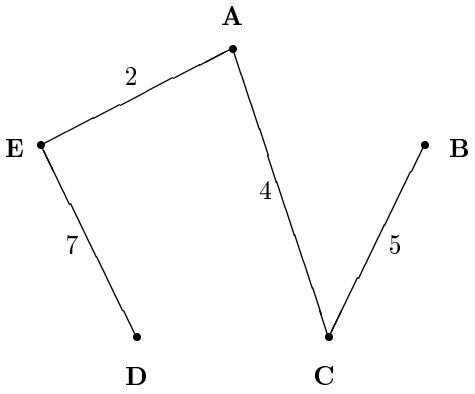
	A	B	C	D	E
D	8	8	9	-	7
E	2	6	4	7	-

Stap 9.
Vergroot je boom met kant AE en knoop E en haal **rij** E uit je tabel.

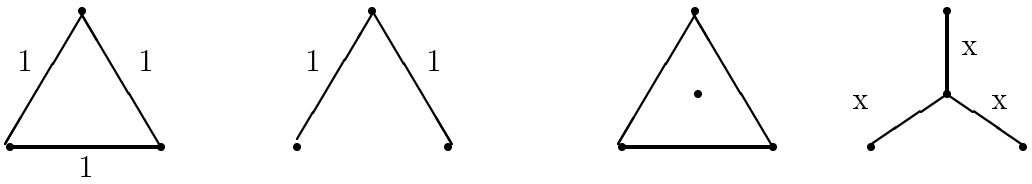


	A	B	C	D	E
D	8	8	9	-	7

Stap 10.
In de **kolommen** A, B, C en E vind je nu onder E de kleinste waarde. De laatste kant die je aan mijn boom plakt is dus kant ED, waarmee je dan tegelijk als laatste knoop D toevoegt. Alle knopen van de originele graaf komen voor in de boom en dit is een kortste opspannende boom van de graaf.



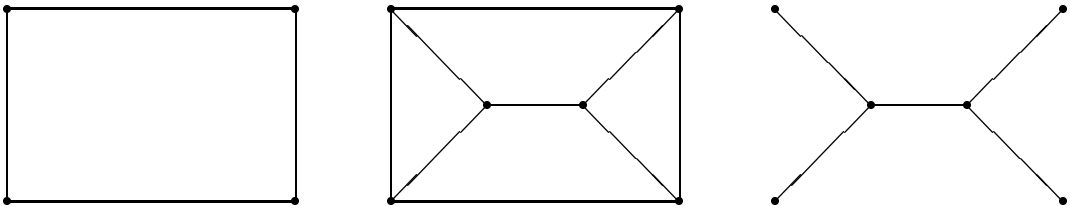
N.B.:
Er zit nog een addertje onder het gras. De boom bestaat uit kanten van de oorspronkelijke graaf. Dat wil zeggen: je mag alleen wegen aanleggen die gegeven plaatsen, dus werkelijke punten uit je graaf, met elkaar verbinden. Als de plaatsen nu eens liggen op de hoeken van een gelijkzijdige driehoek, dan bestaat de boom uit twee van de drie zijden. Maar het is voordeliger een nieuwe plaats te creëren, in het middelpunt van de driehoek, en die met de drie gegeven plaatsen te verbinden. Door punten toe te voegen kun je ook de boom in figuur 5 nog flink verbeteren. Je zoekt dan naar een kortste **Steiner-boom**. Hieronder zie je de Steinder-boom bij een gelijkzijdige driehoek.



2.5 Opgaven

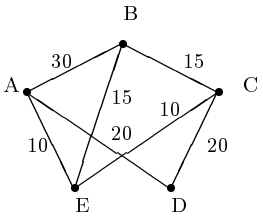
1. Los opgaven 2 t/m 5 uit 2.3 op met behulp van het "Algoritme van Prim-Dijkstra".
2. Bepaal de lengte van de Steiner-boom in het voorbeeld van de gelijkzijdige driehoek.

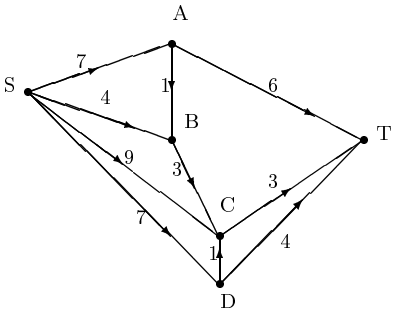
3. Hieronder zie je een graaf op vier punten, met daarnaast de overstap naar de bijbehorende Steiner-boom. Bepaal de positie van de punten.



3 Kortste paden

Er bestaat ook een algoritme waarmee je het kortste pad tussen twee plaatsen kunt vinden. Dat algoritme is vrij ingewikkeld en moet je alleen gebruiken bij grote grafen. Het kortste pad in de linkergraaf tussen A en B is AEB, dat zie je zo. Maar een kortste pad tussen S en T in de rechtergraaf is niet zomaar te zien.

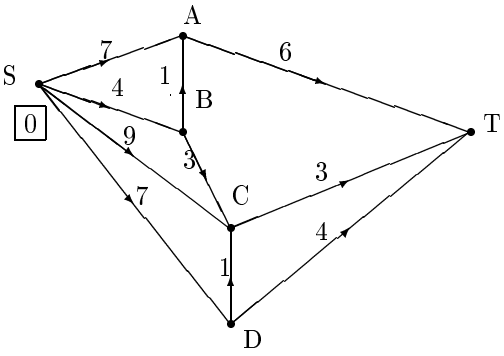




Voordat we het algoritme formuleren passen we het eerst toe met de rechtergraaf als voorbeeld.

3.1 Een algoritme om een kortste pad te vinden

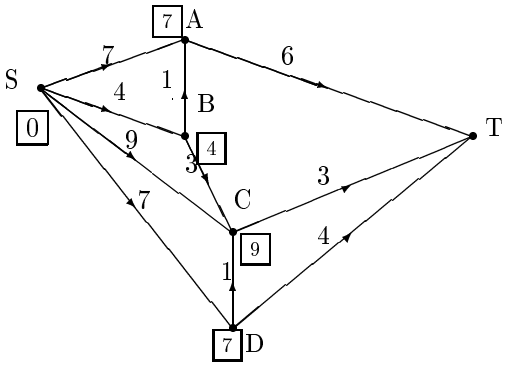
In het hieronderstaande netwerk zoeken we het korste pad van S naar T. Om niet in de war te raken door alle getalletjes die in je graaf komen te staan, stop je alle gegevens die je verzakelt tijdens het zoeken naar het kortste pad in een tabel.



