

Deze tering begint pas bij hoofdstuk 3, de rest is afval.



Hoofdstuk 3 Talen, automaten en berekenbaarheid

3.1 Strings en talen

3.1.1 Strings

Definitie 3.1: Een alfabet is een niet-lege, eindige verzameling. De elementen van de verzameling worden symbolen genoemd.

Voorbeeld 3.1 De verzameling $\{a,b,c,\dots,z\}$ is een **alfabet**.

Wanneer het er niet toe doet wat het gebruikte alfabet precies is, gebruiken we vaak het symbool Σ om naar het alfabet te verwijzen, en noteren we de elementen als $\sigma_1, \sigma_2, \dots$

Definitie 3.2 Een **string** s over een alfabet Σ is een eindige rij symbolen uit Σ .

Het aantal symbolen in die rij noemen we de **lengte** van de string, genoteerd $|s|$.

De string die uit nul symbolen bestaat, noemen we de lege string, en noteren we λ . Elke andere string kan genoteerd worden door de symbolen waaruit de string bestaat achter elkaar op te schrijven.

De verzameling van **alle** strings over een alfabet Σ noteren we Σ^* .

Voorbeeld 3.2 Met een alfabet $\Sigma = \{0,1\}$ zijn 001101, 1111, 1101, en λ allemaal strings, met lengte 6, 4, 4 en 0. Met $\Sigma = \{A,B,C,\dots,Z,a,b,c,\dots,z\}$ zijn goedemiddag en Leuven strings over Σ , maar 120 niet.

Een string heeft steeds een eindige lengte (zie de definitie). Een string met oneindig veel symbolen bestaat dus niet. De verzameling van alle strings is dan wel oneindig, omdat er geen maximale lengte aan de strings opgelegd wordt. Bijvoorbeeld: de verzameling van alle strings over het alfabet $\Sigma = \{a\}$ is de verzameling $\{\lambda, a, aa, aaa, aaaa, aaaaa, \dots\}$.

3.1.2 Talen

Definitie 3.3 Een taal over een alfabet Σ is een verzameling strings over Σ . (Of nog: een taal over Σ is een deelverzameling van Σ^*)

Voorbeeld 3.3 De verzameling van alle Nederlandse woorden is een taal over $\Sigma = \{A, B, C, \dots, Z, a, b, c, \dots, z\}$. De verzameling van alle grammaticaal correcte Nederlandse zinnen en teksten is een taal over een alfabet dat bestaat uit letters, cijfers, leesteken, en de spatie.

Indien we een alfabet Σ vast kiezen, dan kunnen we de volgende bewerkingen uitvoeren op de strings uit Σ^* :

- **De concatenatie of samenstelling.**

Indien $x = \sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma^*$ en $y = \mu_1, \mu_2, \dots, \mu_m \in \Sigma^*$ dan is hun samenstelling

$$xy = \sigma_1, \sigma_2, \dots, \sigma_n \mu_1 \mu_2, \dots, \mu_m \in \Sigma^*$$

In het bijzonder hebben we ook voor elke $x \in \Sigma^*$ dat

$$\lambda x = x\lambda = x \text{ en } \lambda\lambda = \lambda.$$

- Voor $x \in \Sigma^*$ definieren we

$$x^0 = \lambda$$

$$\forall n \in \mathbb{N} : x^{n+1} = xx^n$$

Voorbeeld 3.6 Als $x = abba$ en $y = abcd$, dan is $xy = abbaabcd$. Indien $x = abb$, is $x^3 = abbabbabb$.

We kunnen dezelfde bewerkingen uitbreiden tot talen. Zij $A, B \subseteq \Sigma^*$, dan is

- $AB = \{ab \mid a \in A, b \in B\}$
- $A^0 = \{\lambda\}$
- $\forall n \in \mathbb{N}$ (verzameling van natuurlijke getallen) : $A^{n+1} = AA^n$

Verder definieren we

- $A^* = A_0 \cup A_1 \cup A_2 \cup A_3 \cup \dots = \bigcup_{n=0}^{\infty} A^n.$

A^* wordt de Kleenesluiting van A genoemd

- $A^+ = A_1 \cup A_2 \cup A_3 \cup \dots = \bigcup_{n=1}^{\infty} A^n$

3.1.3 Reguliere talen.

Definitie 3.4 (Reguliere Taal) (kan worden gedefinieerd door een reguliere uitdrukking) Indien Σ een alfabet is, dan wordt de klasse van R van alle reguliere talen over Σ inductief als volgt gedefinieerd:

1. $\emptyset \in R, \{\lambda\} \in R$ en $\forall \sigma \in \Sigma : \{\sigma\} \in R$.
2. Indien $A, B \in R$, dan ook $AB \in R, A \cup B \in R$ en $A^* \in R$

Elke taal uit R wordt een **reguliere taal** genoemd

Voorbeeld 3.9 De taal $L_3 = \{11, 101, 1001, 10001, 100001, \dots\}$ op $\Sigma = \{0, 1\}$ is een reguliere taal, want

1. $\{1\}$ en $\{0\}$ zijn reguliere talen
2. $\{0\}^*$ is een reguliere taal
3. $\{1\}\{0\}^*$ is een reguliere taal
4. $L_3 = \{1\}\{0\}^*\{1\}$ is een reguliere taal

3.1.4 Reguliere uitdrukkingen

Ook wel regular expression genoemd voor de duidelijkheid, is dus niks nieuws. Ze proberen natuurlijk weer speciaal te doen door het te vertalen naar het nederlands. godverdommese taalpuristen.

Voorbeeld 3.10 Veronderstel dat Σ de verzameling van ASCII symbolen is, dan zijn de volgende strings reguliere uitdrukkingen over Σ :

- 123
- Hallo!
- Dag mevrouw.

• xyz	$\{xyz\}$
• x yz	$\{x\} \cup \{yz\} = \{x, yz\}$
• x yz*	$\{x\} \cup \{yz^n n \in \mathbb{N}\} = \{x, y, yz, yzz, yzzz, \dots\}$
• (x y)z*	$\{xz^n n \in \mathbb{N}\} \cup \{yz^n n \in \mathbb{N}\} = \{x, y, xz, yz, xzz, yzz, \dots\}$
• (x yz)*	$(\{x\} \cup \{yz\})^* = \{\lambda, x, yz, xx, xyz, yzx, yzyz, xxx, xxyz, xyzz, xyzyz, yzxx, \dots\}$
• x*(yz)*	$\{x^n n \in \mathbb{N}\} \cup \{(yz)^n n \in \mathbb{N}\} = \{\lambda, x, yz, xx, yzyz, xxx, yzyzzyz, \dots\}$

Definitie 3.5 (Reguliere uitdrukking) Indien Σ een alfabet is, dan wordt een reguliere uitdrukking over Σ op inductieve wijze als volgt gedefinieerd:

1. \emptyset is een reguliere uitdrukking
2. λ is een reguliere uitdrukking
3. Voor elke $\sigma \in \Sigma$ is σ een reguliere uitdrukking
4. Indien A en B reguliere uitdrukkingen zijn, dan zijn ook (A) , A^* , $A|B$ en AB reguliere uitdrukkingen.

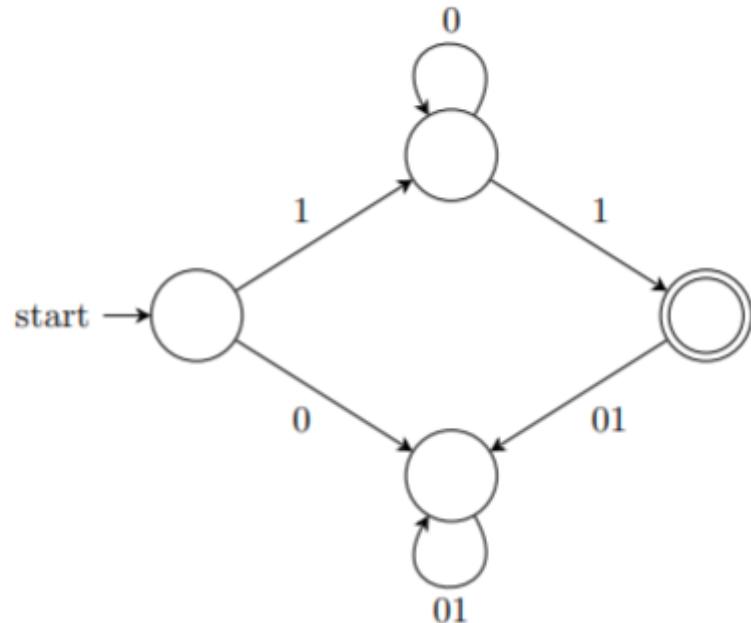
Zoals hierboven aangegeven bepaalt elke reguliere uitdrukking ω een verzameling van strings uit Σ^* . Deze taal wordt de reguliere verzameling bepaald door ω genoemd.

Stelling 3.1 Voor een gegeven alfabet Σ geldt dat de klasse van de reguliere talen op Σ precies samenvalt met de klasse van de reguliere verzamelingen. Voor elke reguliere taal L bestaat er bijgevolg een reguliere uitdrukking ω , waarvan de bijkomende reguliere verzameling precies L is

3.2 Eindige automaten

3.2.1 Eindige automaten

Voorbeeld 3.15 Beschouw het volgende diagram:



Dit schema werkt door een serie van 0'en en 1'en af te gaan en te zien waar je belandt. Het schema zelf bepaalt een taal L. Als je belandt in de dubbele cirkel voldoet de serie aan de taal, als het er niet in belandt dan voldoet deze niet.

Beschouw het voorbeeld 11001. We starten het schema bij de meest linkse cirkel. Omwille van de eerste 1 van de string verplaatsen we ons naar de bovenste cirkel. De tweede 1 uit de string zegt ons naar de rechter cirkel te gaan. Daarna doet de eerste nul ons naar de onderste cirkel gaan en de volgende nul en de laatste 1 doen ons telkens terugkeren naar de onderste cirkel. Op het einde bevinden we ons dus in de onderste cirkel.

Een eindige automaat is een computermodel dat door zo een schema kan worden voorgesteld. De basis-ingredienten van een eindige automaat zijn een eindig aantal toestanden (in het schema voorgesteld door cirkels) waarin de machine zich kan bevinden en vanuit elke toestand "overgangen" afhankelijk van de invoer op dat moment.

Definitie 3.6 (Eindige automaat) Een eindige automaat is een 5-tal $A = (Q, \Sigma, \delta, q_0, F)$ met

- Q een **eindige verzameling**; we noemen de elementen van Q de toestanden van de automaat A .
- $F \subseteq Q$; F is de verzameling van de **aanvaardbare eindtoestanden**. (De letter F is de eerste letter van het Engelse Final).
- $q_0 \in Q$; deze toestand wordt de **begintoestand** genoemd.
- Σ een **alfabet**
- δ een **afbeelding**, de transitieafbeelding genoemd,

$$\delta : Q \times \Sigma \rightarrow Q.$$

Voorbeeld 3.16 We beschouwen een eindige automaat $A = (Q, \Sigma, \delta, q_0, F)$, met

1. $Q = \{E, O\}$, met begintoestand $q_0 = E$.
2. $F = \{E\}$
3. $\Sigma = \{0, 1\}$.
4. De afbeelding δ wordt gegeven door volgende tabel

δ	0	1
O	O	E
E	E	O

De werking van de automaat op de string 1101 is als volgt:

1. Stap 1: de automaat berekent $q_{i_1} = \delta(q_0, 1) = \delta(E, 1) = O$.
2. Stap 2: de automaat berekent $q_{i_2} = \delta(q_{i_1}, 1) = \delta(O, 1) = E$.
3. Stap 3: de automaat berekent $q_{i_3} = \delta(q_{i_2}, 0) = \delta(E, 0) = E$.
4. Stap 4: de automaat berekent $q_{i_4} = \delta(q_{i_3}, 1) = \delta(E, 1) = O \notin F$.

De invoerstring 1101 wordt verworpen.

Deze automaat kan ook zo voorgesteld worden:



3.2.2 Eindige automaten en reguliere talen

Een automaat aanvaardt sommige strings wel, en andere niet. De verzameling van alle strings die aanvaard worden, vormt een taal. We noemen dit de taal bepaald door een automaat; we zeggen ook dat die taal herkend wordt door de automaat.

Definitie 3.7 Als $A = (Q, \Sigma, \delta, q_0, F)$ een eindige automaat is, noemen we

$$L(A) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$$

de taal bepaald door de eindige automaat A. Voor een gegeven taal $L \subseteq \Sigma^*$ zeggen we dat A de taal herkent als $L = L(A)$

Stelling 3.2 De klasse van talen die herkend worden door een eindige automaat valt precies samen met de klasse van de reguliere talen.

Stelling 3.3 (Pompstelling) Voor elke reguliere taal L bestaat er een $n \in \mathbb{N}$, zodat voor elke string $s \in L$ met $|s| \geq n$, het volgende geldt: er bestaan strings x, y, z zodat $s = xyz$ met $|y| > 0, |xy| \leq n$, en $\forall k \in \mathbb{N} : xy^kz \in L$.

Deze stelling wordt in het Engels het “pumping lemma” genoemd. Ze beschrijft een belangrijke basiseigenschap van reguliere talen: Reguliere talen kunnen willekeurig lange strings bevatten, maar vanaf een zekere lengte zijn die strings noodzakelijkerwijs opgebouwd uit een beperkt aantal “bouwstenen” die eventueel meerdere keren herhaald worden. Het bovenstaande voorbeeld toont aan dat, alhoewel eindige automaten in staat zijn om reguliere talen te herkennen (d.w.z., voor elke reguliere taal bestaat er een automaat die die taal herkent), ze toch beperkt zijn: er bestaan relatief eenvoudige talen die ze niet kunnen herkennen. Ze zijn bijgevolg niet geschikt als realistisch computermodel. In een volgend onderdeel zullen we kennismaken met een krachtiger model: de Turingmachine. Maar eerst kijken we nog even naar een ander soort automaten: de niet-deterministische eindige automaten.

3.2.3 Niet-deterministische eindige automaten

De eindige automaten we tot nu toe gezien hebben zijn zogenaamde deterministische machines. Dit betekent dat de machine op elk moment slechts 1 (ondubbelzinnige) instructie te verwerken krijgt en deze instructie slechts op 1 manier kan uitvoeren. We laten nooit de instructie “of” toe (Doe A of Doe B). Indien we echter aannemen dat we over twee onafhankelijke processoren zouden beschikken, dan zou een opdracht als Doe A of Doe B parallel kunnen uitgevoerd worden. De ene processor zou A kunnen uitvoeren, terwijl de andere B zou kunnen uitvoeren.

Indien we maar over 1 processor beschikken, en we willen toch opdrachten met of-statements uitvoeren, dan moet de machine dus op bepaalde momenten een keuze maken welke van de twee (of meer) mogelijke opdrachten ze zal uitvoeren. Dergelijke machines noemt men niet-deterministische machines.

Definitie 3.8 (Niet-deterministische eindige automaat) Een niet-deterministische eindige automaat is een 5-tal $A = (Q, \Sigma, \delta, q_0, F)$ met

- Q een eindige verzameling (de toestanden van de automaat A).
- $F \subseteq Q$ de verzameling van de aanvaardbare eindtoestanden.
- $q_0 \in Q$ de begintoestand van de automaat.
- Σ het alfabet van de automaat.
- δ een afbeelding

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow P(Q).$$

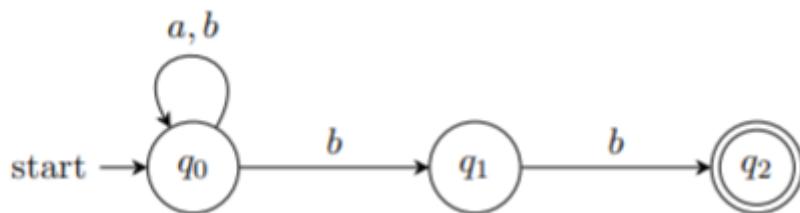
Het eerste verschil tussen een gewone eindige automaat en een niet-deterministische ligt in het feit dat de transitieafbeelding δ geen toestanden als beelden heeft, maar verzameling van toestanden. Die verzamelingen kunnen leeg zijn.

Ook niet-deterministische eindige automaten kunnen we schematisch voorstellen met cirkels en pijlen. We laten bij een niet-deterministische machine echter toe dat er vanuit een cirkel meerdere pijlen met hetzelfde label vertrekken. Het label λ mag eveneens gebruikt worden.

weer veel gezeik, kijk gewoon dit filmpje man

<https://www.youtube.com/watch?v=W8Uu0inPmU8>

Voorbeeld 3.19



Voor de NFA (non-deterministic finite state automaton) zijn er 3 mogelijkheden voor de string abb.

1. De machine eindigt in toestand q_0 omdat er steeds voor de loop gekozen wordt
2. De machine eindigt in toestand q_1 omdat er na 2 loops gekozen wordt om naar q_1 te gaan.
3. De machine eindigt in toestand q_2 .

We zeggen dat een string aanvaard wordt door een niet-deterministische eindige automaat indien er minstens 1 opeenvolging van keuzes gemaakt kan worden, zo dat de automaat eindigt in een aanvaardbare toestand na het verwerken van de string.

Definitie 3.9 (Taal bepaald door een niet-deterministische automaat) Indien $A = (Q, \Sigma, \delta, q_0, F)$ een niet-deterministische eindige automaat is, noemen we

$$L(A) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \cap F \neq \emptyset\}$$

de **taal bepaald door A**

Voor een gegeven taal $L \subseteq \Sigma^*$ zeggen we "A herkent L" als $L = L(A)$

Stelling 3.4 Voor elke niet-deterministische eindige automaat bestaat er een deterministische die dezelfde taal herkent.

Het omgekeerde geldt natuurlijk ook, daar elke deterministische eindige automaat kan aanzien worden als een niet-deterministische automaat. Niet-deterministische eindige automaten kunnen dus niet meer talen herkennen dan deterministische; ze zijn evengoed beperkt tot de klasse van de reguliere talen. Het nut van een niet-deterministische eindige automaat ligt in het feit dat het veel eenvoudiger is om een niet-deterministische eindige automaat te ontwikkelen die een gegeven taal herkent, dan om een deterministische te vinden

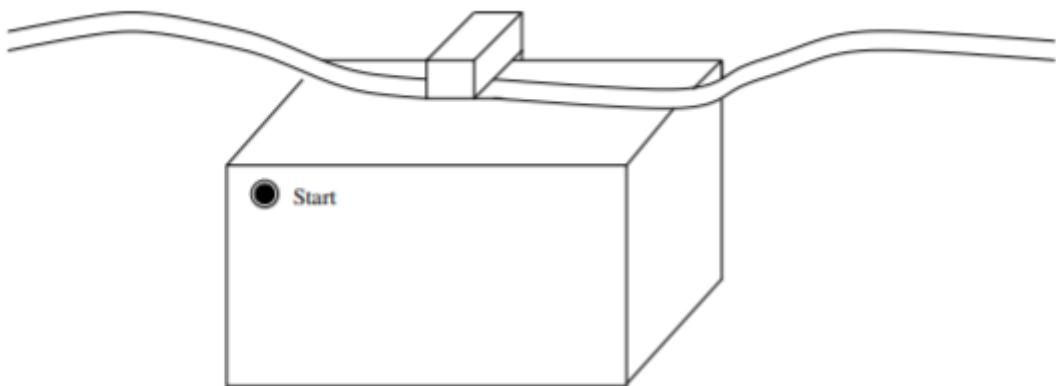
3.3 Turingmachines

3.3.1 Turingmachines

Eindige automaten zijn een wiskundig model voor een zeer eenvoudig soort computers. Ze hebben geen geheugencapaciteit; de enige informatie die ze hebben over de string die ze recent doorlopen hebben, is de toestand waarin ze zich bevinden, en er is maar een eindig aantal verschillende toestanden. Daardoor zijn ze beperkt qua berekeningen: ze kunnen enkel reguliere talen herkennen.

Turingmachines vormen een ander wiskundig model voor een bepaald soort computers, ditmaal wel met een grote geheugencapaciteit. Het model is niet te ingewikkeld, zodat het mogelijk is om bepaalde eigenschappen van dergelijke machines aan te tonen en te behandelen. Aan de andere kant is de klasse van Turingmachines toch krachtig genoeg om alle rekentaken te kunnen uitvoeren die een gewone computer kan uitvoeren.

We kunnen ons een Turingmachine voorstellen als een machine die voorzien is van een magneetband die in twee richtingen kan bewegen. In verschillende fasen van de berekening bevat de band (= het geheugen van dit computermodel!) de invoer voor de berekening, de tussenresultaten, en de uitvoer. De magneetband is een oneindig lange rij van symbolen die bestaat uit een eindige string voorafgegaan en gevolgd door oneindig veel blanco karakters (zie Figuur 3.1).



Figuur 3.1: Een Turingmachine

De machine heeft een lees/schrijfkop en het symbool op de magneetband juist onder de kop noemen we het huidig symbool: de operatie van lezen is daarmee impliciet. De kop kan het huidig symbool overschrijven. Net als een eindige automaat heeft een Turingmachine een eindig aantal mogelijke toestanden, en op elk moment bevindt de machine zich in 1 van die toestanden. Een instructie voor de machine is een voorschrift voor de machine om, in 1 operatie, het huidig symbool te overschrijven met een nieuw symbool, de kop (of de magneetband) 1 positie naar links of naar rechts te bewegen, en naar een nieuwe toestand over te gaan. Welke instructie uitgevoerd wordt, hangt enkel af van de

huidige toestand van de machine en het huidige symbool onder de leeskop. Eens gestart, blijft de machine instructies uitvoeren tot zij in een toestand komt waarbij er voor het huidig gelezen symbool geen instructie meer vorhanden is. Op dat moment kan men controleren of de machine al dan niet in een aanvaardbare toestand gestopt is. We schrijven nu het bovenstaande neer in een formele definitie.

Dit is een hele lang uitleg die ge zou kunnen begrijpen ma als ge het uzelf makkelijk wil maken, kijk dit filmpje: <https://www.youtube.com/watch?v=dNRDvLACg5Q>

Definitie 3.10 (Turingmachine) Een Turingmachine is een 6-tal $M = (Q, \Sigma, T, P, q_0, F)$ met

- Q een eindige verzameling; we noemen de elementen van Q toestanden.
- $F \subset Q$ de verzameling van aanvaardbare eindtoestanden
- $q_0 \in Q$ de begintoestand
- Σ het alfabet van de Turingmachine. Dit alfabet bevat, naast mogelijk andere symbolen, minstens een speciaal symbool, het blanco symbool of lege symbool, dat we noteren als #.
- $T \subseteq \Sigma \setminus \{\#\}$ is de verzameling van invoersymbolen. De elementen van $\Sigma \setminus (T \cup \{\#\})$ worden hulpsymbolen genoemd.
- P een functie (niet noodzakelijk een afbeelding)

$$P : (Q \setminus F) \times \Sigma \rightarrow Q \times \Sigma \times L, R, 0$$

P wordt het programma of de instructieset van de Turingmachine genoemd. (In de verzameling $\{L, R, 0\}$ staat L voor "Links", R voor "Rechts" en 0 voor "Blijf staan", waarmee de beweging van de schrijfkop aangegeven wordt.)

Wanneer we de machine starten bevindt deze zich in de begintoestand q_0 en staat de leeskop boven het meest linkse, niet blanco symbool van de magneetband. De machine voert dan het programma P uit en dit moet als volgt geïnterpreteerd worden: Noem q de toestand waarin de Turingmachine zich op een bepaald moment bevindt en zij $\sigma \in \Sigma$ het symbool dat op datzelfde moment door de kop gelezen wordt. Nu zijn er twee mogelijkheden

1. Het koppel (q, σ) behoort tot het definitiegebied van de functie P . Dit betekent dat $P(q, \sigma)$ bestaat en gelijk is aan een drietal $(q', \sigma', X) \in Q \times \Sigma \times \{L, R, 0\}$. Het effect van de uitvoering van deze instructie is dat de Turingmachine haar toestand (eventueel) zal veranderen in toestand q' , dat het huidig gescande symbool σ (eventueel) vervangen wordt door σ_0 en dat de kop zich beweegt zoals X aangeeft, d.w.z. indien $X = L$, beweegt de kop zich naar links (of de band naar rechts), indien $X = R$ beweegt de kop naar rechts en indien tenslotte $X = 0$ blijft de kop ter plaatse.
2. Indien het koppel (q, σ) niet behoort tot het definitiegebied van P stopt het programma. De toestand q wordt de eindtoestand van de Turingmachine (voor die bepaalde invoer) genoemd.

Indien de Turingmachine gestopt is en de eindtoestand q behoort tot F , dan zegt men dat de invoerstring aanvaard werd, in het andere geval zeggen we dat de invoerstring 29 verworpen werd. Indien een string aanvaard werd, dan beschouwen we de eindige string symbolen op de magneetband (dus zonder de twee oneindig lange uiteinden van blanco symbolen) als de uitvoer van de Turingmachine.

En dan nu een voorbeeld van de werking want deze 3 pagina's aan uitleg waren nog niet genoeg.

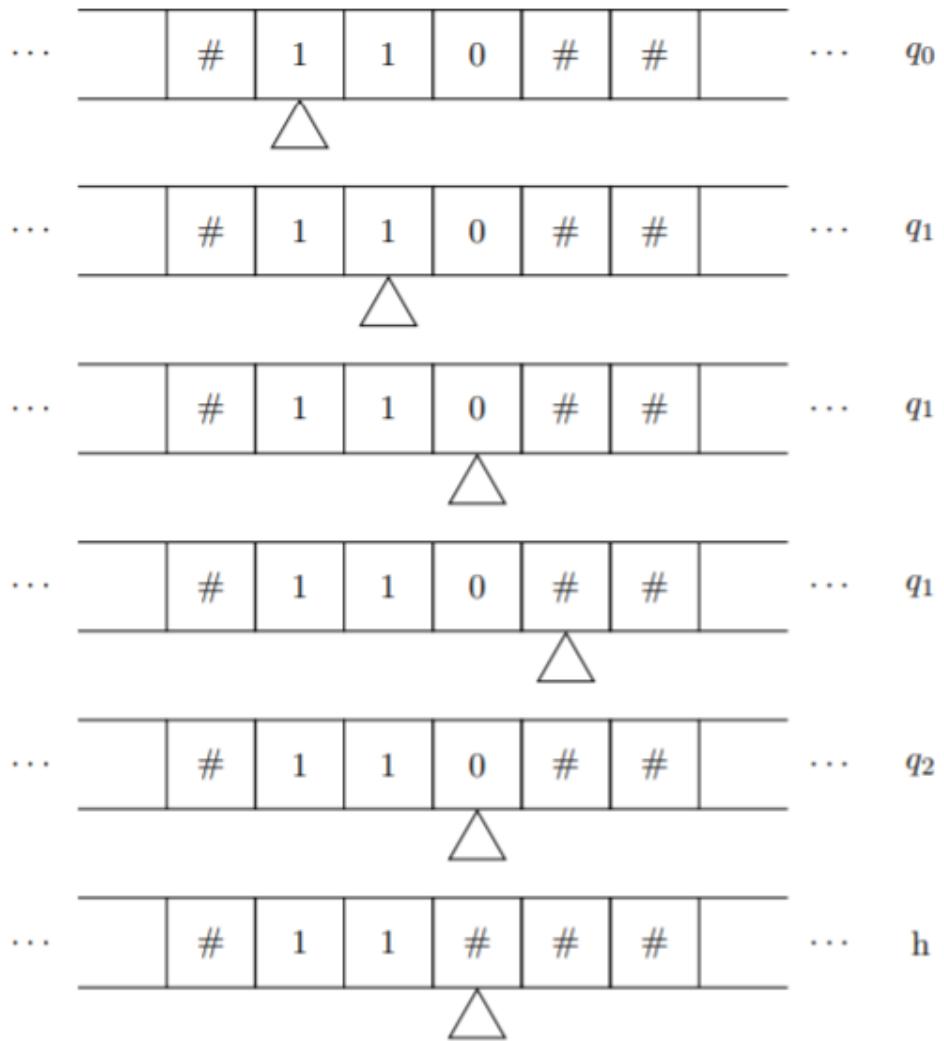
Voorbeeld

We illustreren deze begrippen aan de hand van een Turingmachine die als invoer een string van nullen en enen neemt en daarvan het meest rechtse symbool uitveegt (d.w.z. vervangt door een blanco symbool.)

De werking van een dergelijke Turingmachine zou er als volgt kunnen uitzien:

- Bij het begin staat de leeskop op het meest linkse symbool van de string (toestand q_0)
- Daarna zoekt de machine het meest rechtse symbool van de string door de leeskop telkens een positie naar rechts te verschuiven (toestand q_1).
- Op het moment dat de leeskop het blanco symbool # leest, bevindt de kop zich 1 positie rechts van de string (toestand q_2) en moet de kop weer een positie naar links bewegen.
- Nu moet het ingescande symbool vervangen worden door het blanco symbool # en mag de machine stoppen (toestand h).

Hieronder zie je het hierboven gegeven voorbeeld grafisch voorgesteld.



