Algoritmische grafentheorie

Gunnar Brinkmann

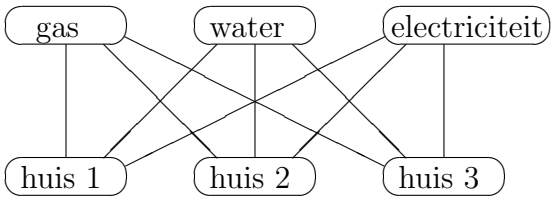
25 januari 2017

Inhoudsopgave

1 Inleiding

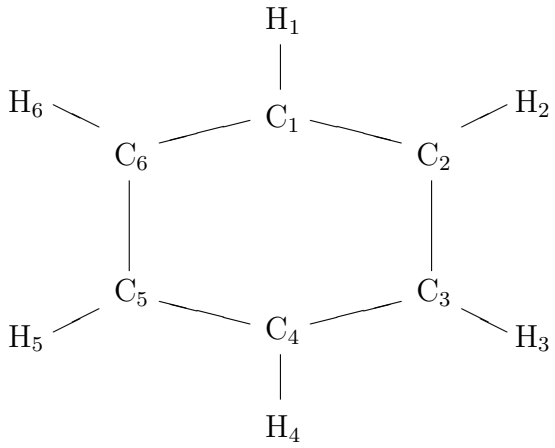
Deze lesnota's zijn niet bedoeld om los van de les gebruikt te worden. Er zijn bv. oefeningen die uitgewerkt **moeten** worden (en dat zal in de lessen gebeuren) en waarvan de juiste oplossing belangrijk is om sommige van de resultaten te verstaan. In de les wordt er natuurlijk voor gezorgd dat de oplossingen zo zijn *als het bedoeld is*. Door deze nauwe samenhang tussen oefeningen en inhoud van de tekst hoop ik dat het oplossen van de oefeningen nog interessanter is. De bedoeling van deze lesnota's is dus alleen de les voor te bereiden en te steunen!

Grafentheorie is een vlak dat volgens sommigen deel uitmaakt van de zuivere wiskunde en volgens sommigen van de toegepaste wiskunde. Wat zeker is, is dat veel resultaten uit de grafentheorie toepasbaar zijn. Of het zuivere of toegepaste wiskunde is hangt er dan sterk vanaf welke resultaten je bedoelt en op welke manier de resultaten voorgesteld worden. Als het om toepassingen gaat dan gaat het heel vaak ook om manieren om invarianten te berekenen of substructuren te vinden – het gaat om algoritmen.
In deze les zullen wij een algoritmisch aanpak gebruiken – dat betekent dat wij (waar mogelijk) de bewijzen op een constructieve algoritmische manier proberen te doen en dat wij er ook in geïnteresseerd zijn hoe moeilijk (tijdscomplexiteit) de problemen zijn die wij bespreken.
Maar omdat er ook studenten aanwezig kunnen zijn die nog helemaal niets over grafentheorie gehoord hebben, beginnen wij natuurlijk met de basisdefinities.
In Figuur 1 zie je een klein netwerk en in Figuur 2 zie je het scheikundige molecuul benzine. Dat zijn voorbeelden van de structuren waarover het hier gaat.



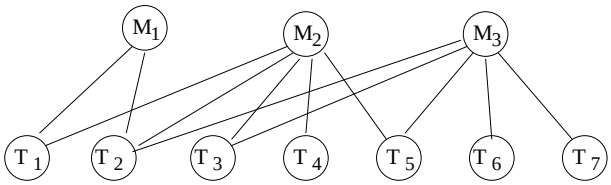
Figuur 1:

Maar nog veel meer structuren zijn duidelijk wat wij als grafen zullen definiëren: bv. schakelschemata, wegennetten, etc.
Maar er zijn ook gevallen die (zoals wij zullen zien) wel *in principe* grafen zijn, maar waar je dat niet onmiddellijk kan zien – zoals bv. lesroosters of taakverdelingen.



Figuur 2:

Een voorbeeld:
Je hebt drie machines en taken $T_1, T_2, \ldots, T_7$ waarbij sommige machines sommige taken kunnen uitvoeren en anderen niet. Deze situatie kan je als volgt modelleren (waarbij een machine met een taak is verbonden als hij die kan uitvoeren):



Figuur 3:

2 Basisdefinities

Nu zullen wij expliciet definiëren wat een graaf is:

1 Definitie *Stel dat $V \neq \emptyset$ een eindige verzameling is en $E \subseteq \binom{V}{2}$ (dus een verzameling van deelverzamelingen van V met 2 elementen. Om geen verwarring te hebben stellen wij altijd dat $V \cap E = \emptyset$.*

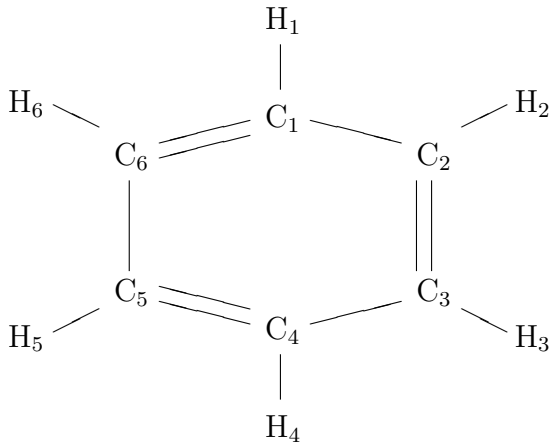
Dan heet het paar $G = (V, E)$ een graaf (engels: graph), een element van V heet een top (vertex) van G en een element van E heet een boog (edge) van G. Voor een boog $e = \{v, w\}$ zeggen wij ook dat v en w de eindpunten zijn en dat e een boog tussen v en w *is en dat v en w buren zijn.*
Doordat wij eisen dat E een verzameling is, is het vastgelegd dat er ten hoogste één boog is tussen twee vaste eindpunten v, w.
Twee toppen v, w heten adjacent als $\{v, w\} \in E$ en een top v en een boog e heten incident als $v \in e$.
Als $E = \binom{V}{2}$ noemen we de graaf compleet en schrijven K_n waarbij $n = |V|$.
De graad (degree) van een top v (wij schrijven $\deg v$) is het aantal met v incidente bogen dus $\deg(v) = \#\{e \in E | v \in e\}$. Voor de kleinste graad van een top in een graaf G schrijven wij $\delta(G)$ en voor de grootste voorkomende graad schrijven wij $\Delta(G)$.
Als alle toppen dezelfde graad k hebben noemen wij de graaf ook k-regulier.

De graaf die het netwerkje voorstelt zou dus
$G = (\{huis1, huis2, huis3, gas, water, electriciteit\}, \{\{huis1, water\},$
$\{huis1, electriciteit\}, \{huis1, gas\}, \{huis2, water\}, \{huis2, electriciteit\}, \{huis2, gas\},$
$\{huis3, water\}, \{huis3, electriciteit\}, \{huis3, gas\}\})$
zijn.
Daarbij zijn *huis*1 en *huis*2 niet adjacent maar *huis*3 en *gas* wel. Bovendien is *huis*1 incident met de boog $\{huis1, water\}$ en elke top heeft graad 3 – het is dus een 3-reguliere graaf.

2 Oefening *Schrijf de grafen van benzine en van ons taakverdelingsvoorbeeld formeel als grafen op en bepaal de graden van de toppen.*

Als jullie het begrip *graaf* in artikels of boeken tegenkomen dan staat het niet altijd voor dezelfde structuur. Er zijn meerdere variaties van grafen en een auteur kiest soms het kortste woord voor de structuur waarmee hij het meest bezig is. In ons voorbeeld van een molecule is al een beetje gesjoemeld: koolstofatomen hebben normaal vier bindingen maar in ons voorbeeld zijn er maar drie. Dit kan je modelleren door te stellen dat er dubbele bogen tussen sommige toppen zijn (zie Figuur 4) – maar dat laat ons model van een graaf natuurlijk niet toe. Wij zouden een dergelijke structuur een *multigraaf* noemen. Voor een multigraaf moet E geen verzameling zijn maar een multiverzameling. Maar wij kunnen de definitie van een multiverzameling ook als deel van de definitie gebruiken:

3 Definitie *Een multigraaf (multigraph) is een 3-tal (V, E, f), waarbij V en E eindige verzamelingen zijn met $V \cap E = \emptyset$. V is de verzameling van toppen*



Figuur 4:

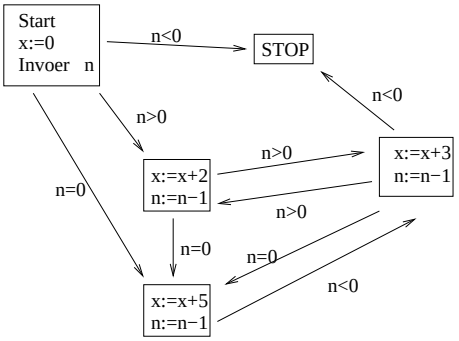
en E is de verzameling van bogen. Bovendien is f een functie $E \to \binom{V}{2}$ die de bogen de eindpunten toekent.
De definities van adjacent, incident en graad zijn dan analoog aan die voor grafen (bv.: Toppen v, w heten adjacent als er een boog $e \in E$ bestaat met $f(e) = \{v, w\}$.)

In het voorbeeld is $V = \{C_1, \ldots, C_6, H_1, \ldots, H_6\}$, $E = \{e_1, \ldots e_{15}\}$ en $f(e_1) = f(e_2) = \{C_1, C_6\}$, $f(e_3) = \{C_1, H_1\}$, $f(e_4) = \{C_1, C_2\}$, $f(e_5) = \{C_2, H_2\}$, $f(e_6) = f(e_7) = \{C_2, C_3\} \ldots$

4 Oefening *Geef de expliciete definities voor* incident *en* graad *voor multigrafen.*

Als mensen voor wat wij multigrafen noemen het begrip grafen gebruiken dan heten de structuren die wij grafen noemen *enkelvoudige grafen* (*simple graphs*).
In sommige toepassingen worden ook structuren gebruikt waar beide eindpunten van een top identiek zijn – dat noemen wij een lus. Formeel moet je dan alleen maar in plaats van $E \subseteq \binom{V}{2}$ eisen dat $E \subseteq \binom{V}{2} \cup V$ resp. $f : E \to \binom{V}{2} \cup V$. Voor de graad van een top v worden lussen van v naar v dubbel geteld.
In andere gevallen – zoals in deze flowchart – hebben bogen een duidelijk begin en een duidelijk eind – ze zijn gericht. dat geeft aanleiding tot *gerichte grafen.*

5 Definitie *Stel dat $V \neq \emptyset$ een eindige verzameling is en $E \subseteq V \times V \setminus$*



Figuur 5:

$\{(v,v)|v \in V\}$ *(dus een verzameling van 2-tallen of vectoren met twee verschillende elementen uit* V*) en* $V \cap E = \emptyset$.
Dan heet het paar $D = (V, E)$ *een gerichte graaf (directed graph of digraph). Als een boog de vorm* (v, w) *heeft dan heet* v *het startpunt en* w *het eindpunt. De ingraad van een top* v *is het aantal bogen van de vorm* (w, v) *en de uitgraad van* v *is het aantal bogen van de vorm* (v, w).

6 Oefening *Ontwikkel een precieze definitie van wat jij een* gerichte multigraaf met lussen *zou noemen en definieer ook wat* adjacent, incident *en* graad *betekenen.*

Nu wij precies weten wat een graaf is, hebben wij nog sommige basisdefinities nodig voordat wij eerste resultaten kunnen bewijzen:

7 Definitie

- *Als* $G = (V, E)$ *een* $G' = (V', E')$ *grafen zijn en bovendien* $V' \subseteq V$ *en* $E' \subseteq E$ *dan heet* G' *een deelgraaf (subgraph) van* G. *Wij schrijven dan ook* $G' \subseteq G$.

- *Als* $G = (V, E)$, $G' = (V', E')$ *en* $G' \subseteq G$ *en bovendien geldt:* $\forall v, w \in V' : \{v, w\} \in E \Rightarrow \{v, w\} \in E'$ *dan heet* G' *een geïnduceerde (induced) deelgraaf.*

- *Als* $G = (V, E)$ *en graaf is en* $V' \subseteq V$ *dan heet de graaf* $G[V']$ *met* $G[V'] = (V', \{\{v, w\}|v, w \in V', \{v, w\} \in E\})$ *de door* V' *in* G *geïnduceerde deelgraaf.*

- *Een padgraaf of pad* P_n *met* $n \geq 0$ *bogen is een graaf* $P = (V, E)$ *met* $V = \{v_0, \ldots, v_n\}$ *(waarbij alle* v_i $0 \leq i \leq n$ *verschillend zijn) en*

$E = \{\{v_i, v_{i+1}\} | 0 \leq i < n\}$. *De* lengte *van een pad is het aantal bogen in het pad.*

- *Als een padgraaf* $P = (V', E')$ *met* $V' = \{v_0, \ldots, v_n\}$ *een deelgraaf is van een graaf* $G = (V, E)$ *dan noemen wij* P *soms ook een* pad in G *van* v_0 *naar* v_n *en schrijven kort* $v_0, \ldots, v_n$ *voor dit pad.*

3 Samenhang en stromen

3.1 Samenhang

8 Definitie • *Een graaf* $G = (V, E)$ *heet* samenhangend *(connected) als voor alle* $v, w \in V$ *een pad van* v *naar* w *bestaat.*

9 Oefening *Gegeven een graaf* $G = (V, E)$. *Voor* $v, w \in V$ *definieer* $v \equiv_{sam} w$ *als er een pad van* v *naar* w *is.*

- *Toon aan dat* $\equiv_{sam}$ *een equivalentierelatie is.*

10 Definitie *Wij noemen de equivalentieklassen van* $\equiv_{sam}$ samenhangscomponenten *of kort* componenten *van de graaf.*

Dat *samenhang* een bijzonder belangrijk begrip is, is onmiddellijk duidelijk. Als je bv. wegennetten of computernetwerken hebt die niet samenhangend zijn dan is dat zeker een probleem. En je wil zeker ook geen computernetwerk waar een enkele computer kan uitvallen en daardoor onmiddellijk veel andere computers niet meer bereikbaar zijn – je wil dus een netwerk met een groot samenhangsgetal. En in het geval van een wegennet zou het ook niet leuk zijn als er werken op een straat zijn en daardoor grote delen niet meer bereikbaar zijn. Hier is het doel zeker een groot boogsamenhangsgetal.

Maar hoe kan je berekenen of een graaf samenhangend is?
Als wij het over algoritmen voor grafen hebben dan stellen wij dat elke top een lijst van zijn buren heeft (dus een lijst van toppen).
Er zijn twee belangrijke manieren om een graaf te doorzoeken diepte eerst (depth first) – dat wordt normaal afgekort als DFS of breedte eerst (breadth first) of BFS. Ze zijn allebei geschikt om vast te stellen of een graaf samenhangend is en vormen de basis voor verschillende andere algoritmen.

11 Algoritme Breadth First Search – BFS
Gegeven een graaf $G = (V, E)$

L zij een lijst van toppen die in het begin alleen maar één arbitraire top s van G bevat.
Alle toppen van G – behalve s – zijn in het begin ongemarkeerd.
De starttop s is gemarkeerd.
Nu herhaal het volgende totdat de lijst leeg is:

- *verwijder het eerste element l uit de lijst*
- *voeg alle buren van l die nog niet gemarkeerd zijn aan het einde van de lijst toe en markeer ze.*

12 Stelling *Als BFS op een graaf G wordt toegepast dan zijn op het einde alle toppen gemarkeerd als en slechts als G samenhangend is.*

Voor dit bewijs is een definitie nuttig van een getal dat je ook gemakkelijk met een kleine variatie van BFS kan berekenen: de afstand tussen twee toppen:

13 Definitie *Gegeven een graaf $G = (V, E)$ en $v, w \in V$. De afstand (distance) $d(v, w)$ is de lengte van een kortste pad van v naar w in G. Als zo'n pad niet bestaat, definiëren wij $d(v, w) = \infty$.*
Of formeel: $d(v, w) = \min(\{l(P) | P \text{is pad in } G \text{ van } v \text{ naar } w\} \cup \{\infty\})$

Bewijs: Wij bewijzen dat elke top in een samenhangende graaf gemarkeerd wordt met inductie in de afstand van de starttop s. Voor afstand 0 (dus s zelf) gebeurt dat onmiddellijk in het begin. Stel dus dat wij een top v met afstand $n > 0$ hebben en dat elke top met afstand ten hoogste $n - 1$ gemarkeerd wordt. Dan is er een pad met lengte n van s naar v. De top v' die net voor v op dit pad ligt, heeft dus een afstand van ten hoogste $n - 1$ (dat is de lengte van het deelpad) – en wordt dus (inductie) gemarkeerd en in de lijst geplaatst. Maar dan wordt ook v gemarkeerd – ten laatste op het moment dat v' uit de lijst wordt gehaald en zijn buren worden bekeken.

De andere richting volgt onmiddelijk omdat als v' vanuit v gemarkeerd wordt duidelijk geldt dat $v \equiv_{sam} v'$ (er is een pad van lengte 1 van v naar v'). Als alle toppen gemarkeed zijn, behoren dus alle toppen tot dezelfde samenhangscomponent – de graaf is dus samenhangend.

■

De complexiteit van dit algoritme is duidelijk $O(|V| + |E|)$ omdat naar elke boog ten hoogste twee keer gekeken wordt.

In het geval van samenhangende grafen is er geen verschil tussen $O(|V|+|E|)$ en $O(|E|)$ omdat er ten minste $|V|-1$ bogen zijn. Maar wij zullen toch al vaak $O(|V|+|E|)$ schrijven omdat zo de lengte van de invoer van een computerprogramma meer beklemtoond wordt. Maar dat is alleen maar een verschil in klemtoon – het resultaat is hetzelfde.

14 Algoritme *Gegeven $G=(V,E)$ en $s \in V$.*
Dit algoritme berekent alle afstanden $d(s,v)$ voor $v \in V$.
Afstand van een top s
Gegeven een graaf $G=(V,E)$ en $s \in V$.
L zij een lijst die in het begin alleen maar s bevat.
De afstand $d(s,s)$ wordt op 0 gezet, alle andere afstanden $d(s,v)$ van toppen $v \in V$ op ∞.
Nu herhaal het volgende totdat de lijst leeg is:

- *verwijder het eerste element l uit de lijst*
- *voeg alle buren v van l met afstand $d(s,v)=\infty$ aan het einde van de lijst toe en definieer $d(s,v)=d(s,l)+1$.*

15 Oefening *Bewijs dat dit algoritme inderdaad de afstanden correct berekent – dus dat op het einde voor alle $v \in V$ inderdaad geldt dat $d(s,v)$ de afstand tussen s en v in G bevat.*

Als je alle afstanden tussen twee punten in een graaf moet berekenen, kan je dit algoritme herhalen voor elk startpunt. De complexiteit is dan $O(|V| * (|V|+|E|))$ dus ten hoogste $O(|V|^3)$. Dit is ook de complexiteit van het algoritme van Floyd-Warshall dat in dergelijke gevallen meestal gebruikt wordt. Wij zullen het in de rest van de les niet nodig hebben, maar het is misschien toch goed om het te kennen:

16 Algoritme *(Floyd-Warshall)*

Gegeven $G=(V,E)$ en $s \in V$.
Dit algoritme berekent alle afstanden $d(v,w)$ voor $v,w \in V$.

Alle afstanden in een graaf (Floyd-Warshall)
Gegeven een graaf $G=(V,E)$ met $V=\{1,2,\ldots|V|\}$.
Maak een 2-dimensionale array $d[][]$ aan met voor $1 \leq i,j \leq |V|$:

$$d[i][j] = \begin{cases} 1 & \text{als } \{i,j\} \in E \\ 0 & \text{als } i=j \\ \infty & \text{anders} \end{cases}$$

Dan pas de volgende drie in elkaar genestelde lussen toe:
voor t *van* 1 *tot* $|V|$
  voor i *van* 1 *tot* $|V|$
    voor j *van* 1 *tot* $|V|$
      als $(d[i][j] > d[i][t] + d[t][j])$ **dan** $d[i][j] = d[i][t] + d[t][j]$

17 Oefening *Bewijs dat op het einde van het algoritme van Floyd-Warshall voor* $1 \leq i, j \leq |V|$ *in* $d[i][j]$ *de afstand tussen top* i *en top* j *staat.*
Tip: t *staat voor* tussenpunt *en in iteratie* t *wordt rekening gehouden met alle paden van* i *naar* j *waar alle tussenpunten in* $1, 2, \ldots, t$ *liggen.*

Een voordeel van BFS is dat het iteratief is en niet recursief. Hoewel recursieve programma's heel mooi zijn om te schrijven en ook heel kort, betekent dat voor de computer toch dat er veel oproepen van functies zijn en dat kan soms duurder zijn dan een iteratief programma.
Het algoritme diepte eerst zoeken (depth first search, DFS) is een typisch voorbeeld van een recursief programma:

18 Algoritme Depth First Search – DFS
Gegeven een graaf $G = (V, E)$ *en een arbitraire starttop* $s \in V$
Alle toppen van G *zijn in het begin ongemarkeerd.*
Dan roep de volgende functie DFS *met parameter* s *op:*

DFS(v) {
markeer v
voor alle buren b *van* v *doe:*

- *als* b *nog niet gemarkeerd is:*
  - *DFS(b)*

}

Ook de complexiteit van dit algoritme is duidelijk $O(|V| + |E|)$. Ook hier wordt naar elke boog ten hoogste twee keer gekeken.

19 Stelling *Als DFS op een graaf* G *wordt toegepast dan zijn op het einde alle toppen gemarkeerd als en slechts als* G *samenhangend is.*

Deze stelling kan op een (bijna) volledig analoge manier als de stelling voor BFS bewezen worden – alleen dat terwijl in het vorige bewijs een lijst gebruikt werd hier een recursie gebruikt wordt. Omdat wij hier niet eerst alle buren

bekijken maar onmiddellijk doorgaan zodra wij een nog niet gemarkeerde buur hebben gevonden, kan je met DFS ook niet de afstanden berekenen.
Maar de volgorde van DFS heeft ook voordelen – en om die te gebruiken hebben wij soms nodig wat wij de DFS-nummering noemen. De DFS-nummering geeft labels aan de toppen in de volgorde waarop DFS de toppen bezoekt. Daarvoor moet je alleen maar de markeringen door labels vervangen:

20 Algoritme Depth First Search Nummering
Gegeven een graaf $G = (V, E)$ en een arbitraire starttop $s \in V$
Alle toppen van G hebben in het begin het DFS-nummer ∞. De variabele nextlabel *is globaal en in het begin is nextlabel*$= 1$.
Dan roep de volgende functie DFS *met parameter s op:*

DFS_num(v) {
DFS$[v]$ =*nextlabel*
nextlabel=*nextlabel*+1
voor alle buren b van v do:

- *als DFS*$[b] = \infty$*: nog niet bezocht*
  - *DFS_num(b)*

}

Op deze manier hebben op het einde alle toppen in een samenhangende graaf een DFS-nummer.
Inderdaad bouwen BFS en DFS beide een substructuur op die wij een *opspannende boom* noemen. Daarvoor hebben wij nog sommige definities nodig:

21 Definitie
- *Een cykelgraaf of cykel C_n met $n \geq 3$ bogen is een graaf $C = (V, E)$ met $V = \{v_1, \ldots, v_n\}$ (waarbij alle v_i $1 \leq i \leq n$ verschillend zijn) en $E = \{\{v_i, v_{i+1}\} | 1 \leq i < n\} \cup \{\{v_1, v_n\}\}$.*
- *Een graaf die geen cykels bevat (geen cykelgrafen als deelgrafen heeft) noemen wij acyclisch – een graaf die wel cykels als deelgrafen heeft noemen wij cyclisch.*
- *Een boom is een samenhangende acyclische graaf.*
- *Een opspannende boom van een samenhangende graaf $G = (V, E)$ is een deelgraaf $T = (V, E')$ die een boom is. Let op: hier wordt geeist dat de verzamelingen van toppen dezelfden zijn!*

22 Oefening *Bewijs dat een graaf dan en slechts dan een cykel is als hij 2-regulier en samenhangend is.*

23 Opmerking *Als $G = (V, E)$ een boom is en $v \in V$, $w \notin V$, dan is $G' = (V \cup \{w\}, E \cup \{\{v, w\}\})$ ook een boom.*

Bewijs: Dat G' acyclisch is, volgt omdat in een cykel elke top graad 2 heeft. Als een cykel een deelgraaf is, is elke graad van een top op de cykel dus ten minste 2. Maar de nieuwe top heeft maar graad 1, de nieuwe boog ligt dus zeker niet op een cykel. Als G' een cykel bevatte dan zou die dus al in G aanwezig geweest zijn.

Dat G' samenhangend is volgt omdat er maar één equivalentieklasse van $\equiv_{sam}$ in G is – en omdat in G' de toppen v en w equivalent zijn levert de transitiviteit van $\equiv_{sam}$ het gewenste resultaat.

■

24 Oefening *Bewijs dat je elke boom op de net beschreven manier uit een enkele top kan opbouwen – of precies:*
Als G een boom met n toppen is, dan bestaat er een reeks $K_1 = G_1, G_2, \ldots, G_n = G$ van bomen zodat voor $1 \leq i \leq n$ de graaf G_i een boom met precies n toppen is, voor $1 \leq i < n$ is G_i een deelgraaf van G_{i+1} en voor $1 \leq i < n$ heeft de (unieke) top die in G_{i+1} zit maar niet in G_i graad 1 in G_{i+1}.