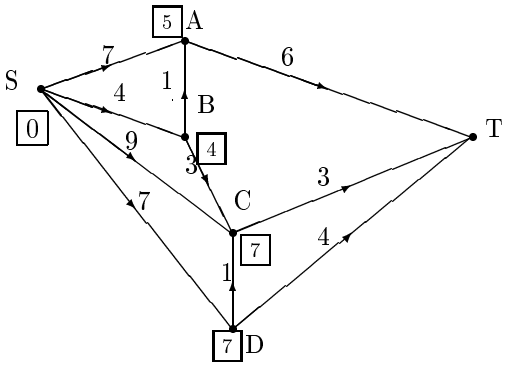
S noemen we punt **1**. De lengte tot nu toe geef je aan met $L(1)$. $L(1)$ is dus 0. De eerste rij in je tabel geeft je de naam van de knoop of knopen die net een lengte hebben gekregen. De eerste rij in je tabel wordt in dit geval dus rij S.

knopen	S	A	B	C	D	T
S	0	7	4	9	7	$\cdots$

Nu kijk je naar knopen die je via één kant vanuit S kunt bereiken. In dit geval zijn dat A, B, C en D. Deze krijgen nu elk een merk, een voorlopige bovengrens van S naar die punten, $M(A) = 7$, $M(B) = 4$, $M(C) = \overline{9 \text{ en } M(D)} = 7$.

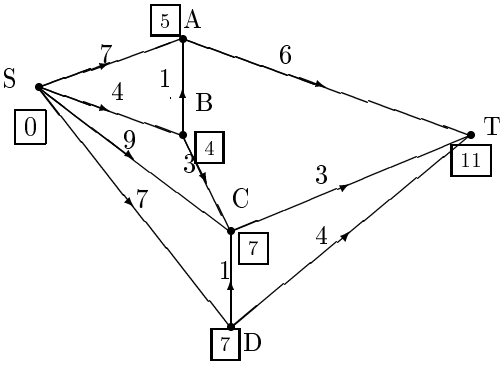


B is de knoop met het kleinste merk. B krijgt nu $L(B) = 4$ en de tweede rij in je tabel wordt rij B. Vervolgens kijk je naar de knopen die direct vanuit B bereikbaar zijn (A en C). De merken van deze knopen worden nu aangepast, $M(A) = 5$ en $M(C) = 7$. Dit verwerk je gelijk in je nieuwe tabel.



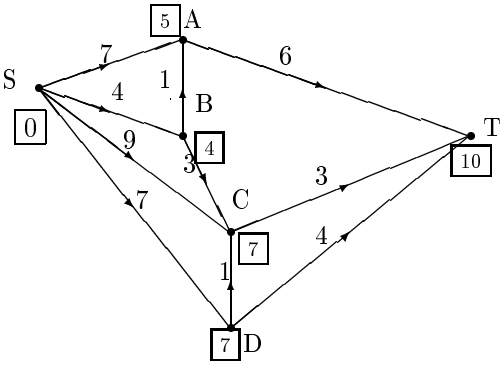
knopen	S	A	B	C	D	T
S	0	7	4	9	7	$\cdots$
B		5	4	7	7	$\cdots$

A heeft nu het kleinste merk, dus $L(A) = 5$. Dan kijk je naar de knopen direct bereikbaar vanuit A. Dit is alleen T. T krijgt $M(T) = 5 + 6 = 11$.



knopen	S	A	B	C	D	T
S	0	7	4	9	7	$\cdots$
B		5	4	7	7	$\cdots$
A		5		7	7	11

C en D hebben nog geen lengte. Ze hebben beide hetzelfde merk en er zijn geen knopen met een kleiner merk. De lengte van C en D wordt dus 7 en de volgende rij in je tabel wordt rij C, D. De enige direct bereikbare knoop vanuit deze punten is knoop T. Vanuit C wordt het merk van T gelijk aan 10 en vanuit D wordt het merk van T gelijk aan 11. We mogen het merk van T dus aanpassen, $M(T) = 10$. Je hebt nu gevonden:



knopen	S	A	B	C	D	T
S	$\boxed{0}$	7	4	9	7	$\cdots$
B		5	$\boxed{4}$	7	7	$\cdots$
A		$\boxed{5}$		7	7	11
C,D				7	7	10
T						$\boxed{10}$

Het kortste pad tussen S en T is $SBCT$. Hieronder vatten we het algoritme nog eens samen.

Het beginpunt van de graaf waarin je een kortste pad zoekt noemen we punt **1**. Dit punt ligt vast. Verder heeft elk punt i in de graaf een **merk** $M(i)$, de voorlopige bovengrens van een kortste pad van **1** naar i. $M(i)$ wordt in elke stap die we maken aangepast. Het punt i krijgt een **lengte** $L(i)$ als we op een gegeven moment bij punt i vanuit **1** zijn en $M(i)$ kan niet meer kleiner worden gemaakt. $M(i)$ wordt dan $L(i)$, dus:

1. We geven het beginpunt **1** een lengte $L(1) = 0$ en zeggen $j = 1$. Alle andere punten i krijgen een merk $M(i) = \infty$. Het punt j is op dit moment het laatste punt dat een lengte heeft gekregen, n.l. 0.

2. We kijken naar alle punten k die nog geen lengte hebben. Als er een kant jk bestaat waarbij $M(k) > L(j) + c_{jk}$, dan veranderen we $M(k)$ in $L(j) + c_{jk}$.

3. Nu kiezen we uit alle punten k dàt punt k^* dat het kleinste merk, $M(k^*)$, heeft en zeggen $L(k^*) = M(k^*)$ en $j = k^*$, dus j is weer het laatste punt dat een lengte heeft gekregen. Zolang er nog punten over zijn zonder lengte, herhalen we het algoritme vanaf punt 2.

Als je tijdens het toepassen van het algoritme ervoor kiest om alle gegevens die je verzamelt in een tabel te stoppen, doe je nog het volgende:

1. Boven elke kolom zet je de naam van een van de knopen in je graaf.

2. Vervolgens geef je de eerste rij de naam van een knoop of knopen die net een **lengte** hebben gekregen.

3. Zet die **lengte** in de gelijknamige kolom en omcirkel deze.

4. Om de rijen af te maken bekijk je de knopen die direct vanuit de net toegevoegde knoop bereikt kunnen worden en zetten daar de bijbehorende **merken** bij. In rij k onder kolom j moet dus $M(k) = L(j) + c_{jk}$ komen te staan.