Figuur 3.2: De verschillende stappen die de Turingmachine M doorloopt

3.3.2 Turingmachines en functies

Computers worden vaak gebruikt om berekeningen uit te voeren. Je kan denken dat de functie 'kwadraat', die van een natuurlijk getal zijn kwadraat bepaalt, berekenbaar is door middel van een computer. Nu heeft een computer geen oneindige opslagcapaciteit en is het dus onmogelijk om de waarde kwadraat(n of zelf het getal n zelf) voor elk natuurlijk getal kan opslaan in zijn geheugen. Toch zien we dit als mogelijk want we kunnen het geheugen van onze computer uitbreiden (in theorie althans). Daarom zeggen we vaak dat de functie **effectief berekenbaar is**. Algemeen noemen we elke functie die berekenbaar is op een machine met onbegrenste opslagcapaciteit effectief berekenbaar.

Stelling 3.5 (These van Church) Een functie is **effectief berekenbaar** als en slechts als die functie Turing-berekenbaar is.

Een functie $f : T^* \rightarrow \sum^*$ is **Turing-berekenbaar** als en slechts als er een Turingmachine M bestaat zodanig dat $f = f_M$

Voorbeeld 3.23 De constante nulfunctie $f : N \rightarrow N : n \rightarrow 0$ is Turing-berekenbaar.

We kiezen voor de unaire representatie van de natuurlijke getallen. Een TM kan de constante nulfunctie als volgt berekenen: Bij om het even wat voor een invoerstring verplaatsen we de leeskop steeds een positie naar rechts waarbij we telkens het ingescande symbool wissen, tot we deze string voorbij zijn. Daarna schrijven we als uitvoer het symbool "1", de unaire representatie voor 0 op de magneetband.

Een formele beschrijving van deze TM $M = (Q, \Sigma, T, P, q_0, F)$ wordt als volgt gegeven:

- $Q = \{q_0, q_1, h\}$, $F = \{h\}$
- $\Sigma = \{1, \#\}$, $T = \{1\}$
- P wordt gegeven door:
 - $P(q_0, 1) = (q_1, \#, R)$
 - $P(q_1, 1) = (q_1, \#, R)$
 - $P(q_1, \#) = (h, 1, 0)$

Opmerking We merken hier op dat de bovenstaande TM niet minimaal is (in het aantal toestanden). Je hoeft namelijk niet meteen van toestand q_0 over te gaan naar een nieuwe toestand. (We illustreren dit in het voorbeeld hieronder.)

3.3.3 Turingmachines en talen

Een TM kan met een bepaalde invoer een bepaalde uitvoer associëren en zo functies berekenen. Maar we kunnen een TM ook gebruiken om talen te herkennen. Als de machine eindigt in een aanvaardbare toestand dan is de invoerstring aanvaard. De taal bepaald door de TM is dan de verzameling van alle strings die aanvaard worden.

Definitie 3.11 De taal bepaald door een Turingmachine M , genoteerd $L(M)$, is de verzameling van alle invoerstrings waarvoor M in een aanvaardbare toestand eindigt. Gegeven een taal L , zeggen we dat L herkend wordt door M als $L(M) = L$.

Definitie 3.12 Een taal L wordt Turing-herkenbaar genoemd, als er een TM bestaat die L herkent. Turing-herkenbare talen worden ook recursief opsombare talen genoemd.

Definitie 3.13 Een TM beslist een taal, als voor elk string $s \in L$ de TM in een aanvaardbare toestand eindigt, en voor elke string $s \notin L$ de TM in een onaanvaardbare toestand eindigt.

Definitie 3.14 Een taal L wordt Turing-beslisbaar of kortweg beslisbaar genoemd, als er een TM bestaat die L beslist. Turing-beslisbare talen worden ook recursieve talen genoemd.

Merk het verschil op tussen herkennen en beslissen. Als een TM de taal L herkent, betekent dat dat de TM alle en alleen de strings in L aanvaardt. Strings buiten L kunnen verworpen worden, of onbeslist blijven (omdat de machine niet stopt). Als een TM L beslist, betekent dat dat de TM voor elke string beslist of de string in L zit of niet; dit is een strikt strenger criterium.

Ik heb dit een lange tijd niet goed begrepen maar heb eindelijk een goede definitie gevonden.

Een machine is "**Turing-herkenbaar**" als en slechts als er een Turing Machine bestaat die

1. Wnr die een string tegenkomt in die taal, die deze string aanvaardt
2. Wnr die een string tegenkomt in die taal, die deze string rejects **en stopt of gewoon weg niet stopt.**

Een machine is "Turing-beslisbaar" als en slechts als er een Turing Machine bestaat die

1. Wnr die een string tegenkomt in die taal, die deze string aanvaardt
2. Wnr die een string tegenkomt niet in die taal, de machine deze string reject **en gewoon stopt.**

3.3.4 Niet-deterministische Turingmachines

Als we spreken over TMs zonder verdere uitleg, bedoelen we steeds deterministische Turingmachines. We kunnen ook hier een niet-deterministische versie definieren. Deze zullen een belangrijke rol spelen in de analyse van de complexiteit van beslissingsproblemen.

Definitie 3.15 (Niet-deterministische Turingmachine) Een niet-deterministische Turingmachine M bestaat uit een zestal $M = (Q, \Sigma, T, P, q_0, F)$ dat aan precies dezelfde voorwaarden voldoet als in definitie 3.10, behalve dat P nu geen functie meer hoeft te zijn, maar slechts een relatie tussen $(Q \setminus F) \times \Sigma$ en $Q \times \Sigma \times \{L, R, 0\}$.

- Een NDTM M stopt met success bij invoer x (en aanvaardt x) indien M bij minstens één van de mogelijke verwerkingen van x in een toestand in F stopt
- De taal $L(M)$ bepaald door een NDTM M is de verzameling van alle strings over T die door M aanvaard worden.
- De NDTM beslist een taal als de NDTM voor elke invoerstring in eindige tijd beslist of de string in $L(M)$ zit of niet.

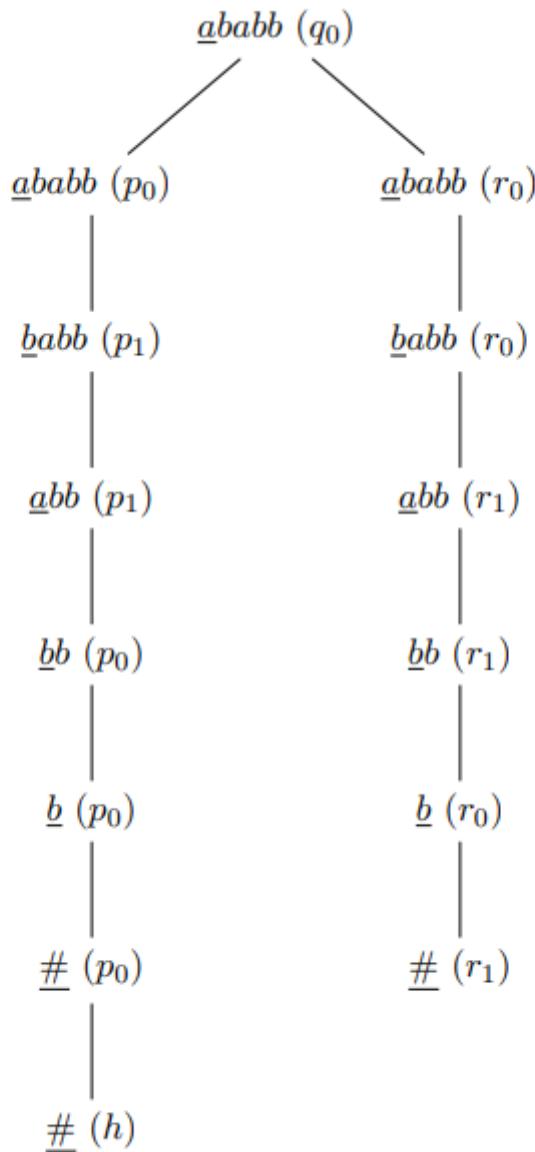
Voorbeeld 3.27 Een NDTM die voor een invoerstring bestaande uit a's en b's nagaat of die string een even aantal a's of b's bevat.

- Toestanden van M : $Q = \{q_0, p_0, p_1, r_0, r_1, h\}$, $F = \{h\}$
- symbolen van M : $\Sigma = \{\#, a, b\}$, $T = \{a, b\}$
- De instructieset P bestaat uit de koppels:

$$\begin{array}{ll} ((q_0, a), (p_0, a, 0)) & ((q_0, a), (r_0, a, 0)) \\ ((q_0, b), (p_0, b, 0)) & ((q_0, b), (r_0, b, 0)) \\ ((p_0, a), (p_1, \#, R)) & ((p_0, b), (p_0, \#, R)) \\ ((p_1, a), (p_0, \#, R)) & ((p_1, b), (p_1, \#, R)) \\ ((r_0, a), (r_0, \#, R)) & ((r_0, b), (r_1, \#, R)) \\ ((r_1, a), (r_1, \#, R)) & ((r_1, b), (r_0, \#, R)) \\ ((r_0, \#), (h, \#, 0)) & ((p_0, \#), (h, \#, 0)) \end{array}$$

In de instructieset hierboven worden de p-toestanden gebruikt om het even of oneven zijn van het aantal a's bij te houden, de r toestanden doen hetzelfde voor het aantal b's. Toestand p_0 (resp. r_0) duidt een even aantal a's (resp. b's) aan, toestand p_1 (resp. r_1) een oneven aantal.

Er zijn twee mogelijke wegen die de NDTM M kan volgen en deze zijn samen geschetst in figuur 3.3. Zoals je ziet op de figuur heeft de NDTM bij de eerste stap de keuze tussen twee opdrachten (namelijk $((q_0, a), (p_0, a, 0))$ of $((q_0, a), (r_0, a, 0))$). We nemen aan dat een NDTM telkens wanneer er meerdere opdrachten mogelijk zijn, elk van deze mogelijkheden apart maar tegelijkertijd (parallel) behandelt. We zeggen dat een NDTM M met succes stopt bij een bepaalde invoer of ook nog dat de NDTM M een bepaalde string aanvaardt, indien 1 van de gevolgde wegen (en misschien ook meer) in een aanvaardbare eindtoestand stopt. De rekentijd van de NDTM komt overeen met het aantal stappen in de kortste van al die wegen.



Figuur 3.3: De twee mogelijkheden voor de NDTM

3.4 Complexiteit van algoritmen

Hoeveel berekeningsstappen heeft een algoritme nodig om de uitkomst te berekenen, hoeveel geheugenruimte is er nodig? Het analyseren van een algoritme om deze vragen te beantwoorden noemen we "complexiteitsanalyse". Dergelijke vragen zijn van belang als we de keuze hebben tussen verschillende algoritmen voor het oplossen van een probleem.

Hoe hangt de benodigde tijd of geheugenruimte af van de "grootte" of de moeilijkheidsgraad van de invoer? Om dat te beantwoorden, moeten we eerst die grootte kwantitatief kunnen uitdrukken met een of andere parameter. Het verband tussen die parameter en de benodigde rekentijd en geheugenruimte noemen we de respectievelijk de tijdscomplexiteit en de ruimtecomplexiteit van het algoritme.

3.4.1 tijdscomplexiteit van algoritmen

Definitie 3.16 (Tijdscomplexiteit) De tijdscomplexiteit van een bepaald algoritme A is een functie $tijd_A(n): \mathbb{N} \rightarrow \mathbb{N}$ die voor een gegeven invoeromvang n het maximum aantal elementaire bewerkingen aangeeft, die door het algoritme A bij een invoeromvang van grootte n zullen worden uitgevoerd.

Uit de definitie volgt onmiddelijk dat $tijd_A(n)$ een slechtste geval maat is. Het kan goed zijn dat het in de meeste gevallen veel minder dan $tijd_A(n)$ elementaire bewerkingen moet uitvoeren om tot een resultaat te komen. Dus maakt men vaak analyses van algoritmen met betrekking tot de gemiddelde complexiteit. Dit is wel wat moeilijker dus beperken wij ons tot het slechtste geval.

O-notatie

Definitie 3.17 (De O-notatie) Indien f en g functies zijn van \mathbb{N} naar \mathbb{R}^+ , dan zeggen we " $f(n)$ is $O(g(n))$ " of " f is $O(g)$ " als

$$\exists c \in \mathbb{R}_0^+, \exists N \in \mathbb{N}, \forall n \in \mathbb{N} : n \geq N \Rightarrow f(n) \leq cg(n)$$

Men zegt ook wel: f is van orde g , of f wordt asymptotisch gedomineerd door g .

Merk op dat het mogelijk is dat " f is $O(g)$ " en " g is $O(f)$ " allebei gelden.

De $O(\dots)$ is eigenlijk een upper bound. Dit betekent dus voor het geval hierboven dat $f(n)$ geupperbound is door $g(n)$.

Definitie 3.18 (Asymptotische equivalentie) Twee functies $f,g: \mathbb{N} \rightarrow \mathbb{R}^+$ worden asymptotische equivalent genoemd indien

$$f \text{ is } O(g) \text{ en } g \text{ is } O(f).$$

We noteren deze situatie door f is $\Theta(g)$ (en dus ook g is $\Theta(f)$).

Definitie 3.9 (Ook asymptotische equivalentie) We schrijven $f(n) \sim g(n)$, of $f \sim g$ als geldt dat

$$\forall \in \mathbb{R}_0^+, \exists N \in \mathbb{N}, \forall n \geq N : |f(n)/g(n) - 1| < \epsilon$$

of, als limiet geschreven, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Het komt neer op het feit dat $f(n)$ en $g(n)$ nauwelijks verschillen. De betekenis van $f \sim g$ is dat, voor grote n , de verhouding tussen $f(n)$ en $g(n)$ heel dicht bij 1 komt te liggen. Niet te verwarring met Θ . f is $\Theta(g)$ betekent dat voor grote n f even snel is als g , op een constante factor na.

Voorbeeld 3.29

Voor de functies $f(n) = 3n$ en $g(n) = 4n$

- f is $\Theta(g)$
- $f \sim g$ geldt niet

Voor de functies $f(n) = 3n$ en $g(n) = 3n + 2$

- f is $\Theta(g)$
- $f \sim g$

Polynomiaal versus exponentieel

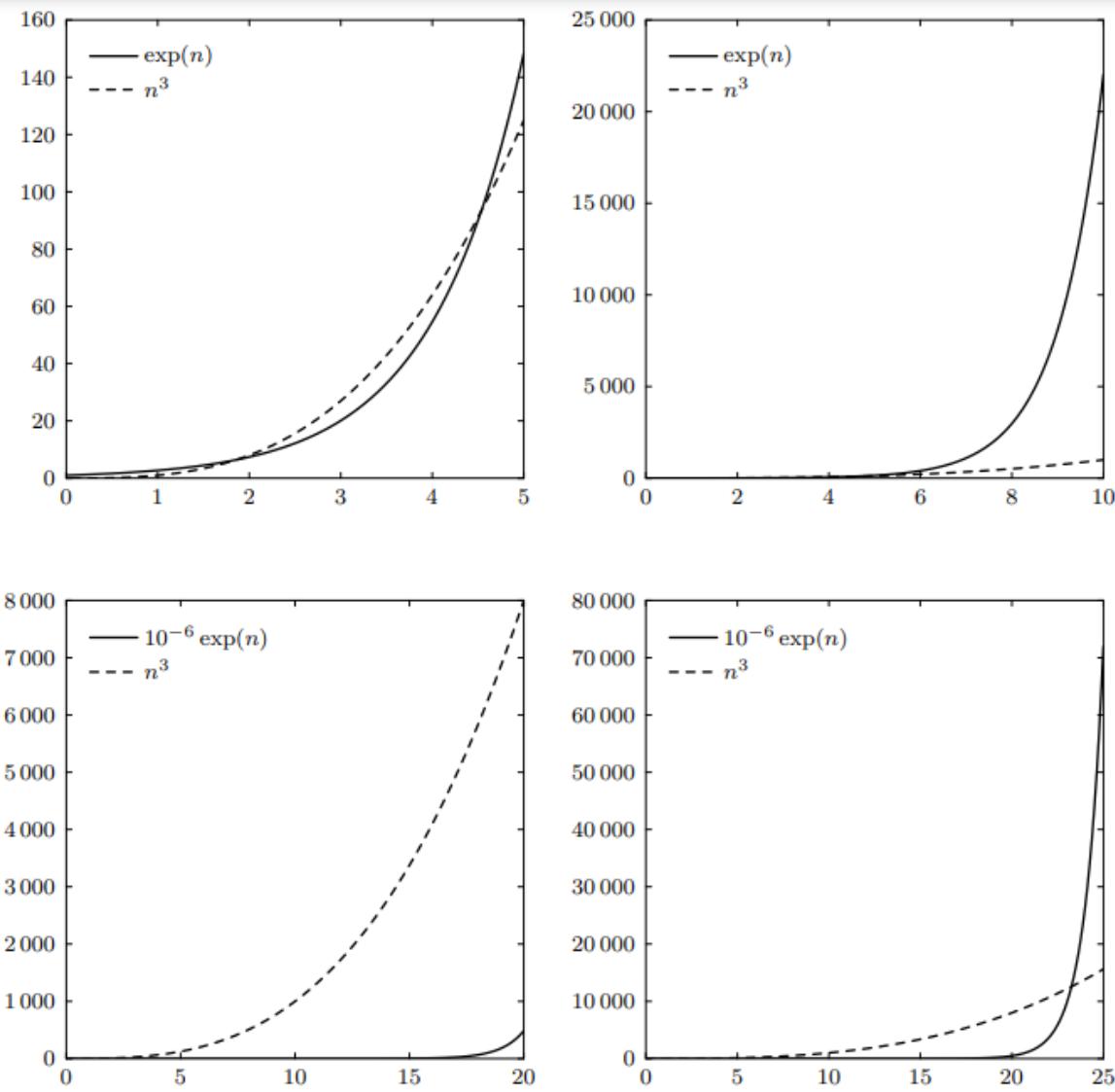
In bepaalde gevallen is het nuttig om een grovere opdeling van complexiteit te hebben. In dat kader wordt vooral verschil gemaakt tussen algoritmen met polynomiale tijdscomplexiteit en algoritmen met exponentiële tijdscomplexiteit.

Definitie 3.20 (Polynomiale tijd) We noemen een algoritme van polynomiale tijd als en slechts als zijn tijdscomplexiteit $O(n^k)$ is voor een $k \in \mathbb{N}$

Definitie 3.21 (Exponentiële tijd) We noemen een algoritme van exponentiële tijd indien zijn complexiteitsfunctie $O(\exp(n^k))$ is, voor een $k \in \mathbb{N}$

Voorbeeld

- Twee algoritmen $tijd_A(n) = n^5$ en $tijd_B(n) = 2^n$
 - $tijd_A(60) < 1.5\text{min}$
 - $tijd_B(60) > 3500\text{jaar}$



Figuur 3.5: Vergelijking van $\exp(n)$ en $10^{-6} \exp(n)$ met n^3 . Let op de schalen van de figuren.

Beheersbare en onbeheersbare problemen

Als we een bepaald probleem hebben kan het zijn dat als we hier een algoritme voor schrijven dat het ons niet lukt om dit polynomiale tijd te doen. Dit kan aan ons liggen maar het kan dat er geen polynomiaal algoritme bestaat voor dit probleem. Dergelijke problemen noemen we in het engels **intractable**; dat betekent zoveel als “onbeheersbaar” of “onhandelbaar”.

Een voorbeeld waarvan men vermoedt dat het onbeheersbaar is. Het probleem heet het Handelreizigersprobleem en wordt als volgt geformuleerd:

Gegeven: n steden c_1, c_2, \dots, c_n . Tussen elk paar steden c_i en c_j kent men ook de afstand $d_{i,j}$.

Probleem: De handelsreiziger vertrekt in 1 welbepaalde stad (zeg c_1) en wil een rondreis maken zodat hij elk van de steden c_2, c_3, \dots, c_n precies 1 keer bezoekt en vervolgens naar zijn beginpunt c_1 terugkeert (zonder nog andere steden te bezoeken). De vraag is nu: welke weg moet de handelsreiziger volgen om een zo klein mogelijke afstand af te leggen.

De huidige oplossing is niet efficiënt want we voeren zeker meer dan een veelvoud van $(n-1)!$ stappen in dit algoritme uit.

3.4.2 Het bepalen van de complexiteit in enkel concrete voorbeelden

Voorbeeld 3.30 Wat is de tijdscomplexiteit van volgend algoritme, dat de som berekent van alle elementen van een $n \times n$ matrix a (in functie van $n = a.length$)?

```
int som=0;                                (1 keer)
for (int i=0; i < a.length; i++)
    { for (int j=0; j < a.length; j++)
        som += a[i][j];
    }
```

Tijdens de uitvoering van dit programma wordt eenmaal de bewerking "som=0", uitgevoerd. Aan de variable i wordt n (met $n = a.length$) keer een waarde toegekend, aan de variable j wordt n 2 keer een waarde toegekend (voor elke keer dat aan i een waarde wordt toegekend, wordt aan j na elkaar de waarden 1,2,...,n toegekend. Het is ook duidelijk dat de bewerking "som+=a[i][j]", n 2 keer zal uitgevoerd worden. Hieruit kunnen we besluiten dat de tijdscomplexiteit van dit algoritme $O(n^2)$ is (op voorwaarde dat de tijdscomplexiteit van de bewerking in de lus constant is).

Het voorbeeld toont ook aan dat bij het bepalen van de tijdscomplexiteit van een geneste lus, het nodig en voldoende is om na te gaan hoe vaak de meest inwendige opdracht (of opdrachten) wordt (worden) uitgevoerd.

Voorbeeld 3.31 Wat is de complexiteit van volgende programma-onderdeel (in functie van n)?

```
for (int i=1; i<=n; i++)
    { for (int j=1; j<=i*i; j++)
        som++;
    }
```

Het volstaat ook hier te berekenen hoeveel keer de bewerking "som++" moet uitgevoerd worden. Noem dit aan tal $T(n)$. In de praktijk blijkt het vaak handig om een verband op te stellen tussen $T(n)$ en $T(n + 1)$ om zodoende meer informatie te bekomen over de functie $T(n)$ zelf.

Indien we n vervangen door $n + 1$ en het algoritme terug uitvoeren, dan doen we een hele tijd precies hetzelfde totdat i de waarde $n + 1$ aanneemt. Bij $i = n + 1$ wordt de bewerking "som++", nog $(n + 1)^2$ keer uitgevoerd (eenmaal voor elke waarde van j tussen 1 en $(n + 1)^2$). We vinden dus dat.

$$T(n + 1) = T(n) + (n + 1)^2 \text{ (bovendien geldt } T(1) = 1\text{)}$$

Een dergelijk verband van de vorm $T(n + 1) = T(n) + f(n)$ (met $f(n)$ een willekeurige functie) wordt een recursie betrekking genoemd.

Eigenschap 3.1 Indien een afbeelding $T : \mathbb{N} \rightarrow \mathbb{R}$ voldoet aan recursiebetrekking

$$\forall n \in \mathbb{N} : T(n+1) = T(n) + f(n)$$

voor een veelterm $f(n)$ van graad $k \geq 0$, dan is $T(n)$ zelf een veelterm van graad $k+1$

3.5 Complexiteitsklassen van beslissingsproblemen

3.5.1 de klassen P en NP

Turingmachines en de klasse P

We hebben eerder gesteld dat er een essentieel verschil is tussen **polynomiale** algoritmen, die we beheersbaar ("**tractable**") noemen, en algoritmen waarvan de tijdscomplexiteit niet door eender welke polynoom **gedomineerd** wordt (onbeheersbaar, of "**intractable**"). Het lijkt dan logisch om die lijn door te trekken naar de complexiteit van problemen. Dat leidt tot de definitie van de klasse **P**.

Definitie 3.22 (Polynomiale beslisbaarheid) Een taal L wordt in polynomiale tijd beslist door een TM M , indien er een $k \in \mathbb{N}$ bestaat zodat M voor elke invoerstring s beslist $s \in L$, in een aantal stappen dat $O(n^k)$ is, met $n = |s|$ de lengte van de invoerstring. Een taal L is **polynomiaal beslisbaar** als er een TM bestaat die de taal in polynomiale tijd beslist.

Definitie 3.23 (De klasse P) Men duidt met P de klasse aan van alle talen waarvoor geldt dat er een TM bestaat die die taal in polynomiale tijd beslist.

Elke reguliere taal is polynomiaal beslisbaar, $R \subset P$

Niet-deterministische Turingmachines en de klasse NP

We hebben eerder niet-deterministische Turingmachines gezien. Analoog aan de klasse **P** van talen die polynomiaal beslist kunnen worden door een **Turingmachine**, kunnen we ook een klasse **NP** definiëren van talen die polynomiaal beslist kunnen worden door een **NDTM**

Definitie 3.24 Tijdscomplexiteit van een NDTM

De tijdscomplexiteit $tijd_M : \mathbb{N} \rightarrow \mathbb{R}^+$ voor een NDTM M wordt gedefinieerd als

$$tijd_M(n) = \begin{cases} 0 & \text{indien er geen enkele string van lengte } n \text{ aanvaard wordt.} \\ \max\{ m \mid \text{er bestaat een string } x \in T^*, \text{ van lengte } n \\ & \text{die in } m \text{ stappen door } M \text{ aanvaard wordt.} \} \end{cases}$$

Definitie 3.25 (NDTM van polynomiale tijd) Een niet-deterministische Turingmachine M heet van polynomiale tijd als en slechts als $tijd_M(n) O(n^k)$ is, voor een $k \in \mathbb{N}$

Definitie 3.26 (Niet-deterministisch polynomiaal herkenbaar) Een taal L wordt herkend in polynomiale tijd door een NDTM M , indien M L herkent en M van polynomiale tijd is. Een taal L is niet-deterministisch polynomiaal herkenbaar als er een NDTM bestaat die L herkent in polynomiale tijd.

definitie 3.27 (De klasse NP) Men duidt met **NP** de klasse aan van alle talen die beslisbaar en niet-deterministisch polynomiaal herkenbaar zijn.

Aangezien elke deterministische TM ook kan aanzien worden als een niet-deterministische TM (maar dan een die toevallig telkens maar 1 mogelijkheid openlaat voor de volgende stap) hebben we onmiddelijk dat

$$P \subseteq NP$$

Polynomiaal verifieerbaar

Definitie: Een taal L is polynomiaal verifieerbaar als en slechts als er een TM M bestaat, zo dat voor elke string s een andere string c bestaat, een zogenaamd certificaat, zodat geldt: M aanvaardt de string (c, s) in een tijd polynomiaal $|s|$ als en slechts als $s \in L$.

(alternatieve) definitie: Men duidt met **NP** de klasse aan van alle polynomiaal verifieerbare talen.

Met andere woorden er bestaat een algoritme dat kan kijken of de gegeven oplossing juist is of niet in polynomiale-tijd.

Definitie 3.31 (Polynomiale equivalentie van talen) Twee talen L_1 en L_2 worden polynomiaal equivalent genoemd, notatie $L_1 \sim L_2$, indien

- $L_1 \propto L_2$ en $L_2 \propto L_1$

Eigenschap 3.4 De relatie \sim is een equivalentierelatie

Definitie 3.30 (Polynomiale transformatie van een taal) Zij gegeven twee talen $L_1 \subseteq T_1^*$ (op een alfabet T_1) en $L_2 \subseteq T_2^*$ (op een alfabet T_2). We zeggen dat L_1 **polynomiaal transformeert** in L_2 indien er afbeelding $f : T_1^* \rightarrow T_2^*$ bestaat waarvoor de volgende twee zaken gelden:

1. $\forall x \in T_1^* : x \in L_1 \Leftrightarrow f(x) \in L_2$
2. Er bestaat een (deterministische) TM die f in polynomiale tijd berekent

We noteren deze situatie door $L_1 \propto L_2$

Stelling 3.7 De relatie \propto is transitief. Met andere woorden, indien we drie talen L_1, L_2 en L_3 gegeven hebben met $L_1 \propto L_2$ en $L_2 \propto L_3$ dan is ook $L_1 \propto L_3$

Stelling 3.8 Gegeven twee talen L_1 en L_2 . Indien $L_1 \propto L_2$ en $L_2 \in P$ dan is ook $L_1 \in P$.

NP: Class of computational decision problems for which any given yes-solution can be verified as a solution in polynomial time by a deterministic Turing machine (or *solvable* by a *non-deterministic* Turing machine in polynomial time).

3.5.2 De klasse NP-Compleet

Naast **P** en **NP** is nog een derde klasse erg belangrijk: de klasse van "**NP-complete**" problemen. Ruwweg kunnen we zeggen dat we de ene taal als "niet moeilijker te beslissen" beschouwen dan een andere, als er een polynomiaal algoritme bestaat om eerder welk woord uit de eerste taal te vertalen in een woord uit de tweede taal.

Definitie 3.32 (De Klasse NPC) Een taal L is **NP-compleet** als en slechts als

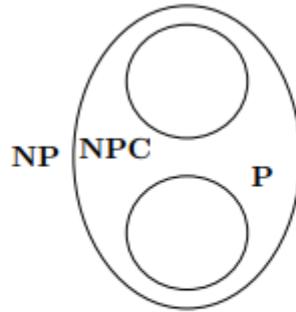
1. $L \in NP$
2. Voor elke andere taal $L' \in NP$ geldt dat $L' \propto L$.