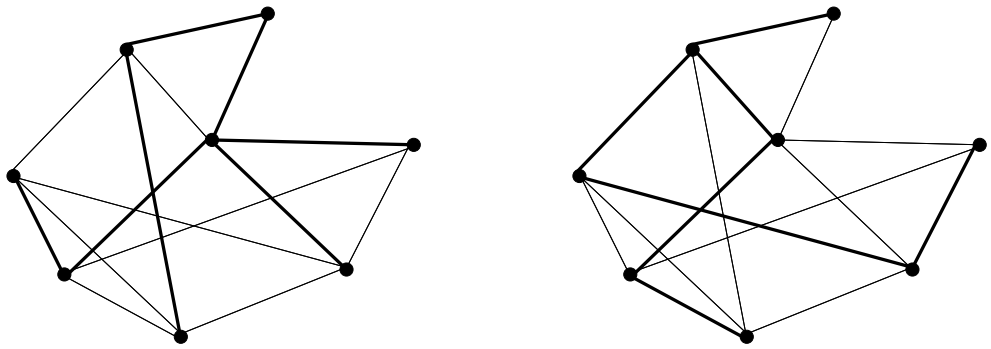
25 Corollarium *Gegeven een samenhangende graaf $G = (V, E)$, een starttop s en een toepassing van DFS resp. BFS op G. Stel dat E_{DFS} resp. E_{BFS} voor de bogen $\{v, w\}$ staat die gebruikt worden als w vanuit v gemarkeerd wordt. Dan zijn (V, E_{DFS}) resp. (V, E_{BFS}) opspannende bomen van G en wij noemen deze bomen soms een DFS-boom resp. een BFS-boom.*

26 Oefening *Gegeven een graaf met een DFS-nummering. Kan je alleen uit deze nummering ook de bogen van de opspannende DFS-boom bepalen? Of anders geformuleerd: kan je als je een DFS-nummering hebt voor een top alleen door naar zijn buren te kijken bepalen vanuit welke buur hij gemarkeerd werd?*

27 Oefening

- *Geef een noodzakelijke eigenschap van een opspannende boom om een DFS-boom te zijn waaraan andere opspannende bomen niet noodzakelijk voldoen.*
- *Geef een noodzakelijke eigenschap van een opspannende boom om een BFS-boom te zijn waaraan andere opspannende bomen niet noodzakelijk voldoen.*

- *Schrijf voor elk van de twee bomen in Figuur 6 of het een DFS of BFS boom kan zijn.*
  - *In het geval van niet: schrijf waarom niet*
  - *In het geval van ja: geef de DFS-nummering (als het een DFS-boom is) resp. de afstanden van de wortel van de boom.*



Figuur 6:

28 Oefening *Bewijs dat in een boom voor elke twee toppen v, w* **precies** *één pad van v naar w bestaat.*

Maar natuurlijk is alleen het feit dat een netwerk samenhangend is nog niet voldoende om het een goed netwerk te maken. Als je bv. een wegennet hebt waar één enkel rondpunt dat niet meer gebruikt kan worden omdat er werken zijn, ervoor zorgt dat het wegennet niet meer samenhangend is dan is dat zeker een slecht wegennet. En hetzelfde geldt natuurlijk voor een computernetwerk waar een enkele kapotte computer ervoor kan zorgen dat sommige computers niet meer bereikbaar zijn. Daarom bestaan ook definities die een sterkere samenhang definiëren:

29 Definitie *Gegeven een graaf $G = (V, E)$*

- *Een verzameling $M \subset V$ heet splitsend als $G[V \setminus M]$ meer samenhangscomponenten heeft dan G. Als er maar één enkele top in een splitsende toppenverzameling zit dan noemen wij de top zoals in het Engels een cutvertex of een splitstop.*
- *G heet k-samenhangend als G samenhangend is, $|V| > k$ en er bestaat geen splitsende verzameling $M \subset V$ met $|M| < k$.*

- *Het samenhangsgetal $\kappa(G)$ is gedefinieerd als*
  $\kappa(G) = \max\{k | G \text{ is } k\text{-samenhangend of } k = 0\}$

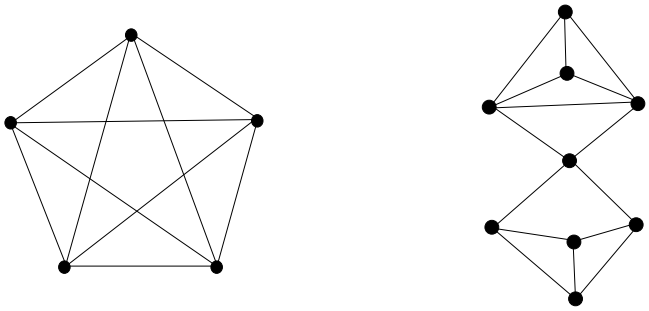
- *Een verzameling $M \subset E$ heet splitsend als $(V, E \setminus M)$ meer samenhangscomponenten heeft dan G. Als er maar één enkele boog in een splitsende boogverzameling zit dan noemen wij de boog een brug.*

- *G heet k-boogsamenhangend als G samenhangend is, $|V| > 1$ en er bestaat geen splitsende verzameling $M \subset E$ met $|M| < k$.*

- *Het boogsamenhangsgetal $\lambda(G)$ is gedefinieerd als*
  $\lambda(G) = \max\{k | G \text{ is } k\text{-boogsamenhangend of } k = 0\}$

Daarbij moet je opletten – als uit de context duidelijk is dat je met boogsamenhang bezig bent dan wordt vaak het begrip k-samenhangend ook voor k-boogsamenhangend gebruikt...

30 Oefening *Bepaal $\kappa()$ en $\lambda()$ voor de volgende twee grafen.*



31 Lemma *Gegeven een k-samenhangende graaf $G = (V, E)$ en k verschillende toppen $v_1, \ldots, v_k \in V$ en een top $w \notin V$.*
Dan is $G' = (V', E')$ met $V' = V \cup \{w\}$ en $E' = E \cup \{\{w, v_1\}, \ldots, \{w, v_k\}\}$ k-samenhangend.

Bewijs: G' is duidelijk samenhangend, het is dus voldoende te bewijzen dat als $M \subset V'$ ten hoogste $k-1$ elementen bevat, $G'[V' \setminus M]$ samenhangend is. Omdat G k-samenhangend is, is $G[V \setminus M] = G'[V' \setminus (M \cup \{w\})]$ zeker samenhangend. Als $w \in G'[V' \setminus M]$ dan is er ook nog ten minste één buur $b \in G'[V' \setminus M]$ (omdat er totaal k buren zijn) en $w \equiv_{sam} b$. Dus behoort w tot dezelfde component als b en dus de rest van $V' \setminus M$. $G'[V' \setminus M]$ is dus samenhangend.

■

32 Oefening
- *Toon aan dat als $G = (V, E)$ een samenhangende graaf en M een minimale splitsende boogverzameling is, dat dan $(V, E \setminus M)$ precies twee componenten heeft.*
- *Geldt iets analoogs ook voor splitsende toppenverzamelingen?*

33 Oefening *Gegeven $G = (V, E)$. Toon aan dat voor elke top $v \in V$ en elke boog $e \in E$ geldt dat*
$\kappa(G[V \setminus \{v\}]) \geq \kappa(G) - 1$ en $\kappa((V, E \setminus \{e\})) \geq \kappa(G) - 1$.
en bewijs daarmee dat voor elke verzameling $S \subseteq E$ geldt dat
$\kappa((V, E \setminus S)) + \lambda(G) \geq \kappa(G)$

Er bestaan nog veel meer definities die proberen een goede maat voor de samenhang te vinden. Bv. de cyclische samenhang waar geeist wordt dat in elk deel ten minste één cykel zit of de expander constante die rekening houdt met de grootte van de delen als er een splitsende verzameling is terwijl het voor de toppen- en boog-samenhang om het even is of er door een splitsende verzameling maar één enkele top wordt afgesplitst of misschien de helft van alle toppen. Maar hier zullen wij het alleen over de meest bekende invarianten hebben.

34 Oefening *Gegeven een graaf $G = (V, E)$.*
Toon aan: $\kappa(G) \leq \lambda(G) \leq \delta(G)$

In een samenhangende graaf zijn de cutvertices en de bruggen dus zoiets als *zwakke schakels* en daarom is het natuurlijk bijzonder belangrijk dergelijke zwakke schakels efficiënt te kunnen vinden.
Wij zullen nu een algoritme zien dat de cutvertices in een graaf in lineaire tijd kan vinden.

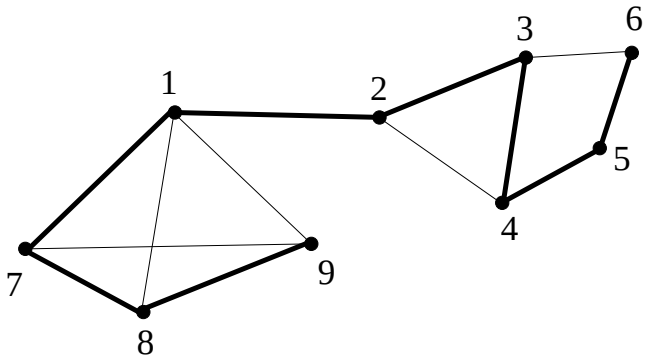
35 Definitie *Gegeven $G = (V, E)$.*
Een samenhangende deelgraaf $B \subseteq G$ heet een blok van G als B geen cutvertices bevat maar voor elke samenhangende graaf S met $B \subsetneq S \subseteq G$ geldt dat S een cutvertex bevat.

36 Oefening *Gegeven $G = (V, E)$. Toon aan:*

- *Een blok van G is altijd een geïnduceerde deelgraaf.*
- *De relatie $a \equiv b$ als $a, b \in V$ en a en b tot hetzelfde blok behoren is geen equivalentierelatie op V.*
- *De relatie "$e \equiv f$ als $e, f \in E$ en e en f tot hetzelfde blok behoren" is een equivalentierelatie op E.*

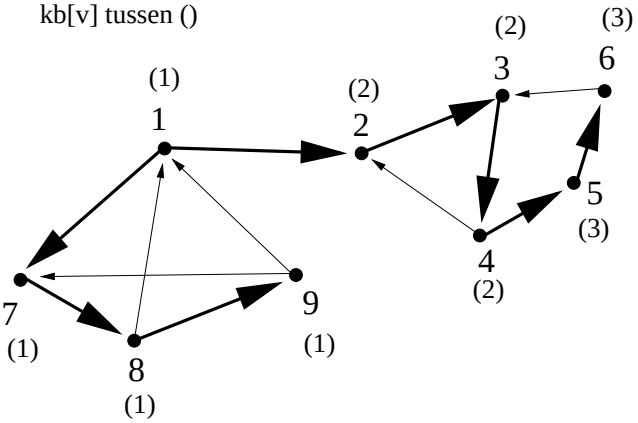
- *Bevatten alle equivalentieklassen van „≡“ op de verzameling van bogen (geïnterpreteerd als grafen) cykels?*
- *Beschrijf hoe sneden tussen blokken er kunnen uitzien.*

Kijk eerst naar Figuur 7. Als je langs de cutvertex gaat dan bezoekt DFS eerst alle toppen in het andere blok voordat het zoeken backtrackt en opnieuw toppen uit de vroegere component bezoekt. Dat betekent dat **alle** DFS-nummers die in dit deel gegeven worden groter zijn dan die van de cutvertex! Maar inderdaad is er ook geen boog naar een al bezochte top met een kleiner DFS-nummer. Deze observatie is de basis van het algoritme van Hopcroft en Tarjan.



Figuur 7:

Eén manier om het algoritme te beschrijven is als volgt:
Elke top v in de DFS-boom is de wortel van een deelboom $T(v)$ bestaande uit alle toppen die bezocht worden voordat de recursie van de top v backtrackt. Stel dat de DFS-boom met de DFS-nummering al is opgebouwd. Nu geven wij richtingen aan de bogen. Als de boog deel uitmaakt van de DFS-boom krijgt hij de richting van de top met het kleinere DFS-nummer naar het grotere en als de boog niet in de boom zit van het grotere naar het kleinere – dat noemen wij dan een terugboog. Nu noemen wij een top w *bereikbaar* vanuit v als er een gericht pad van v naar w is dat alleen bogen van de boom gebruikt – behalve misschien één enkele terugboog op het einde. Wij hebben niet expliciet gedefinieerd wat een *gericht pad* is maar dat is hopelijk duidelijk. Nu definiëren wij voor elke top v de waarde kb[v] als het DFS-nummer van de kleinste bereikbare top. In Figuur 8 zie je de gerichte graaf waarover wij het hebben, de DFS-nummers en de waarden van $kb()$.



Figuur 8:

37 Stelling *Gegeven een graaf $G = (V, E)$ met een DFS-boom B, een DFS-nummering DFS[] en waarden van kb[].*
Dan geldt:

- *De top v met $DFS[v] = 1$ is een cutvertex als en slechts als de graad van v in B groter dan 1 is.*
- *Een top v met $DFS[v] > 1$ is een cutvertex als en slechts als hij een kind w in B heeft met $kb[w] \geq DFS[v]$*

Wij zullen deze stelling natuurlijk nog bewijzen maar eerst zullen wij erover nadenken hoe je kb[] op de meest efficiënte manier kan berekenen. Daarvoor moeten wij gewoon sommige lijnen tot de gewone routine toevoegen die de DFS-nummers berekent. Als wij de recursieve functie voor een top v oproepen, zetten wij kb[v] eerst op zijn DFS-nummer. Elke keer dat een kind w afgewerkt is, wordt kb[v] met de waarde van het kind vergeleken en elke keer dat een boog naar een al bezochte top b gezien wordt met het DFS-nummer van b:

38 Algoritme <u>Cutvertices vinden</u>
Gegeven een graaf $G = (V, E)$ en een arbitraire starttop $s \in V$
Alle toppen van G hebben in het begin het DFS nummer ∞.
De variabele nextlabel *is globaal en in het begin is nextlabel*$= 1$.
Dan roept de volgende functie DFS_cut *met parameters s, s op (waarbij in de eerste stap s natuurlijk niet echt zijn eigen ouder is...):*

DFS_cut(v,ouder) {

DFS$[v] = kb[v] = nextlabel$

nextlabel=*nextlabel*+1
voor alle buren b van v do:

- *als* $DFS[b] = \infty$*: nog niet bezocht – achteraf kb-waarde vergelijken*
  - *DFS_CUT*(b, v)
  - *als* $(kb[b] < kb[v])$ *dan* $kb[v] = kb[b]$
  - *als* $(kb[b] \geq DFS[v]$ *en* $v \neq s)$ *dan is v een cutvertex*
- *anders: al bezocht – DFS-nummer vergelijken als het niet de ouder is*
  - *als* $(b \neq$ *ouder en* $DFS[b] < kb[v])$ *dan* $kb[v] = DFS[b]$

}
Nadat de recursie gedaan is, moet s alleen nog als een cutvertex gemarkeerd worden als de graad groter dan 1 *is.*

het is duidelijk dat dit maar een kleine wijziging van het gewone DFS is en het algoritme natuurlijk nog altijd $O(|V| + |E|)$ tijd vraagt.

39 Oefening *Maak jezelf duidelijk en schrijf op waarom dit algoritme inderdaad de waarde van kb*[] *juist berekent.*

Maar nu moeten wij nog bewijzen dat Stelling 37 ook inderdaad juist is:

Bewijs: (Stelling 37)

Stel dat G ten minste 2 toppen heeft (anders is het triviaal).

Eerst zullen wij naar de wortel van de DFS-boom kijken dus de top v met DFS-nummer 1.

Stel dat v een cutvertex is. Dan begint de recursie in één van de blokken maar omdat geen pad naar een ander blok leidt zonder de top v te gebruiken zal de recursie naar v terugkomen en van daar zal dan een nieuw blok bezocht worden – waardoor de graad van v in de DFS-boom al ten minste twee is.

Een eigenschap van DFS is dat als de recursie voor een top v wordt opgeroepen het markeren precies zo doorgaat alsof het opgestart was op de graaf waarin alle al genummerde toppen verwijderd zijn (voor BFS geldt dat niet!). Als v geen cutvertex is, is $G[V - \{v\}]$ samenhangend

en dat betekent dat alle toppen in $G[V - \{v\}]$ gemarkeerd zijn voordat de recursie van de eerste buur van v terugkomt. Maar dan heeft v in de boom graad 1.

Nu kijken wij naar een top v die niet de wortel is.

Als v een kind b heeft met $\text{kb}[b] \geq \text{DFS}[v]$ dan betekent dat dat er geen pad van b naar een top t met $\text{DFS}[t] < \text{DFS}[v]$ bestaat dat niet de top v bevat (zie Oefening 40). Omdat v niet de wortel is – dus $\text{DFS}[v] \geq 2$ – bestaat zo'n top t. Maar dan is $G[V - \{v\}]$ niet samenhangend, dus v een cutvertex.

Stel nu dat v een cutvertex is en b een kind in een componente C van $G[V - \{v\}]$ die niet de wortel bevat en dat vanuit v gemarkeerd wordt. Dan zijn alle toppen in C nog ongemarkeerd op het moment dat de recursie voor b opgestart wordt en krijgen dus een groter DFS-nummer. Maar er kan geen boog naar een al vroeger bezochte top zijn (behalve naar v) omdat er anders een pad naar deze top zou zijn dat v niet bevat. Dus is er ook geen terugboog naar een top met een kleiner DFS-nummer. Dus is $\text{kb}[b] \geq \text{DFS}[v]$.

■

40 Oefening *In het vorige bewijs staat* Als v een kind b heeft met $\text{kb}[b] \geq \text{DFS}[v]$ dan betekent dat dat er geen pad van b naar een top t met $\text{DFS}[t] < \text{DFS}[v]$ bestaat dat niet de top v bevat.
Maar de definitie van bereikbaar *gebruikt alleen heel speciale paden. Het is dus duidelijk dat er geen gericht pad met op de laatste na alleen bogen uit de boom bestaat, maar waarom bestaat er ook geen ander pad dat de top v niet bevat?*

41 Oefening a.) *Wij hebben in Oefening 36 gezien dat* tot hetzelfde blok behoren *een equivalentierelatie op de bogen is. Modificeer Algoritme 38 zodat nummers aan de bogen worden toegekend en bogen hetzelfde nummer krijgen als en slechts als ze equivalent zijn. Tip: gebruik een stapel voor de bogen en denk erover na wanneer je er een boog moet plaatsen en wanneer je bogen van de stapel moet halen.*

b.) *Beschrijf een lineair algoritme om bruggen te vinden dat gebaseerd is op het algoritme voor cutvertices. Je kan ofwel a.) gebruiken ofwel de graaf zo veranderen dat zekere toppen bruggen voorstellen.*

Er is ook een lineair algoritme (ook van Hopcroft en Tarjan) om de 3-samenhangende componenten van een graaf te vinden, maar dat is al duidelijk

ingewikkelder. Er is geen lineair algoritme gekend dat het samenhangsgetal (of boogsamenhangsgetal) in het algemeen in lineaire tijd kan bepalen.
Natuurlijk is er in principe voor elke k een gemakkelijk algoritme om te testen of de graaf k-samenhangend (resp. k-boogsamenhangend) is: verwijder elke top (boog) één keer en kijk of de graaf nog altijd $(k-1)$-samenhangend ($(k-1)$-boogsamenhangend) is. Voor bv. $k = 3$ is dat ook nog relatief efficiënt maar voor grote k is dat natuurlijk veel te duur.

42 Oefening

- *Als het boogsamenhangsgetal – en dus ook de kleinste graad – in vergelijking met de grootte van de graaf heel groot is dan wordt het opnieuw gemakkelijk. Toon aan:*

  Als $\delta(G) \geq |V|/2$ dan is $\lambda(G) = \delta(G)$.

  geef een voorbeeld dat bewijst dat de grens optimaal is.

- *Bepaal een benedengrens voor $\delta(G)$ zodat $\kappa(G) = \delta(G)$. Bewijs dat jouw grens optimaal is. Wees niet verrast als de grens duidelijk slechter is dan voor λ.*

Om in het algemeen de waarden van κ en λ te berekenen bestaan speciale algoritmen waarvan de meeste gebaseerd zijn op algoritmen om een *maximale stroom* (maximum flow) te bepalen. Deze algoritmen hangen ook samen met een heel belangrijke stelling uit de grafentheorie die wij ook aan de hand van stromen zullen bewijzen:

43 Stelling *(Menger)*
Gegeven een graaf $G = (V, E)$ en twee toppen $u, v \in V$. Dan is de minimale kardinaliteit $s(u, v)$ van een splitsende boogverzameling waarvoor u en v in verschillende componenten zitten gelijk aan het maximale aantal paden van u naar v die paarsgewijs geen boog gemeen hebben.

En met deze stelling heb je onmiddellijk:

44 Corollarium *Een graaf $G = (V, E)$ is k-boogsamenhangend als en slechts als voor elk paar van verschillende toppen u, v er k paden van u naar v bestaan die paarsgewijs geen boog gemeen hebben.*

Er bestaat ook een analoge stelling voor toppensamenhang:

45 Stelling *(Menger)*
Gegeven een graaf $G = (V, E)$ en twee toppen $u, v \in V$ zodat $\{u, v\} \notin E$. Dan is de minimale kardinaliteit $s(u, v)$ van een splitsende toppenverzameling waarvoor u en v in verschillende componenten zitten gelijk aan het maximale

aantal paden van u naar v die paarsgewijs geen top gemeen hebben behalve de eindpunten u en v.

En met deze stelling heb je bijna onmiddellijk:

46 Corollarium *(Whitney/Menger)*
Een graaf $G = (V, E)$ is k-samenhangend als en slechts als voor elk paar van verschillende toppen u, v er k paden van u naar v bestaan die paarsgewijs geen top gemeen hebben behalve de eindpunten.

Bewijs: Het geval waar $\{u, v\} \notin E$ is gewoon Stelling 45. Wij moeten het dus alleen nog over het geval $\{u, v\} \in E$ hebben.

Stel dus dat G k-samenhangend is en er twee toppen u, v met $\{u, v\} \in E$ gegeven zijn. Wij moeten aantonen dat er behalve het pad dat maar de ene boog bevat nog $k - 1$ andere paden van u naar v zijn die paarsgewijs geen top gemeen hebben behalve de eindpunten. Maar dat volgt onmiddellijk met Stelling 45 omdat de graaf $(V, E \setminus \{\{u, v\}\})$ ten minste $k - 1$ samenhangend is en in deze graaf u en v geen buren zijn.

■

47 Oefening *Toon aan: Als G een 2-samenhangende graaf is dan bestaat er voor elk paar van bogen $e_1 \neq e_2$ een cykel die e_1 en e_2 bevat.*

Om de stellingen te bewijzen zullen wij een algoritmische aanpak kiezen en stromen gebruiken maar dat kan natuurlijk ook direct.

48 Oefening *Toon aan:*
Een graaf G met ten minste $2k$ toppen is k-samenhangend als en slechts als voor elk disjunct paar van verzamelingen $M_1, M_2 \subset V$ met $|M_1| = |M_2| = k$ geldt dat er k volledig disjuncte paden van M_1 naar M_2 zijn (dus waar ook de eindpunten disjunct zijn).

49 Oefening *Toon aan: In een k-samenhangende graaf G met $k \geq 2$ bestaat er voor elke verzameling van k toppen een cykel die al deze toppen bevat.*

3.2 Stromen (Flows)

50 Definitie

- *Voor een gerichte graaf $D = (V, E)$ en $W \subseteq V$ definiëren wij*

$$\begin{aligned} W^+ &:= \{e \in E | e = (x, y),\ x \in W,\ y \notin W\} \text{ en} \\ W^- &:= \{e \in E | e = (x, y),\ x \notin W,\ y \in W\} \end{aligned}$$

Voor $v \in V$ schrijven wij kort v^{+-} in plaats van $\{v\}^{+-}$.

- *Een netwerk is een 2-tal $N = (D, c)$ waarbij $D = (V, E)$ een gerichte graaf is en $c : E \to \mathbb{Z}(\mathbb{Q}, \mathbb{R}, \ldots)$ een capaciteitsfunctie die niet negatieve gehele (rationale, reële...) getallen aan de gerichte bogen toekent die wij als de capaciteiten van de bogen interpreteren.*

  Hoewel ∞ zeker geen getal is, laten wij voor één zekere boog (b, d) ook ∞ als capaciteit toe en later in Algoritme 75 zelfs voor 2 bogen. Alles wat wij zullen bewijzen, zal niet veranderen als je ∞ door een heel groot getal vervangt (bv. de som van alle eindige capaciteiten).

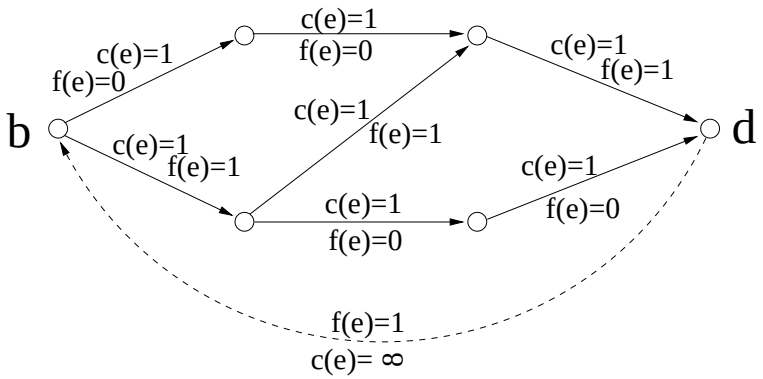
- *Een stroom (flow) op een netwerk $N = (D, c)$ is een functie $f : E \to \mathbb{Z}(\mathbb{Q}, \mathbb{R}, \ldots)$ met $0 \leq f(e) \leq c(e)\ \forall e \in E$ en voor alle toppen $v \in V$ geldt $\sum_{e \in v^-} f(e) - \sum_{e \in v^+} f(e) = 0$ (Wet van Kirchhoff).*
  Wij schrijven $f^{+(-)}(v)$ in plaats van $\sum_{e \in v^{+(-)}} f(e)$.
  Voor $W \subseteq V$ schrijven wij op een analoge manier $f^{+(-)}(W)$.