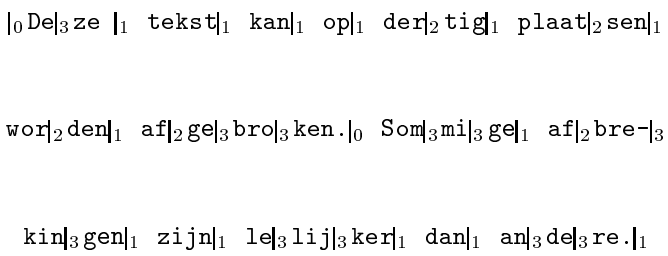
5. Nu zoek je in de net gemaakte rij naar de kleinste waarde die géén **lengte** is. De knoop die boven de kolom staat waarin deze waarde voorkomt is

de naam van je volgende rij. Dit kunnen ook meerdere knopen zijn als er twee gelijke kleinste waarden in je onderzochte rij voorkomen. Dan zet je gewoon twee namen voor je volgende rij.

6. Herhaal vanaf 2 totdat je in je eindknoop van het pad dat je zoekt bent gekomen.

3.1.1 Voorbeeld van een kortste pad.

Kortste paden heb je niet alleen bij wegennetwerken. De tekst die je nu leest is gemaakt met het tekstverwerkingsprogramma LaTeX. Om een alinea in regels in te delen gebruikt LaTeXeen kortste pad algoritme, zie figuur 7. De punten in je graaf worden dan de plaatsen in de zin waar kan worden afgebroken. Als de tekst tussen twee punten i en j op één regel past, dan hebben we daar een kant ij. Hoe vervelender de afbreking des te groter is de lengte c_{ij} van deze kant. Het kortste pad dat LaTeXvindt geeft een alinea-indeling met het grootste leesgemak.



Figuur 7: Het kortste pad geeft de mooiste alinea

3.2 Opgaven

1. Maak van de volgende tabel een gerichte graaf. Zet de juiste lengtes bij de kanten en zoek een kortste pad tussen S en T.

	S	A	B	C	D	E	T
S	-	7	13	28	-	-	-
A	-	-	4	-	25	10	-
B	-	-	-	5	6	-	-
C	-	-	-	-	-	3	-
D	-	-	-	-	-	-	5
E	-	-	-	-	-	-	12
T	-	-	-	-	-	-	-

2. Een bedrijf heeft 5 vestigingen in 5 verschillende steden: A, B, C, D en E. De reiskosten tussen deze steden zie je in onderstaande tabel. Wat is de goedkoopste route tussen elk tweetal steden?

	A	B	C	D	E
A	-	50	40	25	10
B	50	-	20	90	25
C	40	20	-	10	25
D	25	90	10	-	55
E	10	25	25	55	-

3. Maak van de volgende tabel een graaf en zoek dan vanuit S de kortste route naar elk ander punt in de graaf.

	S	A	B	C	D	E	F	T
S	-	1	3	6	-	-	-	-
A	1	-	5	-	2	4	-	-
B	3	5	-	2	-	7	3	-
C	6	-	2	-	-	-	6	-
D	-	2	-	-	-	1	-	7
E	-	4	7	-	1	-	4	5
F	-	-	3	6	-	4	-	2
T	-	-	-	-	7	5	2	-

4. Kun je met behulp van de tabel in opgave 3 ook een langste route vinden?

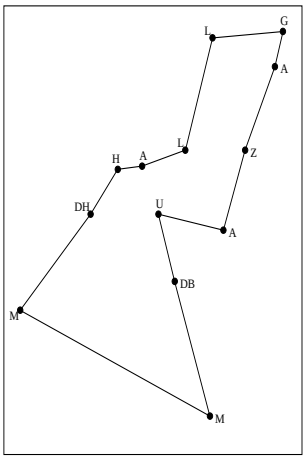
5. [1] Bewijs met **volledige inductie** (zie bijlage) dat je op de beschreven manier inderdaad een kortste pad vindt.

[1] Deze opgave is vrij pittig en kan desgewenst worden overgeslagen.

4 Kortste routes

4.1 Handelsreizigers

Een handelsreiziger moet vanuit zijn woonplaats een aantal andere steden bezoeken.'s Avonds wil hij weer zo vroeg mogelijk thuis zijn, dus moet hij een zo kort mogelijke route langs die steden zien te vinden. Hieronder zie je route van een handelsreiziger langs de provinciehoofdsteden schematisch weergegeven.



Figuur 8: Kortste route door 13 Nederlandse steden

Dit probleem kun je natuurlijk als een graaf voorstellen. De steden (ook de woonplaats van de handelsreiziger) zijn punten. De wegen tussen de steden zijn de kanten, die een lengte c_{ij} hebben die gelijk is aan het aantal kilometers dat je moet rijden tussen twee steden i en j. De vraag wordt dan een circuit te bepalen waarin je precies één keer langs ieder punt gaat, zodat de totale lengte, dus alle lengtes bij elkaar opgeteld, zo klein mogelijk is.

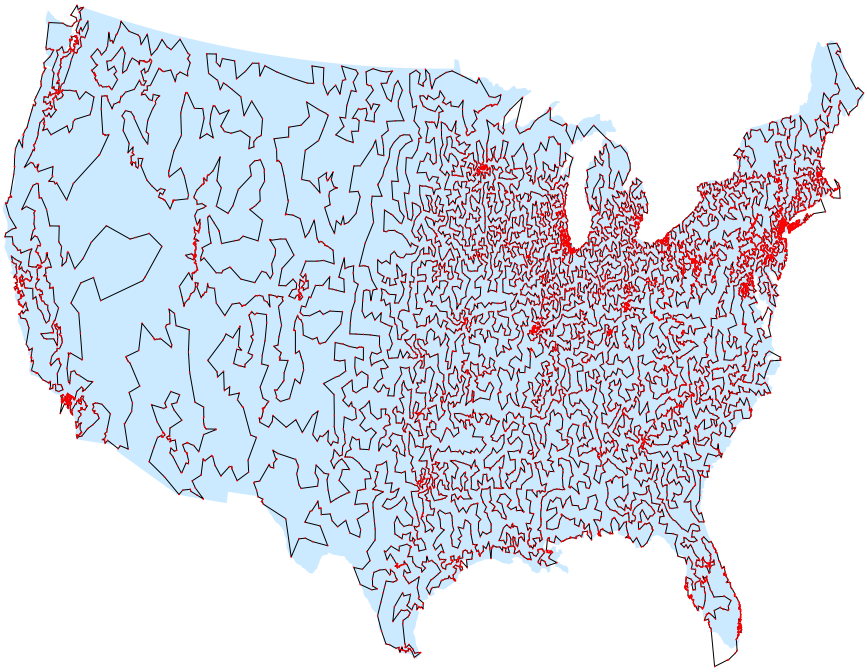
Een circuit dat elk punt van een graaf precies éénmaal bezoekt heet een **Hamiltoncircuit**. Bij het handelsreizigersprobleem zoeken we in een graaf met lengtes op de kanten naar een kortste Hamiltoncircuit.