De klasse van alle **NP**-complete talen duiden we aan door **NPC**

Stelling 3.9

1. **NPC** is een equivalentieklassen voor de relatie \sim
2. Indien $NPC \cap P \neq \emptyset$ dan is $NP = P$

De vorige stelling toont dus aan dat indien er 1 **NPC**-probleem is dat behoort tot de klasse van de problemen die oplosbaar zijn in polynomiale tijd, alle problemen binnen de klasse **NP** in polynomiale tijd oplosbaar zijn. Alles wijst er echter op dat de klasse **NP** niet samenvalt met de klasse **P**.



Figuur 3.6: De (mogelijke) onderlinge ligging van de klassen **P**, **NP** en **NPC**

Stelling 3.10 (Stelling van Cook) *SAT* is **NP-compleet**

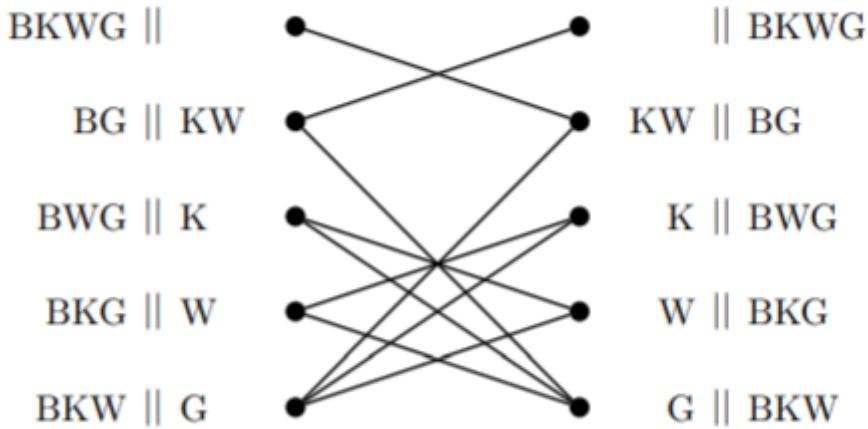
Stelling 3.11 Gegeven twee talen L_1 en L_2 die beide in **NP** zitten. Als $L_1 \propto L_2$ en $L_1 \in NPC$, dan is ook $L_2 \in NPC$.

Hoofdstuk 4 Grafentheorie

4.1 Inleiding

4.1.3 Voorbeeld

Snel voorbeeld aan de hand van een voorbeeld dat iedereen wel kent. De boer, de wolf, de kool en de geit.



Hierboven zie je de graaf die alle mogelijke opties geeft. Als je dan links vanboven begint en dan naar rechts boven probeert te gaan, is dat je oplossing.

Zie boek voor meer voorbeelden uwu

4.2 Grafen

4.2.1 Basisdefinities

Multiverzameling: Een multiverzameling lijkt op een verzameling, met als enige verschil dat hetzelfde element meerdere keren kan voorkomen in een multiverzameling.

Multipaar: Een multipaar is een multiverzameling met 2 elementen.

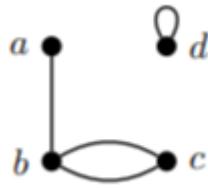
Bv. $\{1,1,2,3,5,5,5\}$ is een multiverzameling waarin 1 twee keer voorkomt, en 5 zelfs drie keer. $\{1,2\}$ is een multipaar, $\{2,2\}$ ook. De volgorde van de elementen in een multiverzameling heeft geen belang.

Definitie 4.1 Graaf Een graaf is een drietal (V, E, φ) , met V een verzameling waarvan de elementen knopen genoemd worden; E een verzameling waarvan de elementen bogen genoemd worden; en $\varphi : E \rightarrow M_2(V)$ een functie die met elke boog twee knopen associeert.

Voorbeeld 4.1 De graaf $G = (V, E, \varphi)$ met $V = \{a, b, c, d\}$, $E = \{e_1, e_2, e_3, e_4\}$, en

- $\varphi(e_1) = (a, b)$
- $\varphi(e_2) = (b, c)$
- $\varphi(e_3) = (b, c)$
- $\varphi(e_4) = (d, d)$

kan als volgt getekend worden:



Definitie 4.2 (Lus) Een lus is een boog e waarvoor geldt dat er een $v \in V$ bestaat zo dat $\varphi(e) = (v, v)$. M.a.w. een lus is een boog die een knoop met zichzelf verbindt.

Definitie 4.3 (Parallelle bogen) In een graaf $G(V, E, \varphi)$ noemen we de bogen e_1 en e_2 parallelle bogen als en slechts als $\varphi(e_1) = \varphi(e_2)$. M.a.w. twee bogen zijn parallel als en slechts als ze dezelfde knopen met elkaar verbinden.

Definitie 4.4 (Enkelvoudige graaf) Een enkelvoudige graaf is een graaf die noch lussen, noch parallelle bogen bevat.

Definitie 4.5 (Graaf) Een graaf is een koppel (V, E) , met V een verzameling knopen, en E een multiverzameling van multiparen uit V .

Definitie 4.6 (Graad) De graad van een knoop v , genoteerd $\delta(v)$, is het aantal bogen dat op v invalt. Een lus telt hierbij voor twee (omdat de lus twee keer op de knoop invalt).

Stelling 4.1 Het aantal bogen in een graaf is steeds de helft van de som van de graden.

Stelling 4.2 De som van de graden van alle knopen in een graaf is steeds even.

Stelling 4.3 Het aantal knopen met oneven graad in een graaf is altijd even.

---> De laatste 3 stellingen zijn allemaal eigenlijk triviaal en is gewoon een logisch van het geval dat als je een boog toevoegt dat deze verbonden is met 2 knopen en dus de som van de graden stijgt met 2.

4.2.2 Paden

Definitie 4.7 (Pad) Een pad in een graaf $G(V, E)$ is een rij bogen van de vorm $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)$ waarbij $\forall i : (v_i, v_{i+1}) \in E$.

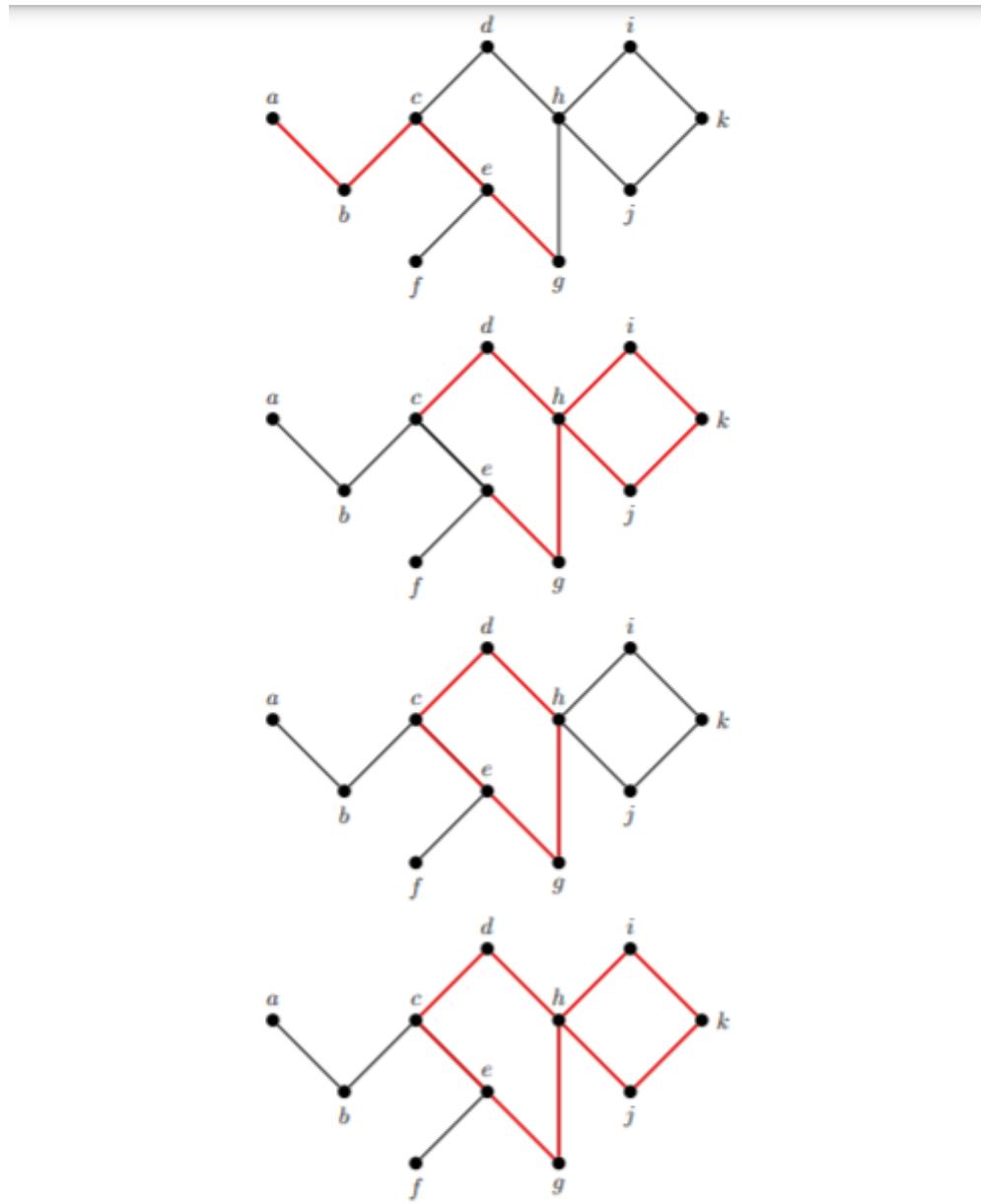
We noteren een pad $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)$ soms ook als $(v_0, v_1, \dots, v_{n-1}, v_n)$. Het aantal bogen in het pad noemen we de lengte van het pad. Het pad van lengte 0 noemen we het lege pad.

Definitie 4.8 (Enkelvoudig pad) Een enkelvoudig pad is een pad $(v_0, v_1, \dots, v_{n-1}, v_n)$ waarvan alle knopen verschillend zijn, d.w.z., $\forall i, j : i \neq j \Rightarrow v_i \neq v_j$.

Definitie 4.9 (Kring, enkelvoudige kring) Een kring is een pad $((v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n))$ waarin alle bogen verschillend zijn, en waarvoor $v_0 = v_n$ (eerste en laatste knoop ugu). Een enkelvoudige kring is een kring waarin ook alle knopen verschillend zijn, afgezien van $v_0 = v_n$.

Definitie 4.10 (Samenhangende grafen) Een graaf is samenhangend als en slechts als tussen elke twee knopen van de graaf een pad bestaat.

Definitie 4.11 (Hamiltoniaans pad, Hamiltoniaanse kring) Zij gegeven een graaf G . Een Hamiltoniaans pad van G is een pad waarin elke knoop van G precies 1 keer voorkomt. Een Hamiltoniaanse kring van G is een enkelvoudige kring waarin elke knoop van G voorkomt.



Figuur 4.2: Vier maal dezelfde graaf, waarin respectievelijk een enkelvoudig pad (a, b, c, e, g) , een niet-enkelvoudig pad $(c, d, h, j, k, i, h, g, e)$, een enkelvoudige kring (c, d, h, g, e, c) en een niet-enkelvoudige kring $(c, d, h, i, k, j, h, g, e, c)$ getoond worden.

Definitie 4.12 (Euleriaans pad, Euleriaanse kring) Zij gegeven een graaf G . Een Euleriaans pad van G is een pad waarin elke knoop van G minstens 1 keer, en elke boog van G precies 1 keer voorkomt. Een Euleriaanse kring van G is een Euleriaans pad dat ook een kring is.

Stelling 4.4 Een graaf G heeft een Euleriaanse kring als en slechts als de graaf samenhangend is en elke knoop een even graad heeft.

Stelling 4.5 Een graaf G heeft een Euleriaans pad als en slechts als de graaf samenhangend is en hoogstens 2 knopen een oneven graad hebben.

4.2.3 Deelgrafen en componenten

Definitie 4.13 (Deelgraaf) Een graaf $G'(V', E')$ is een deelgraaf van een graaf $G(V, E)$, genoteerd $G'' \subseteq G$, als en slechts als $V' \subseteq V$ en $E' \subseteq E$.

Soms willen we, gegeven een graaf $G(V, E)$ en een deelverzameling knopen $V' \subseteq V$, de deelgraaf die de knopen in V' bevat en alle bogen in G tussen de knopen van V' . Die deelgraaf noemen we de deelgraaf geïnduceerd door V' .

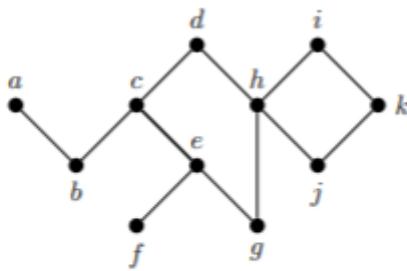
Definitie 4.14 (Geïnduceerde deelgraaf) Gegeven een graaf $G(V, E)$ en een deelverzameling $V' \subseteq V$ noemen we $G'(V', E')$ de deelgraaf van G geïnduceerd door V' als en slechts $E' = \{(v, w) \in E \mid v, w \in V'\}$.

Definitie 4.15 (Component) Een graaf $G(V, E)$ is een component van een graaf $G'(V', E')$ als en slechts als $G \subseteq G'$, G is niet leeg, G is samenhangend, en er bestaat geen samenhangende graaf G'' waarvoor $G \subset G'' \subseteq G'$.

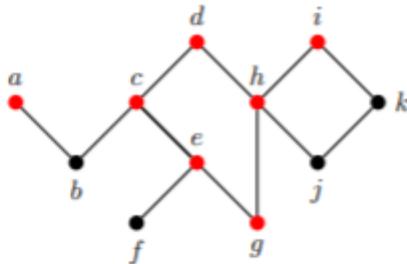
De componenten van een graaf G vormen een partitie van G , d.w.z.: elke knoop en boog komt voor in precies 1 component van G .

Hieronder vind je alles van hierboven uitgelegd in voorbeelden.

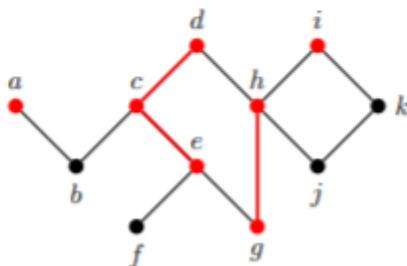
Een graaf G :



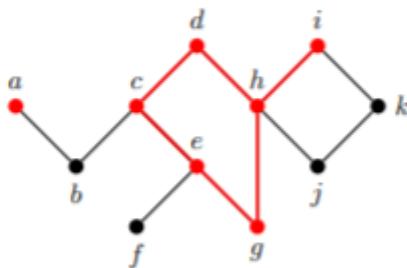
De verzameling V' is een willekeurige deelverzameling knopen:



Een deelgraaf $G'(V', E')$ van G :



De deelgraaf van G geïnduceerd door V' bevat alle bogen van G die tussen elementen van V' lopen:



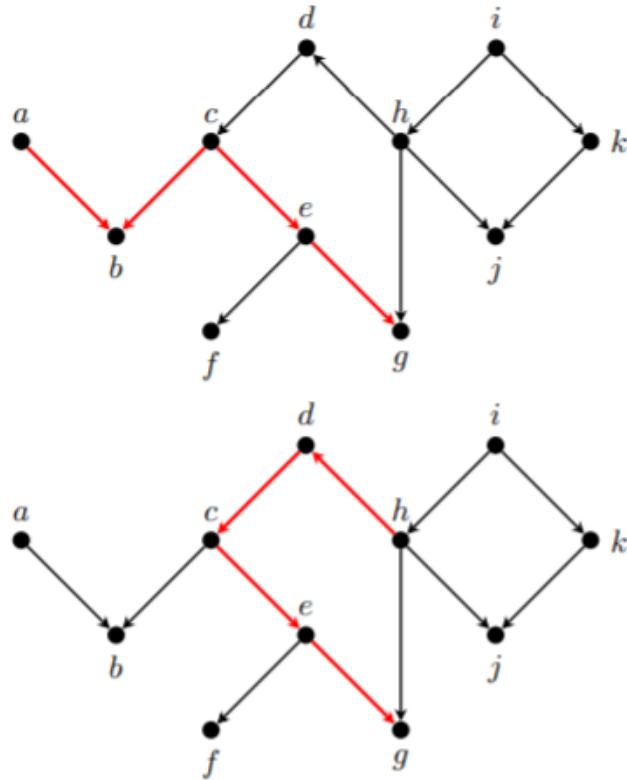
Figuur 4.3: Illustratie van de begrippen “deelgraaf” en “geïnduceerde deelgraaf”

4.2.4 Gerichte grafen

Bij een gewone graaf geven de bogen een symmetrische relatie aan: als v verbonden is met w , is w automatisch ook verbonden met v . Soms hebben we grafen nodig waarin de bogen een richting hebben. Er kan dan een boog van v naar w zijn zonder dat er een boog van w naar v is. Dat leidt tot het concept van een gerichte graaf.

Definitie 4.16 (Gerichte graaf) Een gerichte graaf (Eng. directed graph of digraph) is een koppel (V, E) , met V een verzameling waarvan de elementen knopen genoemd worden, en $E \subseteq V^2$ een verzameling koppels

Definitie 4.17 (Gericht pad, ongericht pad) Een gericht pad in een gerichte graaf $G(V, E)$ is een pad (v_0, v_1, \dots, v_n) met $\forall i : (v_i, v_{i+1}) \in E$. Een ongericht pad is een pad (v_0, v_1, \dots, v_n) met $\forall i : (v_i, v_{i+1}) \in E \vee (v_{i+1}, v_i) \in E$.



Figuur 4.4: Een gerichte graaf. In de bovenste graaf is een ongericht pad tussen a en g aangeduid; in de onderste graaf een gericht pad van h naar g .

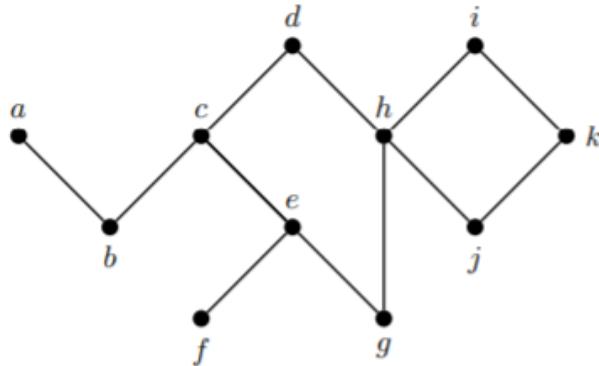
4.3 Voorstellingen van grafen

We hebben grafen tot nog toe gewoonlijk als tekeningen voorgesteld, of als koppels (V, E) met V een verzameling knopen en E een verzameling bogen. Er zijn nog andere voorstellingen mogelijk. In deze sectie bekijken we matrixvoorstellingen van grafen. Een voordeel van deze voorstellingswijze is dat allerlei nuttige berekeningen met grafen dan met behulp van matrixoperaties kunnen gebeuren.

4.3.1 Buurmatrijs (adjacency matrix)

Definitie 4.18 (Buurmatrijs) Gegeven een enkelvoudige graaf $G(V, E)$, met $V = \{v_1, v_2, \dots, v_n\}$, is de buurmatrijs van G een $n \times n$ -matrix A met $A_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$, en $A_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$ (geen element van) E .

Ik had eerst echt totaal niet door wat er met een buurmatrijs bedoelt werd ma das gewoon een matrix van een graaf en als ge een 1 hebt op een plek betekent dat die 2 noden verbonden zijn op de graaf. Eigenlijk niet zo moeilijk.



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad (4.1)$$

Figuur 4.5: Een graaf met bijhorende buurmatrijs. De rijen en kolommen komen overeen met de knopen in alfabetische volgorde.

Stelling 4.7 Als A de buurmatrijs van een enkelvoudige graaf G(V,E) is, geldt voor A^k het volgende: voor alle i en j is A^k_{ij} gelijk aan het aantal verschillende paden van lengte k tussen v_i en v_j

4.3.2 Booleaanse buurmatrijs

4.3.2 Booleaanse buurmatrijs In plaats van matrices met getallen kunnen we ook booleaanse matrices definieren. Booleaanse matrices hebben booleaanse waarden in hun cellen staan. Som en product van dergelijke matrices worden op de gebruikelijke manier gedefinieerd, met dit verschil dat de som van twee getallen hier een disjunctie van twee booleaanse waarden wordt, en het product van twee getallen een conjunctie van booleaanse waarden. Dit wil zeggen dat het product van twee Booleaanse matrices A en B gedefinieerd wordt als $P = A \cdot B \Leftrightarrow \forall i, j : P_{ij} = A_{i1} \wedge B_{1j} \vee A_{i2} \wedge B_{2j} \vee \dots \vee A_{in} \wedge B_{nj} =$

$$\bigvee_k A_{ik} \wedge B_{kj}$$

en hun som als

$$S = A + B \Leftrightarrow \forall i, j : S_{ij} = A_{ij} \vee B_{ij}$$

We kunnen van de buurmatrijs een booleaanse versie definieren

Stelling 4.8 Als B de booleaanse buurmatrix van $G(V, E)$ is, geldt voor B^k het volgende: voor alle i en j is B_{ij}^k equivalent met "er bestaat een pad van lengte k van v_i naar v_j ".

Bewijs: Wordt als oefening gelaten. Dit kan bewezen worden op dezelfde manier als de vorige stelling, of je kan gebruik maken van de vorige stelling en van de eigenschap dat "er bestaat een pad" overeenkomt met "het aantal paden is strikt groter dan 0".

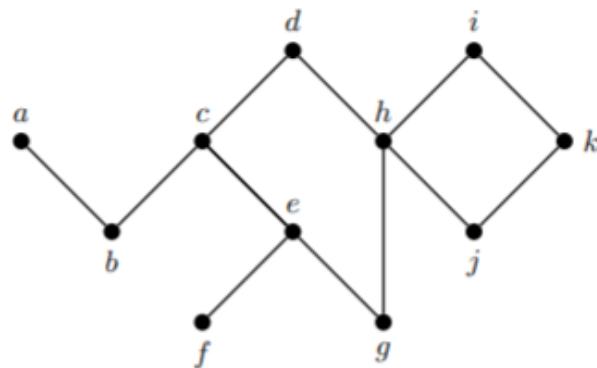
Stelling 4.9 Als B de booleaanse buurmatrix van $G(V, E)$ is, geldt voor $S = \sum_{k=1}^l B^k$ het volgende: voor alle i en j is S_{ij} equivalent met "er bestaat een pad van lengte l of korter tussen v_i en v_j ".

Bewijs: Wordt als oefening gelaten.

4.3.3 Incidentiematrix

Definitie 4.20 (Incidentiematrix) Gegeven een enkelvoudige graaf $G(V, E)$, met $V = \{v_1, v_2, \dots, v_n\}$ en $E = \{e_1, e_2, \dots, e_m\}$, is de incidentiematrix van G een $n \times m$ matrix A met $A_{ij} = 1$ als e_j invalt op v_i , en $A_{ij} = 0$ in alle andere gevallen.

90



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (4.2)$$

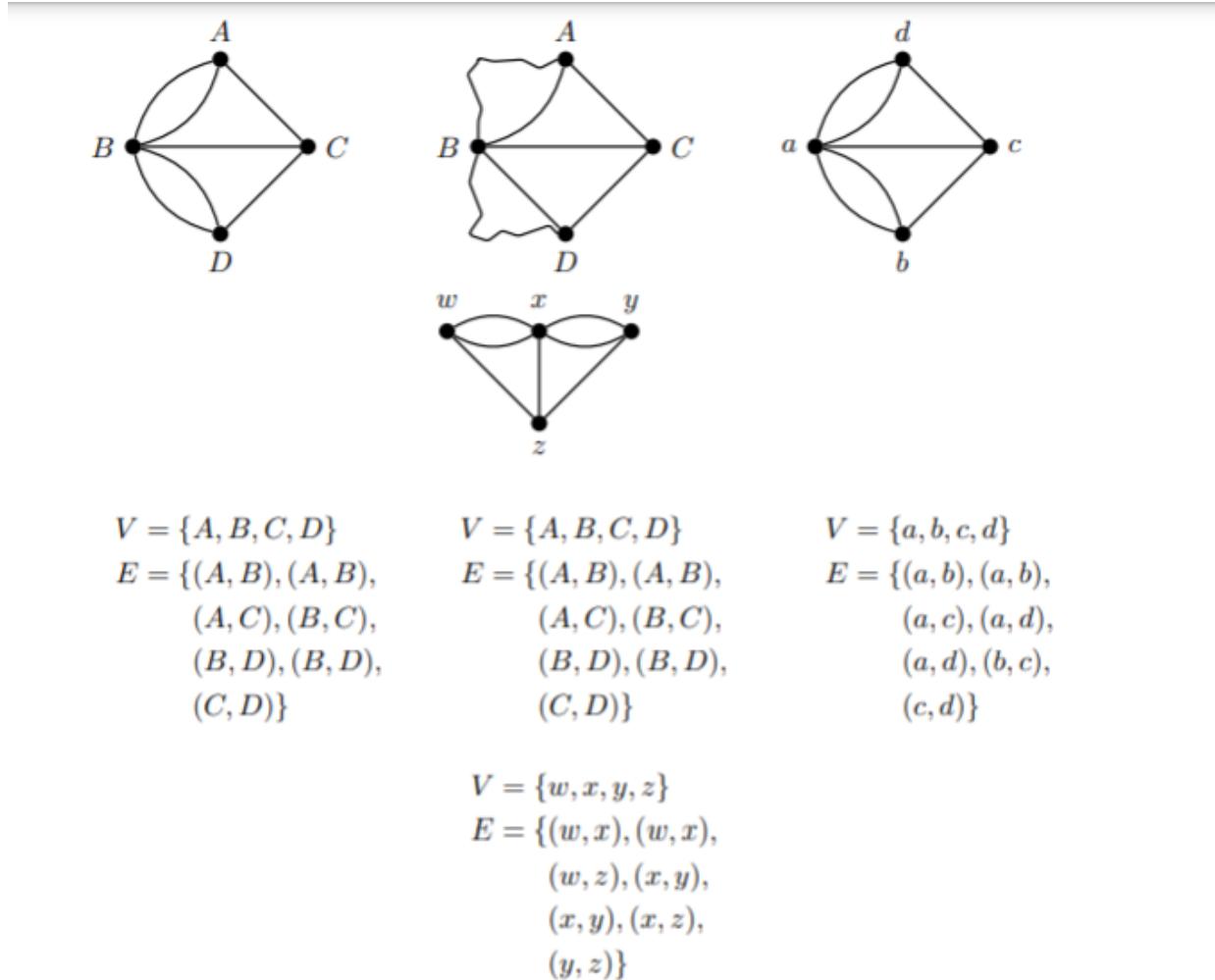
Figuur 4.6: Een graaf met bijhorende incidentiematrix. De rijen komen overeen met knopen in alfabetische volgorde, de kolommen zijn de opeenvolgende bogen (a, b) , (b, c) , (c, d) , (c, e) , ...

4.4 Isomorfisme

Defintie 4.21 (Graaf-isomorfisme) Twee grafen $G(V, E)$ en $G'(V', E')$ zijn isomorf als en slechts als er een bijectie $h : V \rightarrow V'$ bestaat, zo dat $\{h(x) | x \in V\} = V'$ en $\{(h(x), h(y)) | (x, y) \in E\} = E'$

De functie h stelt een “hernoeming” van de knopen voor; ze beeldt elementen van V af op elementen van V' . Als twee grafen isomorf zijn, moeten de knopen van de eerste graaf hernoemd kunnen worden, zo dat, na toepassing van die hernoeming op V en E , V' en E' bekomen wordt

Isomorfisme komt er eigenlijk op neer dat V en E van beide grafen gelijk zijn, op de namen van de elementen na.



Figuur 4.7: Dezelfde graaf op verschillende manieren getekend, met daaronder telkens een mogelijke formele voorstelling van de graaf.

Al deze grafen zijn isomorf!

Stelling 4.10 Twee enkelvoudige grafen zijn isomorf als en slechts als er een ordening van de knopen bestaat zodat hun buurmatrix gelijk is.

Stelling 4.11 Twee grafen G en G' zijn isomorf als en slechts als er een ordening van de knopen en bogen bestaat waarvoor de incidentiematrices van G en G' gelijk zijn.

Bewijs: Het bewijs wordt als oefening gelaten.

4.5 Gewogen grafen

In dit gedeelte bekijken we grafen waarvoor aan elke boog e een gewicht $w(e) \in R_0^+$ toegekend is.

- Het gewicht van een graaf definiëren we als de som van de gewichten van alle bogen in die graaf.
- Het gewicht van een pad definiëren we als de som van de gewichten van de bogen in dat pad.
- Het kortste pad tussen a en b is per definitie het pad tussen a en b met het kleinste gewicht

4.5.1 Het kortste-pad-algoritme van Dijkstra

We beginnen met een lichtjes vereenvoudigde versie van het **Dijkstra** algoritme. Dit algoritme berekent enkel het gewicht van het kortste pad, niet het pad zelf. Daarna zullen we het uitbreiden zodat het ook het pad zelf aanduidt.

4.5.2 Dijkstra, versie 1

Het volgende algoritme berekent het gewicht van het kortste pad van een gegeven knoop a naar een gegeven knoop z in een enkelvoudige, samenhangende, gewogen graaf $G(V, E)$ met $a, z \in V$.