51 Opmerking *Wij zien onmiddellijk dat voor een stroom $f()$ en een toppenverzameling $W \subseteq V$ geldt:*

$$\begin{aligned} f^-(W) - f^+(W) &= \underbrace{\sum_{\substack{e=(x,y)\in E \\ x\notin W,\ y\in W}} f(e) - \sum_{\substack{e=(y,x)\in E \\ y\in W,\ x\notin W}} f(e)}_{\text{definitie}} + \underbrace{\sum_{\substack{e=(x,y)\in E \\ x,y\in W}} f(e) - \sum_{\substack{e=(y,x)\in E \\ x,y\in W}} f(e)}_{0} \\ &= \sum_{\substack{e=(x,y)\in E \\ y\in W}} f(e) - \sum_{\substack{e=(y,x)\in E \\ y\in W}} f(e) = \sum_{y\in W} \underbrace{(f^-(y) - f^+(y))}_{0} = 0 \end{aligned}$$

De vraag die wij in samenhang met dergelijke netwerken zullen onderzoeken is wat de maximale stroom *van één top naar een andere* kan zijn. Hoewel het in de realiteit waar je misschien een netwerk met een bron b en een doel d hebt een natuurlijke vraag lijkt is het antwoord in ons model triviaal

omdat er geen echte *bronnen* kunnen zijn: de stroom is altijd 0 omdat in een netwerk aan **elke** top de totale stroom 0 is. In ons model is het doel dus de stroom op een zekere gerichte boog (d, b) zo groot als mogelijk te hebben. Als wij in de realiteit een echt netwerk (bv. met buizen) met een bron b en een doel d hebben dan voegen wij tot het model van dit netwerk nog een boog (d, b) met heel grote capaciteit (wij schrijven ∞, maar bv. de som van alle andere capaciteiten is ook voldoende) toe en zoeken een stroom op dit netwerk waarvoor de waarde van $f((d, b))$ maximaal is. Als je deze boog dan niet beschouwt dan is deze waarde precies de stroom van b naar d in de rest van het netwerk. In Figuur 9 zien jullie een klein netwerk



Figuur 9: Een klein netwerkje meet een stroom.

Het is natuurlijk altijd gemakkelijk een stroom op een netwerk te vinden: de stroom die constant 0 is voor alle bogen is een geldige stroom. Een standaard aanpak zou dus zijn om met deze stroom te beginnen en dan op een gretige manier de stroom in stappen groter te maken.

52 Algoritme Een gretig stroomalgoritme

- *begin met een stroom die constant* 0 *is.*
- *herhaal de volgende stappen totdat er geen gericht pad meer is van* b *naar* d*:*
  - *Vind een gericht pad van* b *naar* d *(bv. met BFS of DFS). Samen met* (d, b) *is dat een gerichte cykel* C*.*
  - $m = \min\{c(e) - f(e) | e \in C\}$
  - *Voor alle* $e \in C$ *verhoog* $f(e)$ *met* m*.*

– *Verwijder alle bogen met $f(e) = c(e)$ omdat die niet meer tot een grotere stroom kunnen bijdragen.*

Het is duidelijk dat dit algoritme gaat stoppen (en inderdaad ook polynomiaal is) omdat in elke lus ten minste één boog verwijderd wordt. Maar vindt dit algoritme ook altijd een maximale stroom?
Inderdaad toont Figuur 9 al een tegenvoorbeeld. Het is mogelijk dat na de eerste keer dat de stroom groter wordt gemaakt de stroom die in deze figuur getoond wordt geconstrueerd is. Het is gemakkelijk om te zien dat het doel dan vanuit de bron niet meer bereikt kan worden. Maar het is even gemakkelijk om te zien dat deze stroom niet maximaal is maar dat een stroom met waarde 2 op (d, b) mogelijk is.
Maar inderdaad moeten wij dit algoritme maar een klein beetje wijzigen om een algoritme te vinden dat werkt – je moet alleen maar paden toelaten die niet noodzakelijk gericht zijn:

53 Definitie *Gegeven een netwerk $N = (D, c)$ en een stroom $f()$. Bovendien zij een pad P gegeven dat wel gerichte bogen gebruikt maar dat niet noodzakelijk altijd de richting van de bogen volgt: voor bogen (a, b) kan de top b ook voor a in het pad zitten. Dergelijke bogen noemen wij achteruit-bogen en de andere bogen vooruit-bogen. De verzameling van alle vooruit-bogen noemen wij P^+ en de verzameling van alle achteruit-bogen P^-*
Dan definiëren wij de vrije stroom $vs(P)$ als
$vs(P) = \min(\{c(e) - f(e) | e \in P^+\} \cup \{f(e) | e \in P^-\})$

Als P een pad is met $vs(P) > 0$ dan noemen wij P een grotermakend pad.

Als wij een grotermakend pad P van b naar d hebben gevonden (dat niet de boog (d, b) is) dan kunnen wij inderdaad een grotere stroom definiëren. Wij definiëren

$$f'(e) = \begin{cases} f(e) + \text{vs}(P) & \text{als } e \in P^+ \text{ of } e = (d, b) \\ f(e) - \text{vs}(P) & \text{als } e \in P^- \\ f(e) & \text{anders} \end{cases}$$

Het is gemakkelijk om te zien dat dat inderdaad een stroom is omdat de capaciteiten gerespecteerd worden en aan elke top de stroom ofwel niet gewijzigd wordt ofwel $\text{vs}(P)$ meer instroomt en $\text{vs}(P)$ meer uitstroomt.

54 Definitie *$N = (D, c)$ zij een netwerk en $W \subset V(D)$.*

Dan noemen wij

$$c(W) := \sum_{e \in W^+} c(e)$$

de capaciteit van de opdeling W, W^c.

55 Stelling (Ford & Fulkerson, 1956) *$N = (D, c)$ zij een netwerk met bron b en doel d en waarvoor $(d, b) \in E(D)$ met $c((d, b)) = \infty$.*
Dan is de maximale stroom op (d, b) gelijk aan de minimale waarde $c(W)$ voor een verzameling W met $b \in W$ en $d \notin W$.

Bewijs: De ene richting is gemakkelijk:

Stel dat f een maximale stroom is en $W \subset V(D)$ een arbitraire verzameling met $b \in W$, $d \in W^c$.

Dan geldt $f^-(W) - f^+(W) = 0$ dus

$$f((d,b)) + \sum_{\substack{e \in W^- \\ e \neq (d,b)}} f(e) = \sum_{e \in W^+} f(e)$$

$$\Rightarrow \quad f((d,b)) = \sum_{e \in W^+} f(e) - \underbrace{\sum_{\substack{e \in W^- \\ e \neq (d,b)}} f(e)}_{\geq 0} \leq \sum_{e \in W^+} c(e) = c(W).$$

Dus geldt voor elke dergelijke W dat $f((d, b)) \leq c(W)$ dus ook voor een W waarvoor $c(W)$ minimaal is.

Nu moeten wij de andere richting aantonen – dus dat $f((d, b)) \geq c(W)$. Dat zullen wij doen door een algoritme te construeren dat expliciet een stroom construeert met deze eigenschap.

Het algoritme van Ford-Fulkerson:

- Wij beginnen met een stroom die constant 0 is dus $f(e) = 0$ $\forall e \in E(D)$.
- Zoek een grotermakend pad P van b naar d (dat niet (b, d) is).
  - als zo'n pad niet bestaat: stop
  - anders: verhoog de stroom langs dit pad en (d, b) met $\text{vs}(P)$ en herhaal de lus.

Stel dat dit algoritme stopt. Definieer dan W als de verzameling van alle toppen die vanuit b door middel van een grotermakend pad bereikt kunnen worden. Dus $b \in W$ en $d \notin W$ (omdat het algoritme stopt). Dan geldt voor alle $e \in W^+$ dat $f(e) = c(e)$ en voor alle $e \in (W^- \setminus \{(d,b)\})$ dat $f(e) = 0$ omdat anders het andere eindpunt van de boog ook door een grotermakend pad bereikt zou kunnen worden. Dus

$$f((d,b)) + \underbrace{\sum_{e \in W^-, e \neq (d,b)} f(e)}_{0} = \sum_{e \in W^+} f(e) = \sum_{e \in W^+} c(e) = c(W)$$

Dus is $f((d,b))$ zeker ook groter dan of gelijk aan de minimale waarde van een $c(W)$.

Wij moeten dus *alleen nog* bewijzen dat het algoritme inderdaad stopt. Dat is gemakkelijk als de waarden van de capaciteiten in $\mathbb{Z}$ of $\mathbb{Q}$ zijn. Dan kan je de capaciteiten voorstellen als een veelvoud van een vast getal en omdat in elke lus de stroom ten minste om dit vast getal groter wordt en hij bovendien naar boven begrensd is, stopt het algoritme zeker. In het geval van $\mathbb{R}$ is dat niet zo gemakkelijk om te zien (en is het soms zelfs niet waar) maar daar zullen wij het later over hebben.

■

De constructie in dit bewijs levert nog een andere interessante opmerking.

56 Opmerking *Als de capaciteiten in $\mathbb{Z}$ of $\mathbb{Q}$ zijn dan zou je geen grotere stroom kunnen krijgen als je stroomwaarden uit $\mathbb{R}$ toelaat: er is dan altijd ook een maximale stroom waarbij de stroom op elke boog een waarde uit $\mathbb{Z}$ of $\mathbb{Q}$ is.*

Nu zullen wij deze stelling eerst gebruiken om Stelling 43 te bewijzen. Omdat wij daarvoor alleen gehele capaciteiten nodig hebben, is het daarvoor geen probleem dat het geval met reële capaciteiten nog niet afgewerkt is.

Bewijs: (Stelling 43)

Neem de graaf $G = (V, E)$ uit de stelling en bouw een netwerk door elke boog $\{x, y\}$ te vervangen door twee gerichte bogen (x, y) en (y, x) met capaciteit 1.

Stel dat $\{u, v\} \notin E$. (Voor het geval $\{u, v\} \in E$ moet het bewijs maar een klein beetje gewijzigd worden...)

Dan voegen wij nog een boog (v, u) met capaciteit ∞ toe en definiëren u als bron en v als doel. Dan correspondeert een splitsende boogverzameling met een opdeling W, W^c waarbij de bron in W en het doel in W^c ligt – en de splitsende boogverzameling en de capaciteit $c(W)$ zijn even groot.

Maar het aantal paden die geen gemeenschappelijke boog hebben, correspondeert met een maximale stroom: wij weten dat er een maximale stroom bestaat waar alle waarden uit $\mathbb{Z}$ zijn – dus in dit geval 0 of 1. Maar wij mogen ook stellen dat er voor geen boog $\{x, y\} \in E(G)$ geldt dat $f((x, y)) = f((y, x)) = 1$ – of algemener dat er geen gerichte cykels met stroomwaarde 1 zijn. Als dat wel het geval was, kunnen wij voor alle deze bogen (x, y) definiëren $f((x, y)) = 0$ en wij hebben opnieuw een maximale stroom. Maar als wij nu gerichte paden met stroom 1 van u naar v volgen dan kunnen wij op deze manier (bv. door elke keer de bogen die we gevolgd hebben te verwijderen als wij een pad hebben gevonden) even veel paden van u naar v vinden die paarsgewijs geen boog gemeenschappelijk hebben als de stroom groot is.

Stelling 43 volgt dus direct uit Stelling 55.

■

Hebben wij hier Opmerking 56 echt nodig gehad?

57 Oefening *Bewijs Stelling 43 voor het geval dat* $\{u, v\} \in E$.

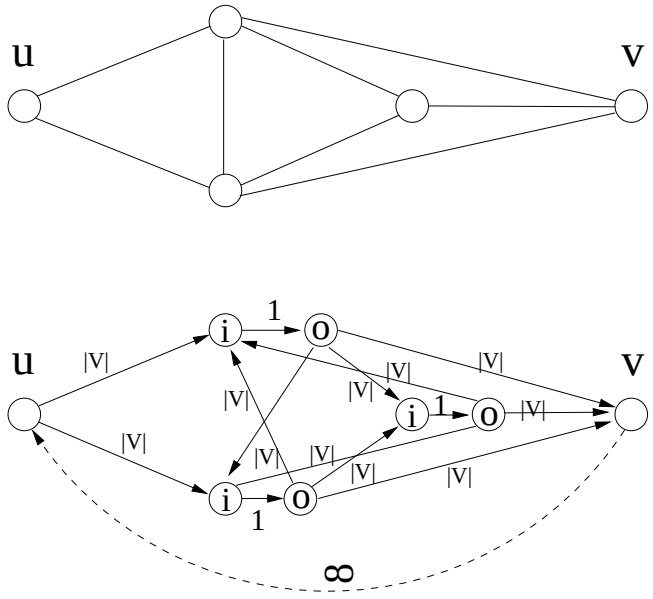
Bewijs: (Stelling 45)

Stelling 45 volgt op een analoge manier uit Stelling 55 – alleen dat hier de graaf G op een beetje een andere manier naar een netwerk N moet vertaald worden.

Wij definiëren opnieuw u als bron en v als doel en voegen de gerichte boog (v, u) met capaciteit ∞ toe.

Maar voor elke top $x \in V \setminus \{u, v\}$ gebruiken wij nu twee toppen: x_{in} en x_{out} en voegen wij een boog (x_{in}, x_{out}) toe met capaciteit 1. Voor elke boog $\{x, y\}$ met $\{u, v\} \cap \{x, y\} = \emptyset$ voegen wij nu de bogen (x_{out}, y_{in}) en (y_{out}, x_{in}) toe, voor elke boog $\{u, x\}$ de boog (u, x_{in}) en voor elke boog $\{v, x\}$ de boog (x_{out}, v). Deze bogen hebben allemaal een heel grote capaciteit – bv $|V|$. Een voorbeeld van deze vertaling zien jullie in Figuur 10.

Elk pad $u, x^1, x^2, \ldots, x^k, v$ in G correspondeert met een gericht pad $u, x^1_{in}, x^1_{out}, x^2_{in}, x^2_{out}, \ldots, x^k_{in}, x^k_{out}, v$ in N en als twee paden in G geen



Figuur 10:

gemeenschappelijke toppen hebben behalve de eindpunten dan geldt hetzelfde voor de paden in N. Als wij l paden van u naar v hebben die paarsgewijs toppendisjunct zijn (dat betekent: geen toppen gemeen hebben behalve de eindtoppen) dan kunnen wij een stroom van 1 langs het corresponderende pad (en terug langs (v, u)) sturen en hebben dus een stroom met waarde ten minste l.

Bovendien zijn alle gerichte paden van u naar v in N van de vorm dat elke tweede boog van de vorm (x_{in}, x_{out}) is omdat er geen andere mogelijkheid is een top x_{in} te verlaten of een top x_{out} binnen te komen. Als wij dus een stroom met grootte l hebben dan kunnen wij zoals in het vorige bewijs l gerichte paden vinden die geen bogen gemeen hebben en volgens onze constructie kunnen zij dan ook geen toppen gemeen hebben behalve de eindtoppen. In G hebben wij dus l corresponderende toppendisjuncte paden van u naar v. Het aantal toppendisjuncte paden van u naar v in G is dus gelijk aan de maximale stroom in N.

Als nu $\{x^1, \ldots, x^l\}$ een minimale u en v splitsende toppenverzameling is en wij definiëren W als de verzameling van alle toppen die je in N vanuit u langs een gericht pad kan bereiken zonder één van de bogen (x^i_{in}, x^i_{out}) te gebruiken, dan is $v \notin W$ en $c(W) = l$.

Kijk nu naar een W met $u \in W$, $v \in W^c$ zodat $c(W)$ minimaal is. Als

wij naar de bogen in W^+ kijken dan is duidelijk dat die allemaal van de vorm (x_{in}, x_{out}) zijn: als je u samen met zijn buren als W neemt, heb je al een cut van deze vorm – die bestaat dus – en elke W waarvoor ook maar een enkele boog in W^+ niet van deze vorm is, heeft al een grotere capaciteit dan deze cut. Maar als je in G alle toppen x verwijdert met $(x_{in}, x_{out}) \in W^+$ dan heb je een toppenverzameling met grootte $c(W)$ die u en v splitst.

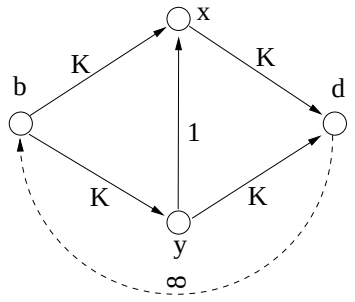
Samen geeft dat dat een minimale u van v splitsende toppenverzameling in G even groot is als de minimale capaciteit $c(W)$ van een verzameling W in N met $u \in W$, $v \notin W$.

Nu volgt Stelling 45 uit Stelling 55.

■

58 Oefening *Formuleer en bewijs een stelling voor gerichte grafen die op één van de stellingen van Menger voor ongerichte grafen lijkt.*

Maar er is niet alleen het probleem dat wij niet weten of het algoritme zou stoppen voor capaciteiten uit $\mathbb{R}$ – zelfs als de capaciteiten uit $\mathbb{Z}$ zijn is er een probleem met de complexiteit. Kijk eens naar Figuur 11.



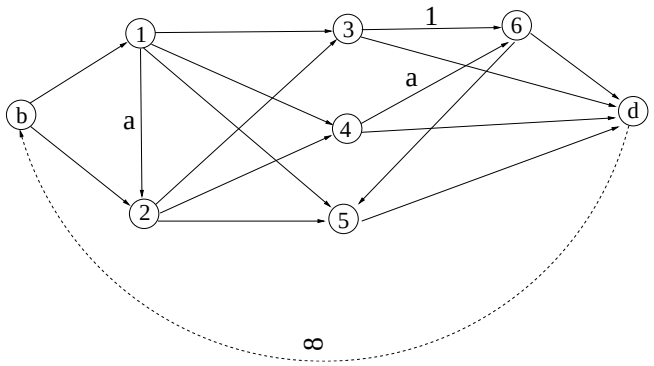
Figuur 11:

Als je hier afwisselend grotermakende paden b, y, x, d en b, x, y, d neemt dan heb je $2K$ iteraties nodig – om het even hoe groot K is – en dat hoewel het een echt klein netwerkje is. Een standaard manier van doen zou zijn grotermakende paden op een DFS-manier te zoeken, maar dan zou inderdaad deze volgorde van grotermakende paden gebruikt kunnen worden en dat is natuurlijk niet goed.

59 Oefening *De invoer voor een algoritme dat een maximale stroom op een netwerk met gehele capaciteiten berekent, is een string die het netwerk beschrijft*

door een lijst van de gerichte bogen te geven, samen met de capaciteiten. Er worden 3 getallen, gescheiden door komma's voor elke boog gebruikt – bv $83, 17, 22$ *voor een boog* $83 \to 17$ *met capaciteit* 22*. De getallen worden in de decimale voorstelling geschreven, de string* $83, 17, 22$ *heeft dus lengte 8 (byte). De codes voor twee gerichte bogen worden door een puntkomma gescheiden.*
Wat is de complexiteit als functie van de invoerlengte als je op de string een algoritme toepast dat het Ford-Fulkerson algoritme gebruikt waarbij je grotermakende paden met DFS zoekt? Een heel grove bovengrens die de $O()$*-notatie gebruikt, is voldoende.*

Als je reële capaciteiten hebt, ziet het er nog erger uit. Het voorbeeld in Figuur 12 is van D.P. Bertsekas:



Figuur 12: Alle capaciteiten die er niet expliciet staan zijn $1 + a$ waarbij a voor $(\sqrt{5} - 1)/2$ staat.

Het is gemakkelijk om te zien dat de maximale stroom $2(1 + a)$ is.
Wij zullen nu het Ford-Fulkerson algoritme toepassen door de stroom die in het begin op elke boog 0 is groter te maken door gebruik te maken van een oneindige herhaling van 6 elkaar opvolgende grotermakende paden:

iteratie $6k + 1$**:** gebruik pad $b, 1, 2, 3, 6, d$ (groei a^{3k+1})

iteratie $6k + 2$**:** gebruik pad $b, 2, 1, 3, 6, 5, d$ (groei a^{3k+2})

iteratie $6k + 3$**:** gebruik pad $b, 1, 2, 4, 6, d$ (groei a^{3k+2})

iteratie $6k + 4$**:** gebruik pad $b, 2, 1, 4, 6, 3, d$ (groei a^{3k+3})

iteratie $6k + 5$**:** gebruik pad $b, 1, 2, 5, 6, d$ (groei a^{3k+3})

iteratie $6k+6$**:** gebruik pad $b, 2, 1, 5, 6, 4, d$ (groei a^{3k+4})

60 Oefening *Voor deze oefening is het nuttig te gebruiken dat voor* $a = (\sqrt{5}-1)/2$ *geldt dat* $a^2 = 1-a$ *en dus* $a^{k+2} = a^k - a^{k+1}$.

- *Werk de eerste (bv. 6) iteraties uit en houdt de stromen op de bogen bij.*
- *Als je deze rij van paden gebruikt convergeert de stroom op* (d, b) *naar een zekere waarde. Ook al stopt het algoritme niet: is deze waarde tenminste de maximale waarde?*
- *Gegeven een arbitrair netwerk. Is er als het algoritme met DFS niet stopt (en de waarde van de stroom dus naar een waarde* x *convergeert) ten minste een bovengrens voor de verhouding* m/x *waarbij* m *de maximale stroom is?*

Het is dus om meerdere redenen belangrijk een grotermakend pad voor het Ford-Fulkerson algoritme op een *goede* manier te kiezen.

61 Algoritme Grotermakende paden met BFS
Gegeven een netwerk N *en een stroom* $f()$.
Een boog (a, b) *heet* vrij vanuit a *als* $f((a, b)) < c((a, b))$. *Een boog* (b, a) *heet* vrij vanuit a *als* $f((b, a)) > 0$.
L zij een lijst van toppen die in het begin alleen maar de bron b *bevat.*
Alle toppen van G *– behalve* b *– zijn in het begin ongemarkeerd.*
De starttop b *is gemarkeerd.*
Nu herhaal het volgende totdat de lijst leeg is of d *gemarkeerd is:*

- *verwijder het eerste element* l *uit de lijst*
- *voeg alle buren van* l *die langs een vrije boog bereikbaar zijn en die nog niet gemarkeerd zijn aan het einde van de lijst toe en markeer ze.*

Als d *op het einde gemarkeerd is, is er een grotermakend pad gevonden – en inderdaad is dat zelfs een kortste grotermakend pad. Als* d *op het einde niet gemarkeerd is, is er geen grotermakend pad meer en de stroom is maximaal.*

Om te bewijzen dat het algoritme met deze manier grotermakende paden te construeren altijd stopt – en inderdaad polynomiaal begrensd is door een functie die alleen van de gerichte graaf afhankelijk is en niet van de capaciteiten – hebben wij eerst een definitie nodig:

62 Definitie *Gegeven een netwerk $N = ((V, E), c)$ waarop het Ford-Fulkerson algoritme met BFS wordt toegepast en toppen $u, v \in V$.*
Eén iteratie zij het zoeken van een grotermakend pad en het groter maken van de stroom.
Bovendien zij k een getal zodat het algoritme ten minste k iteraties uitvoert voordat het stopt.
Dan staat $d_k(u, v)$ voor de lengte van een kortste grotermakend pad van u naar v in het netwerk na k iteraties.
Om niet altijd rekening te moeten houden met het aantal iteraties dat het algoritme doet, definiëren wij voor getallen k met $k > k_{max}$ (waarbij k_{max} het maximale aantal uitgevoerde iteraties is) $d_k(u, v) = d_{k_{max}}(u, v)$.

63 Lemma *Gegeven een netwerk $N = ((V, E), c)$ waarop het Ford-Fulkerson algoritme met BFS wordt toegepast en $u \in V$.*
Dan geldt voor alle $k \geq 0$: $d_{k+1}(b, u) \geq d_k(b, u)$.

Bewijs: Stel dat $v_0 = b, v_1, \ldots v_n = u$ een kortste (beginnend) grotermakend pad van b naar een top u is na $k + 1$ iteraties.

Wij zullen voor alle toppen v_i op dit pad met inductie bewijzen dat $d_{k+1}(b, v_i) \geq d_k(b, v_i)$.

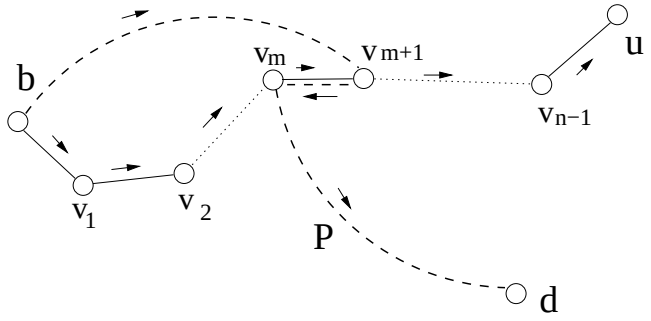
Voor v_0 – dat is b zelf – is dat duidelijk, stel dus dat het bewezen is voor $v_0, \ldots, v_m$.

Wij kijken nu naar de boog $\{v_m, v_{m+1}\}$. Wij schrijven de boog op een ongerichte manier omdat het hier niet echt belangrijk is of het (v_m, v_{m+1}) of (v_{m+1}, v_m) is. Wat belangrijk is, is of de boog vrij vanuit v_m is of niet. Hij is zeker vrij vanuit v_m na $k+1$ iteraties, maar er zijn twee mogelijkheden voor de boog na k iteraties: hij kan ofwel vrij zijn ofwel niet.

vrij: Stel dus dat $\{v_m, v_{m+1}\}$ na k iteraties vrij is vanuit v_m. Dan is $d_k(b, v_{m+1}) \leq d_k(b, v_m) + 1$ omdat een pad eerst naar v_m een mogelijk pad is (als dit pad v_{m+1} al bevat geldt deze vergelijking des te meer). Dus
$d_{k+1}(b, v_{m+1}) = d_{k+1}(b, v_m) + 1 \geq d_k(b, v_m) + 1 \geq d_k(b, v_{m+1})$.

niet vrij: Stel dus dat $\{v_m, v_{m+1}\}$ na k iteraties niet vrij is vanuit v_m. Omdat hij nu wel vrij is, betekent dat dat het grotermakende pad P dat in de laatste iteratie werd gebruikt de boog *vrij heeft gemaakt* – dus er een stroom in de andere richting heeft toegepast. Deze situatie zie je in figuur 13. Maar wees voorzichtig: de paden kunnen meerdere bogen gemeen hebben – de tekening geeft maar

een voorbeeld! Omdat P een kortste grotermakend pad na k iteraties is, geldt dus $d_k(b, v_{m+1}) < d_k(b, v_m)$ – inderdaad geldt zelfs $d_k(b, v_{m+1}) = d_k(b, v_m) - 1$.
Omdat $v_0 = b, v_1, \ldots v_n = u$ een kortste grotermakend pad na $k+1$ iteraties is, geldt $d_{k+1}(b, v_{m+1}) = d_{k+1}(b, v_m) + 1$ en met inductie geldt dan $d_{k+1}(b, v_{m+1}) \geq d_k(b, v_m) + 1 = d_k(b, v_{m+1}) + 2$.



Figuur 13: Geval waar het grotermakende pad en een kortste pad een boog in verschillende richtingen gebruiken.