Het handelsreizigersprobleem kun je in allerlei situaties toepassen. Hier volgen een aantal voorbeelden:

4.1.1 Voorbeelden van het handelsreizigersprobleem.

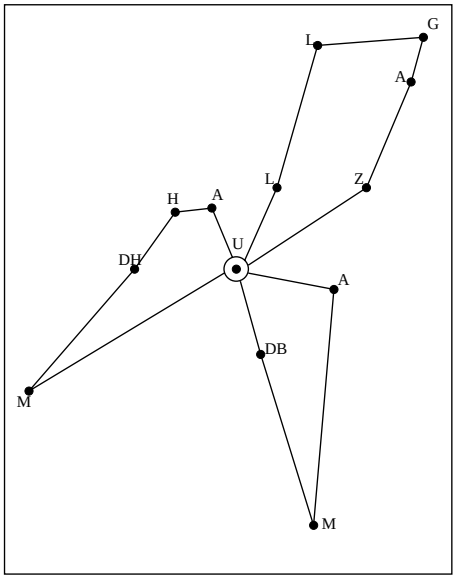
1. De handelsreiziger woont in New York en moet elke plaats in de V.S. van Amerika met tenminste 500 inwoners bezoeken. Hij mag niet verwachten

dat hij dezelfde avond weer thuis is: zijn probleem heeft 13.509 steden. Het is het grootste probleem van dit soort waarvoor een kortste route bekend is. Je ziet hem in figuur 9.



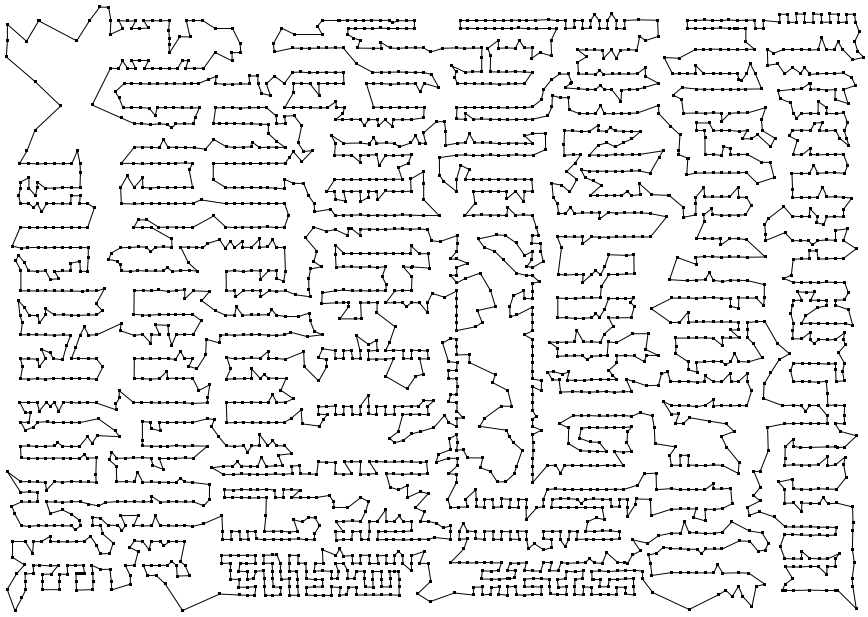
Figuur 9: Kortste route door 13509 plaatsen in Amerika

2. Als Van Gend & Loos pakjes rondbrengt, gebeurt dat vanuit een centraal depot en met meer dan één auto. De planner bepaald eerst welke wagen naar welk adres gaat en lost dan voor elke wagen een handelsreizigers-probleem op; zie figuur 10. In werkelijkheid heb je nog met een heleboel andere factoren te maken: een chauffeur mag maximaal acht uur werken, een klant in een voetgangersgebied moet vóór tien uur 's morgens worden beleverd, de auto staat in een file, enz.



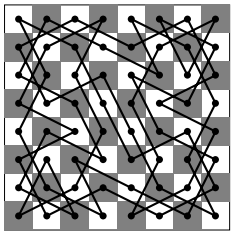
Figuur 10: Routering van drie wagens

3. Een computer bevat vaak rechthoekige platen waarop allerlei onderdelen zijn gemonteerd. Bij de fabricage van zo'n plaat bezoekt een apparaat de talloze montagepuntjes op de plaat om er een druppeltje lijm te deponeren of er met een laserstraal een gaatje in te schieten. Omdat er veel platen moeten worden behandeld, moet het apparaat telkens naar het punt van uitgang terugkeren. Of het nu om lijm of om een laserstraal gaat, het apparaat voert een handelsreizigersroute uit met de montagepuntjes als steden. Dergelijke grote problemen komen in de praktijk vaak voor. In figuur 11 zie je een kortste route voor een probleem met 3038 montage-punten.



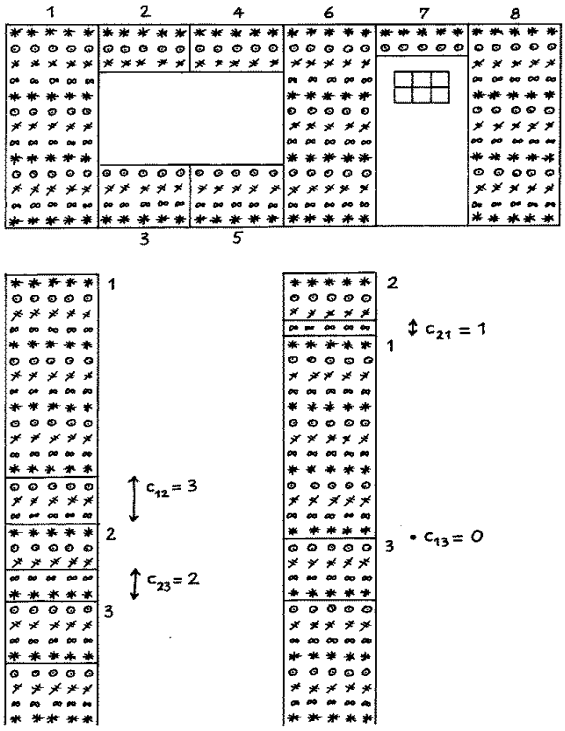
Figuur 11: Kortste route door 3038 montagepunten

4. Is het mogelijk met een paard alle 64 velden van het schaakbord te bezoeken en na 64 zetten op het beginpunt terug te zijn? Dit is een handelsreizigersprobleem met de velden van het schaakbord als steden. De afstand tussen twee steden is het aantal paardensprongen dat nodig is om van het ene veld naar het andere te komen. De vraag is of er een route van lengte 64 is. Het antwoord is ja; zie figuur 12.