Algoritme: Dijkstra's kortste-pad-algoritme

```
R := ∅  
L(a) := 0  
voor alle  $v \neq a$ :  $L(v) := \infty$   
zolang  $z \notin R$ : (*) op dit punt geldt  $I_1$  en  $I_2$   
    kies  $v \in V \setminus R$  zo dat  $L(v)$  zo klein mogelijk is  
     $R := R \cup \{v\}$   
    voor elke boog  $(v, v') \in E$  met  $v' \in V \setminus R$ :  
         $L(v') := \min\{L(v) + w(v, v'), L(v')\}$ 
```

Dit algoritme heeft de volgende eigenschap: na afloop is $L(z)$ gelijk aan het gewicht van het kortste pad van a naar z . Dat is niet zo heel gemakkelijk in te zien, en daarom zullen we het bewijzen.

Alvorens we dat doen, introduceren we eerst het concept invariante eigenschap, ook wel invariante relatie of kortweg invariant genoemd. Wie reeds vertrouwd is met dit begrip, kan de volgende sectie overslaan, of enkel de cursieve paragrafen lezen.

Stelling 4.12 Zij gegeven een gewogen graaf $G(V, E)$, een knoop $a \in V$ en een knoop $z \in V$, waarop het algoritme van Dijkstra uitgevoerd wordt. Wanneer het algoritme van Dijkstra eindigt, is $L(z)$ het gewicht van het kortste pad van a naar z , d.w.z. $L(z) = \lambda(z)$.

Bewijs: Het algoritme eindigt wanneer bij het begin van de lus gedetecteerd wordt dat $z \in R$. Op dat moment geldt I_2 : $\forall v \in R : L(v) = \lambda(v)$, en $z \in R$, waaruit direct volgt dat $L(z) = \lambda(z)$.

Stelling 4.13 Voor elke eindige graaf geldt: het algoritme van Dijkstra eindigt steeds.

Bewijs: Bij elke uitvoering van de buitenste lus wordt een nieuwe knoop v aan R toegevoegd. Als er in totaal n knopen zijn, kan de lus dus niet vaker dan n keer uitgevoerd worden. Een enkele uitvoering van de lus is ook eindig: het vinden van de knoop met laagste L in $V \setminus R$, en het overlopen van alle bogen $(v, v') \in E$, kan in eindige tijd vanwege de eindigheid van V en E . Daaruit volgt het gestelde.

4.5.3 Dijkstra, versie 2

Zoals gezegd berekende het vorige algoritme enkel het gewicht van het kortste pad. Om het kortste pad zelf te kunnen construeren, is het voldoende om bij elke wijziging van een $L(v')$ aan te duiden vanuit welke knoop die wijziging gebeurd is. Dat is namelijk de knoop die op het kortste rode pad de voorlaatste knoop is voor we in v' aankomen. Omdat die informatie voor alle knopen bijgehouden is, kunnen we op deze manier voor elke knoop in R het kortste pad achterwaarts reconstrueren

Algoritme: Dijkstra's kortste-pad-algoritme, versie 2

```

 $R := \emptyset$ 
 $L(a) := 0$ 
voor alle  $v \neq a$ :  $L(v) := \infty$ 
zolang  $z \notin R$ :
    kies  $v \in V \setminus R$  zo dat  $L(v)$  minimaal is
     $R := R \cup \{v\}$ 
    voor elke boog  $(v, v') \in E$ :
        als  $L(v) + w(v, v') < L(v')$  dan:
             $L(v') := L(v) + w(v, v')$ 
             $p(v') := v$ 

```

Als het kortste pad van a naar z gelijk is aan $P = (v_0 = a, v_1, v_2, \dots, v_{i-1}, v_i = z)$, dan geldt na afloop van het algoritme dat $L(z) = w(P)$, en $p(v_i) = v_{i-1}$ voor alle $i \geq 1$.

4.5.4 Complexiteit van Dijkstra's algoritme

- $O(n^2)$
- mits verfijning: $O(|V| \log |V| + |E|)$

4.5.5 Enkele doordenkertjes

oefeningen

4.6 Bijzondere klassen van grafen

Er bestaan veel speciale grafen of klassen van grafen, die grondig bestudeerd zijn omwille van bepaalde interessante eigenschappen die ze hebben. Veel van die grafen of klassen hebben ook vaste namen gekregen.

Elke eigenschap die we gezien hebben definieert een klasse van grafen. bv:

spreken over de klasse van alle samenhangende grafen, alle enkelvoudige grafen, enzovoort. Maar er zijn nog veel meer klassen te bedenken.

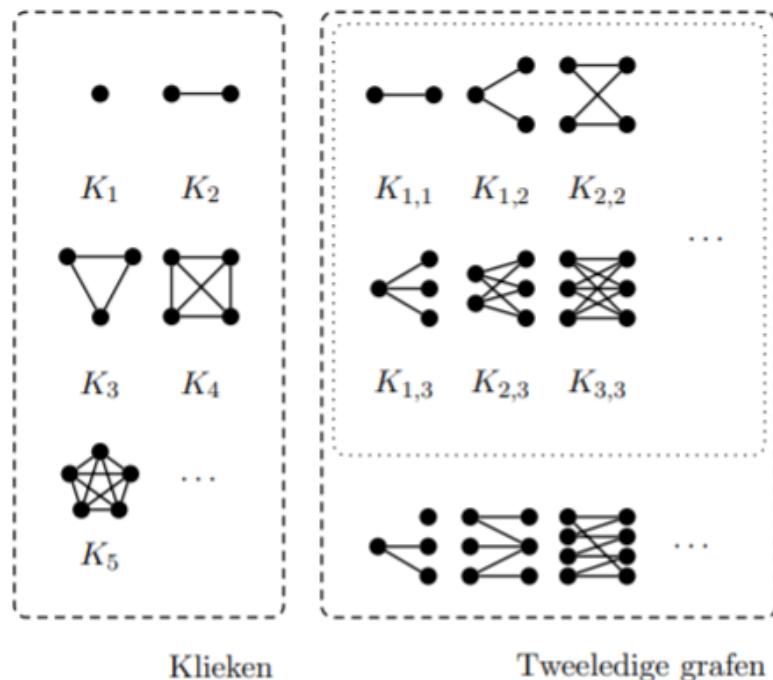
Zo is er bijvoorbeeld de klasse van alle klieken. Een kliek (Eng. clique) is een enkelvoudige graaf waarin elke knoop rechtstreeks verbonden is met elke andere knoop. De kliek met n knopen wordt K_n genoemd. Figuur 4.8 toont een aantal klieken.

Een goede video over tweeledige grafen: <https://www.youtube.com/watch?v=HqIUbSA9cEY> + Pro tip.
When attempting to determine whether a graph is bipartite or not, if two vertices are connected by an edge, due to the definition of a bipartite graph, these two vertices have to be in different sets, and thus, you can colour code them differently. Continue this trend through the graph. If you find a contradiction where two vertices that are connected by an edge are of the same colour, then the graph is not a bipartite graph. Similarly, the opposite can be said to claim that a graph is bipartite.

Een andere klasse zijn de tweeledige grafen. Een enkelvoudige graaf $G(V, E)$ is tweeledig als en slechts als V opgedeeld kan worden in twee deelverzamelingen V_1 en V_2 , zo dat er enkel bogen bestaan tussen V_1 en V_2 , maar nooit binnen V_1 of V_2 . Formeel: $G(V, E)$ is tweeledig als en slechts als er een V_1 en V_2 bestaan zo dat $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, en $E \subseteq \{(v_1, v_2) | v_1 \in V_1, v_2 \in V_2\}$.

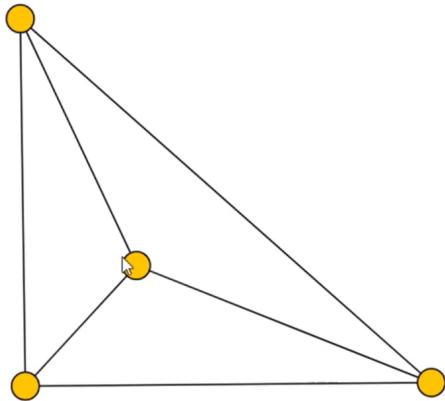
Een tweeledige graaf wordt volledig verbonden genoemd als elke knoop uit V_1 met elke knoop uit V_2 verbonden is. De volledig verbonden tweeledige graaf met m knopen in V_1 en n knopen in V_2 wordt $K_{m,n}$ genoemd.

Figuur 4.8 toont een aantal tweeledige grafen, waaronder enkele die volledig verbonden zijn.



Figuur 4.8: De klasse van alle klieken, en de klasse van alle tweeledige grafen met als subklasse de volledig verbonden tweeledige grafen.

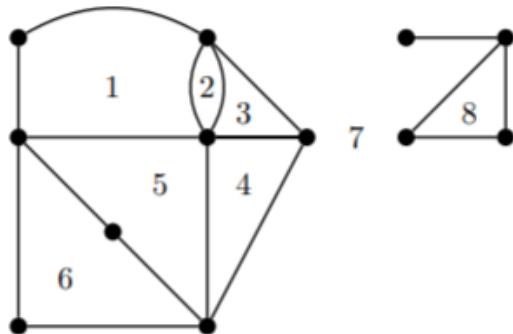
K_4 is vlak zelfs al lijkt het niet zo



4.7 Vlakke grafen

Elke graaf kan getekend worden door met elke knoop een punt in het vlak te associëren, en met elke boog (x, y) een lijn die de punten voor x en y met elkaar verbindt. Sommige grafen kunnen bovendien zo getekend worden dat hun bogen elkaar niet snijden. Dergelijke grafen noemen we vlakke grafen.

Definitie 4.22 Een vlakke graaf is een graaf die getekend kan worden zonder snijdende bogen.



Figuur 4.9: Een vlakke graaf die uit twee componenten bestaat. De zijvlakken zijn genummerd van 1 tot 8. Merk op dat het buitenste gebied ook meetelt als zijvlak.

Dit is een tamelijk informele definitie. Een formelere definitie van vlakke grafen is mogelijk, maar heeft weinig nut voor de rest van deze cursus

Eigenschap 4.1 Elke deelgraaf van een vlakke graaf is vlak.

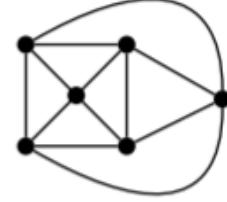
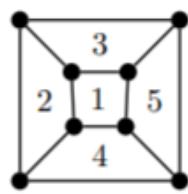
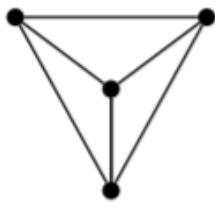
Eigenschap 4.2 Elke kringvrije graaf is vlak en heeft 1 zijvlak.

4.7.1 De formule van Euler

Vlakke grafen hebben de eigenschap dat, eender hoe je de graaf tekent (maar zonder snijdende bogen), het aantal zijvlakken steeds hetzelfde is. Daardoor heeft "het aantal zijvlakken van een vlakke graaf" een eenduidige betekenis. Voor samenhangende vlakke grafen volgt het aantal zijvlakken zelfs direct uit het aantal knopen en bogen. Het verband tussen die drie getallen wordt de formule van Euler genoemd. Alvorens we die bewijzen, geven we eerst een hulpstelling.

Lemma 4.13 In een kringvrije graaf met minstens 1 boog bestaat er steeds een knoop met graad 1.

Stelling 4.14 (Formule van Euler) Zij G een samenhangende vlakke graaf met $v \geq 1$ knopen, e bogen en f zijvlakken. Dan geldt: $v - e + f = 2$



Figuur 4.10: Vlakke grafen die overeenkomen met respectievelijk een regelmatig viervlak, zesvlak en achtvlak. De zes zijvlakken van de kubus zijn genummerd zoals op een dobbelsteen.

We gebruiken ook in het vervolg van deze sectie de conventie dat v , e en f staan voor het aantal knopen, bogen en zijvlakken van een graaf G , en analoog gebruiken we v' , e' en f' voor G' , enzovoort.

Dit volgende stuk is echt verwarring, ik heb dit gecopypaste dus hopelijk is het niet belangrijk

Veelvlakken

Er is een verband tussen vlakke grafen en veelvlakken, die meteen het gebruik van het woord "zijvlak" verklaart. Een veelvlak is een ruimtelijk lichaam dat begrensd is door een aantal zijvlakken; de grenslijnen tussen de zijvlakken worden ribben genoemd, en de punten waar de ribben samenkomen, hoeken. Als we zo'n veelvlak bekijken, is het duidelijk dat je dat veelvlak kan projecteren op het vlak door 1 zijvlak "uit te rekken" tot het zo groot is als alle andere zijvlakken samen (waarbij de andere zijvlakken ook vervormd mogen worden om dit mogelijk te maken). Die projectie in het vlak vormt dan een vlakke graaf waarbij de hoeken knopen zijn, de ribben bogen, en de zijvlakken zijvlakken; het "meest uitgerokken" zijvlak wordt voorgesteld door het buitenste zijvlak. Een regelmatig veelvlak is een veelvlak waarvan elk zijvlak door evenveel ribben omgeven is, en elke hoek evenveel ribben verbindt. Er zijn maar vijf regelmatige veelvlakken; ze worden "Platonische lichamen" genoemd. Met elk dergelijke veelvlak komt een vlakke graaf overeen, waarin elke knoop dezelfde graad heeft en elk zijvlak (ook het buitenste!) evenveel begrenzende bogen. Figuur 4.10 toont de vlakke grafen die overeenkomen met een regelmatig viervlak (of tetra"eder), zesvlak (kubus), en achtvlak (octa"eder). Dat er maar vijf Platonische lichamen bestaan, kan via grafen bewezen worden. Met een Platonisch lichaam komt een vlakke graaf overeen met een constante graad δ voor elke knoop (minstens 3), en een constant aantal begrenzende bogen β per veelvlak (ook minstens 3). We zagen eerder dat de som van de graden, $v\delta$, gelijk is aan $2e$. Omdat elke boog precies twee zijvlakken scheidt, geldt analoog dat $f\beta = 2e$. Dit invullen in de formule van Euler geeft $2e/\delta - e + 2e/\beta = 2$, waaruit $1/\delta - 1/2 + 1/\beta = 1/e$ (deel beide leden door $2e$) en uiteindelijk $1/\delta + 1/\beta = 1/e + 1/2 > 1/2$. Met $\delta \geq 3$ en $\beta \geq 3$ gehele getallen, zijn de enige oplossingen $1/3 + 1/3$, $1/3 + 1/4$, $1/3 + 1/5$, $1/4 + 1/3$,

4.7.2 Karakterisatie van vlakke grafen

We hebben vlakke grafen enkel informeel gedefinieerd. We zullen nu aantonen dat alle vlakke grafen een welbepaalde eigenschap hebben, die andere grafen niet hebben. Die eigenschap kan in principe gebruikt worden voor een alternatieve definitie van vlakke grafen. Een dergelijke alternatieve definitie wordt ook wel een "karakterisatie" genoemd.

Definitie 4.23 (β , B) Gegeven een vlakke graaf G , noteren we het aantal bogen waardoor een zijvlak z begrensd wordt als $\beta(z)$. De som van $\beta(z)$ voor alle zijvlakken z wordt B genoteerd: $B = \sum_{z \in Z} \beta(z)$, met Z de verzameling zijvlakken van de graaf.

Eigenschap 4.3 Voor elke vlakke graaf geldt: $B \leq 2e$

Eigenschap 4.4 Voor elke enkelvoudige vlakke graaf met $f \geq 2$ geldt: $B \geq 3f$.

Eigenschap 4.5 Voor elke enkelvoudige vlakke graaf met $f \geq 2$ geldt: $2e \geq 3f$, of nog, $f \leq (2/3)e$.

We bewijzen nu volgende belangrijke stelling.

Stelling 4.15 K_5 en $K_{3,3}$ zijn niet vlak.

Bewijs: Beide grafen zijn samengehangend, dus als ze ook vlak zijn, moeten ze voldoen aan de formule van Euler. We bewijzen dat ze daar niet aan voldoen, en bijgevolg niet vlak kunnen zijn.

Stel dat K_5 vlak is. K_5 is enkelvoudig en heeft kringen, dus $f \geq 2$, dus geldt (Eig. 4.5). $f \leq (2/3)e$, dus $v - e + f \leq v - e + (2/3)e$. Voor K_5 is $v = 5$ en $e = 10$. Invullen in het rechterlid geeft $v - e + f \leq 5/3$, in strijd met de formule van Euler. Stel dat $K_{3,3}$ vlak is. $K_{3,3}$ bevat kringen. Omdat $K_{3,3}$ enkelvoudig is, bevat elke kring tenminste 3 bogen; omdat $K_{3,3}$ tweeledig is, bevat elke kring bovendien een even aantal bogen, dus minstens 4. Elk zijvlak is dus begrensd door minstens 4 bogen
 $\Rightarrow B \geq 4f \Rightarrow 2e \geq 4f \Rightarrow f \leq e/2 \Rightarrow v - e + f \leq v - e + e/2$. In $K_{3,3}$ is $v = 6$ en $e = 9$; dan is $v - e + f \leq 3/2$, in strijd met de formule van Euler.

Homeomorfisme

In een tekening van een graaf is een knoop met graad 2 in zekere zin overbodig: we kunnen die knoop ook weglaten en de getekende bogen ernaartoe vervangen door een enkele getekende boog; zie Figuur 4.11. We noemen dit een rijdreductie. De tekening van de graaf verandert niet op essentiële wijze door een rijdreductie toe te passen. Wanneer een graaf omgevormd kan worden tot een andere graaf door middel van rijdreducties, zeggen we dat de eerste graaf homeomorf is met de tweede.

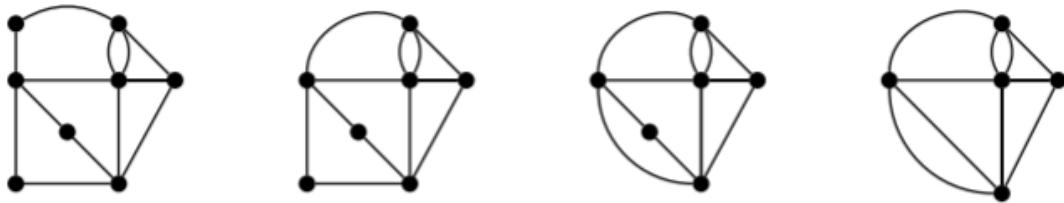
Definitie 4.24 (Homeomorfisme) Een graaf $G(V, E)$ is homeomorf met een graaf $G'(V', E')$ als en slechts als G' uit G bekomen kan worden door een of meer keren een knoop v van graad 2 te kiezen, die samen met zijn invallende bogen (v, w) en (v, u) te verwijderen, en boog (u, w) toe te voegen.

Eigenschap 4.6 Als G homeomorf is met G' , geldt: G is vlak $\Leftrightarrow G'$ is vlak.

De stelling van Kuratowski

De volgende stelling geeft een karakterisatie van vlakke grafen.

Stelling 4.16 (Stelling van Kuratowski) Een graaf is vlak als en slechts als de graaf geen deelgraaf bevat die homeomorf is met K_5 of $K_{3,3}$.



Figuur 4.11: De graaf links wordt via drie rijreducties omgevormd tot de graaf rechts. We zeggen dat de linkergraaf homeomorf is met de rechter.

Bewijs: We moeten de volgende twee implicaties bewijzen: (1) G is vlak $\Rightarrow G$ heeft geen deelgraaf die homeomorf is met K_5 of $K_{3,3}$

(2) G heeft geen deelgraaf die homeomorf is met K_5 of $K_{3,3} \Rightarrow G$ is vlak

We bewijzen hier enkel deel (1). Deel (2) is veel moeilijker; na Kuratowski's oorspronkelijk bewijs zijn er meerdere alternatieve bewijzen voorgesteld, maar geen daarvan past binnen het bestek van deze cursus. Bewijs van deel (1): uit het ongerijmde: stel dat een vlakke graaf G een deelgraaf G_0 heeft die homeomorf is met K_5 of $K_{3,3}$, dan is G_0 niet vlak (wegen Eigenschap 4.6), en dan kan G ook niet vlak zijn (wegen Eigenschap 4.1).

4.7.3 Duale Grafen

Definitie 4.25 (Duale graaf) Zij G een graaf met v knopen, e bogen, f zijvlakken. De duale graaf G' heeft 1 knoop voor elk zijvlak van G , dus $v' = f$ knopen. Voor elke boog tussen de oorspronkelijke zijvlakken is er een boog tussen de overeenkomstige knopen in de duale graaf.

Twee knopen in G' zijn dus verbonden als en slechts als de bijhorende zijvlakken aan elkaar grenzen. Als die grens uit meerdere bogen bestaat, zijn er ook meerdere parallelle bogen tussen de knopen.

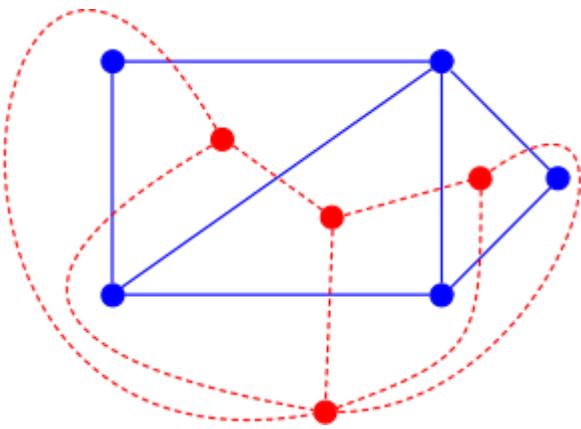
Duale grafen hebben de volgende eigenschap.

Eigenschap 4.7 De duale graaf van een vlakke graaf is ook vlak.

Bewijs: Intuïtief bewijs: Beschouw een tekening van een vlakke graaf. Teken de duale graaf door eerst in het midden van elk zijvlak een knoop te tekenen. Voor aan elkaar grenzende zijvlakken kunnen we altijd een boog tussen hun centrumknopen tekenen zonder andere bogen te snijden.

Merk op: als z een zijvlak is, en x de overeenkomstige knoop in de duale graaf, dan geldt steeds: $\delta(x) = \beta(z)$.

Een duale graaf is een graaf die gemaakt worden wanneer je in elk zijvlak een knoop zet en deze dan verbindt door telkens 1 van de bogen. Zie afbeelding voor betere uitleg.



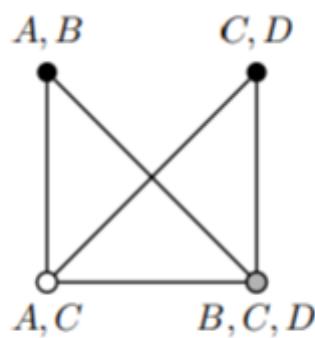
4.8 Kleuringen van grafen

Definitie 4.26 met een kleuring van een graaf (V, E) bedoelt men een toekenning van een kleur aan elke $v \in V$ zodanig dat de kleur van v en w verschilt indien $(v, w) \in E$. Een n -kleuring is een kleuring met n of minder verschillende kleuren. Een minimale kleuring is een n -kleuring met minimale n .

Er zijn heel wat praktische toepassingen van het (minimaal) kleuren van een graaf.

Als voorbeelden:

- Je moet 4 vergaderingen plannen voor 4 personen A,B,C en D. In de eerste vergadering zitten A,B; in de tweede A,C, in de derde zitten B,C,D en in de vierde C,D. Wat is het minimale aantal tijdstippen waarop een vergadering moet gepland worden? Twee vergaderingen moeten op een verschillend tijdstip doorgaan als eenzelfde persoon aan beide vergaderingen moet deelnemen. Je kan het probleem voorstellen door de graaf van Figuur 4.12: elke knoop stelt een vergadering voor en twee vergaderingen zijn verbonden als ze niet op hetzelfde tijdstip mogen doorgaan omdat er iemand in beide vergaderingen moet zijn. Je zoekt er de kleuring met het kleinste aantal kleuren voor en je hebt het antwoord (elke nieuwe kleur komt overeen met een nieuw tijdstip).



Figuur 4.12: De vergaderingsgraaf met kleuring

- Je moet in een warenhuis een aantal goederen opstapelen in de rekken, maar je mag bepaalde goederen niet naast elkaar zetten: bijvoorbeeld benzine mag niet naast brood, porno niet naast kuisproducten enzovoort. Ook hier kan je een graaf opstellen die al die beperkingen voorstelt en waarbij een kleuring van de graaf het probleem oplaat.

Een probleem dat ook een praktisch aspect heeft, maar vooral historisch belangrijk is, is het "vierkleurenprobleem": kan elke landkaart gekleurd worden met vier verschillende kleuren zodanig dat twee aangrenzende landen nooit dezelfde kleur hebben? Beschouw de kaart als een vlakke graaf G waarvan de zijvlakken landen zijn, de bogen de grenzen, en de drielandenpunten knopen. Het inkleuren van de landen in de kaart komt overeen met het kleuren van de duale graaf van G . De duale graaf is eveneens vlak en enkelvoudig. Het probleem van het inkleuren van kaarten herleidt zich dus tot het probleem van het kleuren van knopen in een graaf. Het vierkleurenprobleem wordt dan: Heeft elke vlakke graaf een 4-kleuring?

Het probleem werd voor het eerst gesteld door Francis Guthrie rond 1850. Men was overtuigd dat het kan, maar het vermoeden bleef onbewezen - ondanks verwoede pogingen van menig wiskundige - tot 1976, toen K. Appel en W. Haken bewezen dat er 110 1936 grafen bestaan waarvan er minstens 1 terug te vinden is in elke minimale niet 4-kleurbare graaf en vervolgens bewezen dat zulke grafen niet vlak zijn. Beide stappen in het bewijs werden geleverd door een computerprogramma. Er is tot nog toe geen eleganter bewijs gevonden voor de vierkleurenstelling. Dat vijf kleuren volstaan voor het inkleuren van een vlakke graaf, is wel relatief gemakkelijk te bewijzen, en dat zullen we hier aantonen. We introduceren eerst enkele hulpstellingen.

Stelling 4.17 Voor elke enkelvoudige vlakke graaf met $e \geq 2$ geldt: $e \leq 3v - 6$.

Stelling 4.18 In elke vlakke, enkelvoudige graaf bestaat er minstens een knoop, zeg w , zodanig dat $\delta(w) \leq 5$.

Bewijs: Dit is duidelijk waar voor een graaf met hoogstens 1 boog. Als de graaf minstens 2 bogen heeft, moet gelden: $e \leq 3v - 6$. Stel nu dat de stelling niet voldaan is voor zo'n graaf, d.w.z. alle knopen hebben graad 6 of meer, dan is de som van de graden van alle knopen ($= 2e$) minstens $6v$, en bijgevolg $e \geq 3v$, in tegenspraak met $e \leq 3v - 6$.

De volgende stelling laat zien dat het kleuren van een vlakke graaf altijd kan met vijf kleuren:

Stelling 4.19 Elke enkelvoudige, vlakke graaf $G(V, E)$ heeft een 5-kleuring.

Bewijs: check boek op 116, dees is te kankerlang. Echte tering. KRANK

K_n heeft n kleuren nodig

4.9 Bomen

Bomen zijn een speciaal soort grafen. Er zijn verschillende definities van bomen te geven. De meest intuïtieve is waarschijnlijk deze: Een boom is een samenhangende graaf die geen kringen bevat. Hieronder staan een paar alternatieve definities die equivalent zijn.

1. Een boom is een enkelvoudige graaf waarin tussen elke twee knopen precies 1 pad bestaat.
2. Een boom is een samenhangende graaf met n knopen en $n - 1$ bogen.
3. Een boom is een kringvrije graaf met n knopen en $n - 1$ bogen.
4. In deze definities, en in de rest van dit hoofdstuk, bedoelen we met een pad steeds een enkelvoudig pad.³ We zullen de equivalentie van die definities aantonen. Alvorens we dat doen, geven we een paar hulpstellingen.

³Het is duidelijk dat de eis “er bestaat precies 1 pad” enkel zinvol is als we enkelvoudige paden bedoelen, want anders bestaan er automatisch oneindig veel paden tussen twee verbonden knopen: $(x,y), (x,y,x,y), (x,y,x,y,x,y), \dots$

Stelling 4.20 Tussen twee verschillende knopen v en w die deel uitmaken van dezelfde kring, bestaan steeds minstens 2 verschillende paden.

Intuïtief is dit duidelijk: in een kring kan je linksom of rechtsom naar een andere knoop gaan.

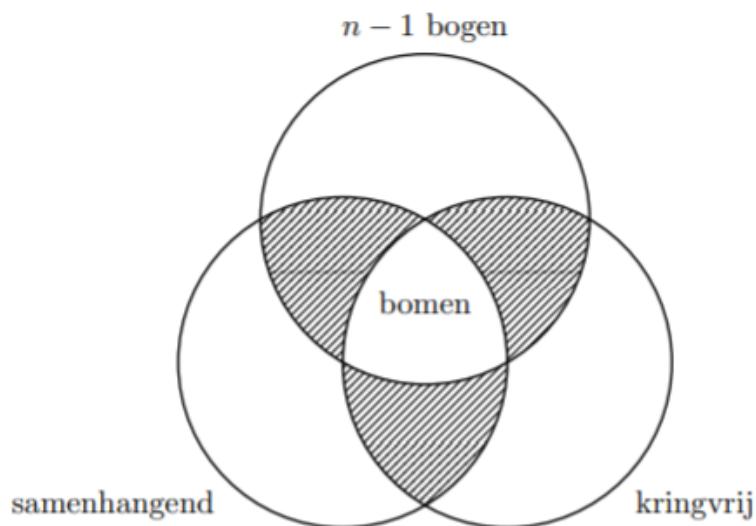
Stelling 4.21 Als tussen twee verschillende knopen in een graaf G twee verschillende paden P_1 en P_2 bestaan, bevat G een kring.