■

Inderdaad hebben wij in het tweede deel van het bewijs al een effect gezien dat ons zal helpen te bewijzen dat het Ford-Fulkerson algoritme met BFS altijd zal stoppen: als een boog één keer in de ene richting op een grotermakend pad vanuit b ligt en één keer in de andere richting dan lijken afstanden te groeien. En natuurlijk zijn afstanden in een graaf naar boven begrensd!
Maar inderdaad heeft Lemma 63 als voorwaarde dat de afstand vanaf b wordt gemeten, maar wij kunnen het ook bewijzen voor een arbitrair startpunt en het doel d:

64 Oefening *Bewijs het volgende lemma:*

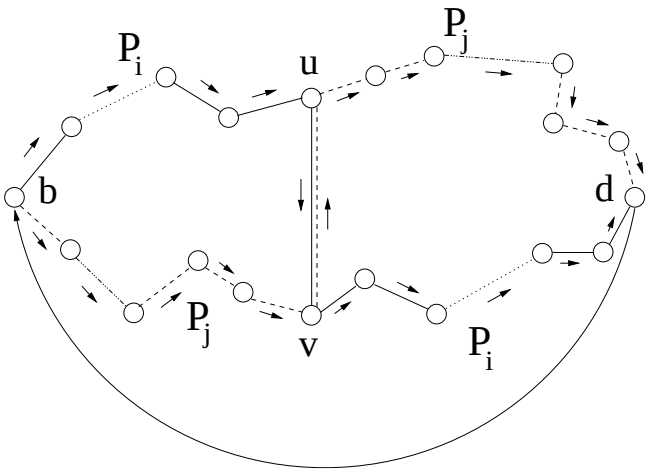
65 Lemma *Gegeven een netwerk $N = ((V, E), c)$ waarop het Ford-Fulkerson algoritme met BFS wordt toegepast en $u \in V$.*
Dan geldt voor alle $k \geq 0$: $d_{k+1}(u, d) \geq d_k(u, d)$.

66 Oefening *Geef een voorbeeld dat toont dat de Lemma's 63 en 65 niet geldig zijn als je afstanden tussen twee arbitraire punten neemt.*
Op welk punt gaat het bewijs niet door?

67 Lemma *Gegeven een samenhangend netwerk $N = (D = (V, E), c)$ met $b, d \in V$ en $(d, b) \in E$.*
Het BFS Ford Fulkerson algoritme toegepast op N zal na $O(|V| * |E|)$ iteraties stoppen. Omdat elke iteratie $O(|E|)$ tijd vraagt, is de complexiteit dus $O(|V| * |E|^2)$.

Bewijs: In elke iteratie is er ten minste één boog die de grens bepaalt voor de grotermakende stroom s – dat is een boog $e = (v, w)$ die in richting $v \to w$ gebruikt wordt en waarvoor geldt dat $s = c(e) - f(e)$ of die in richting $w \to v$ gebruikt wordt en waarvoor geldt dat $s = f(e)$. Dergelijke bogen noemen wij een *bottleneck*. Als een bottleneck voor de volgende keer in een grotermakend pad wordt gebruikt dan moet dat in de andere richting zijn.

Omdat er maar $|E|$ bogen zijn, zal er in een reeks van $|E| + 1$ iteraties dus altijd een boog $\{u, v\}$ (die wel gericht is maar waar de richting hier niet belangrijk is) zijn die in beide richtingen werd gebruikt – b.v in het grotermakende pad P_i van iteratie i en het grotermakende pad P_j van een latere iteratie j. Dat zie je geïllustreerd in Figuur 14. Maar ook hier is de figuur maar een schets – de paden kunnen wel meer bogen gemeen hebben!



Figuur 14: Een boog wordt in verschillende richtingen in twee grotermakende paden gebruikt.

Wij zullen nu aantonen dat $d_j(b, d) \geq d_i(b, d) + 2$:

Wij hebben $d_i(b, d) = d_i(b, u) + d_i(v, d) + 1$ en $d_j(b, d) = d_j(b, v) + d_j(u, d) + 1$.

Volgens Lemma's 63 en 65 geldt

- $d_j(b, v) \geq d_i(b, v)$
- $d_j(u, d) \geq d_i(u, d)$

Bovendien volgt uit de richtingen waarin $\{u, v\}$ gebruikt wordt

- $d_i(b, v) = d_i(b, u) + 1$
- $d_i(u, d) = d_i(v, d) + 1$

Dus

$d_j(b, d) = d_j(b, v) + d_j(u, d) + 1 \geq d_i(b, v) + d_i(u, d) + 1 = d_i(b, u) + d_i(v, d) + 3 = d_i(b, d) + 2$

Dus groeit de afstand tussen b en d langs een grotermakend pad ten laatste alle $|E|$ iteraties ($|E|+1$ iteraties in het begin) met ten minste 2. Omdat $d_0(b, d) \geq 1$, is de afstand na $k|E|+1$ iteraties dus $d_{k|E|+1}(b, d) \geq 2k+1$. Maar de afstand kan ten hoogste $|V|-1$ zijn, dus $2k+1 \leq |V|-1$ dus $k \leq \frac{|V|-2}{2}$.

Het maximale aantal iteraties is dus ten hoogste $\frac{n-2}{2}|E|+1 = O(|V||E|)$

■

68 Oefening *Pas het algoritme van Ford/Fulkerson met BFS toe op het netwerk in Figuur 12.*

De complexiteit van $O(|V||E|^2)$ is natuurlijk niet echt schitterend maar jullie zien ook dat de schattingen heel grof zijn en het algoritme normaal veel sneller zal zijn.
Omdat het probleem maximale stromen te vinden voor veel toepassingen heel belangrijk is, werd er natuurlijk veel onderzoek gedaan om de complexiteit van het algoritme te verbeteren. Er zijn nu algoritmen die in $O(|V|^3)$, $O(|V||E| \log(|V|))$ of $O(|V|^2 \log(\frac{|V|^2}{|E|}))$ draaien. Maar die zijn veel ingewikkelder en ook gebaseerd op het gebruik van efficiënte datastructuren – dus niet ideaal geschikt voor deze les waar natuurlijk vooral de theoretische eigenschappen heel nuttig zijn – zoals de toepassing op de stellingen van Menger (en wij zullen nog meer toepassingen in de grafentheorie zien).
Maar het is ook belangrijk om te zien dat stromen ook op plaatsen toegepast kunnen worden waar je het misschien niet onmiddellijk had verwacht. Toepassingen zoals het optimaliseren van stromen van data op computernetwerken of goederen op wegennetten zijn onmiddellijk herkenbaar als stroomproblemen, maar stromen kunnen ook op vlakken toegepast worden waar het misschien iets verrassender is:

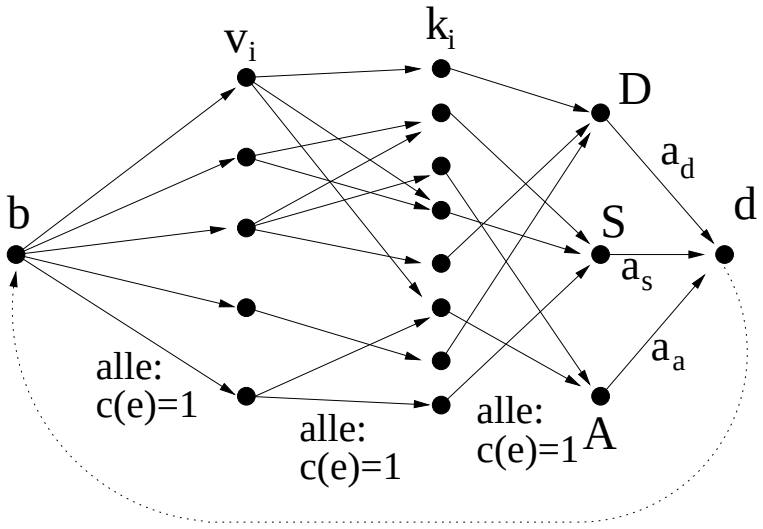
69 Voorbeeld (Hall 1956)

Er worden mensen voor een universiteitscommissie gezocht. Maar omdat het natuurlijk belangrijk is dat de invloed gelijkmatig verdeeld wordt, wordt wel geëist dat uit elk van de vakgroepen $v_1, \ldots, v_k$ precies één lid wordt gekozen (de hele commissie zal dus ook k leden bevatten). Maar ook elk van de drie groepen (doctoraats-) studenten, docenten *en* administratiepersoneel *moet door een op voorhand gekend aantal a_s, a_d en a_a vertegenwoordigd worden (waarbij natuurlijk $a_s + a_d + a_a = k$).*

Stel nu dat je weet wie kandidaat is voor de commissie en dat iemand die lid van meerdere vakgroepen is maar één enkele vakgroep mag representeren. Veronderstel ook dat niemand tot meerdere groepen behoort (bv. studeert maar ook een job in de administratie heeft).

Wat is een geschikt algoritme om uit te vissen of met de gegeven kandidaten een commissie mogelijk is die aan de eisen voldoet?

De oplossing zie je in Figuur 15. Je maakt voor elke vakgroep v_i een top aan en voegt een boog (b, v_i) toe met capaciteit 1*. Dan maak je voor elke kandidaat k_j een top aan en voegt een boog (v_i, k_j) met capaciteit* 1 *toe voor elke vakgroep v_i waarvan k_j lid is. Dan voeg je nog toppen A (administratie) D (docenten) en S (studenten) toe en bogen (k_i, X) met $X \in \{A, D, S\}$ als kandidaat k_i tot de groep X behoort. Ten slotte worden nog bogen $(A, d), (S, d), (D, d)$ met capaciteiten a_a, a_s, a_d toegevoegt en natuurlijk de boog (d, b) met capaciteit oneindig.*



Figuur 15: Het probleem een commissie te vormen wordt naar een stroomprobleem vertaald.

Het is onmiddellijk duidelijk (waarom?) dat de maximale stroom ten hoogste k is. Bovendien kan je een commissie die aan de eisen voldoet ook gemakkelijk vertalen naar een stroom met waarde k en een stroom met waarde k naar een bezetting van de commissie. Dus is het probleem of er een commissie bestaat die aan de eisen voldoet equivalent met het probleem of de maximale stroom waarde k heeft. Het is ook duidelijk dat zo'n netwerk voor een gegeven instantie van een commissieprobleem *snel (in lineaire tijd) opgebouwd kan worden.*
(Wat precies betekent lineaire tijd *in deze samenhang?)*

70 Oefening *Stel nu dezelfde situatie als in voorbeeld 69 – alleen dat de kandidaten* **wel** *tot meerdere groepen kunnen behoren (bv. studeren maar ook een job in de administratie hebben). Maar net zoals iedereen maar n vakgroep mag representeren mag iedereen ook maar n van de groepen* (doctoraats-) studenten, docenten *en* administratiepersoneel *vertegenwoordigen.*
Wat moet je aan jouw netwerk wijzigen om ook deze situatie te kunnen modelleren?

71 Oefening *In de Belgische Ethias-league is de trainer van Optima Gent in een moeilijke positie. De club wil hem ontslaan omdat het nu al zeker is dat ze op het einde van het jaar opnieuw niet op positie* 1 *zullen staan. Nu zou hij graag uitvissen of hij – ten minste theoretisch – toch nog kan winnen. Dat betekent dat er voor de overblijvende spelen een scenario is zodat er op het einde geen ploeg meer punten heeft dan Gent (met het aantal baskets houden wij op dit moment geen rekening).*
De details: Er is geen gelijkspel. De winnaar krijgt 3 *punten en de verliezer krijgt* 1 *punt. Je kent het klassement op dit moment en je weet welke spelen nog gespeeld moeten worden.*
Vertaal het probleem of een gegeven ploeg (bv. Optima Gent) het klassement nog kan winnen naar een stroomprobleem. De vertaling moet daarbij lineair in het aantal **nog te spelen** *wedstrijden plus het aantal ploegen zijn.*
Tip: denk misschien eerst na of de situatie er anders zou uitzien als je 1 *punt voor winst en* 0 *punten voor verlies zou krijgen.*

72 Oefening *In sommige gevallen is het misschien niet voldoende te weten wat de maximale stroom is maar je wil ook weten hoe je die groter kan maken zonder te veel aan het netwerk te moeten wijzigen. Ideaal zou natuurlijk zijn als je alleen de capaciteit van een enkele boog zou moeten verhogen om een grotere maximale stroom te verkrijgen.*

- *Toon aan dat een dergelijke boog niet altijd bestaat.*

- *Geef een algoritme dat een dergelijke boog vindt als die bestaat.*

73 Oefening *Natuurlijk is het belangrijk dat in het geval van vuur in een gebouw van de universiteit met veel leszalen de studenten en medewerkers het gebouw snel kunnen verlaten. Als je een zekere* wandelsnelheid *stelt dan heeft elke uitgang van een leszaal, elke gang en elke uitgang van het gebouw een zekere capaciteit – bepaald bv. door het aantal mensen dat per minuut kan passeren.*
Stel nu dat je leszalen $L_1, \ldots, L_k$ en uitgangen $U_1, \ldots, U_l$ hebt en dat je ook de geometrie van het gebouw kent (welke leszalen liggen aan welke gangen, op welke punten grenzen de gangen aan elkaar, etc). Bovendien ken je ook de capaciteiten van de gangen en de uitgangen van de leszalen en van het gebouw.
Je eist dat voor elke leszaal L_i een zekere aantal m_i van studenten per minuut de leszaal kan verlaten en doorgaan tot één van de uitgangen. Dat moet natuurlijk op een manier mogelijk zijn dat de studenten uit de verschillende leszalen tegelijk het gebouw kunnen verlaten en niet elke leszaal apart.
Beschrijf een algoritme dat vaststelt of het gebouw aan jouw eisen voldoet.

Behalve in Oefening 73 hebben wij tot nu toe stromen op een heel beperkte manier bekeken. Wij hadden altijd maar één bron en maar één doel. Bovendien was de ondergrens voor een stroom altijd 0. In Oefening 73 hebben wij gezien dat het soms nuttig is ook meerdere bronnen toe te laten – en wij hebben ook gezien hoe je dat gemakkelijk kan vertalen naar maar één bron. Analoog kan het in sommige toepassingen ook nuttig zijn een benedengrens voor een stroom te hebben. Of precies:

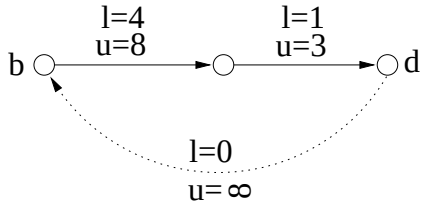
74 Definitie

- *Een <u>netwerk</u> met boven- en benedengrenzen (wij schrijven kort <u>bb-netwerk</u>) is een 3-tal $N = (D, l, u)$ waarbij $D = (V, E)$ een gerichte graaf is, $l : E \to \mathbb{Z}(\mathbb{Q}, \mathbb{R}, \ldots)$ een functie die de benedengrens voor de stroom op de bogen bepaalt en $u : E \to \mathbb{Z}(\mathbb{Q}, \mathbb{R}, \ldots)$ een capaciteitsfunctie die de bovengrens bepaalt. De functies $l()$ en $u()$ kennen niet negatieve gehele (rationale, reële...) getallen toe aan de gerichte bogen.*

  Om triviale gevallen te vermijden, eisen wij bovendien dat voor alle bogen e geldt dat $u(e) \geq l(e)$.

- *Een <u>stroom</u> <u>(flow)</u> op een bb-netwerk $N = (D, l, u)$ is een functie $f : E \to \mathbb{Z}(\mathbb{Q}, \mathbb{R}, \ldots)$ met $l(e) \leq f(e) \leq u(e)$ $\forall e \in E$ en voor alle toppen $v \in V$ geldt de wet van Kirchhoff.*

Het is natuurlijk duidelijk dat een bb-netwerk geen stroom zou kunnen hebben als voor een boog e zou gelden dat $u(e) < l(e)$ (daarom hebben wij dat verboden), maar zelfs in gevallen waar je deze triviale problemen niet hebt, is het mogelijk dat er gewoon geen stroom is – zoals in Figuur 16.

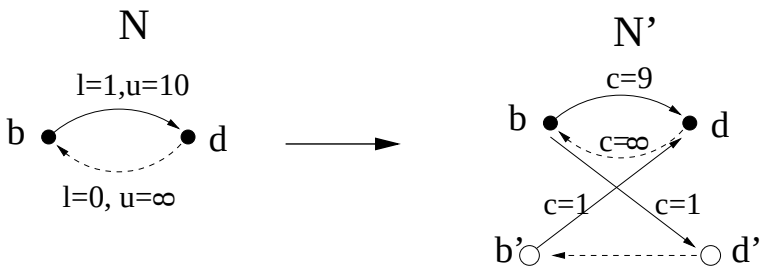


Figuur 16: Een bb-netwerk waarop duidelijk geen stroom mogelijk is.

Dit is natuurlijk een probleem voor het algoritme van Ford/Fulkerson: dat begint met een stroom die overal 0 is, maar hier is dat misschien geen toegelaten stroom! Er zouden zeker nog meer aanpassingen meoten gebeuren om het algoritme toe te passen, maar al de eerste stap is dus niet voor de hand liggend.
Wij zullen het probleem om vast te stellen of een bb-netwerk een stroom heeft (en die te bepalen als die bestaat) gewoon vertalen naar een gewoon maximaalstroomprobleem. Het idee zie je in Figuur 17: je voegt een nieuwe bron en een nieuw doel toe en dan gebruik je een *omleiding* voor de minimale stroom over de nieuwe bron en doel. De capaciteiten van de bogen van/naar de nieuwe toppen worden zo gekozen dat er een stroom op het bb-netwerk bestaat als en slechts als er op het netwerk een maximale stroom bestaat waarbij de volle capaciteit van de bogen die uit b' vertrekken, gebruikt wordt. Een stroom op het bb-netwerk kunnen wij gebruiken om een stroom op het netwerk te vinden die de volle capaciteit van de bogen vanuit b' gebruikt (dat noemen wij *de bogen satureert*). Een stroom op het netwerk dat de bogen vanuit b' satureert kunnen wij gebruiken om een stroom op het bb-netwerk te vinden.
Maar natuurlijk moeten wij dat op een algemene en precieze manier beschrijven:

75 Algoritme Van een bb-netwerk naar een netwerk
Gegeven is een bb-netwerk $N = ((V, E), l, u)$.
Dan definieer voor alle $v \in V$*:*

- $b^{-(+)}(v) := \sum_{e \in v^{-(+)}} l(e)$



Figuur 17: Een bb-netwerk wordt vertaald naar een gewoon netwerk.

- $b(v) := b^-(v) - b^+(v)$

Dus is b^- een instroomgrens – een benedengrens voor de stroom die in deze top moet instromen – en b^+ een uitstroomgrens – een benedengrens voor de stroom die uit deze top moet uitstromen.
$b(v)$ geeft dus het overschot *als je op alle bogen gewoon de ondergrens als stroomwaarde zou vastleggen. Voor sommige toppen zal het overschot dus positief zijn en voor sommige negatief – daar zou dus te veel uitstromen en te weinig instromen. De vraag is dus of je deze overschotten over het net kan laten* afstromen. *Dat vissen wij uit door een stroom op een iets ander netwerk te berekenen. In dit nieuwe netwerk laten wij in de toppen met positief overschot een boog toekomen met capaciteit het overschot (en omgekeerd voor toppen met negatief overschot). Als alle nieuwe bogen door een stroom volledig gesatureerd kunnen worden (dus bv. het overschot in een top met positieve overschot kan instromen en dus ook afstromen – ten slotte is het een stroom die aan de wet van Kirchhoff voldoet), kunnen de overschotten in het netwerk afstromen. Of precies:*

Wij definiëren een netwerk $N'(N) = ((V', E'), c)$ als volgt:

- *Wij voegen eerst een nieuwe bron b' en een nieuw doel d' toe. Dus $V' = V \cup \{b', d'\}$ (waarbij wij stellen dat b' en d' geen toppen zijn die al in V zitten).*

- *Voeg volgende bogen toe: een boog (d', b'), voor elke top $v \in V$ met $b(v) > 0$ een boog (b', v) en voor elke top v met $b(v) < 0$ een boog (v, d'). Dus $E' = E \cup \{(d', b')\} \cup \{(b', v) | b(v) > 0\} \cup \{(v, d') | b(v) < 0\}$*

- *Voor $e \in E'$ definieer*

$$c(e) = \begin{cases} u(e) - l(e) & \text{als } e \in E \\ b(v) & \text{als } e = (b', v) \\ -b(v) & \text{als } e = (v, d') \\ \infty & \text{als } e = (d', b') \end{cases}$$

76 Stelling *$N = ((V, E), l, u)$ zij een bb-netwerk en $N'(N)$ het resultaat als je Algoritme 75 op N toepast.*
Dan bestaat een stroom op N als en slechts als er een stroom met waarde $\sum_{\substack{v \in V \\ b(v)>0}} b(v)$ op (d', b') bestaat.

Bewijs: Stel eerst dat een stroom $f()$ op N bestaat. Dan definieer

$$f'(e) := \begin{cases} f(e) - l(e) & \text{als} \quad e \in E \\ b(v) & \quad e = (b', v) \\ -b(v) & \quad e = (v, d') \\ \sum_{\substack{v \in V \text{met} \\ b(v)>0}} b(v) & \quad e = (d', b') \end{cases}$$

Wij moeten nu aantonen dat dat inderdaad een stroom op N' is:

Het is duidelijk dat $0 \leq f'(e) \leq c(e) = u(e) - l(e)$ voor alle $e \in E$. Maar ook voor bogen $e \in E' \setminus E$ volgt $0 \leq f'(e) \leq c(e)$ direct uit de definities.

Maar ook de wet van Kirchhoff is geldig voor elke top:
Stel dat $v \in N,\ b(v) > 0$

$$\begin{aligned} \sum_{e \in v^-} f'(e) \quad - \quad \sum_{e \in v^+} f'(e) &= \sum_{\substack{e \in v^- \\ e \in N}} (f(e) - l(e)) + \underbrace{b(v)}_{\text{van } b'} - \sum_{\substack{e \in v^+ \\ e \in N}} (f(e) - l(e)) \\ &= \underbrace{\sum_{\substack{e \in v^- \\ e \in N}} f(e) - \sum_{\substack{e \in v^+ \\ e \in N}} f(e)}_{0, \text{ omdat } f \text{ stroom}} + b(v) - \underbrace{(\sum_{\substack{e \in v^- \\ e \in N}} l(e) - \sum_{\substack{e \in v^+ \\ e \in N}} l(e))}_{b(v)} = 0 \end{aligned}$$

Voor v met $b(v) < 0$ of $b(v) = 0$ volgt het volledig analoog.

Voor b' volgt dat de wet van Kirchhoff geldt direct uit de definitie en voor d' geldt

$$\sum_{e \in d'^-} f'(e) - \sum_{e \in d'^+} f'(e) = \sum_{b(v)<0} -b(v) - \sum_{b(v)>0} b(v) = -\sum_{v \in N} b(v) = 0,$$

omdat elke $l(e)$ één keer een positieve en één keer een negatieve bijdrage levert.

Stel anderzijds dat een stroom f' op N' bestaat zodat voor elke boog $e = (b', v)$ of $e = (v, d')$ geldt dat $f'(e) = c(e)$. Dan volgt onmiddellijk dat als wij voor $e \in E$ definiëren $f(e) := f'(e) + l(e)$ inderdaad $f()$ een stroom op N is.

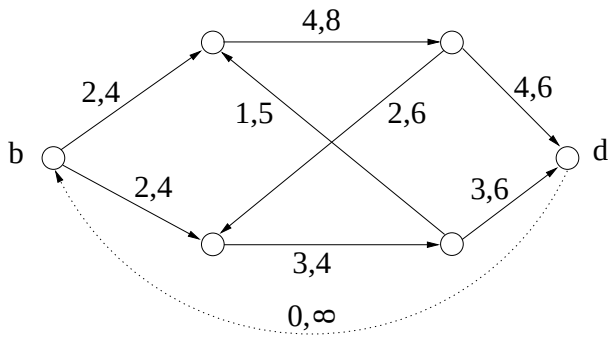
■

77 Oefening *Werk het tweede deel van het bewijs van Stelling 76 expliciet uit.*

78 Oefening *Beschrijf een algoritme dat voor een bb-netwerk niet een arbitraire stroom vindt maar een stroom met maximale waarde op de boog (d, b).*

Inderdaad zijn deze algemenere netwerken met meerdere bronnen of doelen of benedengrenzen voor de stroom op de bogen dus niet echt moeilijker dan de netwerken met alleen een bovengrens op de bogen.

79 Oefening *Gebruik Algoritme 75 om voor het bb-netwerk in Figuur 18 te beslissen of er een stroom bestaat:*



Figuur 18: Een bb-netwerk waarbij x, y aan een boog e betekent dat $l(e) = x$ en $u(e) = y$.

Samenvattend zouden wij dus kunnen zeggen dat stromen niet alleen duidelijk belangrijk zijn voor toepassingen, maar dat er ook een samenhang bestaat met andere stellingen uit de grafentheorie (zoals de stellingen van Menger). Maar in het volgende hoofdstuk zullen wij zien dat er zelfs nog meer stellingen zijn die met de stelling van Ford-Fulkerson samenhangen.

4 Koppelingen (matchings)

80 Definitie *Gegeven een graaf $G = (V, E)$.*

- *Een koppeling of matching in G is een verzameling $M \subseteq E$ zodat voor alle $e, e' \in M$ met $e \neq e'$ geldt dat $e \cap e' = \emptyset$.*
- *Als er een boog $e \in M$ bestaat met $v \in e$ dan heet de top v gesatureerd.*
- *Wij gebruiken het woord maximum voor een verzameling als wij een verzameling met de grootst mogelijke kardinaliteit van een* dergelijke *verzameling bedoelen. Wij gebruiken maximaal als de verzameling niet uitgebreid kan worden tot een grotere* dergelijke *verzameling of als het om getallen gaat (zo als in* maximale afstand*). Daarbij staat* dergelijk *altijd voor de eisen die wij aan de verzamelingen die wij beschouwen stellen. Voor matchings betekent dat: wij noemen een matching M maximum als er geen matching van dezelfde graaf bestaat met meer bogen. (Dat is een sterkere eis dan dat er geen matching M' van de graaf bestaat met $M \subsetneq M'$ – dus dat M maximaal is).*
- *Als elke top gesatureerd is door een koppeling M dan heet M perfect.*

  Het koppelingsgetal of matchinggetal $m(G)$ is gedefinieerd als het aantal bogen in een maximum koppeling, dus
  $m(G) = \max\{|M| \mid M \text{ is koppeling van } G\}$.