Figuur 12: Route van een paard over het schaakbord

5. Je gaat je kamer opnieuw behangen. Je hebt behang uitgezocht met een horizontaal patroon en je wilt dat het patroon over de hele breedte van de muur doorloopt. Twee stukken behang die naast elkaar worden geplakt moeten dus nauwkeurig op elkaar aansluiten. Je wilt de stukken behang nu zodanig uit de rol knippen dat er zo weinig mogelijk verspilling optreedt. De verspilling bestaat uit de stukken behang die je afknipt maar na afloop kunt weggooien. In dit geval zijn de stukken behang die je wilt opplakken de steden. Als je eerst stuk i afknipt en dan stuk j, dan is de lengte van het stuk behang tussen i en j de afstand c_{ij} van i naar j; zie figuur 13. Dit handelsreizigersprobleem is niet helemaal hetzelfde als de andere voorbeelden. In de eerste plaats hoeft de afstand van i naar j niet gelijk te zijn aan die van j naar i. Tot nu toe was dit steeds wel het geval. In de tweede plaats zoek je niet naar een kortste (gesloten) Hamiltoncircuit maar naar een kortste (open) Hamiltonpad. Beide problemen lijken heel veel op elkaar. Als je het circuit-probleem kunt oplossen, kun je het pad-probleem ook aan. Dan laat je immers gewoon de laatste kant weg.



Figuur 13: Het behangen van je kamer

Opmerking

Om de voorbeelden en de opgaven wat eenvoudiger te maken nemen we drie dingen aan:

1. Het handelsreizigersprobleem is een probleem dat wordt beschreven met een **volledige** graaf. M.a.w. tussen elk tweetal steden is er een kant.

2. De afstanden zijn **symmetrisch**, dus $c_{ij} = c_{ji}$ voor elk tweetal steden i, j. We kunnen voor het probleem dus een model maken met een ongerichte graaf. Bij de beschrijving van het probleem hebben we deze veronderstelling eigenlijk al gemaakt. Voorbeelden 1 t/m 5 voldoen aan deze veronderstelling, voorbeeld 6 niet.

3. De afstanden voldoen aan de **driehoeksongelijkheid**, dat wil zeggen: $c_{ik} \leq c_{ij} + c_{jk}$ voor elk drietal steden i, j, k. In woorden: Als je omrijdt schiet je er niets mee op; je legt dan altijd een even lange of langere afstand af dan de rechtstreekse afstand tussen je twee gekozen steden.

Vraag:
Veronderstellingen (2) en (3) betekenen dat $c_{ij} \geq 0$ voor alle ij. De lengtes van de kanten zijn niet-negatief. Kun je dat bewijzen?

4.2 Moeilijkheden van het handelsreizigersprobleem

Het handelsreizigersprobleem kom je zoals je gezien hebt overal tegen: op de weg, in de fabriek en thuis. Het is kenmerkend voor de problemen die we in de **combinatorische optimalisering** tegenkomen:

1. Er worden **discrete** keuzes gemaakt. (Discreet betekent dat je 'ja' of 'nee' kiest, of '0' of '1', 'zwart' of 'wit', er zitten geen grijswaarden tussen. Het probleem is niet continu.) Een kant komt wel of niet in een route voor.

2. Het probleem is makkelijk te omschrijven. Iedereen kan het begrijpen.

3. Maar het probleem is heel lastig optimaal op te lossen.

Nu kun je natuurlijk zeggen: "Hoezo lastig op te lossen?" Veel wiskundigen vinden het juist een heel gemakkelijk probleem. Zij zeggen: "Hoe groot het aantal steden ook is, het aantal mogelijke routes is eindig. Elke route heeft een lengte en onder een eindig aantal routes, die elk een lengte hebben, is er een kortste." Wat ze zeggen is natuurlijk juist; het laat in ieder geval zien dat er een kortste route bestaat. Maar het probleem is die kortste route te vinden, ofwel: het probleem is een **algoritme** te bedenken waarmee je, voor een gegeven probleem, in redelijke tijd een kortste route vindt.

Je zou alle routes één voor één kunnen bekijken, maar dit duurt veel te lang als je veel steden hebt.

Vraag:
Hoeveel routes heb je bij 13 steden?

Als je de bovenstaande vraag hebt kunnen beantwoorden weet je dat er bij 60 steden al meer routes bestaan dan er atomen in het heelal zijn. Toch worden tegenwoordig veel grotere handelsreizigersproblemen snel optimaal opgelost, uiteraard zonder ze allemaal te bekijken.

Wat verstaan we onder snel oplossen? Een algoritme wordt snel genoemd als bij n steden het aantal stappen van de berekening evenredig is met n of n^2 of n-tot-de-een-of-andere constante macht. Een aantal stappen dat evenredig is met 2^n, n^n of $n!$ is traag. Algemeen gesproken noemen we een polynomiale rekentijd redelijk, (uit te drukken in een n-de graads functie, waarbij n een constante is) maar alles wat exponentieel of erger is niet.

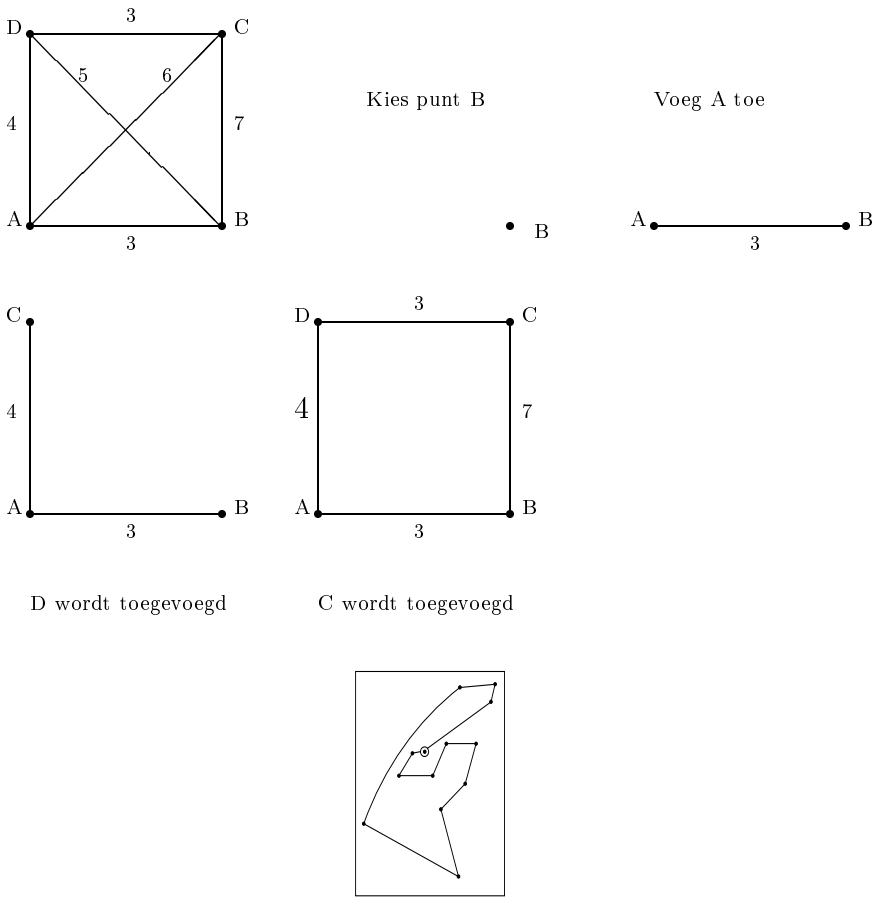
Algoritmes voor het vinden van kortste bomen en kortste paden zijn snel, maar voor het handelsreizigerprobleem bestaat geen snel algoritme dat altijd de snelste route vindt. Tenminste: tot nu toe heeft niemand zo'n algoritme kunnen vinden. Dit betekent dat er keuzes moeten worden gemaakt. Als je echt de kortste route wilt vinden, dan zal dit tijd kosten. Als je die tijd niet hebt, weet je dat je een route vindt die niet per se de kortste hoeft te zijn.