Stelling 4.22 Zij T een graaf met n knopen. Elk van de volgende condities impliceert alle andere:

1. T is enkelvoudig en tussen elke 2 knopen van T bestaat precies 1 pad.
2. T is samenhangend en kringvrij.
3. T is samenhangend en heeft $n - 1$ bogen.
4. T is kringvrij en heeft $n - 1$ bogen.

Vergelijk dit even met de alternatieve definities van boom in het begin van deze sectie!

110



Figuur 4.16: Grafen voldoen steeds aan 0, 1 of 3 van de getoonde eigenschappen. Bomen zijn grafen die aan alle drie voldoen.

4.9.1 Ospannende bomen

Een ospannende boom van een graaf G is een deelgraaf van G die een boom is en die alle knopen van G bevat. Formeel:

Definitie 4.27 $T(V_T, E_T)$ is een ospannende boom voor $G(V, E)$ als en slechts als: T is een boom, $V_T = V$ en $E_T \subseteq E$.

Het is duidelijk dat enkel samenhangende grafen een ospannende boom kunnen hebben. Het bewijs wordt als oefening gelaten.

Stelling 4.23 Als G samenhangend is, dan bestaat er een ospannende boom T voor G .

Bewijs: G is samenhangend. Als G daarenboven geen kringen heeft, is G zelf een boom, en dus bij definitie een ospannende boom voor zichzelf. Als G wel kringen heeft, kunnen we volgende procedure herhalen tot er geen kringen meer zijn: kies een kring, laat een willekeurige boog weg uit die kring; het resultaat T is samenhangend (want door enkel bogen uit kringen weg te laten wordt de samenhangendheid niet gescheiden), en kringvrij, dus een boom; T bevat bovendien alle knopen van G en is dus een ospannende boom.

Uit deze stelling en de stelling uit voorgaande oefening volgt direct:

Stelling 4.24 Een graaf heeft een ospannende boom als en slechts als de graaf samenhangend is.

Bovenstaand bewijs geeft meteen een methode om een ospannende boom (afgekort OB) voor een graaf te construeren: gegeven G , herhaal "kies een kring en laat een boog uit die kring weg" tot er geen kringen meer zijn; het resultaat is een OB voor G . Dit algoritme is niet noodzakelijk erg efficiënt, als G een grote graaf met veel kringen is; bovendien is het zoeken van kringen in een graaf een tamelijk dure operatie.

We kunnen ospannende bomen ook construeren door met een lege boom te beginnen en bogen toe te voegen, in plaats van met de volledige graaf te beginnen en bogen weg te laten. Onderstaande stelling geeft aan dat als we aan een kringvrije deelgraaf van G bogen toevoegen tot we niet meer kunnen zonder dat er kringen ontstaan, we eindigen met een ospannende boom

Stelling 4.25 Als T een maximale kringvrije deelgraaf is van een samenhangende graaf G , dan is T een ospannende boom van G

Dit is om te wenen man.

Bewijs: We moeten bewijzen dat T samenhangend is en alle knopen van G bevat (omdat T kringvrij is, volgt daaruit immers dat T een OB van G is).

1. T is samenhangend. Bewijs (uit het ongerijmde): stel dat T niet samenhangend is, dan valt T uiteen in verschillende componenten T_i . Omdat G wel samenhangend is, is er in G zeker een pad tussen willekeurige knopen $v \in T_1$ en $w \in T_2$. Dat pad begint in T_1 en eindigt buiten T_1 , dus er moet ergens een boog (x, y) zijn waardoor T_1 verlaten wordt, d.w.z. $x \in T_1$ en $y \notin T_1$. Omdat x en y niet tot dezelfde component behoren, is er geen pad in T dat ze verbindt; bijgevolg kan de boog (x, y) geen kring doen ontstaan in T , en kan de boog (x, y) toegevoegd worden aan T . Dat spreekt tegen dat T maximaal was.

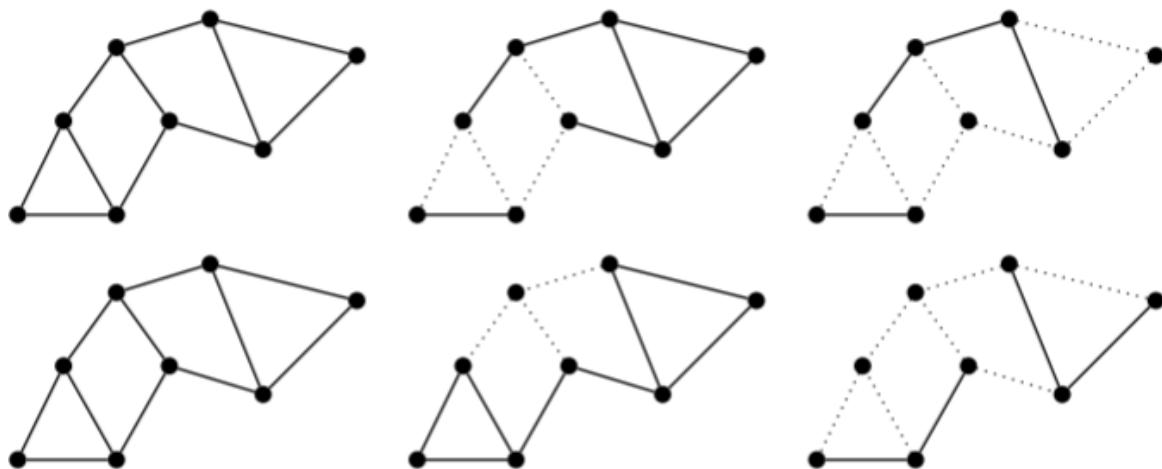
2. T bevat alle knopen van G. Bewijs (uit het ongerijmde): Stel dat er een knoop $v \in G$ bestaat waarvoor $v \notin T$. Omdat G samenhangend is, kan v geen geïsoleerde knoop zijn, er bestaat dus een boog (v, w) in G. Die boog kan toegevoegd worden aan T zonder dat er een kring ontstaat: zo'n kring kan immers alleen ontstaan als er al een pad tussen v en w bestond in T, en omdat $v \notin T$ kan dat niet. Een correct algoritme voor het construeren van een opspannende boom van een graaf G is dus: begin met een lege deelgraaf T; voeg aan T herhaaldelijk een boog uit G toe die geen kring introduceert in T, tot dat niet meer mogelijk is; het resultaat is een OB voor G. Uit al het voorgaande volgt direct:

Stelling 4.26 Als T een kringvrije deelgraaf is van een samenhangende graaf G, en T heeft $n - 1$ bogen, dan is T een opspannende boom voor G.

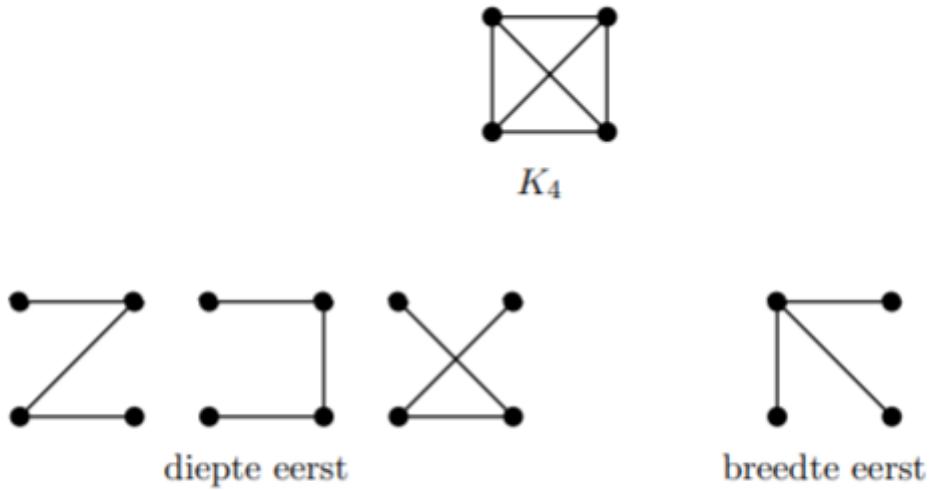
Merk op dat we op elk moment mogen kiezen welke boog we toevoegen, zolang er maar geen kringen ontstaan. We kunnen een aantal systematische manieren bedenken om bogen toe te voegen. Twee daarvan noemen we diepte-eerst en breedte eerst. Deze termen zullen later nog terugkomen.

- Diepte-eerst constructie van een OB: begin met een willekeurige knoop v_0 . Maak vervolgens een zo lang mogelijk pad vanuit v_0 . Dit wil zeggen: voeg een boog (v_0, v_1) toe, vervolgens een boog (v_1, v_2) , (v_2, v_3) , etc., tot het pad niet meer langer gemaakt kan worden zonder een kring te maken. Op dat moment keren we terug naar de laatst toegevoegde v_i van waaruit een alternatieve boog toegevoegd had kunnen worden; construeer vanuit die v_i een pad dat begint met die alternatieve boog, opnieuw zo lang mogelijk. Herhaal dit principe van terugkeren en een alternatief pad maken tot er nergens nog een boog toegevoegd kan worden.
- Breedte-eerst constructie van een OB: begin met een willekeurige knoop v_0 . Voeg alle bogen toe die op v_0 invallen en geen kring maken. Zij V_1 de eindpunten van al die bogen, m.a.w. de verzameling knopen die op die manier aan T toegevoegd worden. Voeg vervolgens zoveel mogelijk bogen (v, w) toe met $v \in V_1$; de eindpunten van al die bogen noemen we V_2 . Zo gaan we verder tot er geen bogen meer toegevoegd kunnen worden.

Figuur 4.17 illustreert het geleidelijk opbouwen van een OB voor een voorbeeldgraaf, volgens diepte-eerst of breedte-eerst.

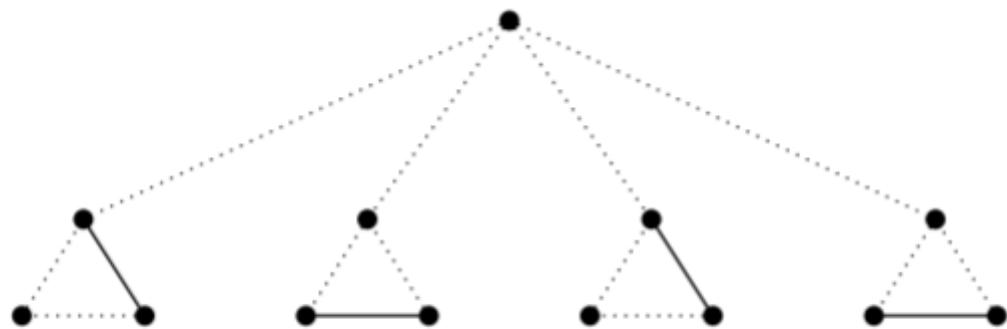


Figuur 4.17: Illustratie van diepte-eerste (boven) en breedte-eerst (onder) constructie van een OB.



Figuur 4.18: Ospannende bomen voor K_4

Figuur 4.19 laat een graaf zien en een ospannende boom, die noch met diepte-eerst, noch met breedte-eerst kan verkregen worden. Het construeren van ospannende bomen is erg belangrijk voor veel toepassingen die een zoekprobleem inhouden. Een zoekprobleem kan vaak herleid worden tot het doorlopen van een graaf op zoek naar een knoop die een oplossing voorstelt voor een gegeven probleem. We beginnen dan bij een bepaalde knoop, volgen een pad dat ons opeenvolgend bij verschillende andere knopen brengt, en als we niet meer verder kunnen keren we op onze stappen terug en volgen een ander pad. Het is niet de bedoeling dat we via dat ander pad weer op bepaalde knopen uitkomen die we al bekijken hebben: dat is tijdverlies, want we hoeven elke knoop maar 1 keer te bekijken. Als we een graaf willen doorlopen zonder dat we een knoop meerdere keren tegenkomen, maar aan de andere kant willen we zeker zijn dat we geen knopen overslaan, dan komt dat eigenlijk neer op het opstellen van een ospannende boom.



Figuur 4.19: Graaf met een hybride ospannende boom

4.9.2 Minimale opspannende bomen

Beschouwen we het volgende probleem: gegeven een aantal steden en stel dat de kost van het bouwen van een weg tussen de steden gegeven is; bepaal welke wegen moeten aangelegd worden om te voldoen aan (1) de totale kost is minimaal, en (2) elke stad is bereikbaar vanuit elke andere stad. Het wegennet dat daaraan voldoet moet een boom zijn, want er kunnen geen kringen zijn (anders heeft het net zeker geen minimale kost), en er is een pad tussen elke twee steden. Dit soort bomen wordt nu gedefinieerd.

Definitie 4.28 Een minimale opspannende boom van een gewogen graaf G is een opspannende boom van G waarvoor geldt dat geen andere opspannende boom een kleiner gewicht heeft.

We bekijken twee verschillende algoritmen om een minimale opspannende boom (afgekort MOB) van een gewogen graaf te vinden.

Het algoritme van Prim

Het algoritme van Prim bouwt een MOB door met een willekeurige knoop te beginnen, en van daaruit een boom op te bouwen door herhaaldelijk een boog toe te voegen aan de reeds opgebouwde boom, op zo'n manier dat opnieuw een boom bekomen wordt, en zo dat, wanneer het algoritme stopt, de gevonden boom een minimale opspannende boom is. Om dat laatste te garanderen, blijkt het voldoende om bij elke toevoeging steeds een boog te kiezen met een zo laag mogelijk gewicht.

Algoritme: Algoritme van Prim

Gegeven: een samenhangende graaf $G(V, E)$ met n knopen

Resultaat: T , een MOB voor G

$T := (\{v_0\}, \emptyset)$

zolang T minder dan $n - 1$ bogen heeft:

kies $(u, v) \in E$ zo dat $u \in T$, $v \notin T$, en $w(u, v) = \min\{w(x, y) | x \in T, y \notin T\}$

voeg (u, v) toe aan T

Bovenstaande beschrijving van het algoritme is het eenvoudigst om te begrijpen (**Blijkbaar**). We kunnen het herschrijven zodat het duidelijker is welke bogen er allemaal kandidaat zijn om toegevoegd te worden, we noemen die verzameling kandidaat-bogen K . We schrijven het algoritme wat formeler en compacter.

Algoritme: Algoritme van Prim

Gegeven: een samenhangende graaf $G(V, E)$ met n knopen

Resultaat: T , een MOB voor G

$T := (\{v_0\}, \emptyset)$

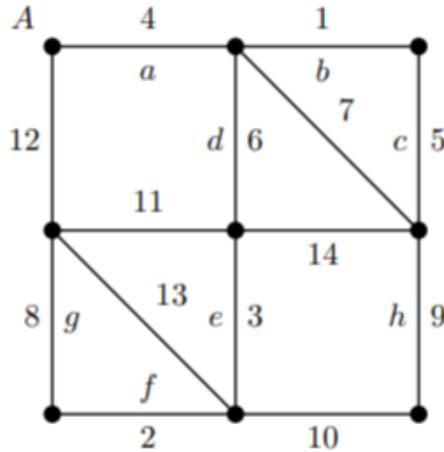
zolang T minder dan $n - 1$ bogen heeft:

$K := \{(u, v) \in E | u \in T, v \notin T\}$

$T := T \cup \{\arg \min_{e \in K} w(e)\}$

We gebruiken hierbij de operator "arg min". Dat betekent: gegeven een functie f met een domein S , geeft de operator $\arg \min_{x \in S} f(x)$ een willekeurige x terug waarvoor geldt dat $f(x)$ minimaal is.

Het algoritme van Prim wordt geïllustreerd in Figuur 4.20: de initiële knoop is A ; de volgorde waarin de bogen worden toegevoegd, staat bij de bogen als a, b, c, \dots, h .



Figuur 4.20: De uitvoering van het algoritme van Prim.

We geven nu het correctheidsbewijs voor het algoritme van Prim

Stelling 4.27 Het algoritme van Prim construeert een MOB van G .

Bewijs: T is op elk moment samenhangend en kringvrij, dus een boom; dus zolang T nog geen $n - 1$ bogen bevat, bevat T nog geen n knopen. Neem een knoop buiten T . Er bestaat een pad van die knoop naar T (want G is samenhangend), en dus een boog (v, w) met $v \in T$ en $w \in V \setminus T$. Dat betekent dat K nooit leeg is; in elke uitvoering van de lus wordt dus precies 1 boog aan T toegevoegd; het algoritme eindigt bijgevolg steeds, en op het einde is T een OB. Dat T op het einde een minimale OB is, tonen we aan door te bewijzen dat T op elk moment een deelboom van een MOB is. Formeel bewijzen we de volgende invariant:

I: er bestaat een MOB T' zodat $T \subseteq T'$.

I geldt zeker bij het begin ($T = (\{v_0\}, \emptyset)$): omdat G samenhangend is bestaat er een MOB, en omdat die elke knoop bevat, bevat hij ook v_0 , en dus T . Als I geldt bij het begin van de uitvoering van de lus, dan ook bij het begin van de volgende uitvoering. Stel dat er bij het begin van de lus een MOB T' bestaat waarvoor $T \subseteq T'$. We voegen vervolgens een boog e aan T toe. Er zijn dan twee mogelijkheden:

- (1) $e \in T'$: samen met $T \subseteq T'$ volgt dan $T \cup \{e\} \subseteq T'$, dus de invariant blijft gelden.
- (2) $e / \in T'$: in dat geval bevat $T' \cup \{e\}$ een kring. Die kring bevat naast e nog een andere boog e' die een knoop van T verbindt met een knoop buiten T .
Beschouw nu $T'' = T' \cup \{e\} \setminus \{e'\}$. T'' is nog steeds een opspannende boom (want samenhangend en $n - 1$ bogen) en $w(T'') = w(T') + w(e) - w(e')$. Omdat $w(e) \leq w(e')$ (we hebben e immers zo gekozen) volgt dan $w(T'') \leq w(T')$. Aangezien T' een MOB was, moet T'' dat dan ook zijn. Verder hebben we $T \cup \{e\} \subseteq T''$. Dus na toevoegen van e aan T blijft de invariant gelden.

Het algoritme van Kruskal

Het algoritme van Kruskal werkt op een gelijkaardige manier als dat van Prim, maar de deelgraaf S die opgebouwd wordt is niet noodzakelijk samenhangend (dus niet noodzakelijk een boom), wel altijd kringvrij.

Algoritme: Algoritme van Kruskal

Gegeven: een samenhangende graaf $G(V, E)$ met n knopen

Resultaat: S , een MOB voor G

$S := (\emptyset, \emptyset)$

zolang S minder dan $n - 1$ bogen heeft:

$(u, v) :=$ de boog $\notin S$ met het kleinste gewicht die geen kring in S maakt
voeg (u, v) toe aan S

Als we dit formeel schrijven dan verschilt het niet veel van Prim, afgezien van een klein verschil in de initialisatie, is het enige verschil de definitie van de verzameling van K van bogen die kandidaat zijn om toegevoegd te worden.

Algoritme: Algoritme van Kruskal

Gegeven: een samenhangende graaf $G(V, E)$ met n knopen

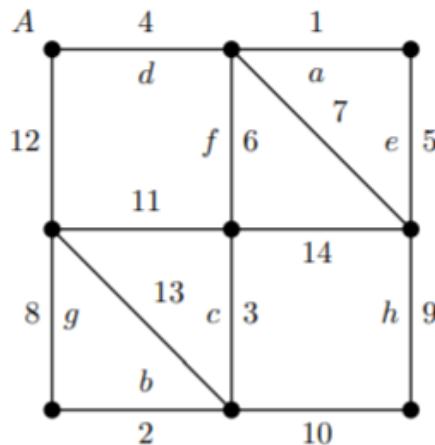
Resultaat: S , een MOB voor G

$S := (\emptyset, \emptyset)$

zolang S minder dan $n - 1$ bogen heeft:

$K := \{e \in E | e \notin S, S \cup \{e\} \text{ is kringvrij}\}$
 $S := S \cup \{\arg \min_{e \in K} w(e)\}$

Merk op dat S tijdens Kruskal's algoritme geen boom hoeft te zijn. De uitvoering van Kruskal wordt in Figuur 4.21 geïllustreerd op dezelfde graaf als in Figuur 4.20.



Figuur 4.21: De uitvoering van het algoritme van Kruskal.

We bewijzen nu dat ook het algoritme van Kruskal een minimale oplettende boom oplevert.

Stelling 4.28 Het algoritme van Kruskal construeert een MOB van G

Bewijs: Het bewijs is bijna identiek aan het bewijs voor Prim. De delen die verschillen zijn cursief weergegeven. *Zolang S nog geen $n-1$ bogen bevat, is S zeker geen OB. Dan zijn er twee mogelijkheden: S bevat nog niet alle knopen van G, of S is niet samenhangend. In beide gevallen zijn er bogen in G die toegevoegd kunnen worden aan S zonder kringen te vormen (namelijk op een pad van S naar een knoop buiten S, of op een pad dat twee componenten van S verbindt, of tussen twee knopen die nog niet tot S behoren).*

Dat betekent dat K nooit leeg is; in elke uitvoering van de lus wordt dus precies 1 boog aan S toegevoegd; het algoritme eindigt bijgevolg steeds, en op het einde is S een OB. Dat S op het einde een minimale OB is, tonen we aan door te bewijzen dat S op elk moment een deelgraaf van een MOB is. Formeel bewijzen we de volgende invariant:

I: er bestaat een MOB T 0 zodat $S \subseteq T_0$.

I geldt zeker bij het begin ($S = (\emptyset, \emptyset)$): omdat G samenhangend is bestaat er een MOB, en omdat S leeg is, omvat die MOB zeker S.

Als I geldt bij het begin van de uitvoering van de lus, dan ook bij het begin van de volgende uitvoering. Stel dat er bij het begin van de lus een MOB T' bestaat waarvoor $S \subseteq T'$. We voegen vervolgens een boog e aan S toe. Er zijn dan twee mogelijkheden: (1) $e \in T'$: samen met $S \subseteq T'$ volgt dan $S \cup \{e\} \subseteq T'$, dus de invariant blijft gelden. (2) $e \notin T'$: in dat geval bevat $T' \cup \{e\}$ een kring. Die kring bevat naast e nog een andere boog e' die niet in S zit (anders zou toevoeging van e aan S een kring gemaakt hebben, namelijk deze kring). Beschouw nu $T'' = T' \cup \{e\} \setminus \{e'\}$. T'' is nog steeds een opspannende boom (want samenhangend en $n - 1$) en $w(T'') = w(T') + w(e) - w(e')$. Omdat $w(e) \leq w(e')$ (we hebben e immers zo gekozen) volgt dan $w(T'') \leq w(T')$. Aangezien T' een MOB was, moet T'' dat dan ook zijn. Verder hebben we $S \cup \{e\} \subseteq T''$. Dus na toevoegen van e aan T blijft de invariant gelden.

4.10 Gewortelde bomen

Definitie 4.29 Een gewortelde boom is een boom waarin een willekeurige knoop aangeduid is als "de wortel".

Formeel is een gewortelde boom een tupel (V, E, w) met $T = (V, E)$ een boom en $w \in V$. In de praktijk schrijven we meestal gewoon T, waarbij V, E en w impliciet gedefinieerd zijn als de verzameling knopen, verzameling bogen, en wortel van de boom. In een gewortelde boom T is elke knoop v verbonden met de wortel w door een uniek pad $P_v = (v_0=w, v_1, v_2, \dots, v_h=v)$. We definiëren dan de volgende termen:

- De lengte van dit pad, h , noemen we de hoogte van v , genoteerd $h(v)$.
- Voor elke i, j met $i < j$ geldt: v_i is een ouder van v_{i+1} , v_{i+1} is een kind van v_i , v_i is een voorouder van v_j , v_j is een afstammeling van v_i .
- Als v en w kinderen zijn van dezelfde ouder, noemen we ze broers of zussen. (Het Engels kent hiervoor een geslachtsneutrale term: siblings).
- Een knoop die geen kinderen heeft, noemen we een blad. Een knoop die geen blad is, is een inwendige knoop.
- De deelboom van T met wortel v is de gewortelde boom die v en al zijn afstammelingen bevat (en de bogen ertussen), en v als wortel heeft.

De hoogte van een gewortelde boom is de maximale hoogte van zijn knopen: $h(T) = \max\{h(v) | v \in T\}$. We tekenen gewortelde bomen gewoonlijk met de wortel bovenaan. Om die reden wordt de hoogte van een knoop ook wel de diepte van die knoop genoemd. In deze sectie bedoelen we, wanneer we over een "boom" spreken, steeds een gewortelde boom.

4.10.1 Binaire bomen

Behalve de wortel heeft elke knoop in een gewortelde boom precies 1 ouder, en behalve de bladeren heeft elke knoop minstens 1 kind. Een boom waarin elke knoop hoogstens twee kinderen heeft, wordt een binaire boom genoemd. Een boom waarin elke inwendige knoop precies twee kinderen heeft, wordt een volledige binaire boom genoemd. In een binaire boom maken we onderscheid tussen het linker- en rechterkind; m.a.w., een boom waarin x_1 het linkerkind van x is en x_2 het rechterkind, is niet hetzelfde als een boom waar dat omgekeerd is. (Formeel is een binaire boom een tupel (V, E, w, λ) met $E \subseteq V \times V$, $w \in V$, $\lambda : V \times \{l, r\} \rightarrow E$, waarbij de graaf $G(V, E)$ een boom is, en de partiële functie λ voor elke inwendige knoop aangeeft welke boog naar het linker- en rechterkind leidt. Dit terzijde; we zullen verder deze formele structuur niet gebruiken.)

Stelling 4.29 Elke volledige binaire boom met i inwendige knopen heeft $i+1$ bladeren en $2i+1$ knopen in totaal.

Bewijs: Elke inwendige knoop heeft 2 kinderen, die verschillen van elkaar en van alle kinderen van andere knopen. Het totaal aantal kinderen is bijgevolg $2i$. Alleen de wortel is geen kind van een andere knoop; het totaal aantal knopen is dus $2i+1$. Het aantal bladeren is het totaal aantal knopen min het aantal inwendige knopen, dus $i+1$

Toepassing Bij een toernooi met directe uitschakeling met n deelnemers.

Stelling 4.30 In een binaire boom met t bladeren en hoogte h geldt: $t \leq 2^h$

Bewijs: Door inductie op de hoogte h

Basis: $h=0$; een boom met $h = 0$ heeft maar 1 knoop, die ook een blad is, dus $t = 1 = 2^0 = 2^h$.

Inductiestap: we bewijzen dat als de formule geldt voor alle bomen van hoogte $h-1$, dan ook voor alle bomen van hoogte h . Neem een boom T van hoogte $h > 0$; T heeft 1 of 2 deelbomen (want anders zou $h = 0$ zijn).

(1) Stel dat T 2 deelbomen heeft, T_l (met hoogte h_l , en t_l bladeren) en T_r (met h_r, t_r). Omdat T hoogte h heeft, is $h_l < h$, dus $h_l \leq h - 1$, en analoog $h_r \leq h - 1$. Uit de inductiehypothese volgt dan $t_l \leq 2^{h_l}$ en $t_r \leq 2^{h_r}$. Alle bladeren van T zijn bladeren van T_l of van T_r , dus $t = t_l + t_r \leq 2^{h_l} + 2^{h_r} \leq 2^{h-1} + 2^{h-1} = 2^h$, wat te bewijzen was.

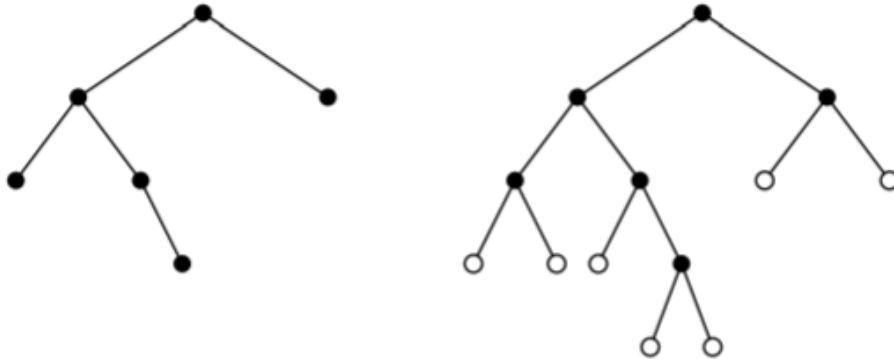
(2) Als T maar 1 deelboom heeft, het maakt niet uit of het de linker- of rechterdeelboom is, dan heeft die deelboom evenveel bladeren als T , en hoogte $h - 1$; dan geldt $t \leq 2^{h-1}$ wegens de inductieveronderstelling, en dus $t < 2^h$.

We kunnen ook schrijven: $h \geq \log_2(t)$, met $\log_2(t)$ de logaritme met basis 2, of binaire logaritme, van t .

Stelling 4.31 In een binaire boom met n knopen en hoogte h geldt: $n+1 \leq 2^{h+1}$.