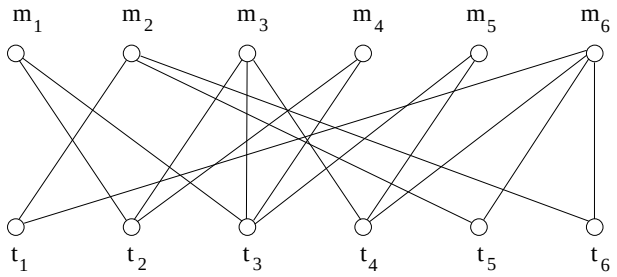
Ook koppelingen zijn heel belangrijk voor toepassingen, waarbij de grafen vaak bipartiet zijn omdat er een natuurlijke splitsing van de toppenverzameling in twee verschillende soorten toppen is. Wij zullen hiervan voorbeelden zien, maar eerst de precieze definitie:

81 Definitie *Een graaf $G = (V, E)$ heet bipartiet als en slechts als er V_1, V_2 bestaan met $V_1 \cap V_2 = \emptyset$ en $V_1 \cup V_2 = V$ en $\forall e \in E : |e \cap V_1| = |e \cap V_2| = 1$.*

82 Oefening *Toon aan: Een graaf G is bipartiet als en slechts als G geen cykels met oneven lengte bevat.*

83 Voorbeeld *In een bedrijf moeten verschillende taken $t_1, \ldots, t_n$ uitgevoerd worden door machines $m_1, \ldots, m_m$. Een machine kan daarbij maar één taak tegelijk uitvoeren. Maar niet elke machine kan ook elke taak uitvoeren. Hoeveel taken kunnen alle machines tegelijk uitvoeren? Dat kan je oplossen door het probleem als een graaf te modelleren:*

De verzameling van toppen is $V = \{t_1, \ldots, t_n, m_1, \ldots, m_m\}$. De verzameling van bogen is $E = \{\{t_i, m_j\} |$ machine m_j kan taak t_i uitvoeren$\}$
Het grootst mogelijke aantal taken die je tegelijk kan uitvoeren is dan het matchinggetal van (V, E).
Deze graaf is duidelijk bipartiet – elke boog bevat één machine en één taak. Een voorbeeld vinden jullie in Figuur 19. Hoeveel taken kan je daar tegelijk uitvoeren?

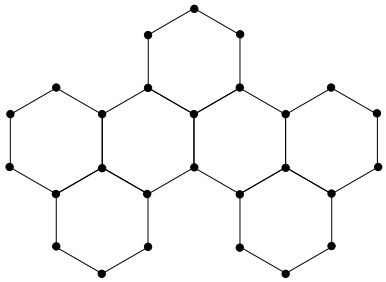


Figuur 19: Een voorbeeld van een als matchingprobleem gemodelleerd taakverdelingsprobleem.

Natuurlijk zijn er heel veel gelijkaardige toepassingsvoorbeelden van matchings die duidelijk bipartiet zijn (secretaressen en teksten die geschreven moeten worden (waarbij niet elke secretaresse elke tekst kan schrijven), piloten en vliegtuigen waarmee gevlogen moet worden (waarbij niet elke piloot met elk vliegtuig of over elke route kan vliegen), ...).

84 Voorbeeld *Wij kunnen nu nog niet precies beschrijven wat een benzenoïde is omdat wij nog niet weten wat vlakke (of planaire) grafen zijn, maar voor een voorbeeld is een informele beschrijving misschien voldoende:*
Een benzenoïde is een uit reguliere zeshoeken opgebouwde graaf zoals in Figuur 20. Natuurlijk zijn niet alle **cykels** *in de graaf zeshoeken, maar alle cykels zijn wel even (dat kunnen wij nu nog niet bewijzen), deze benzenoïden zijn dus bipartiet. De toppen van deze graaf zijn koolstofatomen en elk atoom met graad 2 zit aan de rand en heeft inderdaad nog een waterstofatoom als buur. Maar deze atomen tekenen de scheikundigen nooit omdat ze toch al weten dat die er zijn. Dus heeft elke koolstof inderdaad 3 buren in de hele graaf – ok al zie je ze op de tekening, die alleen de door de koolstoffen geïnduceerde graaf voorstelt niet allemaal. Maar koolstofatomen hebben normaal 4 buren. Het is inderdaad het werk dat Friedrich August Kekulé von Stradonitz – of kort* August Kekule *– hier in Gent heeft gedaan dat dit probleem oplost:*

hij stelde de structuur van benzine als een ring van zes koolstofatomen voor waarbij elke tweede boog dubbel is. Omdat elke top in één dubbele boog zit, heeft elk koolstofatoom dan 4 bindingen. Deze verzameling van bogen die dubbel zijn in de graaf van de koolstofatomen is dus wat wij een perfecte koppeling zouden noemen. Maar de scheikundigen noemen een dergelijke perfecte koppeling nog altijd een Kekulé-structuur.
En hoewel het model van Kekulé nu dat scheikundigen de kwantumchemie ter beschikking hebben zeker niet meer echt actueel is, is het nog altijd zo dat alle benzenoïden die stabiel zijn een Kekulé-structuur lijken te hebben.
Is de benzenoïde in Figuur 20 een kandidaat voor een stabiele molecule?



Figuur 20: Een benzenoïde.

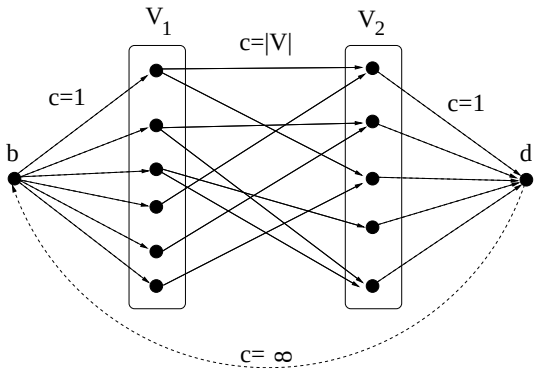
Omdat koppelingen bijzonder belangrijk zijn, bestaan hier veel gespecialiseerde algoritmen om maximum matchings te berekenen (bv. een lineair algoritme voor benzenoïden). Maar toch is het zeker interessant te zien dat ook hier stroomalgoritmen toegepast zouden kunnen worden:

85 Algoritme 4.1 Koppelingen in bipartiete grafen

Maximum matchings in bipartiete grafen door middel van stromen
Gegeven een bipartiete graaf $G = (V, E)$ met partitieklassen V_1 en V_2.
Bouw een netwerk $N = ((V', E'), c)$ als volgt:

- *Definieer $V' = V \cup \{b, d\}$ waarbij wij stellen dat $V \cap \{b, d\} = \emptyset$.*
- *Definieer $E' = \{(v, w) | v \in V_1, w \in V_2, \{v, w\} \in E\} \cup \{(b, v) | v \in V_1\} \cup \{(w, d) | w \in V_2\} \cup \{(d, b)\}$*
- *Definieer $c(e) = |V|$ voor alle $e \in E$, $c(e) = 1$ voor alle $e \notin E, e \in E', e \neq (d, b)$ en $c((d, b)) = \infty$*

Nu bereken (bv. met het Ford Fulkerson algoritme) een maximale stroom op (d, b) met waarden uit $\mathbb{Z}$ op N. Daarbij is de stroomwaarde op de bogen met capaciteit $|V|$ natuurlijk ook altijd 0 of 1 omdat in het beginpunt van de boog ten hoogste stroom 1 kan aankomen. De bogen uit E die bij bogen uit E' behoren met stroomwaarde 1 vormen een maximum koppeling.
Een voorbeeld van de constructie zie je in Figuur 21
De reden dat de capaciteiten van de bogen $e \in E$ niet als 1, maar duidelijk groter gekozen werden, is om Oefening 87 iets gemakkelijker te maken. Voor dit algoritme zou $c(e) = 1$ voor deze bogen even goed werken.



Figuur 21: De vertaling van een matchingprobleem naar een stroomprobleem. De oude bogen zijn vol en de toegevoegde bogen zijn met streepjes.

Bewijs: In dit geval is het vrij gemakkelijk om te bewijzen dat het algoritme werkt en de grootte van een maximale stroom inderdaad gelijk is aan een maximum matching. Wij moeten alleen aantonen dat een stroom van grootte k altijd tot een matching van grootte k leidt en een matching van grootte k tot een stroom met grootte k. Maar de samenhang wordt al in het algoritme gegeven...

■

86 Definitie *Gegeven een graaf $G = (V, E)$.*
Een verzameling $U \subseteq V$ heet toppenoverdekking (vertex cover) als voor alle $e \in E$ geldt $e \cap U \neq \emptyset$.
De invariant $vc(G)$ definiëren wij als de kardinaliteit van een kleinste toppenoverdekking van G.

87 Oefening *Vertaal het probleem een kleinste toppenoverdekking in een bipartiete graaf te vinden naar een probleem met stromen.*

Tip: vergelijk toppenoverdekkingen met capaciteiten van b en d splitsende boogverzamelingen in de door Algoritme 85 opgebouwde graaf.

88 Oefening *Bewijs de stelling van König (1931):*

89 Stelling *(König 1931)*
Voor een bipartiete graaf G geldt $vc(G) = m(G)$.

De volgende stelling van König kan je direct door middel van Ford-Fulkerson bewijzen, maar door Stelling 89 te gebruiken, gaat het iets sneller:

90 Stelling *Elke bipartiete r-reguliere graaf G met $r \geq 1$ heeft een perfecte koppeling.*

Bewijs: In een bipartiete r-reguliere graaf zijn de twee bipartitieklassen V_1, V_2 even groot. Het aantal bogen kan je tellen als $|E| = |V_1| * r$ en als $|E| = |V_2| * r$ waaruit natuurlijk onmiddellijk volgt dat $|V_1| = |V_2|$. Wij schrijven kort n voor $|V_1| = |V_2|$.

Omdat er $|E| = |V_1| * r$ bogen zijn en elke top precies r bogen overdekt, heb je dus ten minste $|V_1| = |V_2|$ toppen voor een toppenoverdekking nodig. Maar aan de andere kant overdekken alle toppen van V_1 (of V_2) zeker alle bogen. Dus $vc(G) = |V_1|$ en met Stelling 89 dus $m(G) = |V_1| = |V_2|$ – er is dus een pefecte koppeling.

■

91 Oefening *Geef een lineair algoritme dat in een reguliere bipartiete graaf een minimale toppenoverdekking berekent.*

92 Oefening *Bewijs dat in een bipartiete graaf G met $\Delta(G) \geq 1$ altijd een matching bestaat zodat alle toppen v met $\deg(v) = \Delta(G)$ gesatureerd zijn.*

93 Oefening *Een boogkleuring van een graaf $G = (V, E)$ met kleuren $1, \ldots, k$ is een afbeelding $c : E \to \{1, \ldots, k\}$ zodat voor alle $e, e' \in E$ met $e \cap e' \neq \emptyset$ geldt dat $c(e) \neq c(e')$. De chromatische index $\chi'(G)$ is het kleinste getal k' zodat er een boogkleuring met kleuren $1, \ldots, k'$ bestaat.*
Er zijn maar twee mogelijke waarden voor $\chi'(G)$: $\Delta(G)$ en $\Delta(G) + 1$ maar in het algemeen is het heel moeilijk (precies: NP-compleet) om te beslissen welk van de twee getallen het is.
Voor bipartiete grafen is het gemakkelijk. Bewijs:

94 Stelling *Voor bipartiete grafen geldt altijd $\chi'(G) = \Delta(G)$.*

De manier om maximum matchings in bipartiete grafen door middel van stromen te vinden, is natuurlijk vooral om theoretische redenen interessant: gewoon om de samenhang tussen stromen en matchings te zien en tussen stellingen uit de grafentheorie die op het eerste gezicht heel verschillend zijn. Algoritmen die gespecialiseerd zijn op matchings, baseren op paden die de matching groter maken – ook dat lijkt dus op stromen. Stel dat een matching M in een graaf G is gegeven.
Wij noemen een pad $P = v_1, \ldots, v_n$ in G grotermakend als het aan de volgende eisen voldoet:

- v_1 en v_n zijn ongesatureerd.
- $\{v_i, v_{i+1}\} \notin M$ voor alle oneven i, $1 \leq i < n$.
- $\{v_i, v_{i+1}\} \in M$ voor alle even i, $1 \leq i < n$.

Als wij een grotermakend pad P hebben dan kunnen wij gewoon de bogen van $P \cap M$ in M vervangen door de bogen van $P \setminus M$ en hebben een matching M' die één boog meer bevat. (Dus $M' = M \setminus (P \cap M) \cup (P \setminus M)$.)
Wij kunnen dus een gretig algoritme toepassen op om het even welke beginkoppeling en die door middel van grotermakende paden groter maken. Wij zullen zien dat dit algoritme inderdaad een maximum matching vindt.

95 Oefening *Onderzoek het verband tussen een grotermakend pad (in de betekenis van matchings) in een bipartiete graaf met een matching en een grotermakend pad (in de betekenis van stromen) in het netwerk dat met deze graaf correspondeert met een stroom die met de matching correspondeert.*

96 Algoritme Maximum matching berekenen
Gegeven een (niet noodzakelijk bipartiete) graaf G.

- *Kies een beginmatching M bv. op een gretige manier: kies bogen tussen twee ongesatureerde toppen voor M totdat je geen bogen meer kan toevoegen.*
- *Herhaal de volgende stappen tot aan de stopvoorwaarde is voldaan:*
  - *Zoek een grotermakend pad P. Als er geen grotermakend pad is, stop dan.*
  - *Kies $M \setminus (P \cap M) \cup (P \setminus M)$ als nieuwe koppeling M.*

Het is duidelijk dat de eerste stap in principe niet nodig zou zijn. *Bogen tussen ongesatureerde toppen kiezen* is inderdaad hetzelfde als korte grotermakende paden te kiezen. Maar deze stap kan je ook door een andere snelle heuristiek vervangen waarvan je verwacht dat die al relatief grote matchings oplevert.
Ook de stap *zoek een grotermakend pad* is natuurlijk niet voldoende duidelijk voor een algoritme – hoe doe je dat?
Maar eerst zullen wij bewijzen dat het algoritme inderdaad maximum matchings oplevert:

97 Stelling Tutte 1957
Gegeven een (niet noodzakelijk bipartiete) graaf G en een matching M van G. Dan geldt:
$|M| = m(G)$ als en slechts als er geen grotermakend pad is.

Bewijs: De richting $|M| = m(G) \Rightarrow$ *er bestaat geen grotermakend pad* is duidelijk omdat als zo'n pad voor M wel zou bestaan, er een koppeling met meer bogen zou bestaan, dus $|M| < m(G)$.

Stel nu dat $|M| < m(G)$. Er bestaat dus een koppeling M' met $|M'| > |M|$. Wij moeten aantonen dat er een grotermakend pad P bestaat.

Kijk naar de graaf $G' = (V, M'')$ met $M'' = \underbrace{(M \cup M') - (M \cap M')}_{M+M'}$.

De graad van een top in deze graaf kan ten hoogste twee zijn – één boog uit elke koppeling. De samenhangscomponenten zijn dus geïsoleerde toppen, cykels of paden.

Omdat $M' = (M' \cap M) \dot\cup (M' \cap M'')$, dus $|M'| = |M \cap M'| + |M' \cap M''|$ en analoog $|M| = |M \cap M'| + |M \cap M''|$, volgt met $|M'| > |M|$ direct

$$|M' \cap M''| > |M \cap M''|.$$

Er moet dus ten minste één samenhangscomponent zijn die meer bogen van M' bevat dan van M. In de samenhangscomponenten die geïsoleerde toppen of cykels zijn, zitten even veel bogen uit M als uit M' dus moet er een samenhangscomponent zijn die een pad is en die meer bogen van M' bevat dan van M. Maar omdat nooit twee bogen uit M of twee bogen uit M' een top gemeen kunnen hebben moet dit pad precies de structuur van een grotermakend pad hebben.

■

98 Oefening *Gegeven een graaf G met koppelingsgetal $m(G)$.*
Geef een benedengrens voor de grootte van de beginmatching die de eerste stap in Algoritme 96 opbouwt en bewijs dat jouw benedengrens optimaal is.

Nu moeten wij dus alleen nog beschrijven hoe de stap *vind een grotermakend pad* geïmplementeerd kan worden. Wij zullen zien dat dat in het geval van een bipartiete graaf veel gemakkelijker is. Inderdaad heb je daar alleen maar een BFS-achtige routine nodig die alternerende paden bouwt:

99 Oefening *Bestaat er in een bipartiete graaf G met $\Delta(G) \geq 1$ ook altijd een* **maximum** *matching zodat alle toppen v met $\deg(v) = \Delta(G)$ gesatureerd zijn?*

100 Algoritme *Gegeven een bipartiete graaf $G = (V, E)$ met een koppeling M die ook leeg kan zijn en een ongesatureerde top s.*
Gezocht is een grotermakend pad dat in s begint.
L zij een lijst van toppen die in het begin alleen maar s bevat.
De afstand $d(s, s)$ wordt op 0 gezet, alle andere afstanden $d(s, v)$ tot toppen $v \in V$ op ∞. De afstanden hier zijn afstanden langs een pad waarin bogen uit M en M^c alterneren.
Nu herhaal het volgende totdat de lijst leeg is:

- *verwijder het eerste element l uit de lijst*
- – *als $d(l, s)$ even: voeg alle toppen v met afstand $d(s, v) = \infty$ waarvoor er een boog $\{l, v\} \in M^c$ bestaat toe aan het einde van de lijst en definieer $d(s, v) = d(s, l) + 1$. Als één van de toppen ongesatureerd is, is een grotermakend pad gevonden. In dit geval stop.*
  – *als $d(l, s)$ oneven: als er een top v met afstand $d(s, v) = \infty$ bestaat waarvoor er een boog $\{l, v\} \in M$ bestaat, dan voeg v toe aan het einde van de lijst en definieer $d(s, v) = d(s, l) + 1$.*

Als de hele lijst afgewerkt is en er werd geen grotermakend pad gevonden dan bestaat het ook niet.

Bewijs: Wij moeten natuurlijk aantonen dat als er een grotermakend pad is dat van s vertrekt, er ook één wordt gevonden.

Wij zullen iets sterkers aantonen: dat alle toppen die vanuit s door middel van een alternerend pad bereikt kunnen worden ook via een alternerend pad in de opgebouwde boom bereikt kunnen worden.

Stel dat de partitie van V gegeven is door $V = V_1 \cup V_2$ en dat $s \in V_1$. Wij zullen dat opnieuw bewijzen door inductie te gebruiken over de afstand. Wij gebruiken de afstand langs een alternerend pad. Het resultaat is duidelijk voor afstand 0 of 1 – stel dus dat elke top die via een alternerend pad van lengte l vanuit s bereikt kan worden ook inderdaad door middel van een pad in de opgebouwde alternerende boom bereikt kan worden. Stel nu dat v een top met alternerende afstand $l+1$ van s is. Dan is er een pad $P = s, v_1, \ldots, v_l, v$ van lengte $l+1$ van s naar v. De alternerende afstand van v_l is dus ten hoogste l – hij wordt dus bereikt. Noem het alternerende pad in de boom $P' = s, w_1, \ldots, w_n = v_l$. Nu komt een belangrijke stap: Omdat G bipartiet is, hebben l en n dezelfde pariteit – anders zou je een cykel van oneven lengte hebben (vergelijk het bewijs van Oefening 82). Als $v_l, v \notin M$ (dus l even) dan is n even en het algoritme zal op het moment dat v_l afgewerkt wordt de boog v_l, v toetsen en v toevoegen (als die nog niet in de boom zit). Het geval $v_l, v \in M$ is volledig analoog. Dus wordt v ook langs een grotermakend pad in de boom bereikt – en precies dat moesten wij bewijzen.

■

Je kan het basisalgoritme onmiddellijk een beetje verbeteren:

101 Oefening *Moet je als je in een bipartiete graaf naar een grotermakend pad zoekt van alle ongesatureerde toppen vertrekken om naar grotermakende paden te zoeken? Als niet: van welke toppen wel?*

De volgende optimalisatie is iets moeilijker om te bewijzen, maar helpt ook meer – ook voor de theoretische analyse:

102 Oefening *Gegeven een graaf $G = (V, E)$ waarop één van de geziene matchingalgoritmen wordt toegepast. Stel dat $v \in V$ een ongesatureerde top is en dat nadat n keer de matching door middel van een grotermakend pad groter werd gemaakt er geen grotermakend pad wordt gevonden dat vanaf v start.*
Toon aan dat er dan ook na $m > n$ iteraties geen grotermakend pad zal zijn – dat je dus nooit meer vanuit de top v naar een grotermakend pad moet zoeken.

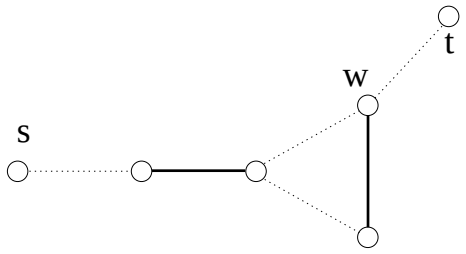
Als je deze twee optimalisaties toepast (of ten minste de tweede) zie je onmiddellijk dat de complexiteit van dit algoritme $O(|V| * |E|)$ is (ten minste als wij – zoals altijd – stellen dat $|E| = \Omega(|V|)$). Dat is al niet slecht en inderdaad is het zo dat de uitbreidingsstap in veel gevallen niet vaak gebruikt

moet worden omdat de snelle heuristiek die een beginmatching berekent al een vrij grote koppeling berekent.
Maar omdat het een heel belangrijk probleem is, werden nog snellere algoritmen ontwikkeld – bv. een $O(\sqrt{|V|}|E|)$ algoritme van Hopcroft en Karp dat de grotermakende paden door middel van een simultane BFS vanuit alle ongesatureerde toppen berekent.

103 Oefening *Geef een algoritme dat in een bipartiete graaf een maximum koppeling vindt die alle toppen met maximale graad satureert. De complexiteit van het algoritme moet $O(|V| * |E|)$ zijn.*

4.2 Koppelingen in algemene grafen

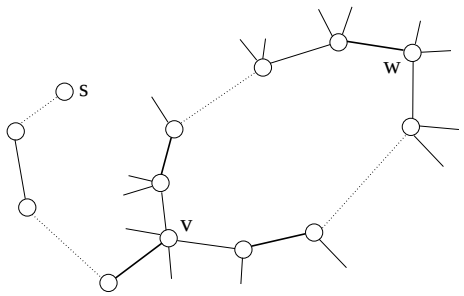
Maar koppelingen zijn natuurlijk ook interessant in gevallen waar de graaf niet bipartiet is. Waar loopt het algoritme voor bipartiete grafen mis? In Figuur 22 zie je een geval waar het mis kan lopen. Je kan t vanuit s via een alternerend pad bereiken, maar als je de routine voor bipartiete grafen toepast, wordt w via een niet-matching boog bereikt en je zou t dus niet bereiken omdat de volgende boog deel van de matching zou moeten uitmaken. De routine zou dus vertrekkend van s geen grotermakend pad vinden ook al bestaat dat! In het voorbeeld zou je wel van t kunnen vertrekken en ook met het algoritme voor bipartiete grafen een grotermakend pad naar s vinden – maar je kan gemakkelijk een voorbeeld geven waar ook dat niet kan.



Figuur 22: Een geval waar je dezelfde top w via een boog uit M en via een boog uit M^c langs een alterend pad vanuit s kan bereiken.

104 Oefening *Geef een voorbeeld van een graaf en een matching zodat er een grotermakend pad bestaat maar het algoritme voor bipartiete grafen er geen vindt – om het even met welke top het algoritme begint.*

Stel nu dat je een top w vanuit s langs een alternerend pad P via een matching boog **en** langs een alternerend pad P' via een niet-matching boog kan bereiken. Stel daarbij dat er maar één zo'n punt is (anders kies P gewoon korter). Dan bestaat er een laatste top v op P voor w die ook op P' ligt. Deze top ligt in P op het einde van een boog uit M omdat anders de volgende top – het einde van de unieke boog uit M die v bevat – ook nog in beide paden zou liggen (wij hadden gesteld dat deze toppen op dezelfde manier bereikt worden – dus dat ook P' de top via een niet-matching boog zou bereiken). Wij hebben dus twee paden $v \to w$ die alleen de eindpunten gemeen hebben. Ze vormen een oneven cykel waarin elke top – behalve v – adjacent is met één boog uit M en één boog uit M^c. De top v is adjacent met twee bogen uit M^c. Deze situatie zie je in Figuur 23.



Figuur 23: Een cykel waar je elke top langs een boog uit M en langs een boog uit M^c kan bereiken.

Wat kan je nu over alternerende paden vanuit s zeggen die via v langs een dergelijke cykel C lopen:

- C wordt altijd langs een boog uit M bereikt.
- C wordt altijd langs een boog uit M^c verlaten.
- Voor elke boog e uit M^c met precies één eindpunt op C is er een alternerend pad vanuit s dat via de cykel loopt en de cykel met e verlaat.

Met andere woorden: de cykel heeft ongeveer dezelfde eigenschappen als een enkele top die via een boog uit M bereikt wordt. En dat is de eigenschap die Edmond's blossom algoritme gebruikt: deze cykels worden gewoon *samengeperst* tot een enkele top en dan wordt doorgegaan. Als een grotermakend pad in de samengeperste graaf is gevonden, moeten de cykels gewoon gedecomprimeerd worden om het pad in de originele graaf te vinden. De naam

blossom algorithm (bloem algoritme) moet zeker ook aan de gewone manier van doen herinneren: daar bouwen wij een boom. Hier bouwen wij bloemen (de cykel is de bloesem en het pad ernaartoe is de stengel).