4.3 Algoritmes voor het benaderen van het optimum voor het handelsreizigersprobleem

4.3.1 Het beste-buur-algoritme

Een eenvoudig en snelle regel voor het vinden van een route werkt als volgt:

1. We beginnen in een willekeurige stad.
2. Als je nog niet in alle steden bent geweest dan ga je naar de dichtstbijzijnde nog niet bezochte stad.
3. Als alle steden zijn bezocht, ga dan vanuit de stad waar je bent naar het beginpunt.



Figuur 14: Beste-buur-route vanuit Amsterdam

In figuur 14 zie je hoe dit algoritme het probleem uit voorbeeld 1 heeft opgelost. Het resultaat is helemaal niet optimaal. Een beste-buur-algoritme is meestal matig tot slecht.

4.3.2 Het invoegingsalgortime

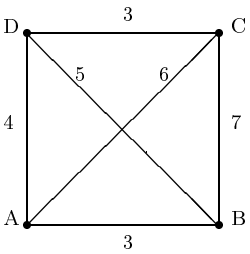
Het beste-buur-algoritme construeert een route door aan een pad telkens een kant toe te voegen. Het invoegingsalgoritme maakt een route door aan een gedeeltelijke route, waar niet alle steden in zitten, steeds een stad toe te voegen. Dit werkt op de volgende manier:

1. Begin met een gedeeltelijke route bestaande uit een willekeurig gekozen stad.

2. Als deze willekeurig gekozen gedeeltelijke route nog niet uit alle steden

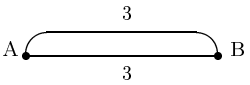
bestaat, kies dan twee steden k en j, met j wel in de route en k niet, zodanig dat c_{jk} minimaal is.

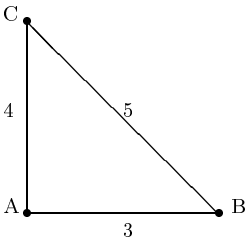
3. Als i een buur van j in de route is, vervang dan de kant ij in de route door ik en kj.