Bewijs: Zij T een binaire boom. Maak een volledige binaire boom T' door aan elk blad van T 2 kinderen toe te voegen, en aan elke inwendige knoop van T die maar 1 kind heeft, een tweede kind (alle toegevoegde knopen zijn bladeren). Voor T' geldt $i' = n$ (elke knoop van T is een inwendige knoop van T'), dus $t' = i' + 1 = n+1$ (wegen stelling 4.29), en $h' = h+1$ (de constructie verhoogt de hoogte van de boom met precies 1). Door invullen in $t' \leq 2^{h'}$ (stelling 4.30) volgt dan direct $n+1 \leq 2^{h+1}$.

Figuur 4.22 toont een boom en zijn uitbreiding zoals gedefinieerd in stelling 4.31: de toegevoegde knopen zijn getekend als niet opgevulde cirkels. Merk ook op dat als de oorspronkelijk boom al volledig is, de uitbreiding toch verschilt!



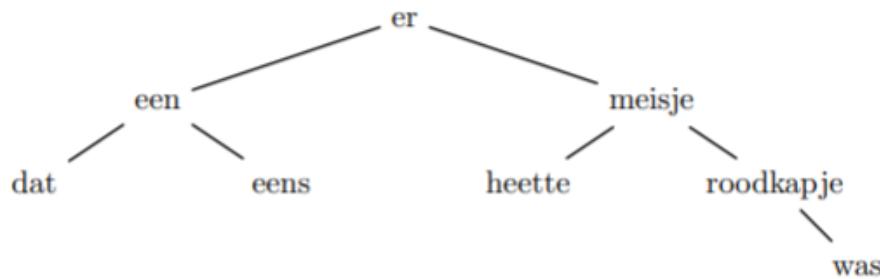
Figuur 4.22: Een binaire boom en zijn uitbreiding tot een volledige binaire boom

4.10.2 Zoekbomen

Een binaire zoekboom is een binaire boom waarin met elke knoop v een waarde $w(v)$ geassocieerd wordt, en die aan de volgende eigenschap voldoet: voor elke knoop v geldt dat alle waarden geassocieerd met een knoop in de linkse deelboom van v kleiner zijn dan $w(v)$, en alle waarden geassocieerd met een knoop in de rechtse deelboom groter dan $w(v)$.

Definitie 4.30 (Binaire zoekboom) Een binaire zoekboom is een binaire boom waarin met elke knoop v een waarde $w(v)$ is geassocieerd (bv. een getal) zodanig dat als l behoort tot de linker- en r tot de rechterdeelboom van v , dan $w(l) < w(v) < w(r)$

Een binaire zoekboom wordt ook soms gesorteerde binaire boom genoemd. Figuur 4.23 toont een binaire zoekboom waarin de waarde van de knopen woorden zijn; de orde is alfabetisch.



Figuur 4.23: Een binaire zoekboom voor de woorden uit de zin “Er was eens een meisje dat Roodkapje heette”

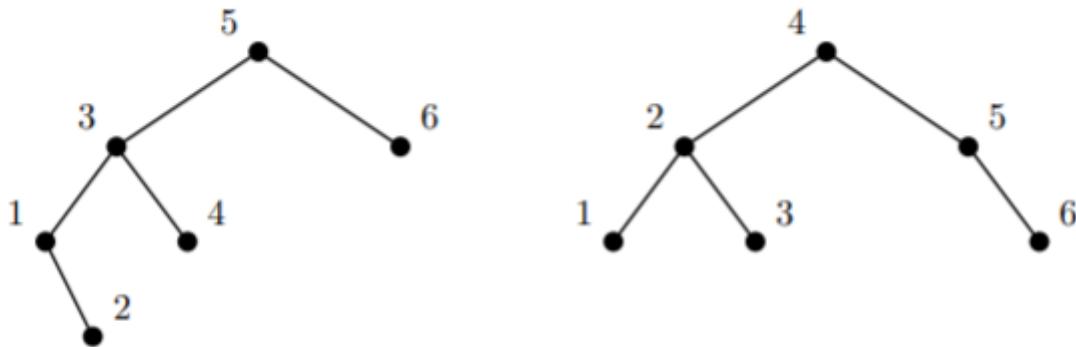
Het volgende algoritme zoekt of een bepaalde waarde aanwezig is in een binaire zoekboom: het algoritme is geschreven als een pseudo-Java methode die TRUE teruggeeft als de waarde gevonden is, anders FALSE;

```

boolean zoek(T boom, W waarde);
{
    P = wortel(T);
    doevoort = not(empty(T));
    zoek = FALSE;
    while (doevoort)
    {
        if (waarde(P) == W)
        {
            doevoort = FALSE;
            zoek = TRUE;
        }
        else
        if (waarde(P) < W)
            P = rechterkind(P);
        else P = linkerkind(P);
        if (empty(P))
            doevoort = FALSE
    }
}
  
```

We kunnen de complexiteit van dit algoritme uitdrukken in het aantal keer dat het lichaam van delus wordt uitgevoerd. In het slechtste geval zit de gezochte waarde niet in de boom en zoeken we langs het langste pad, vertrekend aan de wortel; dat pad heeft een lengte die gelijk is aan de hoogte h van de boom en lus wordt dus $h+1$ keer uitgevoerd. Uit Stelling 4.31 weten we dat $\text{blog}(n+1) \leq h+1$ en voor een vaste n kunnen we het slechtste geval dus niet beter maken dan $\text{blog}(n+1)$.

Een voorbeeld van twee binaire bomen met dezelfde knopen zie je in Figuur 4.24: de rechterboom is beter gebalanceerd dan de linkerboom en is dus minder hoog.



Figuur 4.24: Twee bomen met dezelfde knopen en verschillende hoogte

4.10.3 Praktische voorstelling van bomen

Voorstelling van gewortelde bomen in computerprogramma's

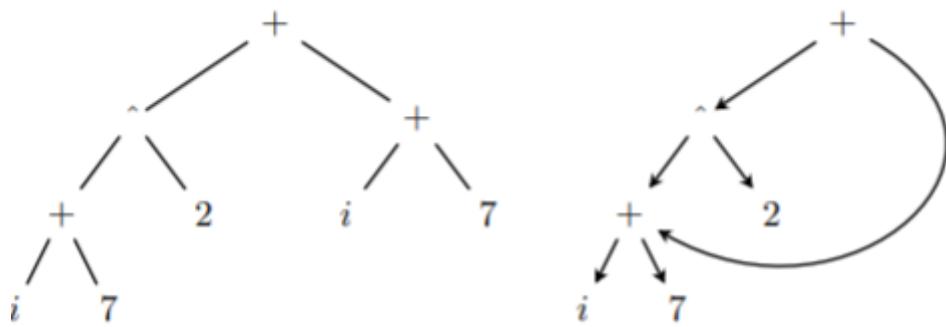
Bomen kunnen net als andere grafen als matrix voorgesteld worden, maar in computerprogramma's wordt vaak een andere structuur gebruikt, waarbij voor elke knoop een verwijzing naar de kinderen van die knoop opgeslagen wordt.

Bijvoorbeeld, voor een binaire boom die getallen als inhoud bevat, zouden we in Java de volgende code kunnen schrijven:

```
public class BinaireBoom {  
    private int inhoud;  
    private BinaireBoom linkerkind, rechterkind;  
  
    ...  
}
```

Een meer compacte voorstelling voor bomen

Dikwijls wordt een boom voorgesteld met gerichte bogen; die voorstelling benadrukt dat de functies linkerkind en rechterkind dikwijls explicet aanwezig zijn,, maar niet de functie ouder. Daarenboven kan het ook nuttig zijn een boom als een (gerichte) graaf voor te stellen die geen boom meer hoeft te zijn; neem bijvoorbeeld de boomvoorstelling van de rekenkundige uitdrukking $(i + 7)2 + i + 7$ in Figuur 4.25 links; daarin zie je twee deelbomen die precies dezelfde zijn: de deelboom voor de deeluitdrukking $i + 7$; de overeenkomstige gerichte graaf (die geen boom meer is) staat in Figuur 4.25 rechts. De voorstelling m.b.v. een gerichte graaf is meer compact omdat het voorkomen van de deeluitdrukking $i + 7$ door twee deelbomen gedeeld wordt (in het Engels: shared).



Figuur 4.25: Twee voorstellingen van $(i + 7)^2 + i + 7$

4.10.4 Het doorlopen van gewortelde bomen

Met het “doorlopen van een boom” bedoelen we het 1 en voor 1 behandelen van de knopen van die boom, waarbij uiteindelijk elke knoop van de boom precies 1 keer aan bod komt. In principe komt het doorlopen van een boom $G(V, E)$ gewoon overeen met het oplijsten van de elementen van V . In de praktijk gebruiken computerprogramma’s voor het opslaan van een gewortelde boom vaak een datastructuur zoals net besproken, die het niet zo evident maakt om alle knopen te overlopen. In dit deel gaan we ervan uit dat er voor het doorlopen van een gewortelde boom twee operaties beschikbaar zijn:

1. gegeven een boom, kunnen we de wortel van die boom opvragen;
2. gegeven een knoop in een boom, kunnen we de kinderen van die knoop opvragen.

Wanneer we beperkt zijn tot deze twee operaties, kunnen we bijvoorbeeld niet zomaar de ouder van een knoop opvragen.

Het volledig doorlopen van een gewortelde boom

Volgend algoritme geeft een methode voor het doorlopen van een gewortelde boom, waarbij ervan uit gegaan wordt dat enkel bovenstaande operaties toegelaten zijn.

Algoritme : volledig doorlopen van een gewortelde boom

```

 $S := \{w\}$ 
 $R := \emptyset$ 
zolang  $S \neq \emptyset$ :
    kies een willekeurige knoop  $a \in S$ 
     $S := S \setminus \{a\} \cup \{k | k \in \text{kinderen}(a)\}$ 
     $R := R \cup \{a\}$ 
    behandel  $a$ 

```

Stelling 4.32 Bovenstaand algoritme behandelt elke knoop in de boom precies 1 keer.

Bewijs: We bewijzen dit in zes stappen.

1. Een knoop wordt niet vaker behandeld dan hij toegevoegd wordt. Op het moment dat een knoop in S gekozen wordt om behandeld te worden, wordt die knoop uit S verwijderd. De knoop kan niet opnieuw gekozen worden voor behandeling zonder eerst opnieuw toegevoegd te worden. Hij moet dus minstens evenveel keer toegevoegd worden aan S als hij behandeld wordt.

2. Elke knoop wordt hoogstens 1 keer toegevoegd aan S . We bewijzen dit door inductie op de diepte van de knoop. Basis: de wortel is de enige knoop op diepte 0. Deze knoop wordt tijdens de initialisatie aan S toegevoegd. Binnen de while-lus worden enkel nog knopen aan S toegevoegd die een kind zijn van een ander knoop; de wortel is geen kind van een andere knoop en wordt hier dus nooit toegevoegd.

Besluit: alle knopen van diepte 0 worden hoogstens 1 keer toegevoegd aan S .

Inductiestap: als alle knopen op diepte i (met $i \geq 0$) hoogstens 1 keer toegevoegd worden, dan worden ook alle knopen op diepte $i + 1$ hoogstens 1 keer toegevoegd. Bewijs: neem een willekeurige knoop v op diepte $i + 1$. Die knoop heeft maar 1 ouder, en kan alleen toegevoegd worden op het moment dat die ouder behandeld wordt. De ouder heeft diepte i . Vanwege de inductieveronderstelling wordt de ouder hoogstens 1 keer toegevoegd, dus (vanwege punt 1) hoogstens 1 keer behandeld, en dus wordt v hoogstens 1 keer toegevoegd.

3. Elke knoop wordt hoogstens 1 keer behandeld. Dit volgt direct uit 1 en 2.
4. Elke knoop die in S zit, wordt ooit behandeld. Bewijs: Stel dat de boom n knopen heeft. Neem nu een willekeurige knoop v die op een bepaald moment in S zit, en stel dat S op dat moment m elementen bevat. Bij elke volgende uitvoering van de lus wordt een knoop uit S gehaald. Er kunnen intussen ook knopen toegevoegd worden, maar niet meer dan $n-m$, want elke knoop in de boom wordt hoogstens 1 keer toegevoegd. De lus kan dus hoogstens nog n keer uitgevoerd worden alvorens S leeg wordt. Op dat moment moet v behandeld zijn.

5. Elke knoop wordt precies 1 keer aan S toegevoegd. Bewijs: door inductie op de diepte van de knoop.

Basis: de wortel is de enige knoop op diepte 0. Deze knoop wordt minstens 1 keer toegevoegd, namelijk tijdens de initialisatie, en hoogstens 1 keer (vanwege punt 2), dus precies 1 keer.

Inductiestap: als alle knopen op diepte i (voor een bepaalde $i \geq 0$) precies 1 keer toegevoegd worden aan S , dan worden ook alle knopen op diepte $i + 1$ precies 1 keer toegevoegd aan S . Bewijs: beschouw een willekeurige knoop v op diepte $i + 1$. v heeft precies 1 ouder, en die heeft diepte i . Wegens de inductieveronderstelling wordt die ouder precies 1 keer toegevoegd, en dus (wegen punt 3 en 4) ook precies 1 keer behandeld. Op dat moment (en alleen dan) worden al zijn kinderen toegevoegd, dus ook v .

6. Elke knoop wordt precies 1 keer behandeld. Dit volgt nu direct uit 3, 4 en 5.

Merk op dat uit het bovenstaande ook volgt dat het algoritme altijd eindigt. Er zijn n knopen en elke knoop wordt precies 1 keer gekozen voor behandeling, dus de lus wordt precies n keer uitgevoerd. Een enkele uitvoering van de lus eindigt steeds.

Diepte-eerst en breedte-eerst doorlopen

Het bovenstaande algoritme kiest steeds willekeurig een knoop in S om te behandelen. Volgende algoritme doet hetzelfde als het voorgaande, maar houdt voor elke knoop in S de diepte van de knoop bij, en kiest systematisch steeds een knoop op maximale diepte. -> diepte eerst doorlopen

Algoritme: diepte-eerst doorlopen van een boom

```

 $S := \{(w, 0)\}$ 
 $R := \emptyset$ 
zolang  $S \neq \emptyset$ :
    kies een willekeurige  $(a, n)$  uit  $S$  met maximale  $n$ 
     $S := S \setminus \{(a, n)\} \cup \{(v, n+1) | v \in \text{kinderen}(a)\}$ 
     $R := R \cup \{a\}$ 
    behandel  $a$ 

```

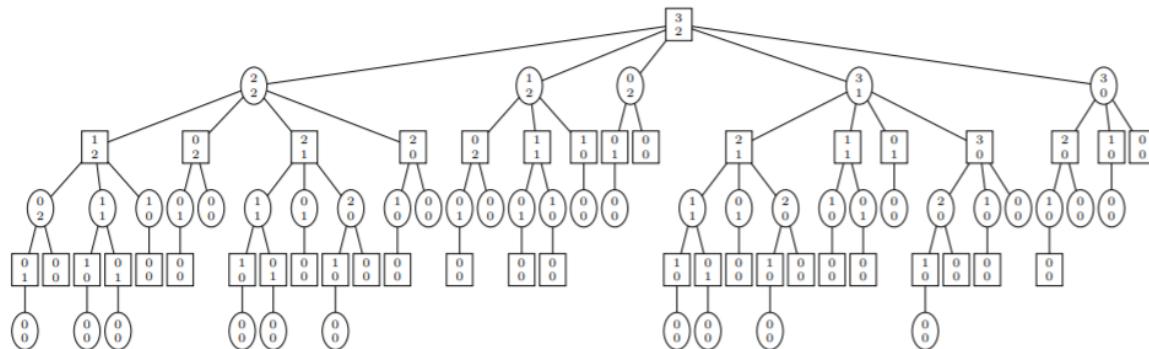
Als we steeds een knoop op minimale diepte kiezen, in plaats van maximale, verkrijgen we de breedte-eerst variant (Eng. breadth-first).

Oefening op p135 in de cursus is lezen

4.10.5 Spelbomen

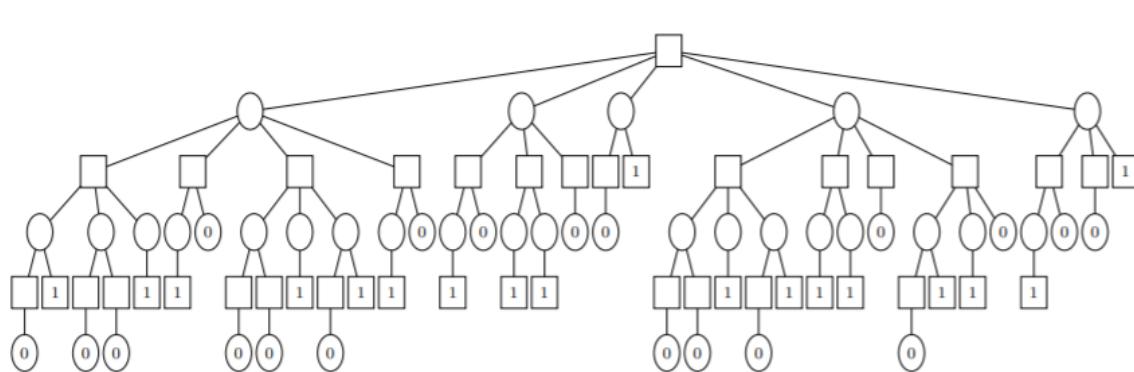
Minimaxprocedure

Beschouwen we nim: een spel waarbij er n stapels munten zijn, en twee spelers die om beurten van 1 willekeurige stapel 1 of meer munten wegnemen. Degene die de laatste munt wegneemt verliest het spel. We analyseren nim voor twee stapels, waarbij de eerste stapel 3 munten en de tweede stapel 2 munten telt bij het begin van het spel. De spelers heten Doos en Bol en Doos begint. 5 mogelijke zetten en zou graag weten wat de gevolgen zijn van elk van die mogelijkheden. Daartoe tekent hij een boom met als wortel een doosje met daarin de beginsituatie van het spel, namelijk $\frac{3}{2}$. Als kinderen van die wortel tekent hij die vijf mogelijkheden, maar die tekent hij in een cirkel, omdat het dan de beurt zal zijn aan Bol. En bij elk van de vijf mogelijkheden voor zijn eerste zet, tekent Doos de mogelijke zetten van Bol enzovoort: vermits er bij elke zet minstens een munt verdwijnt, is deze boom eindig; je ziet de volledige boom die Doos tekende in Figuur 4.32.



Figuur 4.32: De spelboom voor nim(3,2)

Elke eindknoop heeft $\frac{0}{0}$ en een eindknoop in de vorm van een doosje, toont een overwinning van Doos: Doos wil natuurlijk het spel zo laten verlopen dat hij bij een doosje aankomt op het einde van het spel. En Bol zal hem dat proberen te verhinderen. Die volledige spelboom helpt niet veel voor Doos om zijn keuze te maken bij de eerste zet. Daarom gaat hij alle eindknopen markeren met 0 of 1: 0 als hij verliest als het spel in die toestand komt, 1 als hij wint. Je krijgt de boom van Figuur 4.33.



Figuur 4.33: De eindknopen hebben een label

Het is duidelijk dat Doos in een eindknoop wil geraken waar een 1 in staat, want dan wint hij. Maar er is nog geen aanduiding in de wortel van de spelboom over zijn eerste zet. Stel dat een spelboom maar een niveau diep is, zoals in tekening 4.31.



Figuur 4.31: Propagatie van kinderen naar ouders

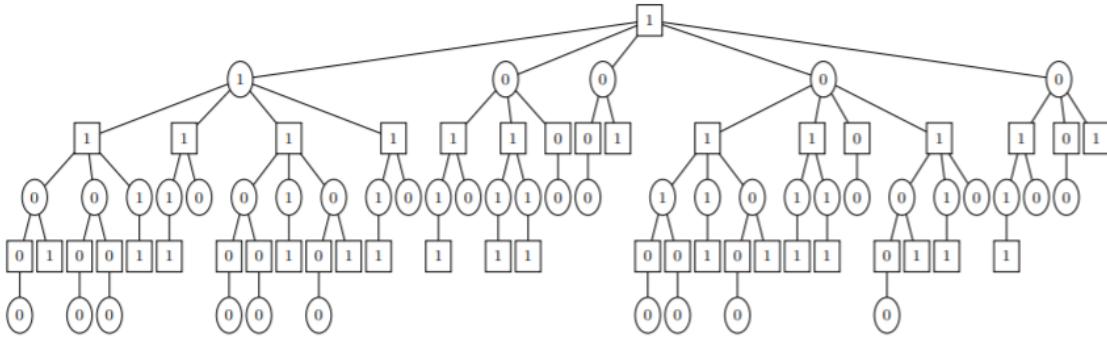
Als van de doos-knoop alle kinderen een 0 hebben, dan zal Doos verliezen, gelijk wat hij doet, en daarom kan hij de wortel van de boom als label 0 geven: een hij in die situatie verzeild is, verliest hij zeker. Daarentegen, als er minstens 1 kind is met label gelijk aan 1, kan Doos bij zijn zet daar naartoe gaan: daarom kan hij de doos-knoop het label 1 geven. Een doos-knoop krijgt dus het maximum van de labels van zijn kinderen. In Figuur 4.31 links krijgt de wortel dus label 1.

Langs de andere kant, als een bol-knoop alleen maar kinderen met een 1 heeft, dan is de situatie een win voor Doos, want gelijk wat Bol kiest, hij verliest. Maar als er minstens 1 kind met een label gelijk aan 0 is, kan Bol die zet kiezen en is hij dus in een situatie om Doos te doen verliezen, bijgevolg kan de bol-knoop het label 0 krijgen. Een bol-knoop krijgt dus het minimum van de labels van zijn kinderen. In Figuur 4.31 rechts krijgt de wortel dus label 0.

Deze procedure van labeling berekent een label voor elke knoop waarvan de labels van zijn kinderen al bekend zijn. We passen ze toe op Figuur 4.33 en verkrijgen Figuur 4.34. Zoals aangegeven, labelt deze procedure in een pad van de bladeren naar de wortel afwisselend met het minimum en maximum van de labels van de kinderen: daarom wordt dit de minimaxprocedure genoemd.

Figuur 4.34 laat het resultaat zien van de minimaxprocedure uitgevoerd op 4.33.

Figuur 4.34: Alle knopen hebben een label



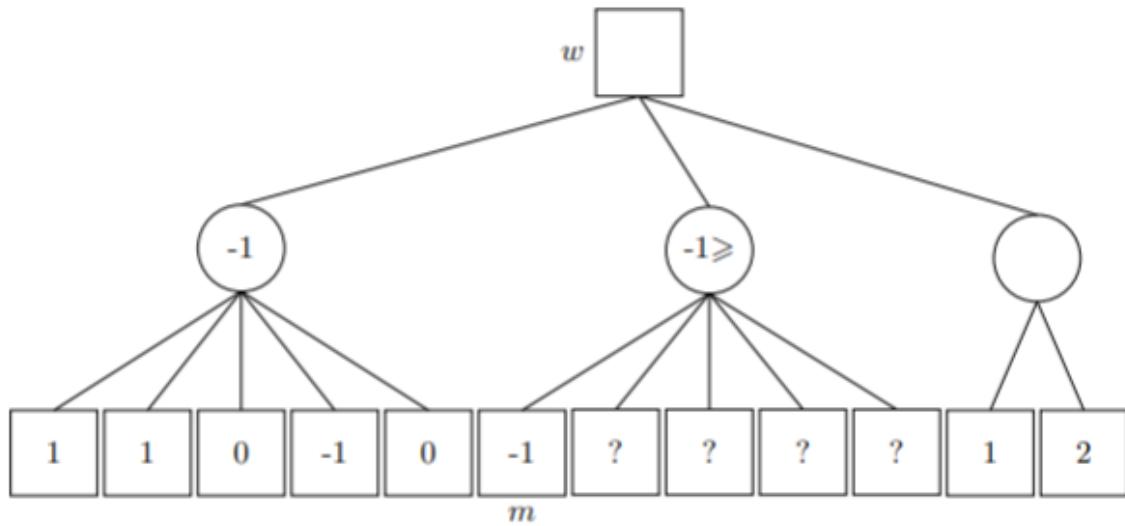
Evaluatiefunctie

In principe kan je van veel andere spelen ook de hele spelboom opstellen, maar in parktijk is dat niet te doen omdat de spelboom te diep is en de vertakkingsgraad te groot. Bv schaak, 20 openingszetten waarop telkens 20 mogelijke antwoorden zijn en het aantal mogelijke zetten stijgt nadien nog. Dit eindigt uiteindelijk in een enorm groot getal. We kunnen de boom dus maar analyseren tot een bepaalde diepte d . Dit betekent dat we de bladeren niet kennen. Daarom gaan we een evaluatiefunctie E gebruiken. E kent aan elke knoop diepte d een getal toe dat groter is naarmate de positie beter is voor Doos. Deze functie is niet onfeilbaar maar komt dichtbij.

α -snede en β -snede

Soms kan vermeden worden om sommige spelposities - knopen in de spelboom - te evalueren. In figuur 4.37 zie je dat als berekend is dat m waard -1 heeft, de evaluatie van de broers en zusters van m niet meer moet gebeuren: immers, het label van de ouder van m zal hoogstens -1 zijn (want we zullen het minimum van de labels van de kinderen nemen) en er is op dat niveau al een label -1 , dus gelijk wat de vraagtekens opleveren, het label van de wortel w – die het maximum wordt van de labels van zijn kinderen – zal er niet door beïnvloed worden. We noemen dit een α -snede (cutoff).

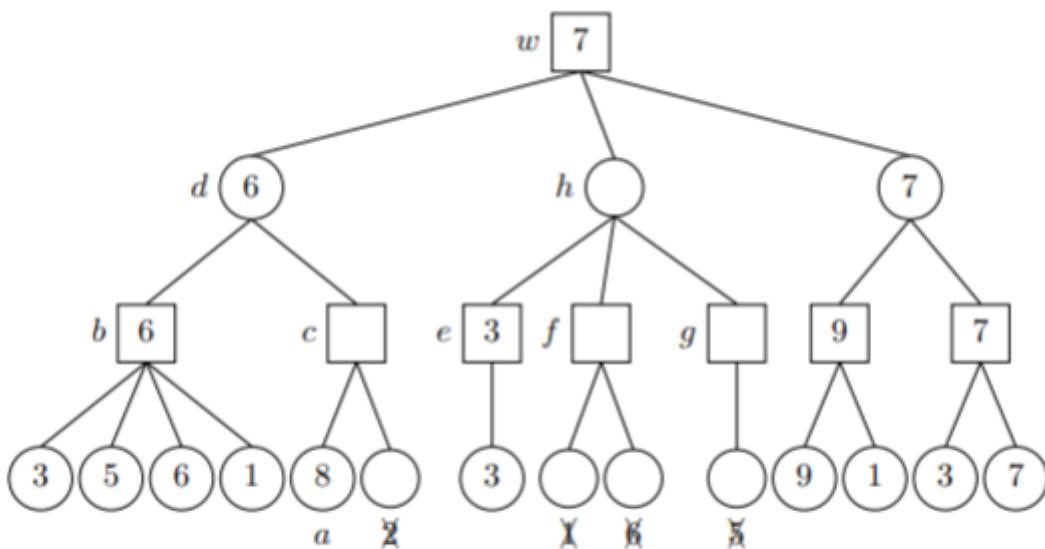
Analoog aan de α -snede is er ook een β -snede: een β -snede komt voor bij een boknoot w als een kleinkind m een waarde heeft die groter of gelijk is aan de β -waarde van w ; de β -waarde van w is de tot dan toe kleinste waarde van een kind van w . In die situatie mag de deelboom met wortel de ouder van m verwijderd worden uit de spelboom.



Figuur 4.37: α snede in de labeling

Als bijkomend voorbeeld van het α - β -algoritme, beschouw de figuur 4.38: de waarde van de bladeren staat eronder, maar is nog niet berekend.

De spelboom wordt weer diepte-eerst doorlopen - en hier ook van links naar rechts. Vlak voor de evaluatie van knoop a gebeurt (zie Figuur 4.39) is de β -waarde van d = 6. Vermits $E(a) = 8$ hebben we een β -snede die mogelijk maakt om de evaluatie van de andere kinderen van c over te slaan. De definitieve waarde van d is dan bekend en gelijk aan 6. Daardoor wordt de α -waarde van w = 6. Later is de waarde van e al bekend: 3 en $3 < \alpha$ -waarde van w bijgevolg hebben we een α -snede voor w, t.t.z. de deelboom die begint bij h (de ouder van e die de snede mogelijk maakte) hoeft niet geëvalueerd te worden. Figuur 4.39 toont de niet-berekende waarden van eindknopen en lege cirkels/dozen voor knopen die weggesneden werden



Figuur 4.39: Aanduiding van het effect van de $\alpha - \beta$ -sneltes

4.11 Netwerkmodellen en Petri-netten

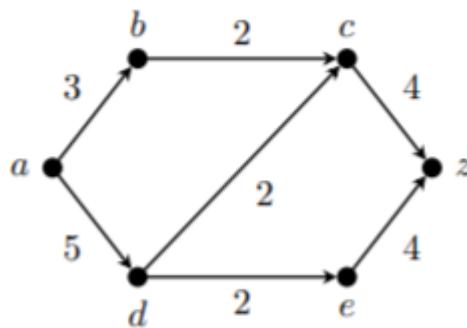
Een netwerk van verbindingen, elk met hun eigen capaciteit, kan gemodelleerd worden als een gerichte, gewogen graaf. Als voorbeeld kan je denken aan een wegennetwerk, een elektrisch netwerk of aan een stel oliepijplijnen. Het belangrijkste probleem i.v.m dit soort netwerken is het optimaliseren van een stroming, zonder capaciteitoverschreiding. We zullen dit optimalisatieprobleem oplossing in de context van grafentheorie. Ook andere problemen die op het eerste zicht niets met optimalisatie van stroming te maken hebben, kunnen gemodelleerd worden als een netwerkprobleem: personeelstoewijzing, toekenning van resources en ook het partnerkeuzeprobleem ("The marriage problem"). Petrinetten modelleren gelijktijdige acties en vormen een kader om bijvoorbeeld "deadlock" problemen te bekijken.