105 Algoritme Edmond's blossom algoritme

Wij zullen niet alle details beschrijven, maar gewoon de wijzigingen die nodig zijn in vergelijking met Algoritme 100. Voor even afstand *schrijven wij hier* door een boog uit M bereikt *en voor* oneven afstand *schrijven wij* door een boog uit M^c bereikt

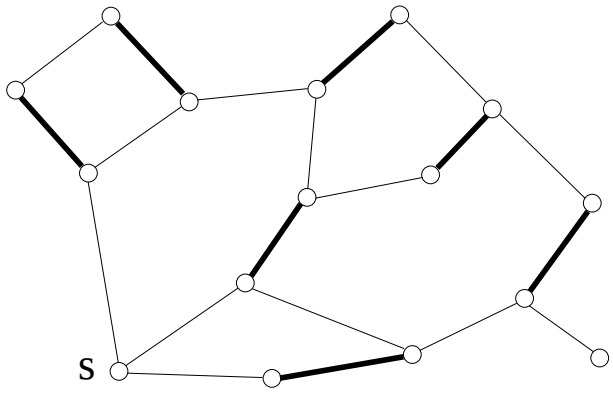
Gegeven een graaf $G = (V, E)$ met een koppeling M die ook leeg kan zijn en een ongesatureerde top s.

Gezocht is een grotermakend pad dat in s begint.

- *Pas Algoritme 100 toe met de volgende wijzigingen:*
  - *Als een top die al via een boog uit M bereikt is ook via een boog uit M^c bereikt kan worden (of omgekeerd) dan wijzig de graaf door alle toppen door de cykel die door de laatste boog gevormd wordt tot één top samen te vatten. Met de gewijzigde graaf wordt dan doorgegaan waarbij alle buren van de nieuwe top kandidaten zijn om nu aan de BFS lijst als* bereikbaar door een boog uit M^c *te worden toegevoegd.*
  - *Als een grotermakend pad is gevonden, worden langs dit pad de gecomprimeerde cykels in de omgekeerde volgorde van het comprimeren* ontplooid *en de toppen vervangen door de deelpaden langs de cykel.*

Wij hebben dus nog altijd een boom met alternerende paden. Maar sommige van de toppen staan voor hele cykels en de boom is inderdaad niet noodzakelijk een boom in de graaf waarmee wij zijn begonnen.
Omdat het algoritme recursief werkt, kan het ook gebeuren dat sommige cykels die samengevat worden al toppen bevatten die cykels voorstellen – en die cykels kunnen ook toppen bevatten die cykels voorstellen, etc.
Als je dit implementeert moet je er zorgvuldig op letten hoe je de contracties van de cykels implementeert. De rechtstreekse manier van doen vraagt tijd $O(|V|^4)$. Je moet ten slotte de toppen in jouw lijst zoeken die tot een cykel behoren die samengevat wordt, etc. Een manier die betere datastructuren en een betere analyse gebruikt werd later voorgesteld (Ahuja, Magnanti, Orlin). Die vraagt tijd $O(|V|^3)$. Maar intussen zijn ook betere (en **veel** ingewikkeldere...) algoritmen gekend die in tijd $O(\sqrt{|V|} * |E|)$ draaien (Micali, Vazirani).

106 Oefening *De beste manier om het algoritme te verstaan, is het gewoon toe te passen. Pas het algoritme toe op de graaf in Figuur 24.*



Figuur 24: Pas het blossom algoritme toe en zoek en grotermakend pad dat in s start.

Complexiteit en tijd:
Maar als het om algoritmen en echte toepassingen gaat, moet je wel voorzichtig zijn. De asymptotische slechtste geval analyse beschrijft dan niet noodzakelijk wat je echt wil weten. In de realiteit zijn ook de constanten belangrijk en de kans dat zo'n slecht geval zich echt voordoet!
Als je bv. eerst een heuristiek toepast om een goede beginkoppeling te vinden dan wijzigt dat de complexiteit van het slechtste geval helemaal niet – hoewel dat in de praktijk de programma's veel sneller kan maken. Een ander idee zou zijn, gewoon het algoritme voor bipartiete grafen – dat een veel kleinere overhead heeft dan welk algoritme dan ook voor algemene grafen – eerst toe te passen om een goede beginkoppeling te vinden. Testen die met toevallig gekozen grafen gedraaid werden, toonden echter dat dit algoritme al in meer dan 99% van de gevallen inderdaad een maximum koppeling vond. In deze gevallen was de ingewikkeldere routine helemaal niet meer nodig (bv. omdat de matching perfect was of er andere redenen waren om te zien dat het niet beter kon) of alleen om vast te stellen dat het niet beter kon. Maar zelfs in de gevallen waar er geen maximum matching werd gevonden, hielp het natuurlijk sterk al een relatief grote matching te hebben... Alleen met de algoritmen die jullie hier hebben gezien, kunnen jullie matchings in (toevallig gekozen) grafen met 10.000 toppen en meer in minder dan een seconde vinden en dat is voor de meeste toepassingen zeker voldoende.

Een extreem voorbeeld voor het verschil tussen de complexiteit in de theorie en de praktijk is het volgende: Een 3-reguliere samenhangende graaf heet een Yutsis-graaf als de toppenverzameling V zo in V_1, V_2 gesplitst kan worden dat V_1 en V_2 beide een boom induceren. Aan de ene kant is bewezen dat dit probleem NP-compleet is. Het is zelfs bewezen dat het NP-compleet blijft als je alleen maar naar planaire grafen kijkt. (Planaire grafen zijn grafen die je in het vlak kan tekenen zonder dat bogen kruisen. Veel problemen zijn gemakkelijker voor deze soort grafen.) Wij zullen hier op dit moment niet uitleggen wat NP-compleet betekent – voor de doeleinden hier is het voldoende te weten dat de NP-complete problemen een klasse van problemen is die allemaal *ongeveer even moeilijk* zijn en waarvoor geen polynomiale algoritmen gekend zijn – en misschien gewoon niet bestaan (zie ook Hoofdstuk 6). Maar aan de andere kant zijn er heuristieken die heel goed werken voor toevallige grafen met meer dan 100.000 toppen en waarvoor testen getoond hebben dat de kans een grote graaf tegen te komen waarvoor de heuristieken niet werken ongelofelijk klein is – in de testen is het gewoon nooit gebeurd. Een beetje meer over dit probleem zien jullie op het einde van de lesnota's in Hoofdstuk 6.
Hoewel de slechtste geval analyse dus heel belangrijk is (en zeker theoretisch interessant!) moeten jullie ook weten dat het normaal niets zegt over de kans dat dit slechtste geval zich voordoet en dat het voor de praktijk ook belangrijk en interessant is gewoon testen te draaien met de typische grafen waarop het algoritme toegepast moet worden.

107 Oefening *Stel voor de volgende uitspraken dat het matching algoritme voor bipartiete grafen op een niet noodzakelijk bipartiete graaf wordt toegepast zonder eerst een heuristiek toe te passen om een beginkoppeling te vinden. Welke van de volgende uitspraken is waar?*

- *Er bestaat een constante $c < 1$ zodat voor elke $n \in \mathbb{N}$ een graaf G met ten minste n toppen bestaat zodat als het matching algoritme voor bipartiete grafen op G wordt toegepast de gevonden koppeling kleiner dan $c * m(G)$ is – om het even in welke volgorde de toppen en bogen van de graaf worden beschouwd.*

- *Er bestaat een constante $c > 0$ zodat voor elke graaf G geldt dat als het matching algoritme voor bipartiete grafen op G wordt toegepast de gevonden koppeling ten minste $c * m(G)$ is.*

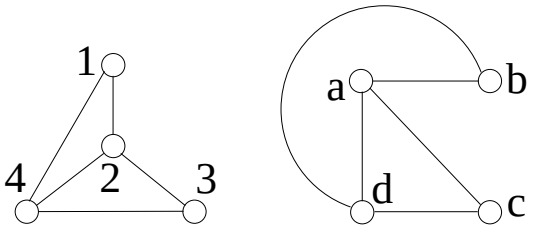
- *Gegeven een (niet noodzakelijk bipartiete) graaf G met koppeling M en een top s zodat er een grotermakend pad vanuit s bestaat. Dan bestaat er een volgorde waarop de buren van elke top worden afgewerkt zodat de*

routine voor het vinden van grotermakende paden in bipartiete grafen het grotermakende pad vindt.

- *Gegeven een (niet noodzakelijk bipartiete) graaf G met koppeling M en een top s zodat er een grotermakend pad vanuit s bestaat. Stel dat wij DFS in plaats van BFS voor het zoeken van een grotermakend pad in het algoritme voor bipartiete grafen gebruiken. Dan bestaat er een volgorde waarop de buren van elke top worden afgewerkt zodat de routine het grotermakende pad vindt.*

108 Oefening *Gegeven een graaf G en een top $v \in V$ met $\deg(v) \geq 1$. Bestaat er gegarandeerd een* **maximum** *matching zodat v gesatureerd is?*

5 Isomorfie



Figuur 25: Twee verschillende grafen met *dezelfde structuur*.

Hoewel de toppen natuurlijk heel belangrijk zijn in een graaf is het (meestal) niet echt belangrijk wat de toppen precies zijn. Of het nu getallen zijn of letters of huizen of kruispunten of... heeft wel veel met de betekenis en interpretatie te maken, maar niet met wat wij *de structuur* van de graaf zouden noemen. In Figuur 25 zien jullie twee duidelijk verschillende grafen – ten slotte zijn al de toppenverzamelingen verschillend – maar qua structuur lijken ze toch sterk op elkaar: beide grafen hebben 4 toppen en er ontbreekt maar één boog om compleet te zijn (compleet betekent dat $E = \binom{V}{2}$ dus dat alle mogelijke bogen aanwezig zijn). Als twee wiskundige objecten dezelfde structuur hebben dan wordt dat normaal isomorf genoemd. En dat doen wij ook hier:

109 Definitie *Twee grafen $G = (V, E)$ en $G' = (V', E')$ heten isomorf als en slechts als er een bijectieve afbeelding $f : V \to V'$ bestaat met $\{v, w\} \in E \Leftrightarrow \{f(v), f(w)\} \in E'$.*
Wij schrijven dan $G \cong G'$.
De afbeelding $f()$ heet een isomorfisme en als $G = G'$ dan heet $f()$ een automorfisme.

Grafenisomorfie is natuurlijk een heel belangrijk probleem en er is al veel onderzoek op dit vlak gebeurd, maar toch is het in het algemeen nog niet geweten wat de complexiteit ervan is. De meeste mensen geloven dat het niet mogelijk is altijd in polynomiale tijd te beslissen of twee grafen isomorf zijn, maar men gelooft ook niet dat het probleem NP-compleet is.
Wat wel bekend is, is dat als de maximale graad $\Delta(G)$ begrensd is, het probleem wel in polynomiale tijd opgelost kan worden (Luks 1982, waarbij het polynoom van $\Delta(G)$ afhangt). Maar jammer genoeg is het algoritme niet alleen heel ingewikkeld, maar voor alle praktische toepassingen ook trager dan een algoritme dat in het slechtste geval exponentieel is. Hoewel dit algoritme (nauty van Brendan McKay) in het slechtste geval exponentieel is, is het voor alle praktische toepassingen voldoende. Alleen in gevallen waar **heel veel** grafen getest moeten worden, is het soms de moeite om zich nog met optimalisaties bezig te houden.
De *gewone manier van doen* als twee grafen op isomorfie getest moeten worden, is niet expliciet de bijectie uit de definitie te construeren, maar een invariant te vinden die de graaf op een unieke manier beschrijft.

110 Definitie *Een invariant is een functie $f()$ die aan elke graaf een getal (of een string) toekent en waarvoor geldt dat als twee grafen G, G' isomorf zijn dat dan ook $f(G) = f(G')$.*

Het is niet moeilijk zo'n invariant te vinden: definieer gewoon $f(G) = 0$ voor alle G. Als wij dergelijke invarianten willen gebruiken om isomorfie te testen dan hebben wij meer nodig:

111 Definitie *Wij noemen een invariant $f()$ een isomorfiebepalende invariant van een graaf als $f(G) = f(G')$ als en slechts als $G \cong G'$.*

Grafenisomorfie is ook één van de problemen waar al veel mensen dachten dat ze een polynomiaal algoritme hebben gevonden – maar dan bleek het toch niet te kloppen. Vaak worden invarianten gedefinieerd en gesteld dat ze inderdaad ook de vereiste eigenschap van Definitie 111 hebben zonder dat echt te bewijzen... Als je invarianten combineert wordt het aantal grafen waarvoor alle invarianten tegelijk identiek zijn natuurlijk kleiner (bv. aantal

toppen **en** aantal bogen **en** matching getal **en** lengte van de kortste cykel **en** samenhangsgetal **en**...) maar ook de combinatie zal normaal geen isomorfiebepalende invariant zijn.
Een string invariant die vooral in de scheikunde heel vaak gebruikt wordt is de volgende: de graaf wordt voorgesteld als een adjacentiematrix. De adjacentiematrix van een graaf met toppen $1, \ldots, n$ is de matrix met een 1 in rij i en kolom j als en slechts als de boog $\{i, j\}$ in de graaf zit. Alle andere getallen in de matrix zijn 0. Dit is natuurlijk een gewone matrix zoals gekend uit de algebra en je kan er de eigenwaarden van berekenen. De (gesorteerde) string van eigenwaarden is een heel populaire invariant in de scheikunde omdat deze string niet allen toelaat voor veel grafen aan te tonen dat ze niet isomorf zijn, maar ook een scheikundige interpretatie heeft. Het is niet onmiddellijk duidelijk dat deze string een invariant is. Je kan ook veel wetenschappelijke artikels terugvinden over *isospectral graphs*. Voor veel (deel-) klassen is het interessant om te weten of daarvoor deze string misschien isomorfiebepalend is.

112 Oefening *Toon expliciet aan dat $m(G)$ een invariant is.*

113 Oefening *Wat is het kleinste aantal toppen waarvoor er twee niet isomorfe grafen zijn waarvoor het aantal toppen en het aantal bogen en het matching getal en de lengte van de kortste cykel en het samenhangsgetal gelijk zijn?*

Als je niet op efficiëntie let dan is het gemakkelijk een isomorfiebepalende invariant te definiëren:
Stel eerst dat $G = (V, E)$ met $V = \{1, \ldots, n\}$. Dan kunnen wij de graaf als string voorstellen door de buren van elke top te sorteren en dan de lijsten van buren in de volgorde die door de nummers van de toppen wordt gegeven en gescheiden door bv. een „|“op te schrijven. De eerste graaf in Figuur 25 zou dan door de string

$$\underbrace{2\ \ 4}\ |\underbrace{1\ \ 3\ \ 4}|\ \underbrace{2\ \ 4}\ |\underbrace{1\ \ 2\ \ 3}$$

buren van 1 buren van 2 buren van 3 buren van 4

voorgesteld worden. Dat noemen wij rep(G).
Maar dat is natuurlijk **geen** invariant en zelfs niet voor alle grafen gedefinieerd! Maar wat wij nu kunnen doen, is de invariant minlist() definiëren:

114 Definitie *Gegeven een graaf $G = (V, E)$ met n toppen. F zij de verzameling van alle bijectieve afbeeldingen $V \to \{1, \ldots, n\}$ en voor $f \in F$ zij $f(G)$ de graaf $(\{1, \ldots, n\}, \{\{f(x), f(y)\} | \{x, y\} \in E\})$.*
Strings kunnen natuurlijk lexicografisch geordend worden. Wij stellen dat „|“groter is dan elk getal en dat de getallen enkele letters in de string zijn

(om het even hoe groot) en niet als strings van cijfers worden voorgesteld, maar dat zijn dingen die niet van fundamenteel belang zijn.
Nu definiëren wij:
$\mathit{minlist}(G) := \min\{\mathit{rep}(f(G)) | f \in F\}$

Informeel kan je dus zeggen dat wij op alle mogelijke manieren de nummers $\{1, \ldots, n\}$ aan de toppen toekennen, daarvan de string-voorstelling berekenen en dan de kleinste kiezen.
Dit is natuurlijk verschrikkelijk inefficiënt als je de definitie direct toepast, maar er zijn ook algoritmen die deze invariant iets efficiënter berekenen (maar ook die zijn exponentieel).

115 Oefening *Bepaal $\mathit{minlist}(G)$ waarbij G één van de grafen uit Figuur 25 is. Voor hoeveel manieren om nummers te geven, krijg je deze minimale representatie?*
Is er een samenhang met de automorfismen van de graaf?

116 Definitie *Met de invariant $\mathit{minlist}(G)$ correspondeert een manier om nummers aan de toppen toe te kennen (een afbeelding $f : V \to \{1, \ldots, |V|\}$) zodat $\mathit{rep}(f(G))$ inderdaad $\mathit{minlist}(G)$ is. Dat noemen wij een kanonische labeling (voor deze invariant) en een graaf G met $\mathit{rep}(G) = \mathit{minlist}(G)$ noemen wij een kanonische representant van zijn isomorfieklasse. De kanonische labeling $f()$ beschrijft dus tegelijk een isomorfisme van de graaf G naar de kanonische representant van zijn isomorfieklasse.*
De twee begrippen representant *en* kanonische labeling *passen wij dus niet alleen op $\mathit{minlist}()$ toe maar ook op andere isomorfiebepalende invarianten die gebaseerd zijn op een labeling van de graaf – of equivalent: waar je een verzameling R van representanten hebt zodat als je met een verzameling S van grafen werkt voor elke isomorfieklasse precies één representant in R zit.*

117 Oefening *Geef een definitie van een kanonische nummering die in plaats van $\mathit{minlist}(G)$ de adjacentiematrix gebruikt. Kies de definitie zo dat dezelfde nummeringen kanonisch zijn – en bewijs deze eigenschap.*

118 Oefening *Gegeven een graaf G met $\mathit{rep}(G) = \mathit{minlist}(G)$. De bogen $\{x, y\}$ van G zijn zo opgeschreven dat altijd $x < y$. Dan kan je een orde op de bogen definiëren door ze lexicografisch te sorteren. Stel dat e de grootste boog is.*
Toon aan of geef een tegenvoorbeeld:
Dan geldt voor $G' = G - \{e\}$ dat $\mathit{rep}(G') = \mathit{minlist}(G')$.

Inderdaad zijn alle gekende (en juiste) isomorfiebepalende invarianten gebaseerd op een manier van doen die erop neerkomt labels aan de toppen toe te kennen.

5.1 Isomorfie van bomen

Wat wij nu zullen zien, is een andere isomorfiebepalende invariant dan minlist(). Het is een speciale invariant voor bomen die in lineaire tijd berekend kan worden. Het is een *interpretatie* van een algoritme van R.C. Read.

119 Definitie *Wij definiëren op een recursieve manier wat een center $ctr(T)$ van een boom $T = (V, E)$ is:*

- *als $|V| \leq 2$ dan is $ctr(T) = V$ het center van de boom. In veel boeken wordt in het geval van $|V| = 2$ ook de boog het center genoemd. Maar natuurlijk is in dat geval de boog gelijk aan V. Wij noemen de elementen van $ctr(T)$ ook centers.*
- *als $|V| > 2$ dan is de graaf $T' = T[\{v \in V | \deg(v) > 1\}]$ (dus de graaf waar alle toppen met graad 1 verwijderd zijn) ook een boom en wij definiëren $ctr(T) = ctr(T')$*

120 Oefening *Toon aan dat $ctr(T)$ voor een boom T in lineaire tijd berekend kan worden.*

121 Oefening *In algemene grafen wordt het center anders gedefinieerd:*
G zij een graaf.
De eccentriciteit $ecc(v)$ van een top v is de maximale afstand van een andere top, dus $ecc(v) = \max\{d(v, w) | w \in V\}$. De radius $rad(G)$ van G is de minimale eccentriciteit, dus $rad(G) = \min\{ecc(v) | v \in G\}$.
Dan is $ctr(G) = \{v \in V | ecc(v) = rad(G)\}$
Bewijs dat de twee definities voor het center op bomen overeenstemmen.

122 Definitie *De hoogte $hg(v)$ van een top v in een boom T is de kortste afstand tot een top $c \in ctr(T)$.*

123 Oefening *Toon aan dat een isomorfisme tussen twee bomen een bijectieve afbeelding tussen de centers induceert.*

124 Oefening *Toon aan dat voor een boom T in lineaire tijd aan elke top v zijn hoogte $hg(v)$ en ook bovendien aan elke top $v \notin ctr(T)$ de eerste top $voorg(v)$ op het pad naar het center toegekend kan worden.*

125 Definitie *Een gewortelde boom is een boom $T = (V, E)$ samen met een top $r \in V$ die wij de wortel noemen.*
Twee gewortelde bomen heten isomorf als er een isomorfisme is dat de wortels op elkaar afbeeldt.

Het algoritme zal nu – beginnend met toppen met maximale hoogte – ordenummers toekennen aan de toppen zodat twee toppen v, w op dezelfde hoogte hetzelfde nummer krijgen als en slechts als de gewortelde bomen met wortels v resp. w bestaande uit deze toppen en al hun kinderen isomorf zijn. "Kinderen van top v" betekent hier "de toppen die vanuit v bereikbaar zijn zonder ooit een top met kleinere hoogte dan v te bezoeken".

126 Algoritme Ordenummers toekennen
Input is een boom T waarvoor al voor elke hoogte $0, \ldots, m$ een lijst met toppen aangemaakt werd. Daarbij is m de maximale hoogte.
Bovendien heeft elke top een lijst $l(v)$ van ordenummers van kinderen die in het begin leeg is.

- *Schrijf het getal 0 in de lijst van alle bladeren – om het even op welke hoogte die zitten.*
- *Doe de volgende stappen voor hoogte h beginnend met $h = m$ t.e.m $h = 0$:*
  - **(*)** *Sorteer de lijst van de k toppen op hoogte h in lexicografische volgorde van hun lijsten.*
  - *Nu krijgen de toppen nummers $n()$:*
    Voor de eerste top v_1 definieer $n(v_1) = 1$ en voor de i-de top v_i definieer $n(v_i) = n(v_{i-1})$ als de lijsten van v_i en v_{i-1} gelijk zijn en $n(v_i) = n(v_{i-1}) + 1$ anders.
  - **(**)** *Als $h > 0$ worden nu de lijsten van de toppen op hoogte $h-1$ opgebouwd:*
    Voor elke top $v_i = v_1, \ldots, v_k$ (in deze volgorde) voeg $n(v_i)$ toe aan de lijst van voorg(v_i) (dus $l($voorg$(v_i))$). Deze lijsten zijn dan automatisch gesorteerd.

Gebaseerd op dit algoritme dat volgordes voor elke hoogte bepaalt, kan je nu op verschillende manieren een isomorfiebepalende invariant definiëren. Een manier van doen die vaak gebruikt wordt is een reeks van haakjes. Wij zullen hier – ook om een kanonische labeling te hebben – gewoon een functie $f()$ gebruiken die labels aan de toppen toekent en dan kunnen wij rep$(f(T))$ als invariant gebruiken.
Daarvoor stellen wij dat in stap **(**)** niet alleen gesorteerde lijsten van ordenummers van kinderen opgebouwd worden maar ook een lijst $k[]$ van kinderen in de volgorde van de ordenummers.

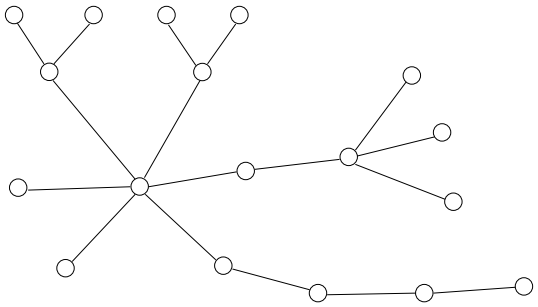
127 Algoritme Nummers geven
Gegeven een boom T waarop Algoritme 126 al werd toegepast.
In het begin is $f()$ voor geen top gedefinieerd en de variabele nextnummer *is gelijk aan 1. Als er twee toppen op hoogte* 0 *zijn dan voeg de tweede (volgens de volgorde die door Algoritme 126 gegeven is) toe aan de lijst $k[]$ van de eerste (bv. op het einde) en noem de eerste zijn voorganger.*
Dan pas de volgende recursieve routine nummer() toe met als parameter de eerste top op hoogte 0*:*

nummer(v)

- *voor alle kinderen $k[i] = k[1], \ldots, k[n]$ van v (in de volgorde die door Algoritme 126 is gegeven) doe nummer($k[i]$).*
- *$f(v) =$ nextnummer; verhoog nextnummer met* 1*.*

Merk op dat als je de nummers $f()$ in de volgorde van de lijst $k[]$ schrijft, die al gesorteerd zijn en dat de voorganger altijd een hoger nummer heeft dan alle kinderen. Als wij rep($f(T)$) willen berekenen kan dat dus gemakkelijk in lineaire tijd.

128 Oefening *Gebruik het algoritme om nummers aan deze boom toe te kennen:*



129 Oefening *Toon aan dat rep($f(T)$) inderdaad een isomorfiebepalende invariant is en dat de manier om nummers toe te kennen een kanonische labeling is.*

130 Oefening *Algoritme 127 werkt op een DFS manier. Geef een algoritme dat nummers op een BFS manier toekent.*

Het is ook duidelijk dat Algoritme 127 in lineaire tijd draait en in Algoritme 126 is ook voor bijna alle stappen duidelijk dat de lus voor een gegeven hoogte h maar $O(k)$ stappen vraagt als er k toppen op die hoogte zijn. Het

enige probleem is stap **(*)**. Als alle sorteerstappen samen in tijd $O(|V|)$ gedaan zouden kunnen worden, zou het hele algoritme in lineaire tijd draaien. Elke top duikt ten hoogste twee keer in de te sorteren strings op – één keer misschien als blad en één keer met een ordenummer in een string van een top met een hoogte die één kleiner is. De som van de lengten van alle te sorteren strings is dus $O(|V|)$. Als op een hoogte n strings gesorteerd moeten worden waarbij de lengte van string i precies l_i is dan moeten wij aantonen dat wij met $L = \sum_{i=1}^{n} l_i$ die in tijd $O(L)$ kunnen sorteren. Ook in de rest van dit deel zal L voor $\sum_{i=1}^{n} l_i$ staan.
Jammer genoeg is er in het algemeen geen algoritme gekend dat zo snel kan sorteren. Hier moeten wij dus een bijzondere manier van sorteren toepassen die gebruik maakt van de omstandigheden in dit algoritme – of precies: dat de getallen in de strings ten hoogste $|V|$ kunnen zijn. Wij gebruiken daarvoor iets dat het idee van bucket sort gebruikt. En hoewel het onderwerp van deze les natuurlijk vooral grafenalgoritmen zijn, moeten wij ten minste één keer ook echt de details van algoritmische stappen uitwerken die belangrijk zijn voor de complexiteit – ook al hebben ze niets te maken met grafen... Wij zullen ons algoritme in drie stappen beschrijven – zo kan het gemakkelijker verstaan worden:
Eerst zullen wij bucket sort zelf bespreken.