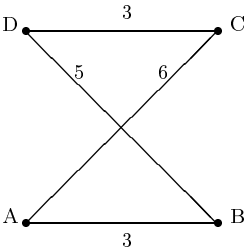
Kies punt B

Voeg A en route AB toe



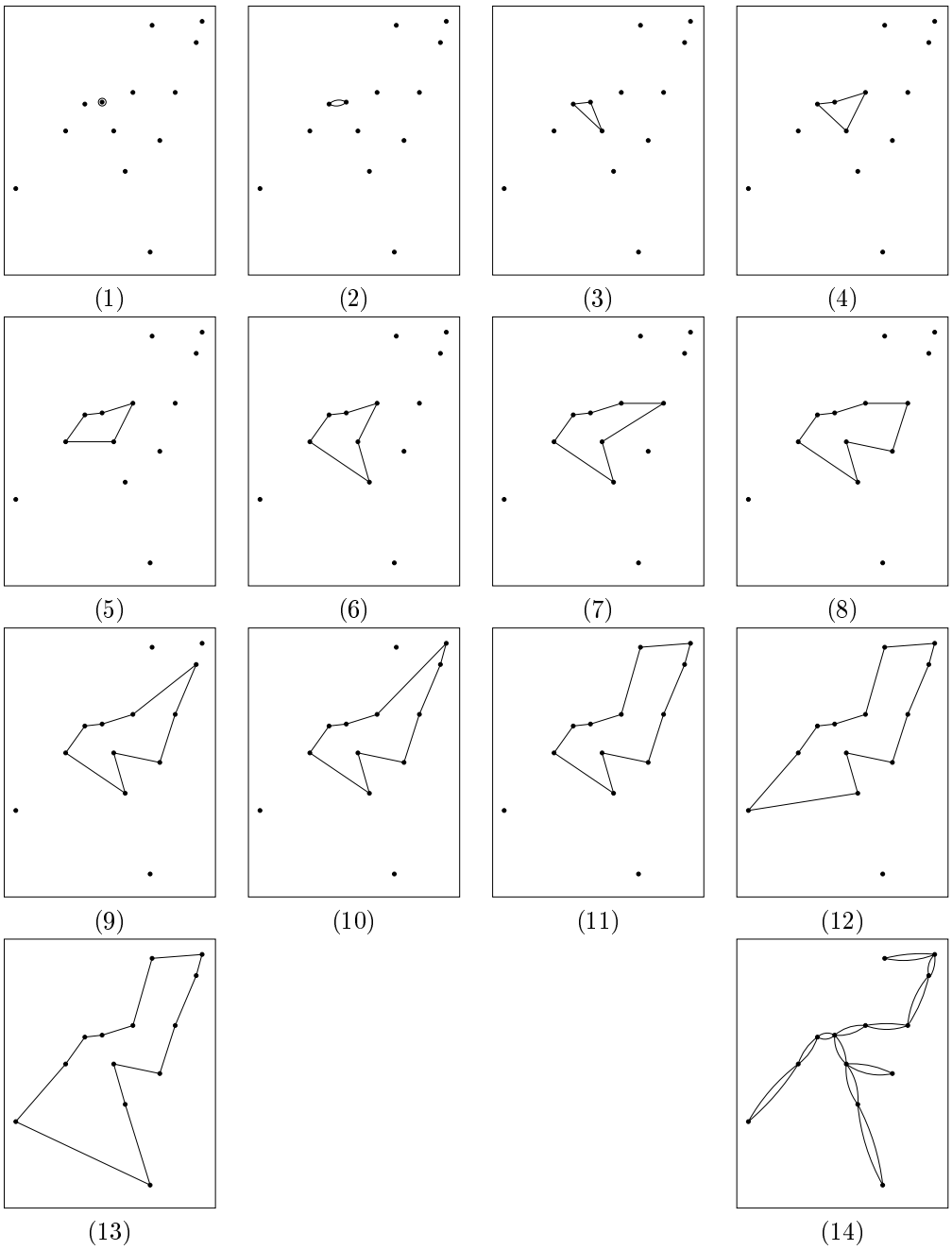


D wordt toegevoegd,
de kanten AD en DB
vervangen AB



C wordt toegevoegd,
de kanten AC en CD
vervangen AD

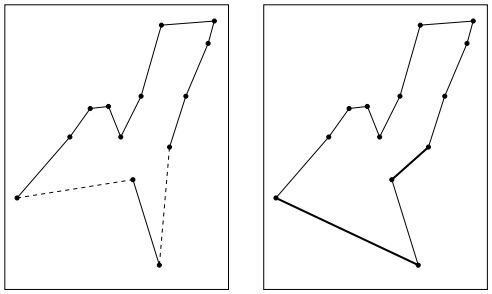
Deze methode lost het probleem uit voorbeeld 1 optimaal op, (zie figuur 14, plaatjes 1 - 13). Het invoegingsalgoritme geeft over het algemeen een beter resultaat dan het beste-buur-algoritme. Er kan nog steeds een groot verschil met het optimum zijn, maar dit verschil blijft begrensd: de route die je met het invoegingsalgoritme vindt is gegarandeerd korter dan twee keer de kortste route.



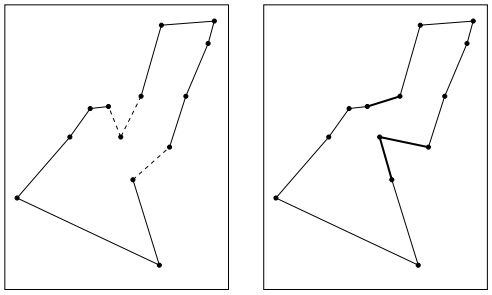
Figuur 15: Het invoegingsalgoritme in actie

4.3.3 Uitwisselingsalgoritmen

Het beste-buur-algoritme en het invoegingsalgoritme construeren stapsgewijs een route. Uitwisselingsalgoritmen gaan uit van een volledige route en proberen die stapsgewijs te verbeteren. Dit gaat meestal via het uitwisselen van kanten. Je kunt natuurlijk zoveel kanten uitwisselen als je zelf zou willen, één, twee of meer. We hebben het dan over 1-uitwisseling, 2-uitwisseling etc. Een t-uitwisseling is dus een uitwisseling waarbij t kanten van een route vervangen worden door t andere kanten, zodanig dat er weer een route ontstaat. Figuur 15 laat een 2-uitwisseling en een 3-uitwisseling zien. Als de nieuwe route korter is dan de oude is er een verbetering bereikt en begint het proces van voren af aan. Dat gaat door totdat de route door een t-uitwisseling niet verder kan worden verbeterd. We noemen de route dan **t-optimaal**.



(a) 2-Uitwisseling



(b) 3-Uitwisseling

Figuur 16: t-Uitwisselingen

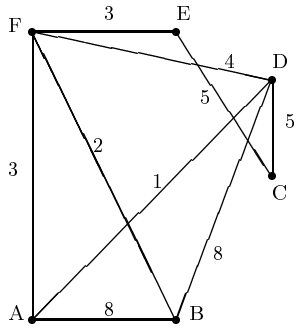
In tegenstelling tot de constructieve algoritmen zijn uitwisselingsalgortimen niet snel. Bovendien kunnen de resultaten net als bij het beste-buur-algoritme behoorlijk slecht zijn, maar dit is dan wel het ergst denkbare geval. In werkelijkheid valt het met de rekentijd en de prestaties erg mee en worden deze algoritmen vaak gebruikt.

4.4 Opgaven

1. Probeer eens een voorbeeld van het beste-buur-algoritme te maken waarvoor het nog slechter gaat dan in figuur 14.

2. Leg uit waarom de route die je met het invoegingsalgoritme vindt korter is dan twee keer de kortste route.

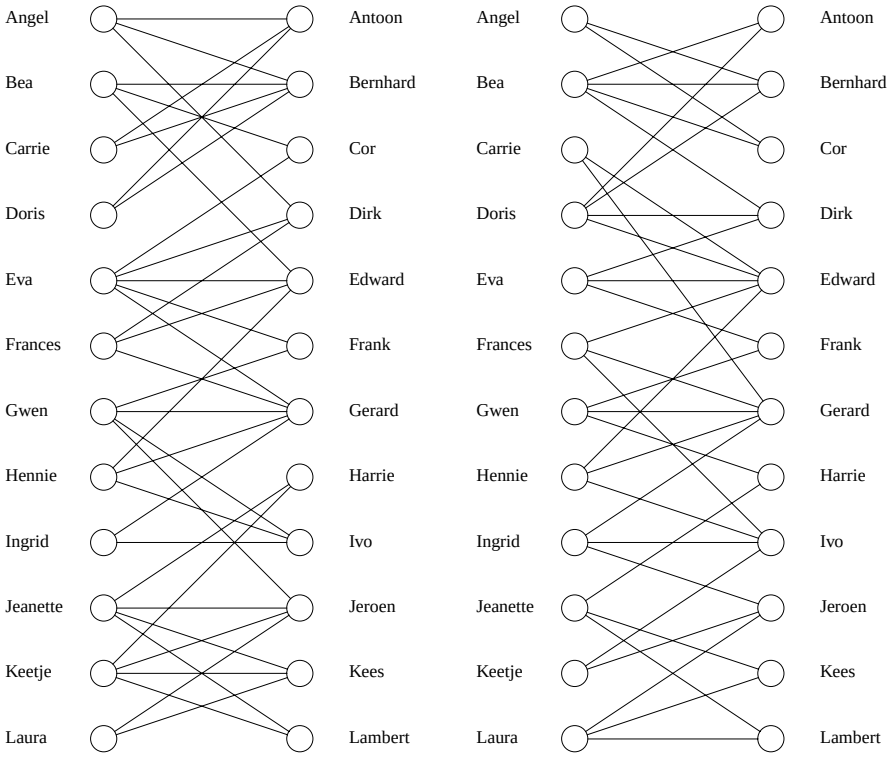
3. Bepaal in onderstaande graaf met alle besproken algoritmes de kortste route tussen $ABCDEF$ N.B. De kanten AE, AC, BC, BE en CF zijn om de graaf overzichtelijk te houden niet getekend. Zij hebben alle lengte 10.