4.11.1 Transportnetwerk

Definitie 4.31 (Transportnetwerk) Een transportnetwerk (of simpelweg een netwerk) is een enkelvoudige, gewogen, gerichte graaf G die voldoet aan:

1. Er is juist 1 knoop in G zonder binnengrenende bogen; deze knoop wordt de **bron** genoemd (Eng.: source)
2. Er is juist 1 knoop in G zonder buitenstaande bogen; deze knoop wordt de **put** genoemd (Eng.: sink)
3. Het gewicht van $C_{i,j}$ van de (gerichte) boog (i,j) is positief en wordt de **capaciteit** van de boog genoemd
4. G is samenhangend

Figuur 4.40 toont een netwerk: de bron is de knoop a en de put is de knoop z ; de capaciteit van elke boog is bij de boog geschreven. Het netwerk modelleert bijvoorbeeld een stel eenrichtingsstraten in een stad tussen het station (knoop a) en de markt (knoop z); de capaciteit is het aantal voertuigen dat per minuut kan passeren door elke straat.



Figuur 4.40: Een transportnetwerk

Definitie 4.32 (Stroming) Voor een netwerk $G(V,E)$ met capaciteiten $C_{i,j}$, $i,j \in V$ is F een stroming als F een afbeelding is van E naar \mathbb{R}^+ zodanig dat

1. $F(i,j) \leq C_{i,j}$

2. voor elke knoop j die niet de bron of de put is geldt:

$$\sum_{i \in V} F(i, j) = \sum_{i \in V} F(j, i)$$

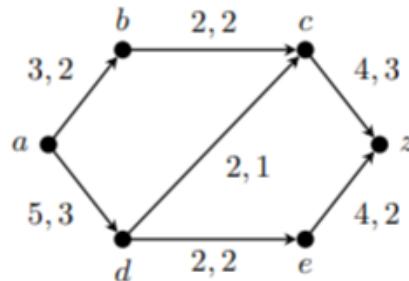
We noemen $F(i,j)$ de stroming in boog (i,j) . Voor een knoop j noemen we $\sum_{i \in V} F(i, j)$ de stroming naar of in j en $\sum_{i \in V} F(j, i)$ de stroming uit j .

Formule 2 in definitie 4.32 drukt het behoud van stroming uit: alles wat binnenkomt in een knoop, gaat er weer buiten en alles wat buiten komt, is binnengekomen; dat verhindert dat er een ophoping of productie gebeurt in de knopen.

Figuur 4.41 toont een stroming voor het netwerk van Figuur 4.40; de stroming is gedefinieerd door:

$$\begin{aligned} F(a, b) &= 2 \\ F(b, c) &= 2 \\ F(c, z) &= 3 \\ F(a, d) &= 3 \\ F(d, c) &= 1 \\ F(d, e) &= 2 \\ F(e, z) &= 2 \end{aligned}$$

en telkens naast de capaciteit van de overeenkomstige boog gezet.



Figuur 4.41: Een stroming in een transportnetwerk

Je kan nagaan dat formule 2 in definitie 4.32 voldaan is voor elke knoop behalve de bron en de put.

Stelling 4.33 Van een stroming F in een netwerk $G(V, E)$ is de stroming uit de bron gelijk aan de stroming in de put, of meer formeel:

$$\sum_{i \in V} F(a, i) = \sum_{i \in V} F(i, z)$$

Bewijs: Het is duidelijk dat

$$\sum_{j \in V} (\sum_{i \in V} F(i, j)) = \sum_{i \in V} (\sum_{j \in V} F(i, j))$$

De omwisseling van de \sum 's mag omdat de grafen eindig zijn. De tweede gelijkheid geldt wegens hernoeming van i en j .

Daaruit volgt:

$$\begin{aligned}
0 &= \sum_{j \in V} \left(\sum_{i \in V} F(i, j) - \sum_{i \in V} F(j, i) \right) \\
&= \left(\sum_{i \in V} F(i, z) - \sum_{i \in V} F(z, i) \right) + \left(\sum_{i \in V} F(i, a) - \sum_{i \in V} F(a, i) \right) \\
&\quad + \sum_{j \in V \setminus \{a, z\}} \left(\sum_{i \in V} F(i, j) - \sum_{i \in V} F(j, i) \right) \\
&= \sum_{i \in V} F(i, z) - \sum_{i \in V} F(a, i)
\end{aligned}$$

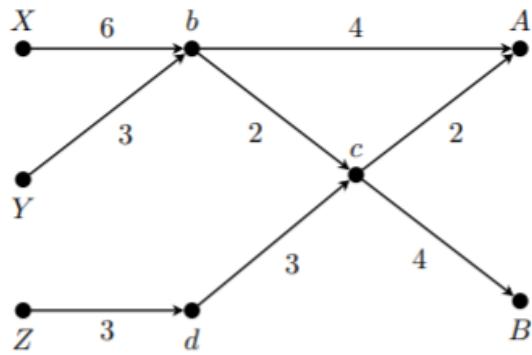
vermits $F(z, i) = 0 = F(i, a)$ voor $\forall i \in V$ (door definitie 4.32) en $\sum_{i \in V} F(i, j) = \sum_{i \in V} F(j, i)$ voor $\forall j \in (V \setminus \{a, z\})$ (door formule 2 in definitie 4.32)

Op basis van stelling 4.33, kunnen we nu definiëren:

Definitie 4.33 (Grootte van een stroming) De grootte van een stroming F in een netwerk $G(V, E)$ met bron a en put z is gedefineerd door $\sum_{i \in V} F(a, i)$ of $\sum_{i \in V} F(i, z)$. We noteren de grootte van F als $|F|$

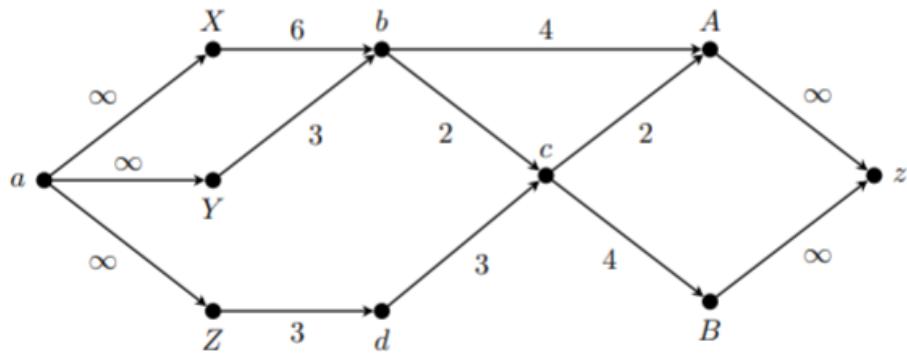
Het netwerkprobleem kan nu als volgt geformuleerd worden: voor een gegeven netwerk G , vind de maximale stroming, of m.a.w. vind de stroming met de maximale grootte.

Tot slot van deze sectie, nog iets over netwerken met meer dan 1 bron of put: het netwerk in figuur 4.42 stelt de waterbevoorrading van de steden A en B voor, vanuit de bronnen X, Y en Z en over verdeelstations b, c en d.

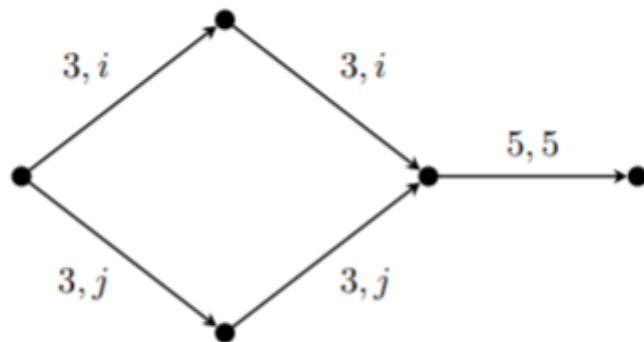


Figuur 4.42: Een transportnetwerk met meerdere bronnen en putten

Door een superbron a en een superput z toe te voegen aan dit netwerk, verkrijgen we terug een gewoon netwerk: vanuit a voeg je ook een gerichte boog toe naar elke bron van het originele netwerk, en vanuit elke oude put een gerichte boog naar z ; de toegevoegde bogen krijgen alle de capaciteit ∞ .



Figuur 4.43: Een transportnetwerk met een superbron en superput



Figuur 4.44: Een netwerk met oneindig veel maximale stromen

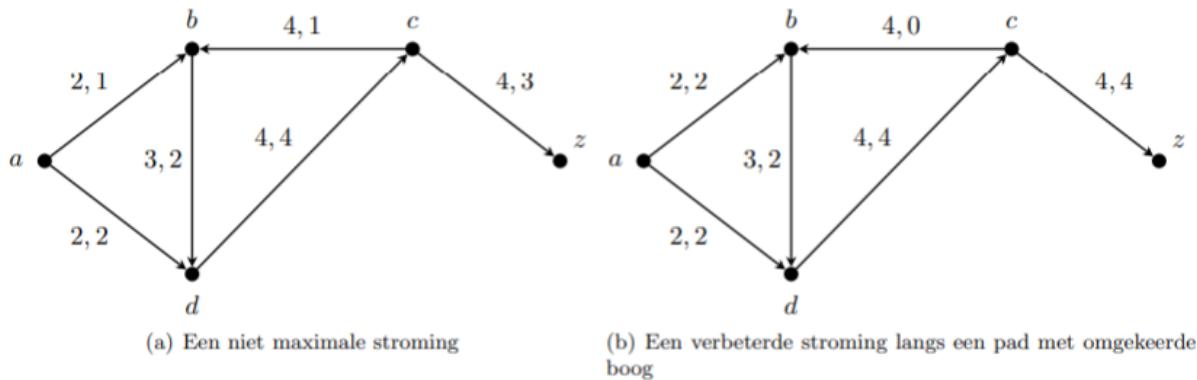
4.11.2 Maximale stroming

We zullen een algoritme zien om een maximale stroming te berekenen. Het basisidee is eenvoudig: vertrek van een stroming en verbeter die totdat dat niet meer mogelijk is; je hebt nu een maximale stroming. Een intuïtieve beschrijving van hoe je een stroming verbetert, gaat als volgt:

- Neem een pad P van de bron a naar de put z
- zoek het minimum Δ van $C_b - F(b)$ over alle bogen $b \in P$
- bepaal de nieuwe stroming langs het pad P door bij elke $F(b) \Delta$ bij te tellen.

Een paar opmerkingen bij dit voorschrift

- De operatie verhoogt de stroming enkel als het pad P (toevallig) zo is dat $\Delta > 0$
- De beschrijving geldt alleen voor een pad waarvan elke boog de goede richting heeft, maar we mogen niet alleen paden van a naar z zoeken langs de gerichte bogen: bekijk Figuur 4.45(a); daarin zie je dat geen enkel pad van a naar z met enkel goede bogen "verbeterd" kan worden met bovenstaande methode; maar de stroming langs het pad (a, b, c, z) kan wel verbeterd worden door de stroming getoond in Figuur 4.45(b); we moeten dus ook paden beschouwen waarvan bogen "omgekeerd" lopen en we moeten voor die verkeerde bogen niet iets optellen bij de stroming, maar er iets aftrekken: als een "omgekeerde" boog een stroming draagt kan het immers zijn dat er alleen maar iets rondstroomt zonder ooit de put te bereiken; we formaliseren dat als volgt:



Figuur 4.45: Verbetering van stroming

Definitie 4.34 (Goede / slechte boog) In een gerichte graaf $G(V, E)$ met pad (v_1, v_2, \dots, v_n) noemen we de boog (v_i, v_{i+1}) goed (gericht) indien $(v_i, v_{i+1}) \in E$ en anders slecht (gericht).

In een pad P zullen we de goede bogen noteren door P_+ en de slechte door P_- .

Stelling 4.34 (Verbeteren van een stroming) Zij P een pad van a naar z in een netwerk $G(V, E)$, waarbij

1. $\forall (i, j) \in P_+ : F(i, j) < C_{i,j}$
2. $\forall (i, j) \in P_- : 0 < F(i, j)$

Laat bovendien $\Delta = \min(\min_{(i,j) \in P_+} \{C_{i,j} - F(i,j)\}, \min_{(i,j) \in P_-} \{F(i,j)\})$; definieer de functie F' als volgt:

$$\begin{aligned} F'(i,j) &= F(i,j) \quad \forall (i,j) \notin P \\ &= F(i,j) + \Delta \quad \forall (i,j) \in P_+ \\ &= F(i,j) - \Delta \quad \forall (i,j) \in P_- \end{aligned}$$

Dan is F' een stroming waarvan de grootte Δ meer is dan die van F .

Algoritme: Constructie van een maximale stroming

Algoritme: Constructie van een maximale stroming

Laat $G(V, E)$ een netwerk zijn met bron a en put z en capaciteit C , waarbij alle capaciteiten positief en geheel zijn. Onderstel een orde op de knopen met $a = v_0, v_1, \dots, v_n = z$. Met \mathcal{B} zullen we de verzameling beschouwde knopen aanduiden; met \mathcal{L} de verzameling van knopen met een label; de operaties op \mathcal{B} zullen we explicet maken; die op \mathcal{L} zijn impliciet.

1. **Initialisatie:** Definieer $\forall (i,j) \in E : F(i,j) = 0$
2. **Label de bron:** Geef a het label $(-, \infty)$; zet $\mathcal{B} = \emptyset$
3. **Aangekomen?:** Indien z een label heeft, verbeter de stroming en ga terug naar **Label de bron**.
Verbeter de stroming gaat als volgt: er is juist één pad P van a naar z dat achteruit kan geconstrueerd worden te vertrekken van z , dan de eerste component van het label van z en zo verder tot in a . De tweede component van het label van z is een grootheid Δ die nu wordt bijgeteld bij F voor goede bogen in P en afgetrokken van F voor slechte bogen van P . Daarna worden alle labels in heel het netwerk gewist.
4. **Kies volgende knoop:** Indien $\mathcal{L} \setminus \mathcal{B} = \emptyset$, **stop**: de stroming F is maximaal.
Noem v de nog niet beschouwde knoop v_i met een label en met kleinste index i : voeg v toe aan \mathcal{B} .
5. **Label buren:** Stel dat het label van v gelijk is aan (α, Δ) . Behandel nu elke boog van de vorm (v, w) of (w, v) in de volgorde $(v, v_0), (v_0, v), (v, v_1), (v_1, v), \dots$ waarbij w nog geen label heeft.
 - voor elke boog (v, w) (dus een boog die wegloopt uit v):
indien $F(v, w) < C_{v,w}$ geef w het label $(v, \min\{\Delta, C_{v,w} - F(v, w)\})$ anders geef je w geen label
 - voor elke boog (w, v) (dus een boog die toekomt in v):
indien $F(w, v) > 0$ geef w het label $(v, \min\{\Delta, F(w, v)\})$ anders geef je w geen label

Ga naar **Aangekomen?**

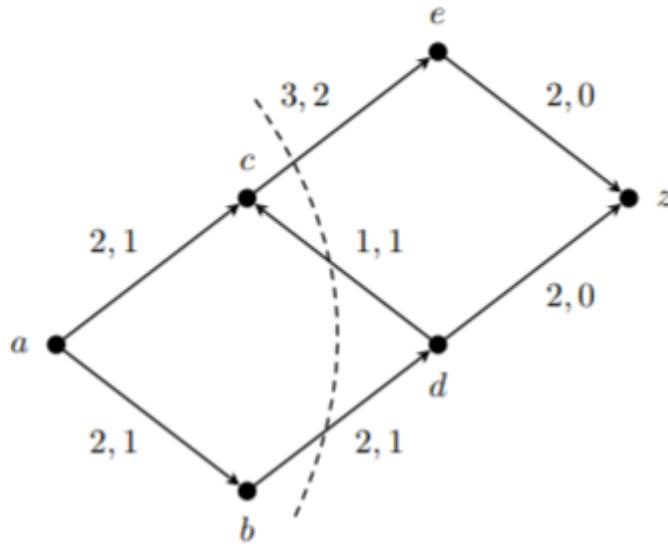
Stelling 4.35 Het voorgaande algoritme eindigt steeds

Bewijs: Punt 3 in het algoritme is cruciaal: zolang de uitgang stop niet genomen wordt, wordt punt 3 herhaaldelijk uitgevoerd. Vanuit punt 3, gaat de uitvoering ofwel naar punt 2 of punt 4. De overgang van punt 3 naar punt 2 kan maar een eindig aantal keer gebeuren, want die overgang impliceert dat de stroming verbeterd wordt met $\Delta > 0$; Δ is geheel omdat alle capaciteiten geheel zijn en vermits de

maximale stroming begrensd is (door de som van de capaciteiten van de bogen die vertrekken in de bron) kan de gegeven stroming dus slechts een eindig aantal keer verbeterd worden. Na een eindig aantal overgangen van punt 3 naar punt 2, wordt dus steeds opnieuw de overgang van punt 3 naar punt 4 genomen. In punt 4 wordt telkens een knoop toegevoegd aan B (die nooit meer terug op \emptyset wordt gezet). Bijgevolg zal na verloop van tijd $L = B$ en op dat moment stoppen het algoritme.

Definitie 4.35 (Snede) Een snede van een netwerk $G(V, E)$ met bron a en put z , is een 2-tal (P, \bar{P}) zodanig dat $a \in P, z \in \bar{P}, P \cup \bar{P} = V$ en $P \cap \bar{P} = \emptyset$

Zie hier een netwerk in twee verdeelt door een snede



Figuur 4.46: een netwerk met een stroming en een snede

We kunnen checken hoeveel stroming er globaal loopt over de snede door de pijlen van de bogen van links naar rechts die over de snede lopen op te tellen en de pijlen van rechts naar links af te trekken.

$$F(c,e) + F(b,d) - F(d,c) = 2+1-1 = 2$$

Als we dit vergelijken met de stroming in a of z dan zien we dat die ook gelijk is aan 2. Is dit toeval? find out in the next episode of dragon ball z.

Als we proberen een schatting te maken van hoeveel stroming er maximaal van links naar rechts over de snede kan lopen, door optimistisch te veronderstellen dat er misschien een stroming bestaat die voor alle goede bogen over de snede maximaal is, dus gelijk aan de capaciteit van die boog en voor alle slechte bogen nul. Dit is de capaciteit.

Bijvoorbeeld:

$$C(c,e)+C(b,c) = 3 + 2 = 5$$

Definitie 4.36 (Capaciteit van een snede) De capaciteit van een snede (P, \bar{P}) is

$$C(P, \bar{P}) = \sum_{i \in P} \sum_{j \in \bar{P}} C_{i,j}$$

Stelling 4.36 De capaciteit van een snede is niet kleiner dan een stroming, m.a.w.

$$\sum_{i \in P} \sum_{j \in \bar{P}} C_{i,j} \geq \sum_{i \in V} F(a, i)$$

Bewijs:

$$\begin{aligned} \sum_{i \in V} F(a, i) &= \sum_{i \in V} (F(a, i) - F(i, a)) + \sum_{j \in P \setminus \{a\}} \sum_{i \in V} (F(j, i) - F(i, j)) \\ &= \sum_{j \in P} \sum_{i \in V} (F(j, i) - F(i, j)) \\ &= \sum_{j \in P} \sum_{i \in P} F(j, i) + \sum_{j \in P} \sum_{i \in \bar{P}} F(j, i) - \sum_{j \in P} \sum_{i \in P} F(i, j) - \sum_{j \in P} \sum_{i \in \bar{P}} F(i, j) \\ &= \sum_{j \in P} \sum_{i \in \bar{P}} F(j, i) - \sum_{j \in P} \sum_{i \in \bar{P}} F(i, j) \\ &\leq \sum_{j \in P} \sum_{i \in \bar{P}} F(j, i) \\ &\leq C(P, \bar{P}) \end{aligned}$$

Stelling 4.37 (Max flow, min cut) Voor een snede (P, \bar{P}) en stroming F in een net $G(V, E)$ geldt dat als $C(P, \bar{P}) = |F|$ (of dus als $\sum_{i \in P} \sum_{j \in \bar{P}} C_{i,j} = \sum_{i \in V} F(a, i)$)

dan is de stroming maximaal en de snede minimaal.

Bovendien is die gelijkheid equivalent met

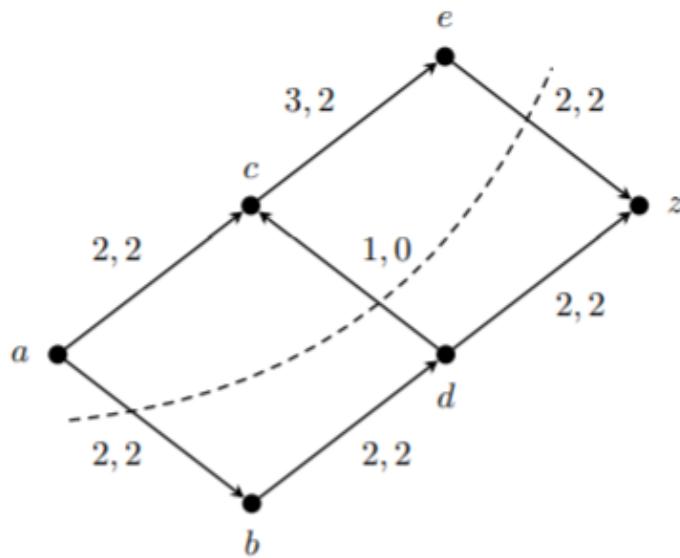
1. $\forall i \in P, j \in \bar{P} : F(i, j) = C_{i,j}$ en
2. $\forall i \in \bar{P}, j \in P : F(i, j) = 0$

d.w.z. goede bogen over de snede hebben een stroming gelijk aan hun capaciteit en slechte bogen een nul stroming.

Bewijs: Dit volgt onmiddelijk uit de ongelijkheden van stelling 4.36 (I guess)

Merk op dat we niet bewezen dat er voor elke maximale stroom een minimale snede is met die gelijkheid noch omgekeerd.

Figuur 4.47 toont een maximale stroom met minimale snede.



Figuur 4.47: Een netwerk met een maximale stroming en een minimale snede

Stelling 4.38 Wanneer het algoritme voor constructie van een maximale stroming eindigt, geldt dat F maximaal is.

Bewijs: zie boek op p157

4.11.3 Matching

Als je dit snel wilt begrijpen raad ik aan om deze video te bekijken: <https://www.youtube.com/watch?v=chdr2aj4FUc>

Definitie 4.37 (Matching) Voor een gerichte, tweeledige graaf $G(V \cup W, E)$ waarbij $V \cap W = \emptyset$ en $E \subseteq V \times W$, is M een overeenkomst of matching indien

- $M \subseteq E$ en
- $\forall (x, y), (i, j) \in M : \text{indien } (i, j) \neq (x, y) \text{ dan is } i \neq x \text{ en } j \neq y$ (t.t.z. in elke knoop komt hoogstens 1 boog aan en er vertrekt er hoogstens 1)

Een maximale matching heeft een maximaal aantal bogen in M . Een matching is volledig indien $\forall v \in V, \exists w \in W : (v, w) \in M$.

Een gerichte, tweeledige graaf $G(V \cup W, E)$ waaraan een bron en put wordt toegevoegd, bogen van de bron naar de knopen in V en bogen van de knopen in W naar de put, en waar aan elke boog de capaciteit 'e' en wordt toegekend, noemen we het matching netwerk dat van G is afgeleid.

De volgende stelling formaliseert de overeenkomst tussen een maximale stroming in een matching netwerk en een maximale matching. Een gehele stroming F is zodanig dat F enkel gehele groottes heeft.

Stelling 4.39 Voor een gerichte, tweeledige graaf $G(V \cup W, E)$ waarbij $V \cap W = \emptyset$ en $E \subseteq V \times W$ geldt dat

- Een gehele stroming F in het overeenkomstige matching netwerk, geeft een matching in $G : v \in V$ komt overeen met $w \in W$ als en slechts als $F(v, w) = 1$
- Een maximale gehele stroming komt overeen met een maximale matching.
- Een gehele stroming met grootte $\#V$ komt overeen met een volledige matching.

Bewijs van de stelling mag je zelf uitwerken (Jij denkt lol)

Er is een manier om snel te kunnen zien of er volledige matching is in een graaf of niet. Als je n knopen hebt maar samen hebben deze knopen minder dan n verschillende bogen dan gaat die niet volledig matching zijn.

Veralgemeend dus:

- Definieer de afbeelding ($P(S)$ stelt de machtsverzameling van S voor)

$$R : P(V) \rightarrow P(W)$$

$$S \rightarrow \{w \in W | \exists v \in S \text{ met } (v, w) \in E\}$$

- Indien G een volledige matching heeft, dan moet $\#S \leq \#R(S), \forall S \subseteq V$

Stelling 4.40 (De trouwstelling van Hall) Voor een gerichte, tweeledige graaf $G(V \cup W, E)$ waarbij $V \cap W = \emptyset$ en $E \subseteq V \times W$, heeft een volledige matching als en slechts als $\#S \leq \#R(S), \forall S \subseteq V$

Bewijs zie boek

In kort:

Maximale matching

Een matching P , van graaf G , is maximaal als er geen edges meer kunnen worden toegevoegd aan P omdat elke node gematched is aan een andere node.

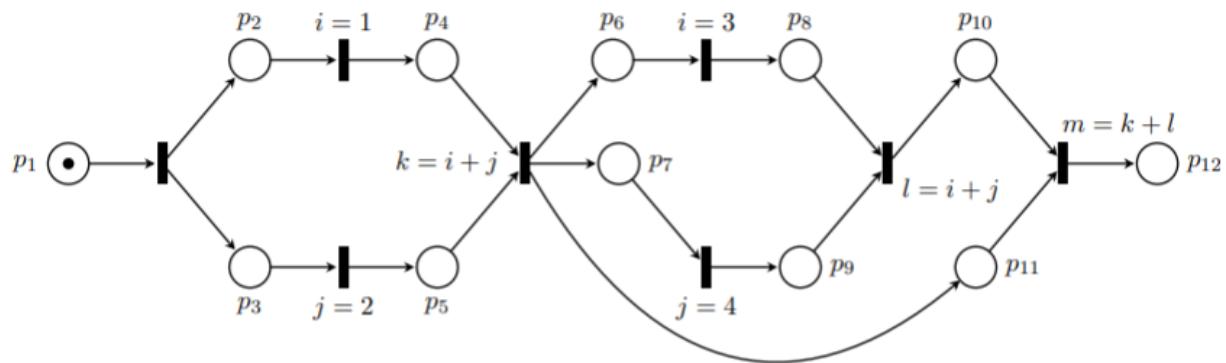
Maximum matching

Een matching is een maximum matching als het de grootste hoeveelheid mogelijk edges bezit. Simpel gezegd, een maximum matching is een maximale matching met de maximum hoeveelheid edges. Elke maximum matching is maximaal, maar niet elke maximale matching is een maximum matching.

4.11.4 Petrinetten

Een voorbeeld van een Petrinet zie je in Figuur 4.56; het is de voorstelling van het stukje programma

```
i = 1;  
j = 2;  
k = i+j;  
i = 3;  
j = 4;  
l = i+j;  
m = k+l;
```



Figuur 4.56: Petrinetvoorstelling van een programma

Een petrinet is een gerichte tweeledige graaf met twee soorten knopen, een cirkel dat we plaatsen noemen en een transitie dat we staafjes noemen. De plaatsen komen overeen met programmapunten tussen instructies(states); de transities komen overeen met de instructies in het programma. De gerichte bogen lopen steeds van een plaats naar een transitie en van een transitie naar een plaats. De plaatsen zijn leeg, behalve p_1 , waarin een zwart bolletje is getekend: een token. Een plaats met een token symboliseert het feit dat een preconditie voor het mogen uitvoeren van een instructie verbonden met de plaats is voldaan. Eenzelfde preconditie kan gelden voor meer dan 1 instructie, doch 1 token geeft aanleiding tot het uitvoeren van maar 1 instructie. Als een transitie is gebeurd - de instructie is uitgevoerd - dan verschijnt er een token op elke plaats verbonden met de transitie. De idee achter deze voorstelling van het programma is dat door de aanwezigheid van het token in p_1 , het programma "gestart" wordt: de eerste twee instructies $i = 1$ en $j = 2$ mogen worden uitgevoerd en zijn zelfs niet afhankelijk van elkaar en daarom is het beëindigen van de ene geen voorwaarde voor het begin van de andere. Hier wordt o.a. gemodelleerd dat die twee instructies in parallel mogen worden uitgevoerd. De instructie $k = i + j$ is afhankelijk van het beëindigen van de eerste twee instructies en zal pas mogen uitgevoerd worden - de terminologie is "de transitie vuurt" - indien er een token zit zowel in p_4 als in p_5 . Als die transitie gevuurrd heeft, dan is er al een preconditie voldaan voor de uitvoering van de laatste toekenning en daarom zal er al een token verschijnen in p_{11} . Doch p_{10} heeft nog geen token en daarom moet de instructie nog wachten.