131 Algoritme Bucket sort
Gegeven een reeks $g_1, \ldots, g_n$ van n getallen met $0 \leq g_i \leq m$ $\forall 1 \leq i \leq n$. Dan kunnen deze getallen in tijd $O(n+m)$ gesorteerd worden (dus in $O(n)$ als $m = O(n)$):

- *maak $m+1$ lege lijsten $l_0, \ldots, l_m$ aan.*
- *voor $1 \leq i \leq n$ verplaats getal g_i naar de lijst l_{g_i}. De te sorteren lijst is op het einde leeg.*
- *voor $0 \leq i \leq m$ (in deze volgorde) verplaats de inhoud van lijst l_i naar de te sorteren lijst.*

De volgende stap zal nu zijn strings te sorteren die allemaal dezelfde lengte hebben. Als het grootste getal m is en de lengte van de strings is k, dan zijn er $(m+1)^k$ mogelijke waarden – wij kunnen dus niet gewoon bucket sort met één emmer voor elke mogelijke waarde toepassen als wij een algoritme willen dat lineair in n is en $n = o((m+1)^k)$.
Als wij zeggen *verplaats string s* bedoelen wij natuurlijk dat een pointer naar de string verplaatst wordt (wat in constante tijd kan gebeuren) en niet de hele string! Het volgende algoritme zal k iteraties nodig hebben. Na iteratie i

zijn de strings volgens de **laatste** i getallen in de strings juist gesorteerd. Na k iteraties dus volledig gesorteerd. Deze manier van sorteren heet radix sort en kan bv. ook op getallen toegepast worden als ze als strings van bits geïnterpreteerd worden:

132 Algoritme Strings van gelijke lengten sorteren
Gegeven een reeks $s_1, \ldots, s_n$ van n strings met lengte k (dus $L = k * n$) van getallen met $0 \leq g \leq m$ voor alle getallen g die in de strings voorkomen. Dan kunnen deze strings in tijd $O(k * (n + m))$ gesorteerd worden. Merk op dat in dit geval $\Theta(k * n)$ stappen al nodig zijn om één keer naar elk element te kijken.

- *maak $m + 1$ lege lijsten $lijst_0, \ldots, lijst_m$ aan. Dit zijn first-in-first-out (FIFO) lijsten!*
- *herhaal de volgende stappen voor $i = k \ldots 1$ in deze volgorde*
  - *voor $1 \leq j \leq n$ verplaats string s_j naar de lijst $lijst_{s_j[i]}$ waarbij $s_j[i]$ het i-de element van string s_j is. De te sorteren lijst is op het einde leeg.*
  - *voor $0 \leq j \leq m$ (in deze volgorde) verplaats de inhoud van lijst $lijst_j$ naar de te sorteren lijst. De lijst is dan in niet dalende volgorde van de laatste $k - i + 1$ elementen gesorteerd.*

De reden waarom dit werkt is dat strings die hetzelfde i-de element hebben – dus in de iteratie met index i in dezelfde emmer terechtkomen – al gesorteerd zijn volgens de latere elementen. Als ze in de emmers geplaatst worden dan worden dus de elementen met de *kleinere rest* eerst geplaatst en komen dan als de emmer leeg gemaakt wordt ook voor de anderen terecht. Het beste kan je dat zien door de volgende oefening te doen:

133 Oefening *Sorteer de volgende binaire getallen door ze als strings met lengte 5 te interpreteren:*
00101, 10101, 10001, 11101, 10100, 11111

Wij zouden dit algoritme kunnen toepassen op strings van verschillende lengten door die op te vullen totdat ze dezelfde lengte hebben. Maar dan zou de complexiteit kwadratisch kunnen zijn, omdat de som van de lengten achteraf $\Omega(n^2)$ zou kunnen zijn als de som van de lengten van de strings in het begin $O(n)$ was. Dat is dus geen goed idee. Maar als de langste string lengte $\Omega(L)$ heeft, zou het al kwadratische kost kunnen hebben voor elke lengte m emmers aan te maken!
Om efficiënt te kunnen werken, hebben wij dus twee dingen nodig:

- Voor elke lengte moeten wij vaststellen welke strings gesorteerd moeten worden.
- Voor elke lengte moeten wij vaststellen hoeveel emmers echt nodig zijn!

En natuurlijk moeten ook deze dingen op een efficiënte (lineair in L) manier berekend worden!
Deze stappen verwezenlijken wij op de volgende manier:

- Voor elk getal g dat in een string s op positie i staat, maak een paar (i, g) (een string van lengte 2) aan. Dat vraagt $O(L)$ stappen en het resultaat zijn L strings.
- Sorteer deze strings met Algoritme 132. Dat vraagt $O(2 * (L + m) = O(L)$ stappen als voor alle getallen g in de strings geldt dat $g \leq L$.
- Ken voor elke positie i die in de gesorteerde lijst voorkomt nummers $1, \ldots, a$ toe, waarbij a het aantal verschillende getallen is die op positie i voorkomen. Kleinere getallen krijgen daarbij ook kleinere nummers en dezelfde getallen dezelfde nummers. Omdat de lijst gesorteerd is, kan dat zeker in tijd lineair in de lengte van de lijst, dus $O(L)$. Wij noemen deze nummers $n_i()$. De volgorde als wij volgens $n_i()$ sorteren is dus dezelfde als wanneer wij volgens de echte getallen sorteren.
- Maak voor elke lengte een lijst van strings aan met deze lengte. Dat gaat zeker in tijd $O(L)$.

Nu zijn de voorbereidingen klaar en wij kunnen echt beginnen de strings te sorteren:

134 Algoritme Strings sorteren
Input: n strings $s_1, \ldots, s_n$. De lengte van s_i is l_i, de getallen zijn allemaal positief en het grootste getal is ten hoogste $L = \sum_{i=1}^{n} l_i$. De grootste lengte is $l_{\max}$.
Doel: de strings in lexicografische volgorde sorteren waarbij wij een leeg teken als kleiner dan een getal interpreteren.
De preprocessing is al gedaan, dus hebben wij voor $1 \leq i \leq l_{\max}$ lijsten $lijst_i$ met de strings van lengte i en hebben wij de nummers $n_i()$ die wij in constante tijd kunnen opzoeken. Het getal $n_{i,\max}$ staat voor het grootste getal $n_i()$.
De lijst S waar wij op het einde de strings in gesorteerde volgorde zullen hebben, is in het begin leeg.
Dan herhaal de volgende stappen voor $i = l_{\max}$ t.e.m. 1 in deze volgorde:

- *Maak $n_{i,\max}$ emmers aan (FIFO).*
- *Verplaats de elementen van lijst$_i$ naar de emmers. Een string s komt in emmer j terecht als $n_i(s[i]) = j$.*
- *Verplaats de elementen van S naar de emmers. Een string s komt in emmer j terecht als $n_i(s[i]) = j$.*
- *Verplaats de inhoud van de emmers $1, \ldots, n_{i,\max}$ naar S.*

Het principe is duidelijk hetzelfde als in Algoritme 132 – het is dus ook even duidelijk dat de strings op het einde echt gesorteerd zijn. Elke lus voor een vaste i heeft een kost $O(a_i)$ als a_i het aantal strings met lengte ten minste i is. De hele reeks van lussen heeft dus een kost $O(\sum_{i=1}^{l_{\max}} a_i)$. Een reeks met lengte k duikt dus k keer in de som op – het is dus inderdaad $O(L)$.
Als wij dit algoritme toepassen voor het sorteren in Algoritme 126 dan draait het algoritme dus inderdaad in lineaire tijd, omdat elke top – gerepresenteerd door het nummer dat de gewortelde boom met die top als wortel voorstelt – maar één keer in een te sorteren lijst opduikt en het nummer is natuurlijk nooit groter dan het aantal toppen. Elke keer dat een lijst gesorteerd wordt, gebeurt dat dus lineair in de lengte van de lijst en de som van alle lengten is ten hoogste $|V|$.

135 Oefening *Pas Algoritme 134 toe op de volgende strings:*
$(1, 2, 4, 5, 6)$, $(1, 2, 3)$, $(2, 4, 7)$, $(3, 2, 4)$, $(7, 6)$, $(2, 4, 6, 7, 2, 5)$

136 Oefening *Algoritme 132 sorteert n strings van positieve getallen $g \leq m$ van lengte k in tijd $O(k*(n+m))$. In het geval dat $m = c*(k*n)$ – het grootste getal dus van de orde van het aantal posities is – is dat $O(k^2*n)$ dus $O(k*L)$ met – zoals gewoon – $L = \sum_{i=1}^{n} |s_i|$.*
Geef een algoritme dat de taak in tijd $O(L)$ vervult.

5.2 Planaire grafen en isomorfie

Als wij het over planaire grafen of in het algemeen over ingebedde grafen hebben, dan hebben wij het in principe niet alleen over discrete structuren maar over topologie. Een planaire graaf is meestal gedefinieerd als een graaf die je *in het vlak of op een bol kan tekenen zonder dat de bogen kruisen of door toppen lopen.* Maar omdat jullie niet veel over topologie hebben gehoord en omdat het ook niet echt het onderwerp van deze les is, zullen wij de zaak op een helemaal andere manier benaderen en dan alleen zeggen dat het inderdaad iets met het vlak te maken heeft...

137 Definitie *In het volgende zullen wij het over georiënteerde bogen hebben. Dat zijn paren (v, e) waarbij $v \in e \in E$ voor een graaf $G = (V, E)$. Dus $e = \{v, w\}$ voor een $w \in V$. Om het korter te kunnen opschrijven, schrijven wij voor een dergelijk paar gewoon $[v, w]$. Wij noemen $[w, v]$ ook de inverse boog van $[v, w]$ (hoewel het normaal eerder* inverse georiënteerde boog *zou moeten heten...) en schrijven $[w, v] = [v, w]^{-1}$. Voor elke boog zijn er twee georiënteerde bogen. Hoewel wij* georiënteerde bogen *in de toekomst vaak gewoon bogen zullen noemen is het (hopelijk) altijd uit de samenhang duidelijk wanneer bogen en wanneer georiënteerde bogen bedoeld worden.*
Een ingebedde graaf $G = (V, E, opv())$ is een graaf (V, E) samen met een functie opv() die aan elke georiënteerde boog een georiënteerde boog met dezelfde begintop toekent zodat voor alle georiënteerde bogen $[v, w]$ geldt dat $\{[v, w], opv([v, w]), opv^2([v, w])), \ldots, opv^{deg(v)-1}([v, w])\} = \{[v, y] | \{v, y\} \in E\}$.
Daarbij (en in de volgende tekst) staat $opv^i([v, w])$ voor $\underbrace{opv(\ldots opv(}_{i \text{ keer}}[v, w]) \ldots)$.

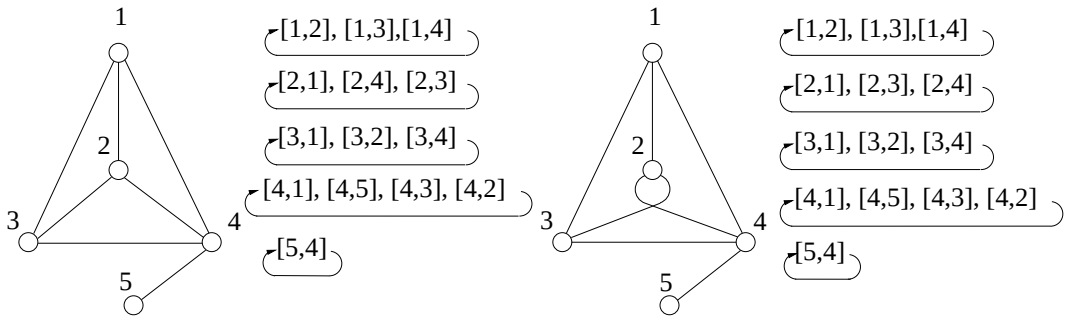
Wij noemen $opv([v, w])$ ook de opvolger van $[v, w]$.
Informeel kan je dus zeggen dat rond elke top v een volgorde van de bogen die met v incident zijn, vastgelegd wordt.

138 Oefening *Gegeven een ingebedde graaf $G = (V, E, opv())$ en $\{x, y\} \in E$.*

- *Toon aan dat $opv^{\deg(x)}([x, y]) = [x, y]$*
- *Toon aan dat de functie opv() bijectief is – dus dat de functie $opv()^{-1}$ bestaat.*
- *Voldoet $opv()^{-1}$ ook aan de eisen voor een opvolgerfunctie – is $G = (V, E, opv()^{-1})$ ook een ingebedde graaf?*

Je kan de ingebedde grafen als een tekening voorstellen door de bogen rond een top zo te tekenen dat de volgorde in wijzerzin de functie opv() beschrijft. Een voorbeeld zie je in Figuur 26. In deze les zijn tekeningen **alleen** manieren om de toppen- en boogverzamelingen en de opvolgerfunctie voor te stellen!

139 Definitie *Nu zullen wij definiëren wat wij combinatorisch als vlakken willen verstaan. Wij zullen in dit deel altijd stellen dat de graaf samenhangend is en dat er ten minste één boog in de graaf zit.*
Gegeven een ingebedde graaf $G = (V, E, opv)$.
Wij zeggen dat $[v, w]$ en $[w, x]$ een hoek vormen als $[w, x] = opv([w, v])$ (hier is de volgorde van v en w verwisseld). Kijk eens naar Figuur 26 om te zien wat daar bv. de hoeken zijn.



Figuur 26: Twee inbeddingen waarbij de grafen gelijk zijn en alleen de opvolgerfunctie verschillend. De volgorde van bogen rond top 1 zou je ook kort als $1 : 2, 3, 4$ kunnen schrijven en daarbij veronderstellen dat de lijst natuurlijk cyclisch bedoeld is.

Een georiënteerde boog maakt dus deel uit van twee hoeken. Als wij naar de vlakkengraaf $VL(G)$ van deze ingebedde graaf G kijken – dat is de graaf waar de georiënteerde bogen de toppen zijn en twee georiënteerde bogen adjacent zijn als ze een hoek vormen, dan is dat dus een 2-reguliere graaf. De samenhangscomponenten van $VL(G)$ heten de vlakken van de ingebedde graaf.
Wij schrijven $F(G)$ voor de verzameling van vlakken van G.

Dat lijkt misschien heel abstract (en is het ook. . .), maar als jullie eens kijken wat de vlakken van de eerste graaf in Figuur 26 zijn, dan vinden jullie de definitie misschien iets natuurlijker.
In de figuren en met de interpretatie van *opv beschrijft de volgorde in wijzerzin* kan je de net gedefinieerde vlakken begrijpen als de vlakken die *links* van deze gerichte cykel liggen. Maar dat is natuurlijk informeel en intuïtief. . .
De vlakken van een graaf G kan je gemakkelijk door DFS in de vlakkengraaf vinden of in de originele graaf waarbij je gebruik maakt van georiënteerde bogen:

140 Algoritme *Input: een samenhangende ingebedde graaf $G = (V, E, opv)$ en een boog $[v, w]$ met $\{v, w\} \in E$.*
Gezocht: het vlak dat $[v, w]$ bevat – dus een verzameling F van georiënteerde bogen.

Begin met $F = \{[v, w]\}$.
Dan herhaal de volgende lus met in het begin $s = v$ en $t = w$ totdat aan de stopvoorwaarde is voldaan:

- $e = opv([t, s])$. *Als* $e = [a, b]$ *dan zij* $s = a$ *en* $t = b$.
- **als** $s = v$ **en** $t = w$**:** *STOP*

  anders: *voeg* $[s, t]$ *aan* F *toe en herhaal de lus.*

141 Oefening *Bereken de verzamelingen F van vlakken van de twee ingebedde grafen in Figuur 26.*

Nu zullen wij de Euler karakteristiek van een ingebedde graaf definiëren. Voor ons is dat gewoon een invariant van de inbedding, maar inderdaad is dat een topologische invariant van een *2-dimensionale oppervlakte*. Daarover zullen wij later nog een klein beetje meer zeggen.

142 Definitie *Gegeven een samenhangende ingebedde graaf $G = (V, E, opv)$ met $|E| \geq 1$ waarvoor F de verzameling van vlakken is.*
Dan heet $ek(G) = |V| - |E| + |F|$ de Euler karakteristiek van de ingebedde graaf.
$gen(G) = \frac{2-ek(G)}{2}$ heet het genus of het geslacht van de ingebedde graaf.
Als $G = (V, E)$ een (niet ingebedde) graaf is, dan is het genus van G gedefinieerd als het minimum van alle genera van ingebedde grafen $G' = (V, E, opv)$.

Natuurlijk is de berekening van het genus van een graaf heel duur als je gewoon alle volgorden van bogen rond een top wil testen...

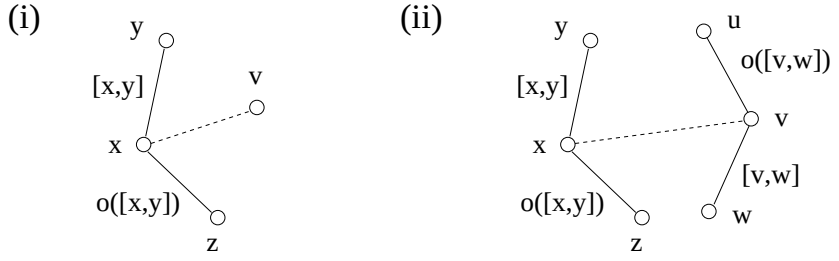
143 Oefening *Bereken de genera en de Euler karakteristieken van de twee ingebedde grafen in Figuur 26.*

Maar nu zullen wij eerst kijken hoe de Euler karakteristiek zich ontwikkelt als wij een boog zo tot een (samenhangende) ingebedde graaf $G = (V, E, \text{opv})$ toevoegen dat het resultaat $G' = (V', E', \text{opv}')$ nog steeds samenhangend is. Er zijn twee mogelijke bewerkingen:

bewerking (i): Een boog samen met een nieuwe top wordt toegevoegd.

bewerking (ii): Een boog tussen twee al bestaande toppen wordt toegevoegd.

Met *een boog toevoegen* bedoelen wij dat niet alleen E gewijzigd wordt maar ook de volgorde rond de eindtoppen van de nieuwe boog. Informeel zou je kunnen zeggen dat de nieuwe boog ergens *tussengevoegd* moet worden. Formeel kan je dat als volgt opschrijven: als $\{x, v\}$ de nieuwe boog is en x al in de graaf zit dan bestaan er bogen $[x, y]$ en $[x, z] = \text{opv}([x, y])$ zo dat voor $\text{opv}'()$ geldt dat $\text{opv}'([x, y]) = [x, v]$ en $\text{opv}'([x, v]) = \text{opv}([x, y]) = [x, z]$. Als



Figuur 27: De mogelijkheden om een boog toe te voegen aan een ingebedde graaf.

v ook al in de graaf zit, dan geldt iets analoogs voor v en als v nieuw is dan is er maar één boog, dus $\text{opv}'([v,x]) = [v,x]$. Voor alle andere (georiënteerde) bogen e geldt $\text{opv}'(e) = \text{opv}(e)$.

144 Lemma *$G = (V, E, opv)$ zij een samenhangende ingebedde graaf. Als wij bewerking (i) of bewerking (ii) toepassen en het resultaat is $G' = (V', E', opv')$ dan geldt $ek(G') \in \{ek(G) - 2, ek(G)\}$.*

Bewijs: Het is duidelijk hoe $|V|$ en $|E|$ veranderen in de twee gevallen, maar hoe verandert $|F|$?

(i): Vlakken waarin geen enkele hoek gewijzigd word, veranderen natuurlijk niet en er is maar één van de bestaande hoeken die gewijzigd wordt: $[y,x],[x,z]$. Als wij nu naar het vlak kijken waarin de hoek $[y,x],[x,z]$ zit en $[y_1,y_2],[y_2,y_3],\ldots,[y_{k-1},y_k] = [y,x],[x,z] = [y_k,y_{k+1}],\ldots,[y_n,y_1]$ is het vlak in G dan is er één vlak $[y_1,y_2],[y_2,y_3],\ldots,[y_{k-1},y_k] = [y,x],[x,v],[v,x],[x,z] = [y_k,y_{k+1}],\ldots,[y_n,y_1]$ dat alle oude bogen van het vlak bevat en de nieuwe. Het aantal vlakken is dus niet veranderd. Dus

- $|V'| = |V| + 1$
- $|E'| = |E| + 1$
- $|F'| = |F|$
- dus $\text{ek}(G) = |V| - |E| + |F| = |V| + 1 - |E| - 1 + |F| = |V'| - |E'| + |F'| = \text{ek}(G')$

(ii): Nu zijn twee hoeken gewijzigd: $[y,x],[x,z]$ en $[w,v],[v,u]$. Er zijn twee gevallen:

(ii)a: De twee hoeken liggen in hetzelfde vlak van G.

(ii)b: De twee hoeken liggen in verschillende vlakken van G.

(ii)a: Het vlak was dus
$[y_1, y_2], [y_2, y_3], \ldots, [y_{k-1}, y_k] = [y, x], [x, z] = [y_k, y_{k+1}], \ldots, [y_{l-1}, y_l] = [w, v], [v, u] = [y_l, y_{l+1}], \ldots [y_n, y_1]$

Als je nu Algoritme 140 toepast dan zie je dat er door de twee plaatsen waar er wijzigingen gebeurd zijn (en het algoritme dus andere bogen kiest) twee vlakken ontstaan zijn, namelijk
$[y_1, y_2], [y_2, y_3], \ldots, [y_{k-1}, y_k] = [y, x], [x, v], [v, u] = [y_l, y_{l+1}], \ldots [y_n, y_1]$
en
$[x, z] = [y_k, y_{k+1}], \ldots, [y_{l-1}, y_l] = [w, v], [v, x]$. Dus

- $|V'| = |V|$
- $|E'| = |E| + 1$
- $|F'| = |F| + 1$
- dus $\text{ek}(G) = |V| - |E| + |F| = |V| - |E| - 1 + |F| + 1 = |V'| - |E'| + |F'| = \text{ek}(G')$

(ii)b: Er waren dus twee vlakken

$[y_1, y_2], [y_2, y_3], \ldots, [y_{k-1}, y_k] = [y, x], [x, z] = [y_k, y_{k+1}], \ldots, [y_n, y_1]$ en
$[t_1, t_2], [t_2, t_3], \ldots, [t_{l-1}, t_l] = [w, v], [v, u] = [t_l, t_{l+1}], \ldots, [t_m, t_1]$.

Als je nu Algoritme 140 toepast dan is het resultaat één groot vlak:

$[y_1, y_2], [y_2, y_3], \ldots, [y_{k-1}, y_k] = [y, x], [x, v], [v, u] = [t_l, t_{l+1}], \ldots, [t_m, t_1]$,
$[t_1, t_2], [t_2, t_3], \ldots, [t_{l-1}, t_l] = [w, v], [v, x], [x, z] = [y_k, y_{k+1}], \ldots, [y_n, y_1]$.

Dus

- $|V'| = |V|$
- $|E'| = |E| + 1$
- $|F'| = |F| - 1$
- dus $\text{ek}(G) = |V| - |E| + |F| = |V| - |E| - 1 + |F| - 1 + 2 = |V'| - |E'| + |F'| + 2 = \text{ek}(G') + 2$ of $\text{ek}(G') = \text{ek}(G) - 2$

■

145 Oefening *Kijk naar de volgende definitie:*

146 Definitie *Gegeven een ingebedde graaf $G = (V, E, opv)$ en drie verschillende bogen*
$\{v, x\}, \{v, y\}, \{v, z\} \in E$.
Dan zeggen wij dat $[v, y]$ tussen $[v, x]$ en $[v, z]$ ligt als er $k, k' \in \mathbb{N}$ bestaan met $0 < k < k'$ en $opv^k([v, x]) = [v, y]$, $opv^{k'}([v, x]) = [v, z]$.

Beschrijft deze definitie precies wat wij als "in wijzerzin tussen" *zouden willen zien? Of ontbreekt er een eis? Als dat zo is voeg deze eis dan toe zodat de definitie onze intuitie weerspiegelt.*

147 Oefening *Intuïtief is duidelijk wat bedoeld is als wij het over een (ingebedde) deelgraaf van een ingebedde graaf hebben. Geef een precieze definitie van dit begrip.*

148 Lemma *Als $G' = (V', E', opv')$ een (samenhangende ingebedde) deelgraaf van een (samenhangende ingebedde) graaf $G = (V, E, opv)$ is, dan geldt $ek(G) \leq ek(G')$.*

Bewijs: Het is gemakkelijk om aan te tonen dat je G uit G' kan verkrijgen door de bewerkingen (i) en (ii) toe te passen. Maar tijdens deze bewerkingen kan ek() volgens Lemma 144 alleen maar dalen – daarmee volgt het resultaat onmiddellijk.

■

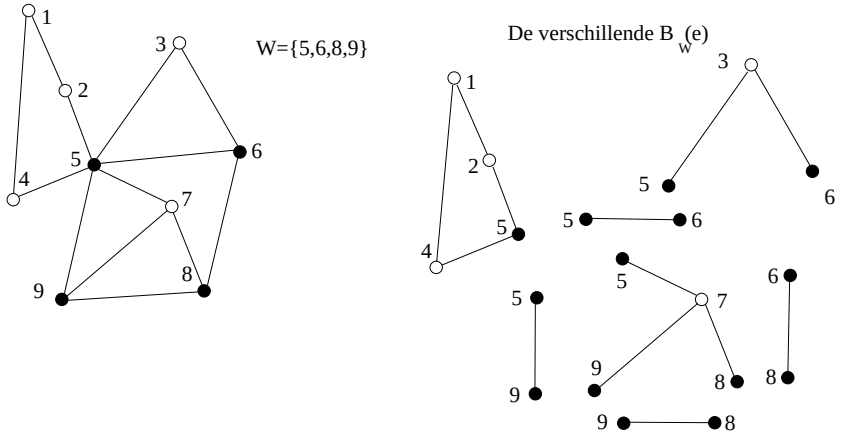
149 Corollarium *Voor elke graaf $G = (V, E, opv)$ geldt dat $ek(G) \leq 2$ en dat $ek(G)$ even is.*
Dus is $gen(G) \geq 0$ en is het een geheel getal.