4. Verzin zelf een leuke toepassing bij het handelsreizigerprobleem.

5 Gemengde Opgaven

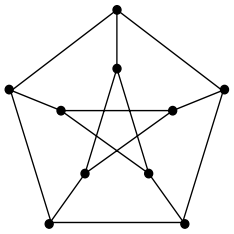
1. Een dansschool heeft als regel dat je je als een stel moet aanmelden, d.w.z.: een jongen en een meisje moeten zich samen aanmelden. Iedereen heeft dan één danspartner. We maken hiervan een voorstelling met een graaf, waarin de punten links de meisjes en de punten rechts de jongens voorstellen. Als een jongen en een meisje mogelijke partners zijn, dan geven we dit aan met een kant tussen de betreffende punten. In figuur 17 zie je twee mogelijke situaties. In die situaties zijn er per persoon ongeveer drie mogelijke danspartners. Zoek voor beide situaties een verdeling in twaalf stellen.



Figuur 17: Problemen rond de dansles

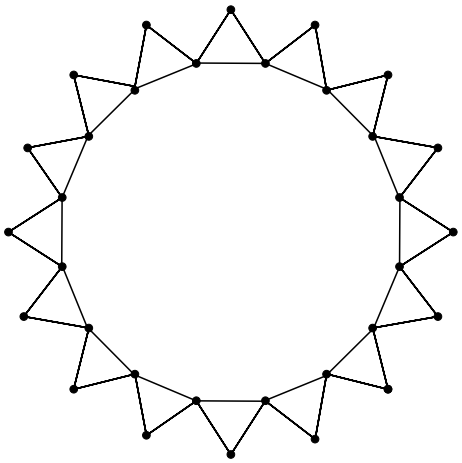
2. In de paragraaf over kortste paden staat in figuur 7 een stukje tekst waarin de mogelijke afbrekingen zijn aangegeven en hoeveel elke afbreking kost. Plaats deze tekst op een bladzijde waarop maar 15 tekens naast elkaar kunnen staan.

3. In figuur 18 zie je de zogenaamde **Petersen**-graaf. Heeft de Petersen-graaf een Hamiltoncircuit?



Figuur 18: Petersen-graaf

4. Het stratenplan in figuur 19 heeft 32 punten. Alle kanten hebben een lengte 1. De afstand tussen twee punten is de lengte van het kortste pad ertussen. Hoe lang is de kortste route langs deze 32 punten? Laat zien dat de beste-buur-algoritme een route zou kunnen vinden (door af en toe een slechte keus te maken wanneer meer dan één adres het dichtstbij ligt) die tweemaal zolang is als de kortste.



Figuur 19: Stratenplan voor 32 adressen

BIJLAGE

Volledige inductie

Volledige inductie is een manier om te bewijzen dat een algoritme, een formule, een patroon voor elk positief geheel getal waar is. Hoe het werkt kun je zien in het volgende voorbeeld:

We tellen steeds opvolgende oneven getallen, beginnend bij 1, bij elkaar op, dus:

$$\begin{array}{lll} 1+3 & = 4 & = 2^2 \\ 1+3+5 & = 9 & = 3^2 \\ 1+3+5+7 & = 16 & = 4^2 \\ 1+3+5+7+9 & = 25 & = 5^2 \\ 1+3+5+7+9+11 & = 36 & = 6^2 \end{array}$$

Het lijkt erop dat de optelling van n opvolgende oneven getallen, beginnend bij 1, steeds n^2 oplevert. Je denkt natuurlijk dat dit altijd zo doorgaat, maar hoe kun je dat bewijzen? We kijken eerst wat we precies willen laten zien. Dat is in dit geval:

$$1 + 3 + 5 + 7 + \ldots + (2n-1) = n^2$$

Dit is een bewering en we noemen deze bewering even $P(n)$. Met **volledige inductie** gaan we nu laten zien dat $P(n)$ waar is voor alle mogelijke n. Dit gaat niet door maar willekeurig een heel groot getal in te vullen. Er is namelijk altijd een mogelijkheid dat het voor een nog groter getal niet meer geldt en het is natuurlijk onmogelijk om alle getallen te controleren. Eigenlijk hebben we maar twee dingen nodig. Eerst moeten we laten zien dat $P(n)$ waar is voor $n = 1$ (dat wil zeggen $P(1)$ is waar) en dan willen we laten zien dat als $P(n)$ waar is voor een willekeurige n, dat dan automatisch $P(n+1)$ ook waar is. Je kunt je dit misschien voorstellen als een soort domino-effect. Als één dominosteen valt dan valt de volgende ook en die daarna ook en die daarna, enzovoorts. Dus als we de allereerste steen omgooien (als $P(1)$ waar is), dan valt de hele rij dominostenen ($P(n)$ is dan waar voor alle n), omdat elke steen door zijn voorganger zal worden omgetikt, en zelf weer zijn opvolger omgooit.

Dus in ons geval kijken we eerst of geldt:

$$1 = 1^2$$

Dit is natuurlijk flauw. Laten we nu eens aannemen dan $P(n)$ waar is voor een willekeurige n. Dus neem aan dat:

$$1 + 3 + 5 + 7 + \ldots + (2n-1) = n^2$$

Het volgende oneven getal na $(2n-1)$ is $(2n+1)$. We tellen dit bij allebei de kanten van het gelijkteken in onze bewering op:

$$1 + 3 + 5 + 7 + \ldots + (2n - 1) + (2n + 1) = n^2 + (2n + 1)$$

We weten dat $(n + 1)^2 = n^2 + 2n + 1$, dus wordt het voorgaande nu:

$$1 + 3 + 5 + 7 + \ldots + (2n - 1) + (2n + 1) = (n + 1)^2$$

en dat is precies $P(n + 1)$. We hebben dus laten zien dat *als* $P(n)$ waar is voor een willekeurige n *dan* is ook $P(n + 1)$ waar.

We hadden n willekeurig gekozen en daarom mogen we nu ook elk geheel getal dat we kunnen bedenken invullen. Als ik $n = 1$ neem dan is $P(1)$ waar, dat was duidelijk. Je ziet uit het voorgaande dat dan $P(2)$ ook waar is. Maar als $P(2)$ waar is, dan is ook $P(3)$ waar, enz. De bewering zal altijd gelden dus hebben we met behulp van volledige inductie kunnen vaststellen dat $P(n)$ inderdaad voor alle positieve gehele getallen n waar is.