Definitie 4.38 (Petrinet) Een petrinet is een enkelvoudige gerichte graaf $G(V, E)$ waarbij $V = P \cup T$, $P \cap T$ en $E \subseteq P \times T \cup T \times P$; P noemen we plaatsen en T transities.

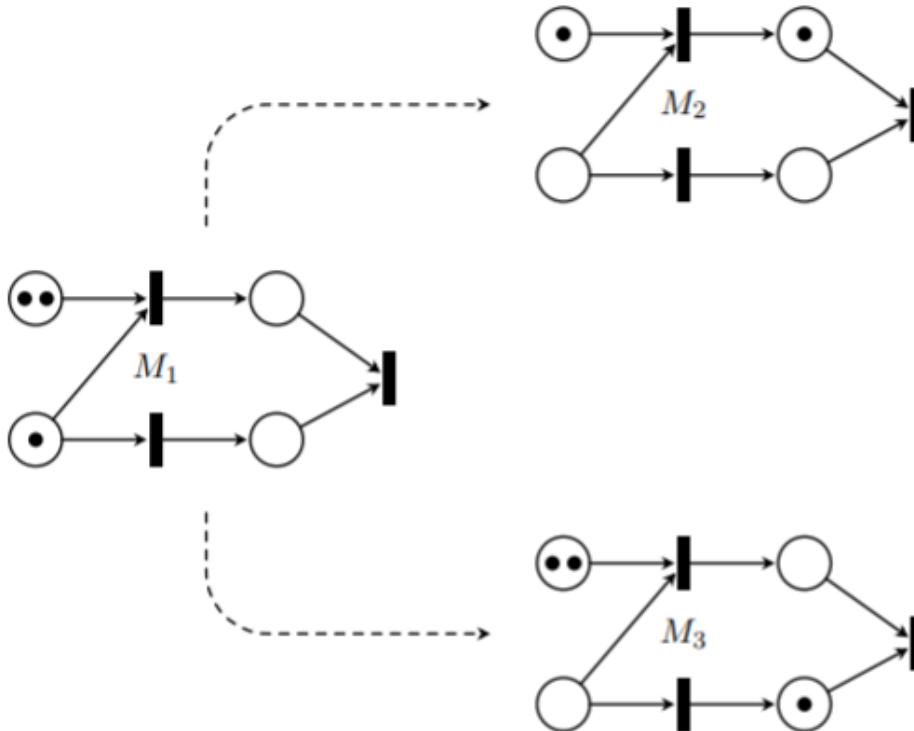
In het algemeen hoeft een Petrinet niet enkelvoudig te zijn, maar voor de eenvoud zullen we altijd onderstellen dat er geen parallelle bogen zijn

Definitie 4.39 (Markering van een Petrinet) Een markering van een Petrinet is een toekenning van een natuurlijk getal aan elke plaats van het Petrinet.

Plaatsen modelleren voorwaarden, terwijl transities gebeurtenissen voorstellen. Op zijn minste 1 token in een plaats betekent dat de voorwaarde voldaan is.

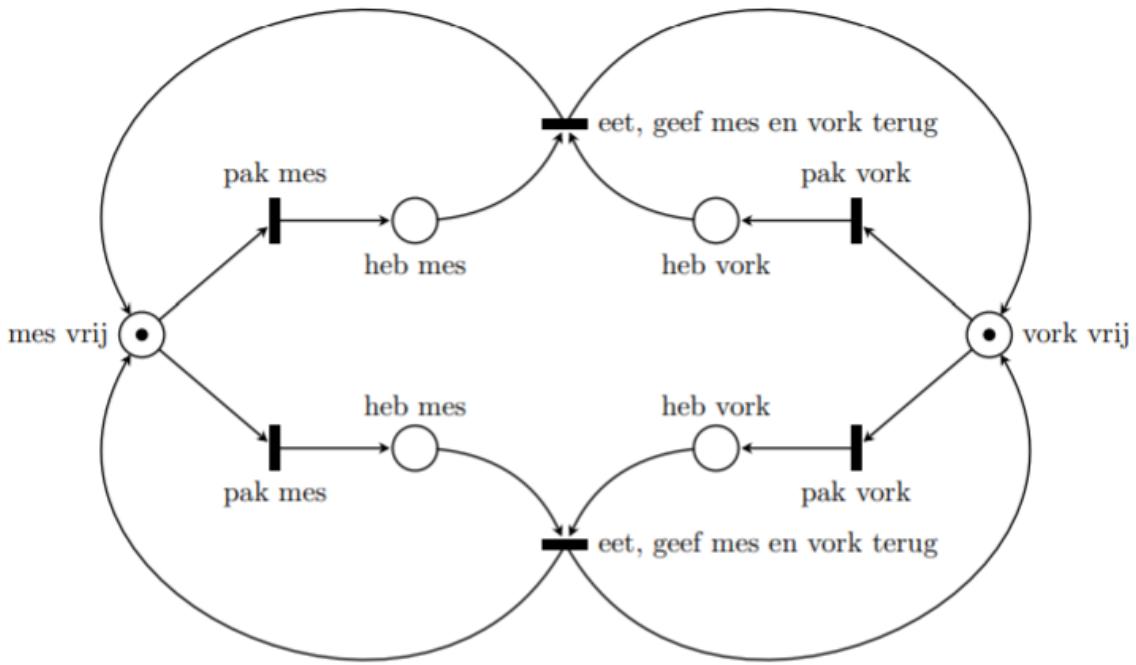
Definitie 4.40 (Bereikbare markering) Een markering M' is bereikbaar vanuit een markering M , indien er vertrekend van M een salvo is dat M' als resultaat heeft.

Vanuit een bepaalde markering is soms meer dan 1 markering bereikbaar met slechts 1 vurende transiti. Zie bijvoorbeeld Figuur 4.57.



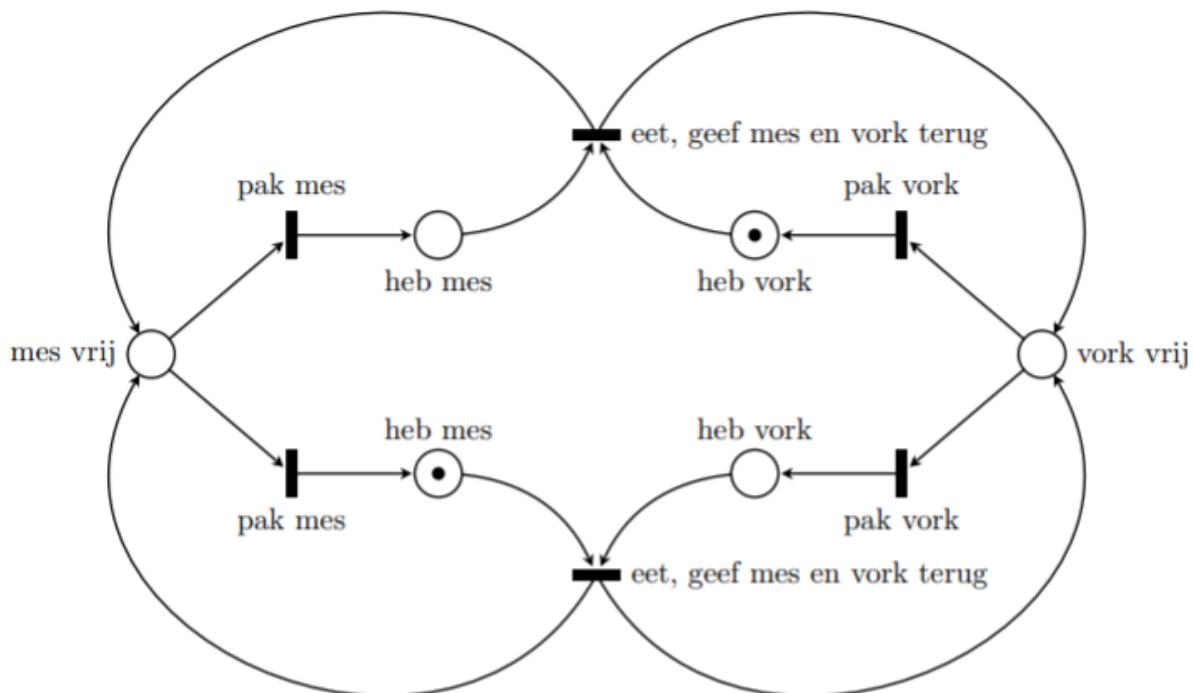
Figuur 4.57: Twee bereikbare markeringen

Wat uitlegt bij deze figuur. Om het algoritme te kunnen doen beginnen hebben we in elke startplaats een token nodig. Dit is het geval bij M_1 . Als we dan naar staat M_2 gaan zien we dat de 2 tokens 'geconsumeerd' zijn en we vast zitten omdat er niet genoeg tokens meer zijn om verder te gaan. Dit noemen we een deadlock. Hieronder een concreet voorbeeld voor nog wat verduidelijking. Twee personen willen iets eten maar hebben daar een mes en vork voor nodig terwijl we maar 1 mes en 1 vork hebben. Indien de eerste persoon het mes neemt en de tweede neemt de vork, dan kunnen geen van beide beginnen eten en zitten beide te wachten op het andere stuk bestek.



Figuur 4.58: Model van twee personen die willen eten met één mes en één vork

Dit betekent natuurlijk niet dat als er geen deadlock kan voorkomen dat alle transities in een petrinet aan bod gaan komen. Het petrinet kan er bijvoorbeeld voor zorgen dat 1 persoon nooit iets kan eten, zelfs als ze iets zouden "afspreken". Daarom is er de notie van een levend markering voor een Petrinet.



Figuur 4.59: Klassiek voorbeeld van een deadlock

Zie hier een voorbeeld van een deadlock en sta verstelt.

Definitie 4.41 (Levende markering) Een markering M van een Petrinet is levend, indien na elk salvo te beginnen bij M , er voor elke transitie t een salvo bestaat zodanig dat na dat salvo t actief wordt.

De definitie zegt niet dat de transitie t ook vuurt, enkel dat alle precondities voor t voldaan zijn. Als een markering levend is, dan is het duidelijk dat die markering geen aanleiding kan zijn tot deadlock. Een andere eigenschap die belangrijk is bij Petrinetten, is de hoeveelheid tokens die een bepaalde plaats ooit kan bevatten: je kan de plaatsen zien als een register met een teller die aangeeft hoeveel tokense er zijn en je wil graag weten dat een bepaalde capaciteit van dat register nooit overschreden kan worden.

Definitie 4.42 (Begrensde markering) Een markering M is begrensd indien er een eindige n bestaat zodanig dat bij gelijk welk salvo, geen enkel plaats ooit meer dan n tokense bevattet. Een begrensde markering waarvoor $n = 1$ wordt een veilige markering genoemd.

Hoofdstuk 5 Vastepuntstheorie

5.1 Inleiding

deze inleiding is cools certified

Ik vind deze inleiding volledig waardeloos dus als ge het zou willen gaan lezen dan moet ge naar de cursus :).

5.2 Orderrelaties

5.2.1 Basisbegrippen

Van cruciaal belang in de theorie van de vaste punten is het begrip orderrelatie. Ter herinnering: een relatie R op een verzameling S is niets anders dan een deelverzameling van de productverzameling $S \times S$. Een relatie bestaat dus uit koppels (x, y) met $x, y \in S$. We noteren zowel $(x, y) \in R$ als xRy om aan te geven dat het koppel (x, y) behoort tot de relatie R .

Definitie 5.1 (Partiële orderrelatie)

Een relatie R op een verzameling S is een (**partiële**) **orde(relatie)** als en slechts als voldaan is aan de volgende drie voorwaarden:

1. R is reflexief $\forall x \in S : x R x$
2. R is antisymmetrisch: $\forall x, y \in S : x R y \text{ en } y R x \Rightarrow x = y$
3. R is transitief: $\forall x, y, z \in S : x R y \text{ en } y R z \Rightarrow x R z$.

Effe opfrisser voor mensen die vergeten zijn wat deze termen betekenen (zoals ik (:)

Definitie 7.1. Een relatie $R \subseteq A \times A$ is **reflexief** als en slechts als

$$\forall x \in A : (x, x) \in R.$$

Definitie 7.2. Een relatie $R \subseteq A \times A$ is **anti-reflexief** als en slechts als

$$\forall x \in A : (x, x) \notin R.$$

Definitie 7.3. Een relatie $R : A \times A$ is **symmetrisch** als en slechts als

$$\forall x, y \in A : (x, y) \in R \Leftrightarrow (y, x) \in R.$$

Definitie 7.4. Een relatie $R : A \times A$ is **anti-symmetrisch** als en slechts als

$$\forall x, y \in A : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y.$$

Definitie 7.5. Een relatie $R : A \times A$ is **transitief** als en slechts als

$$\forall x, y, z \in A : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R.$$

In het volgende voorbeeld, en ook verder in de cursus, stelt $P(S)$ de machtsverzameling van een verzameling S voor, m.a.w. de verzameling van alle deelverzamelingen van S .

Voorbeeld is de relatie " \leq "

Definitie 5.2 (Boven- en ondergrens)

Zij S een verzameling met een partiële orde \leq .

1. $a \in S$ is een bovengrens (of majorant) voor een deel $X \subseteq S \Leftrightarrow \forall x \in X : x \leq a$.
2. $b \in S$ is een ondergrens (of minorant) voor een deel $X \subseteq S \Leftrightarrow \forall x \in X : b \leq x$

Opmerking Met de notatie S, \leq duiden we in het vervolg aan dat S een verzameling is, voorzien van een partiële orderrelatie " \leq " (bv. N, \leq en $N, |$). Indien S, \leq een partiële geordende verzameling is en $x, y \in S$ dan noteren we ook $x \geq y$ wanneer we $y \leq x$ bedoelen

Voorbeeld 5.2

1. In \mathbb{N}, \leq
 - (a) is 7 een bovengrens voor $X = \{2, 3, 4\}$
 - (b) is 2 een ondergrens voor $X = \{2, 3, 4\}$
 - (c) heeft $X = \{2, 4, 6, 8, \dots\}$ geen bovengrens.
2. In $\mathbb{N}, |$
 - (a) is 12 een bovengrens voor $X = \{2, 3, 4\}$, maar 7 niet.
 - (b) is 1 de enige ondergrens voor $X = \{2, 3, 4\}$
 - (c) heeft $X = \{2, 4, 6, 8, \dots\}$ het getal 0 als bovengrens.
3. In $\mathcal{P}(A), \subseteq$ met $A = \{a, b, c, d\}$ is
 - (a) $\{a, b, c\}$ een bovengrens voor $X = \{\{a, b\}, \{a, c\}\}$
 - (b) $\{a\}$ een ondergrens voor $X = \{\{a, b\}, \{a, c\}\}$
 - (c) \emptyset een ondergrens voor elke $X \subseteq \mathcal{P}(A)$
 - (d) A een bovengrens voor elke $X \subseteq \mathcal{P}(A)$

Wij zullen speciale aandacht hebben voor “de kleinste mogelijke bovengrens” en “de grootste mogelijke ondergrens”.

Definitie 5.3 (Sup en inf)

Zij S, \leq een geordende verzameling en $X \subseteq S$.

1. $a \in S$ is een kleinste bovengrens van X of een supremum van X , notatie $a = \sup(X)$ als en slechts als a een bovengrens is voor X en voor elke bovengrens a' van X geldt dat $a \leq a'$
2. $b \in S$ is een grootste ondergrens van X of een infimum van X , notatie $b = \inf(X)$ als en slechts als b een ondergrens is voor X en voor elke ondergrens b' van X geldt dat $b \geq b'$

Opmerking Indien een supremum (infimum) bestaat, is het uniek (Waarom?). We kunnen dus in het vervolg spreken van **het supremum (infimum)**.

Voorbeeld 5.3

1. Beschouw het interval $X = [2, 3[$ in \mathbb{R}, \leq . Dan is $2 = \inf(X)$ en $3 = \sup(X)$.
2. In \mathbb{Z}, \leq heeft $X = \{0, 2, 4, 6, 8, \dots\}$ wel een infimum, namelijk $\inf(X) = 0$, maar geen supremum, want X heeft geen bovengrenzen.
3. In \mathbb{Q}, \leq heeft $X = \{q \in \mathbb{Q} \mid \sqrt{2} < q < \Pi\} =]\sqrt{2}, \Pi[\cap \mathbb{Q}$ noch een supremum, noch een infimum ($\sqrt{2} \notin \mathbb{Q}$ en $\Pi \notin \mathbb{Q}$). Nochtans bestaan er zowel ondergrenzen als bovengrenzen voor X .

Definitie 5.4 (Complete tralie)

Een partieel geordende verzameling L, \leq is een complete tralie als $\sup(X)$ en $\inf(X)$ bestaan voor elke niet lege deelverzameling $X \subseteq L$.

Voorbeeld 5.4

1. De geordende verzamelingen \mathbb{N}, \leq , \mathbb{Z}, \leq , \mathbb{Q}, \leq en \mathbb{R}, \leq zijn geen complete tralies.
(waarom?)
2. Neem het gesloten interval $[a, b] \subseteq \mathbb{R}$, dan is $[a, b], \leq$ wel een complete tralie.
3. Voor een willekeurige verzameling A geldt dat de geordende verzameling $\mathcal{P}(A), \subseteq$ een complete tralie is. Inderdaad, neem $X \subseteq \mathcal{P}(A)$. Er geldt dat

$$\sup(X) = \bigcup_{C \in X} C \quad \text{en} \quad \inf(X) = \bigcap_{C \in X} C$$

Aangezien in een complete tralie L, \leq het supremum en het infimum bestaat voor elke $X \subseteq L$, kunnen we ook $X = L$ beschouwen.

Definitie 5.5 (Top- en bodemelement)

In een complete tralie L, \leq noemen we

$$\begin{aligned}\top &= \text{topelement} = \sup(L) \\ \bot &= \text{bodemelement} = \inf(L)\end{aligned}$$

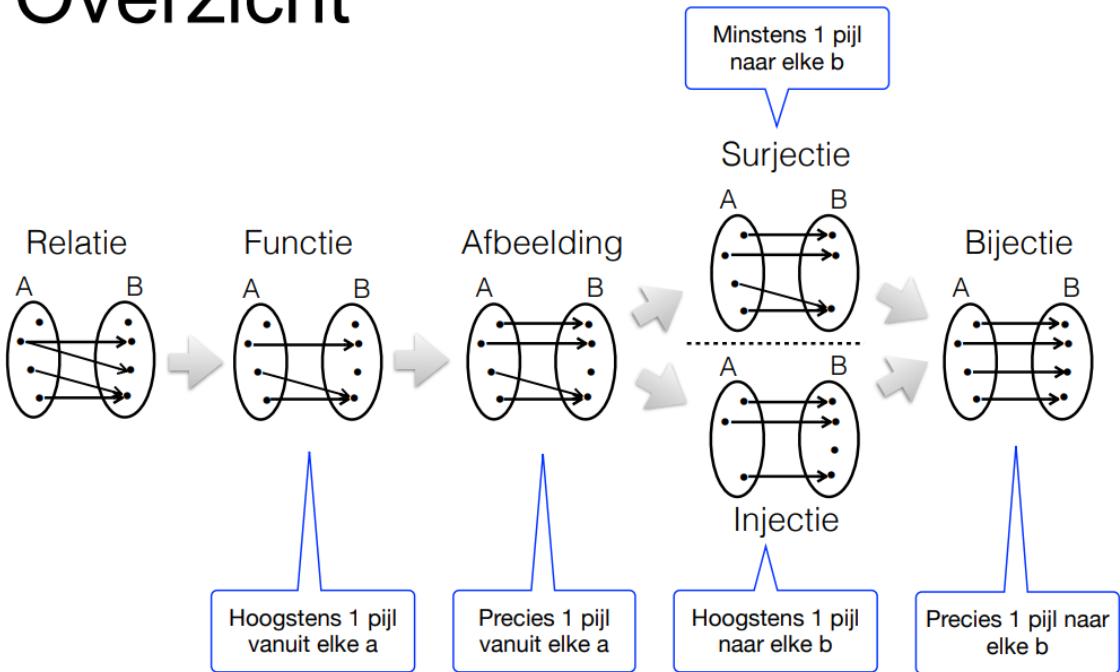
Voorbeeld 5.5

In de complete tralie $\mathcal{P}(A)$ (A een willekeurige verzameling) is

$$\top = A \quad \text{en} \quad \bot = \emptyset$$

5.2.2 Monotone en continue afbeeldingen

Overzicht



Definitie 5.6 (Monotone afbeelding)

Zij L, \leq een geordende verzameling. Een afbeelding $T: L \rightarrow L$ noemen we monotoon als uit $x \leq y$ volgt dat $T(x) \leq T(y)$

Voorbeeld 5.6

1. In \mathbb{R}, \leq is elke stijgende afbeelding een monotone afbeelding. Vb. de afbeelding $T: \mathbb{R} \rightarrow \mathbb{R}: x \mapsto x^3$ is een monotone afbeelding. Ook de constante afbeelding $T: \mathbb{R} \rightarrow \mathbb{R}: x \mapsto 4$ is monotoon.
2. Neem een willekeurige verzameling A en $B \subseteq A$. Dan is

$$T: \mathcal{P}(A) \rightarrow \mathcal{P}(A): X \mapsto X \cap B$$

een monotone afbeelding op de geordende verzameling $\mathcal{P}(A), \subseteq$, want uit $X \subseteq Y$ volgt dat $T(X) = X \cap B \subseteq Y \cap B = T(Y)$.

Definitie 5.7 (Gerichte verzameling)

Beschouw een willekeurige deelverzameling X van een complete tralie L, \leq . X is gericht \Leftrightarrow elke eindige deelverzameling van X heeft een bovengrens in X .

Voorbeeld 5.7

1. Zij L, \leq een complete tralie, dan is L gericht, want elke deelverzameling X van L , dus ook de eindige, hebben een bovengrens, namelijk het topelement \top , in L .
2. In $[a, b], \leq$ is elke deelverzameling een gerichte verzameling.
3. Zij A een oneindige verzameling en beschouw de verzameling X van alle eindige deelverzamelingen van A :

$$X = \{B \subseteq A \mid B \text{ is eindig}\}.$$

X is gericht, want elke eindige deelverzameling $\{B_1, B_2, \dots, B_r\} \subseteq X$ heeft een bovengrens in X , namelijk $B = B_1 \cup B_2 \cup \dots \cup B_r$.

Merk echter op dat X zelf (X is een oneindige deelverzameling van zichzelf) geen bovengrens heeft in X . (Zie oefening 3).

Definitie 5.8 (Continue afbeelding)

een afbeelding $T: L \rightarrow L$ op een complete tralie L, \leq wordt een continue afbeelding genoemd, indien

$$T(\sup(X)) = \sup(T(X))$$

voor elke gerichte deelverzameling X van L . ($T(X) = \{T(x) \mid x \in X\}$)

Voorbeeld 5.8

De monotone afbeelding $T: \mathcal{P}(A) \rightarrow \mathcal{P}(A) : X \mapsto X \cap B$ (A en B zoals hierboven) is continu. Inderdaad, voor elke $X \subseteq \mathcal{P}(A)$ geldt dat (met $X = \{X_i \mid i \in I\}$):

$$\begin{aligned} T(\sup(X)) &= \left(\bigcup_{i \in I} X_i \right) \cap B \\ &= \bigcup_{i \in I} (X_i \cap B) \\ &= \sup(\{X_i \cap B \mid i \in I\}) \\ &= \sup(\{T(X_i) \mid i \in I\}) \\ &= \sup(T(X)) \end{aligned}$$

De volgende eigenschap geeft een verband tussen continue en monotone afbeeldingen.

Eigenschap 5.1 Voor een afbeelding $T: L \rightarrow L$ op een complete tralie L, \leq geldt

$$T \text{ is continu} \Rightarrow T \text{ is monotoon}$$

Bewijs: Veronderstel dat $T : L \rightarrow L$ continu is. Neem twee willekeurige elementen $x, y \in L$ met $x \leq y$. We moeten aantonen dat $T(x) \leq T(y)$.

Neem $X = \{x, y\}$, dan is X een gerichte verzameling van L , want y is een bovengrens voor elke (eindige) deelverzameling van X . Omdat T continu is geldt

$$\begin{aligned} \sup(T(X)) &= T(\sup(X)) \\ &\Downarrow \\ \sup\{T(x), T(y)\} &= T(y) \\ &\Downarrow \\ T(x) &\leq T(y) \end{aligned}$$

Voorbeeld 5.9

De afbeelding

$$f : [0, 1] \rightarrow [0, 1] : x \mapsto \begin{cases} \frac{1}{2}x & \text{als } 0 \leq x < \frac{1}{2} \\ \frac{1}{2}x + \frac{1}{2} & \text{als } \frac{1}{2} \leq x \leq 1 \end{cases}$$

op de complete tralie $[0, 1]$, \leq is een monotone afbeelding die niet continu is.
(Zie oefening 5)

Wel geldt altijd de volgende ongelijkheid:

Eigenschap 5.2 Zij $T : L \rightarrow L$ een monotone afbeelding op een complete tralie L, \leq , dan geldt voor elke deelverzameling $X \subseteq L$ dat

$$\sup(T(X)) \leq T(\sup(X))$$

Bewijs:

$$\begin{aligned} &\forall x \in X : x \leq \sup(X) \\ \Rightarrow &\forall x \in X : T(x) \leq T(\sup(X)) \\ \Rightarrow &\sup(T(X)) \leq T(\sup(X)) \end{aligned} \tag{1}$$

5.3 De stellingen van Tarski en Kleene

5.3.1 De stelling van Tarski

Definitie 5.9 (Vaste punten)

Zij L, \leq een complete tralie en beschouw een afbeelding $T : L \rightarrow L$

1. We noemen $a \in L$ een vast punt voor T als $T(a) = a$
2. Een kleinste vast punt voor T is een vast punt a voor T zodat voor elk ander vast punt b voor T geldt dat $a \leq b$.
3. Een grootste vast punt voor T is een vast punt a voor T zodat voor elk ander vast punt b voor T geldt $a \geq b$.

Voorbeeld 5.10

Voor de monotone afbeelding $T : \mathcal{P}(A) \rightarrow \mathcal{P}(A) : X \mapsto X \cap B$ die we reeds een aantal keer ontmoet hebben geldt:

1. X is een vast punt $\Leftrightarrow X \cap B = X \Leftrightarrow X \subseteq B$
2. Het kleinste vast punt is \emptyset
3. Het grootste vast punt is B

Stelling 5.1 (Stelling van Tarski) Zij $T : L \rightarrow L$ een monotone afbeelding op een complete tralie L, \leq , dan heeft T een kleinste vast punt a en een grootste vast punt b .

Bovendien geldt:

$$\begin{aligned} a &= \inf\{x | T(x) = x\} = \inf\{x | T(x) \leq x\} \\ b &= \sup\{x | T(x) = x\} = \sup\{x | T(x) \geq x\} \end{aligned}$$

Gevolg 5.1 Zij $T : L \rightarrow L$ een monotone afbeelding op een complete tralie.

Veronderstel dat $a \in L$ en $a \geq T(a)$, dan bestaat er een vast punt a' zodat $a \geq a'$.

Veronderstel dat $b \in L$ en $b \leq T(b)$, dan bestaat er een vast punt b' zodat $b \leq b'$.

5.3.2 De stelling van Kleene

Notatie: Zij $T : L \rightarrow L$ een monotone afbeelding op een complete tralie L . Dan noteren we

- $T \upharpoonright 0 = \perp$
- $T \upharpoonright (n + 1) = T(T \upharpoonright n)$, voor elke $n \in \mathbb{N}$
- $T \upharpoonright \omega = \sup\{T \upharpoonright n \mid n \in \mathbb{N}\}$

Stelling 5.3 De stelling van Kleene

Zij $T : L \rightarrow L$ een continue afbeelding op een complete tralie L, \leq , dan is $T \upharpoonright \omega =$ het kleinste vast punt van T .

5.3.3 Voorbeeld: de transitieve sluiting van een relatie

Definitie 5.10 Zij $R \subseteq A \times A$ een relatie op een verzameling A . De transitieve sluiting van R is de kleinste relatie op S op A zodat $R \subseteq S \subseteq A \times A$ en S is transitief

Eigenschap 5.3 De transitieve sluiting van een relatie R op een verzameling A bestaat altijd en is gelijk aan

$$S_0 = \bigcap_{S \in \tau} S, \text{ met } \tau = \{S \subseteq A \times A \mid R \subseteq S \text{ en } S \text{ is transitief}\}$$

Definitie 5.11 Op L, \leq beschouwen we vanaf nu de afbeelding

$$T : L \rightarrow L : S \mapsto T(S)$$

$$\text{met } T(S) = S \cup \{(a, b) \in A \times A \mid \exists c \in A, (a, c) \in S \text{ en } (c, b) \in S\}.$$

Eigenschap 5.5 De transitieve sluiting van de relatie R is precies het kleinste vast punt van T .