Bewijs: De Euler karakteristiek van de unieke samenhangende graaf met maar één boog is 2 en die is een deelgraaf van elke ingebedde graaf. Als wij bewerkingen (i) en (ii) toepassen, kan de Euler karakteristiek ofwel niet veranderen ofwel met twee verminderen – het getal blijft dus even.

■

150 Definitie *Een (samenhangende ingebedde) graaf $G = (V, E, opv)$ met $ek(G) = 2$ heet vlak.*
Een samenhangende graaf $G = (V, E)$ heet planair als er een functie opv() bestaat zodat $G = (V, E, opv)$ vlak is.

151 Definitie *Gegeven een graaf $G = (V, E)$ en $W \subseteq V$.*
Dan is een W niet kruisend pad een pad P waarvan ten hoogste de eindpunten in W zitten. Een W niet kruisende cykel is een cykel waarvan ten hoogste één punt in W zit.
Als $e \in E$ dan is de vanuit e zonder W bereikbare deelgraaf $B_W(e)$ gedefinieerd als $B_W(e) = \bigcup_{\{P|P \text{ is pad of cykel, bevat } e \text{ en } P \text{ kruist } W \text{niet}\}} P$.
De vereniging van paden is daarbij de vereniging van de toppenverzamelingen en de boogverzamelingen van de paden.
Het is onmiddellijk duidelijk dat de $B_W(e)$ altijd samenhangend zijn.



Figuur 28: Een voorbeeld voor Definitie 151.

152 Oefening *Als je voor een gegeven graaf $G = (V, E)$ en $W \subseteq V$ definieert dat voor twee bogen $e, e' \in E$ geldt dat $e \equiv_B e'$ als $e' \in B_W(e)$, is $\equiv_B$ dan een equivalentierelatie op de bogen?*

153 Definitie *Gegeven is een ingebedde graaf $G = (V, E, opv)$ en een cykel $C = v_0, v_1, \ldots, v_{n-1}, v_0$ in G waarbij wij hier echt op de volgorde letten – dus een verschil maken met $C' = v_0, v_{n-1}, v_{n-2}, \ldots, v_1, v_0$. Dan zeggen wij dat een met (de toppen van) C incidente boog $\{v_i, x\}$ (met $x \notin \{v_{i+1 \mod n}, v_{i-1 \mod n}\}$) rechts van C ligt als (v_i, x) tussen $[v_i, v_{i+1 \mod n}]$ en $[v_i, v_{i-1 \mod n}]$ ligt. Anders zeggen wij dat de boog links van C ligt. Hoewel de boog op de positie van v_i in de cykel alleen maar ofwel rechts ofwel links kan liggen, kan het best zijn dat als x ook op C ligt dezelfde boog op de positie van de top x rechts ligt – dus één keer rechts en één keer links van C.*

Opgelet: Je moet voorzichtig zijn met het gebruik van deze begrippen *rechts* en *links*. Ze zijn zo gekozen om de interpretatie van de tekeningen gemakkelijker te maken maar wij zouden even goed *binnen* en *buiten* hebben kunnen kiezen of slechts a en b. Als je een tekening gebruikt en jouw intuïtieve definitie van *links* en *rechts*, kijk dan vooral of het met de betekenis uit de definitie overeenkomt...

Met deze definities kunnen wij nu een gemakkelijke versie van een heel bekende stelling uit de topologie bewijzen: de Jordanse kromme stelling. In de topologie (of analyse) is deze stelling veel moeilijker om te bewijzen maar de reden is omdat de stelling dan veel sterker is en een cykel een veel ingewikkeldere structuur kan hebben dan een cykel in een graaf! Informeel zou je kunnen zeggen dat het feit dat de Jordanse kromme stelling geldig is, toont dat je op de bol of in het vlak bezig bent.

154 Stelling van de Jordanse kromme
Gegeven is een vlakke graaf $G = (V, E, opv)$ en een cykel C in G. Dan is er geen $e \in E, e \notin C$ zodat $B_{V(C)}(e)$ een boog bevat die links van C ligt **en** *een boog die rechts van C ligt.*

Informeel zou je kunnen zeggen dat de cykel dan een binnenkant en een buitenkant heeft en je niet van binnen naar buiten kan gaan zonder de cykel te kruisen. Maar dat is heel informeel...

Bewijs: De cykel als ingebedde graaf heeft een Euler karakteristiek gelijk aan 2 waarbij wij precies twee vlakken hebben: Als $C = v_0, v_1, \ldots, v_{n-1}, v_0$ dan zijn de vlakken $[v_0, v_1], [v_1, v_2], \ldots, [v_{n-1}, v_0]$ en $[v_0, v_{n-1}], [v_{n-1}, v_{n-2}]$, $\ldots, [v_1, v_0]$.

Als er nu een $B_{V(C)}(e)$ zou zijn die een boog e' bevat die links van C ligt en een boog e'' die rechts van C ligt, dan zouden de eindbogen e', e'' van een pad P dat met e' begint en met e'' eindigt (waarbij $e' = e''$ mogelijk is) in G tussen bogen f_0, f_1 resp. g_0, g_1 liggen zodat f_0^{-1}, f_1, resp. g_1^{-1}, g_0 in C hoeken van verschillende vlakken vormen. Maar net zoals in Lemma 144 (ii)b geldt voor $\text{ek}(C \cup P)$ dan $\text{ek}(C \cup P) = \text{ek}(C) - 2 = 0$ en dus (Lemma 148) $\text{ek}(G) < 2$ in tegenstelling tot de voorwaarde dat G vlak is (dus $\text{ek}(G) = 2$).

■

Omdat in dit geval $B_{V(C)}(e)$ ofwel alleen maar bogen rechts van C ofwel alleen maar bogen links van C kan bevatten, is het dus gerechtvaardigd ook te zeggen dat $B_{V(C)}(e)$ rechts (of links) van C ligt als het een boog bevat die rechts (of links) van C ligt.

155 Oefening *Geef een ingebedde graaf en een cykel die aantonen dat de Jordanse kromme stelling niet geldig is als de graaf niet vlak is.*

156 Oefening *Geldt de omkering van de stelling van de Jordanse kromme: als een graaf niet vlak is, bestaan er dan een cykel C en een boog e zodat $B_{V(C)}(e)$ een boog bevat die links van C ligt en één die rechts van C ligt?*

157 Stelling *Gegeven een 2-samenhangende vlakke graaf G en een vlak F van G.*
Dan vormt de verzameling C van bogen $\{x, y\}$ waarvoor $[x, y] \in F$ of $[y, x] \in F$ een cykel in G.

Bewijs: Het is duidelijk dat voor elke $\{x, y\} \in C$ ten minste één boog $\{x, v\} \in C$ met $v \neq y$ bestaat omdat x anders graad 1 zou hebben. Elke top is dus incident met ten minste twee bogen. Omdat de door C gegeven graaf duidelijk samenhangend is, moeten wij dus alleen nog bewijzen dat **precies** twee dergelijke bogen bestaan.

Een vlak zo als wij het in Definitie 139 gedefinieerd hebben, vormt een cykel waarbij de toppen de georiënteerde bogen zijn. Als elke top in maar één hoek (in maar één paar op elkaar volgende georiënteerde bogen) voorkomt (en geen top graad 1 heeft) heeft elke top ook twee incidente buren in C – C is dus een cykel.

Stel dus dat een top y meer dan één keer opduikt en dat de cykel van georiënteerde bogen
$\ldots, [x, y], [y, v_1], [v_1, v_2], \ldots, [v_k, y], [y, x'], \ldots$
is waarbij y zo gekozen is dat er in de reeks $y, v_1, v_2, \ldots v_k$ geen top twee keer opduikt.

Kijk nu naar de cykel $C' = y, v_1, v_2, \ldots, v_k, y$. De enige bogen incident met toppen van C' die er links van liggen, vertrekken vanuit y en liggen tussen $[y, v_k]$ en $[y, v_1]$ (kijk naar de definitie van *links liggen*). Eén van deze bogen is zeker $\{x, y\}$ omdat $[x, y], [y, v_1]$ een hoek vormen. Omdat G 2-samenhangend is, moet er een pad P van v_1 naar x zijn dat y niet bevat en geldt dat $v_1 \neq x$ (omdat $\deg(y) > 1$).

Als $x \in \{y, v_1, v_2, \ldots, v_k\}$ dan moet de boog $[x, y]$ op de plaats waar x in C' zit dus rechts van C' liggen – maar dan is $B_{V(C')}(\{x, y\})$ in tegenstrijdigheid met Stelling 154.

Als $x \notin \{y, v_1, v_2, \ldots, v_k\}$ krijg je op dezelfde manier een tegenstrijdigheid als je naar de laatste boog in P kijkt die één top uit C'

en één top niet uit C' bevat (waarom moet een dergelijke boog bestaan?). Ook deze boog moet rechts van C' liggen – maar hij behoort tot $B_{V(C')}(\{x, y\})$ – opnieuw een tegenstrijdigheid.

■

158 Oefening *Gegeven een vlakke graaf $G = (V, E, opv)$, een cykel C in G en een vlak F.*
Toon aan dat het vlak niet tegelijk bogen links van C en bogen rechts van C kan bevatten.

159 Lemma *Gegeven een vlakke graaf $G = (V, E, opv)$, een vlak F, toppen $x, y \notin F$ en 3 toppendisjuncte paden P_1, P_2, P_3 van x naar y.*
Dan bestaat er een P_i, $1 \leq i \leq 3$ zodat P_i geen top van F bevat.

Bewijs: Vergelijk met Figuur 29.

Stel dat F toppen uit alle 3 paden bevat en dat $v_1, v_2, \ldots, v_k$ een kortst mogelijk randpad in F is met $v_1 \in P_1$, $v_k \in P_3$ en $v_j \in P_2$ voor een j met $1 < j < k$ (waarbij de indices van de paden misschien aangepast moeten worden). Kies de index j minimaal met deze eigenschap.

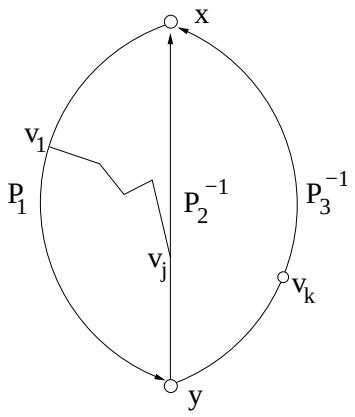
Dan liggen de bogen $\{v_1, v_2\}, \ldots, \{v_{j-1}, v_j\}$ aan dezelfde kant van de cykel P_1, P_3^{-1} als de bogen van P_2 – stel links (anders moet je in de volgende argumentatie rechts en links verwisselen). Als je nu naar de rotatie rond x kijkt, zie je dat de bogen van P_3 rechts van de cykel P_1, P_2^{-1} liggen. Maar omdat $\{v_1, v_2\}$ van P_1 vertrekt, ligt de boog nog steeds links – ook als je naar P_1, P_2^{-1} kijkt in plaats van P_1, P_3^{-1}. Maar $\{v_{k-1}, v_k\}$ is incident met een interne top van P_3 en ligt dus rechts van P_1, P_2^{-1} – maar dat is een tegenstrijdigheid met het resultaat van Oefening 158.

■

160 Oefening *Is het mogelijk dat in een 3-samenhangende vlakke graaf twee vlakken drie of meer toppen gemeen hebben?*

161 Oefening

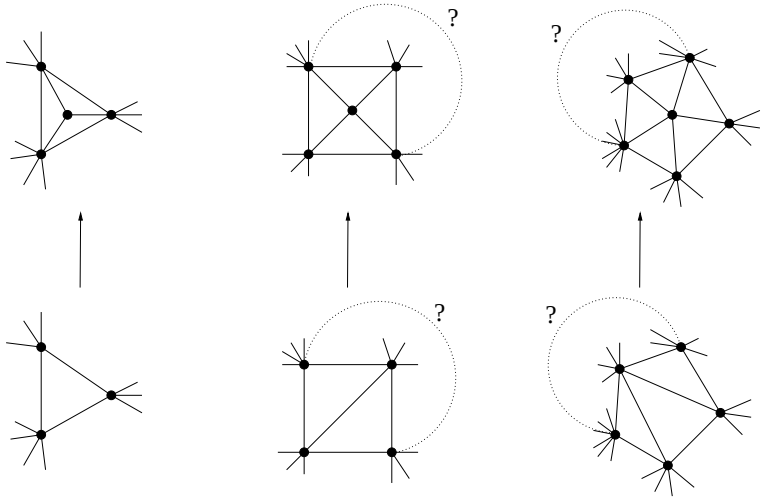
- *Toon aan dat in een planaire graaf geldt dat $|E| = O(|V|)$. Tip: gebruik dat vlakken (voor $|V| \geq 3$) altijd ten minste 3 georiënteerde bogen bevatten (anders zou het een multigraaf zijn).*
- *Toon aan dat in een planaire graaf altijd toppen met graad 5 of kleiner aanwezig zijn.*



Figuur 29: Drie paden en een deel van een vlak.

162 Oefening

- *Bewijs dat de graaf K_5 (de graaf met 5 toppen waar elke top met elke andere is verbonden) geen planaire inbedding bezit.*
- *Iets moeilijker: Bewijs dat de graaf $K_{3,3}$ (de bipartiete graaf met 6 toppen waarbij je twee bipartitieklassen met 3 toppen hebt en elk top in de ene klasse met alle toppen in de andere is verbonden) geen planaire inbedding bezit.*



Figuur 30: De operaties van Eberhard om triangulaties op te bouwen. De bogen met vraagtekens kunnen maar moeten niet aanwezig zijn.

163 Definitie *Een triangulatie (van de bol) is een vlakke graaf waarin elk vlak een driehoek is, dus 3 geörienteerde bogen bevat.*

In 1891 heeft Eberhard (min of meer) de volgende stelling bewezen:

164 Stelling *Elke triangulatie (van de bol) kan opgebouwd worden door een reeks van operaties zoals in Figuur 30 op een driehoek toe te passen. Ook na elke tussenstap is de graaf een triangulatie.*

In de figuur zijn de operaties als een tekening voorgesteld maar het is duidelijk wat dat betekent als je het uitdrukt als wijzigingen van V, E en opv().

165 Oefening *Het bewijs van Eberhard was niet grafentheoretisch maar hij gebruikte operaties zoals* snijden *langs een vlak en paste dat toe op* convexe polyeders.
Gebruik nu jouw grafentheoretische kennis om deze stelling te bewijzen. Let op: tijdens het opbouwen mogen nooit multigrafen ontstaan!

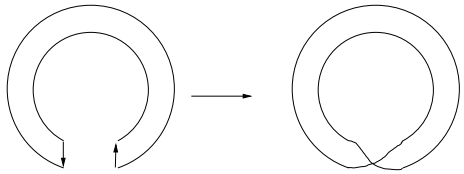
166 Oefening *Gegeven een 3-samenhangende vlakke graaf $G = (V, E, opv)$, waarbij F de verzameling van vlakken is. Dan hebben 2 verschillende vlakken ten hoogste één gemeenschappelijke boog.*
Dan definieren wij de duale graaf $G_d = (V_d, E_d, opv_d)$ door $V_d = F$, $E_d = \{\{f_1, f_2\} | f_1, f_2$ hebben een boog $e(f_1, f_2)$ gemeen.$\}$ en in $opv_d(f)$ wordt de cyklische volgorde van de bogen $\{f, f_i\}$ door de volgorde van de $e(f, f_i)$ in het vlak f gegeven.

Bewijs: de duale graaf is vlak.

5.2.1 Heel informeel: aanschouwelijke betekenis van het genus

Wij hebben het genus op een manier gedefinieerd waarvoor wij helemaal geen topologie of analyse nodig hebben. Maar inderdaad heeft het iets te maken met topologie. Deze samenhang willen wij hier alleen **heel grof schetsen**. Omdat de meesten van jullie informatici zijn, kennen jullie misschien sommige van de basisbegrippen niet die nodig zijn om het iets formeler te doen – en het is ook niet gemakkelijk! Omdat de samenhang met *oppervlakten* aan de ene kant wel belangrijk is om te weten maar wij hem in toekomst niet echt nodig zullen hebben, is een grove schets misschien een goed compromis. Omdat het maar een schets is, zullen wij de *definitieachtige* delen hier niet echt definitie noemen en het ook niet over *stellingen* hebben.
Een 2-dimensionale oppervlakte is *iets* dat er lokaal zo uitziet alsof het $\mathbb{R}^2$ was (voor elk punt bestaat er een omgeving die homeomorf is met een omgeving van een punt in $\mathbb{R}^2$). Je kan bv. aan een bol of de oppervlakte van een binnenband van een fiets denken. Zo'n oppervlakte heet oriënteerbaar als je aan elk punt een *wijzerzin* kan toekennen en dat op een *continue* manier. Dat is bv. mogelijk als je de oppervlakte van een bol neemt, maar niet als je

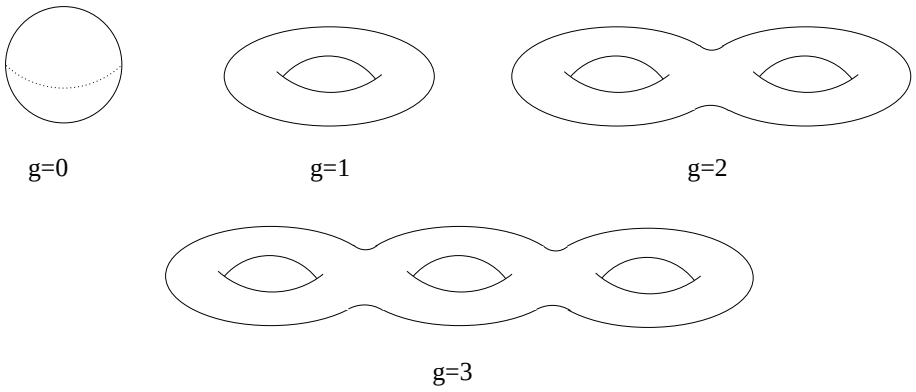
een band van Möbius neemt. Zie Figuur 31 voor een tekening en een beetje uitleg. Als je daar een wijzerzin neemt en die bijhoudt terwijl je naar het punt gaat dat als de band van papier zou zijn de achterkant van het eerste punt was, maar in de (oneindig dunne) band van Möbius natuurlijk hetzelfde punt, dan ga je zien dat je plotseling een andere wijzerzin hebt...



Figuur 31: De band van Möbius krijg je als je de twee pijlen zo aan elkaar plakt dat ze in dezelfde richting wijzen. Maar je mag het je niet zo voorstellen alsof hij van papier is, maar alsof de band oneindig dun is zodat je als je op een stuk papier aan de achterkant bent, het punt inderdaad identiek is aan het punt op de voorkant. Bovendien moet je de rand van de band verwijderen om aan onze definitie van een oppervlakte te voldoen – punten op de rand zien er lokaal niet uit als $\mathbb{R}^2$.

Bovendien zijn wij alleen maar geïnteresserd in *compacte* oppervlakten. Intuïtief is dat zoiets als *eindig* of *gesloten* – maar niet helemaal. De oppervlakte van een bol is bv. compact – maar als je één enkel punt verwijdert, is het resultaat niet meer compact (inderdaad is het resultaat voor topologen hetzelfde als het vlak – daarom kan je een graaf ook op een bol tekenen als en slechts als je dat in het vlak kan doen). Iets formeler kan je zeggen dat een oppervlakte compact is als elke overdekking met (open) omgevingen een eindige deeloverdekking bevat. In de topolgie wordt er geen verschil gemaakt tussen oppervlakten die je in elkaar kan *vervormen* – waarbij bv. een voetbal en een rugbybal topolgisch als equivalent beschouwd worden. Maar je kan een voetbal bv. niet in een binnenband van een fiets vervormen zonder te snijden en te plakken – deze oppervlakten worden als verschillend beschouwd.
Er is een classificatie van de 2-dimensionale compacte oriënteerbare oppervlakten: voor elke dergelijke oppervlakte bestaat er een g zodat de oppervlakte equivalent is aan de oppervlakte van een *bol waarin er g gaten zitten die helemaal door de bol gaan* (zie Figuur 32). Deze g heet dan ook de genus of het geslacht van de oppervlakte. Voor $g = 0$ is dat de oppervlakte van de bol, voor $g = 1$ de oppervlakte van een binnenband van een fiets, etc.

Als een ingebedde graaf genus g heeft, dan betekent dat dat je hem op de oppervlakte van zo'n bol met g gaten kan tekenen. Als je de genus van een graaf G definieert als de minimale g zodat je G op de oppervlakte van een 2-dimensionale compacte oriënteerbare oppervlakte met genus g kan tekenen (waarbij je *tekenen* natuurlijk precies moet definiëren) dan is dat een definitie die equivalent is met die die wij gegeven hebben.



Figuur 32: Voorbeelden van de oppervlakten met $g \in \{0, 1, 2, 3\}$.

Maar het feit dat jullie nog geen topologie gevolgd hebben, is niet de enige reden dat onze aanzet niet van de topologische kant komt. Er is nog een belangrijkere reden: als jullie bv. planaire grafen op de gewone manier met homeomorfismen van het interval $[0, 1]$ definiëren, kunnen jullie dat voor algoritmen die op een computer moeten draaien heel slecht gebruiken. Dan moeten jullie al benaderingen toepassen (bv. eisen dat de beelden van de bogen stuksgewijs lineair zijn) om het in de computer te kunnen verwerken. Inderdaad is onze aanpak een aanpak die heel geschikt is voor algoritmen voor ingebedde grafen. Inderdaad kan je gewoon een lijst van buren als datastructuur gebruiken – net zoals voor gewone grafen – alleen dat in dit geval de volgorde belangrijk is en in het geval van gewone grafen – waar de lijst een verzameling voorstelt – niet. En bovendien (dat is natuurlijk belangrijk!) bevat onze voorstelling van een ingebedde graaf alle informatie die nodig is om dat te coderen wat wij *de bijzonder interessante structuren* van een ingebedde graaf zouden kunnen noemen – dat is wat de vlakken zijn, welke bogen aan welke vlakken grenzen, etc. Wij zijn er normaal niet echt in geïnteresseerd welke *vorm* de getekende bogen hebben en dus is de combinatorische informatie voldoende. In gevallen waar de exacte vorm wel

belangrijk is, moet dan natuurlijk met een andere voorstelling van een ingebedde graaf gewerkt worden. Er zal ook een oefening op dit vlak zijn (die natuurlijk een beetje informeel opgelost moet worden omdat wij niet alle nodige werktuigen hebben...).

167 Oefening *Schets een bewijs voor de stelling dat elke planaire graaf op een manier in het vlak getekend kan worden zodat alle bogen rechte lijnen zijn. Tips:*

- *Is elke planaire graaf een deelgraaf van een triangulatie?*
- *Kijk naar Stelling 164.*
- *Kan je voor een gegeven inbedding van de graaf het buitenvlak voor de tekening met rechte lijnen kiezen?*

5.2.2 Isomorfie van ingebedde grafen

Als wij het over *isomorfie* van ingebedde grafen hebben, dan willen wij natuurlijk dat er ook rekening met de inbedding – dus de opvolgerfuncties – gehouden wordt:

168 Definitie

- *Twee (samenhangende ingebedde) grafen $G = (V, E, opv)$ en $G' = (V', E', opv')$ heten oriëntatiebewarend isomorf als en slechts als er een bijectieve afbeelding $f : V \to V'$ bestaat met $\{v, w\} \in E \Leftrightarrow \{f(v), f(w)\} \in E'$ en $opv([v, w]) = [v, x] \Leftrightarrow opv'([f(v), f(w)]) = [f(v), f(x)]$. Wij schrijven $G \cong_b G'$.*
- *G en G' heten oriëntatie-omkerend isomorf als en slechts als er een bijectieve afbeelding $f : V \to V'$ bestaat met $\{v, w\} \in E \Leftrightarrow \{f(v), f(w)\} \in E'$ en $opv([v, w]) = [v, x] \Leftrightarrow opv'([f(v), f(x)]) = [f(v), f(w)]$. Wij schrijven $G \cong_o G'$.*
- *Twee ingebedde grafen heten isomorf als ze oriëntatie-omkerend isomorf of oriëntatiebewarend isomorf zijn. Wij schrijven $G \cong_i G'$.*
- *Twee ingebedde grafen heten isomorf als grafen als de grafen zonder de inbedding isomorf zijn. Wij schrijven zoals vroeger $G \cong G'$.*

169 Oefening *Zijn de eisen voor de geïnduceerde operatie op E en E' nodig of volgen die uit de eisen voor $opv()$ en $opv'()$?*

170 Oefening *Is de volgende definitie equivalent met die in Definitie 168?*

$G = (V, E, opv)$ en $G' = (V', E', opv')$ heten oriëntatie-omkerend isomorf als en slechts als $G = (V, E, opv)$ en $G'' = (V', E', (opv')^{-1})$ orientatie bewarend isomorf zijn.

Het verschil tussen *oriëntatiebewarend isomorf* en *oriëntatie-omkerend isomorf* heeft ermee te maken of je een verschil wil maken tussen spiegelbeelden of niet. Als G oriëntatie-omkerend isomorf is met G' dan kan je de grafen als spiegelbeelden van elkaar zien. Of je dat wil toelaten hangt van de toepassing af waarmee je bezig bent. In veel gevallen zou je zeggen dat de structuur van het spiegelbeeld identiek is aan de structuur van het origineel (bv. vlakken van dezelfde grootte, etc.). Maar in andere gevallen is het wel belangrijk of je een ingebedde graaf hebt of zijn spiegelbeeld. Het belangrijkste en droevigste geval waar het verschil tussen een molecuul en zijn spiegelbeeld een grote impact had (wel het spiegelbeeld in 3 dimensies) is het geval van een medicament dat in België als Softenon verkocht werd (in Duitsland als Contergan). Het had de eigenschap dat het molecuul dat voor de werking verantwoordelijk was, onschadelijk was in de ene vorm – maar in de andere vorm – die qua bindingen identiek was aan de eerste maar qua realisatie in de ruimte het spiegelbeeld van de eerste – zware schade aan de kinderen veroorzaakte als zwangere vrouwen het gebruikten. Voor moleculen is het dus soms **heel** belangrijk een verschil te maken tussen spiegelbeelden!

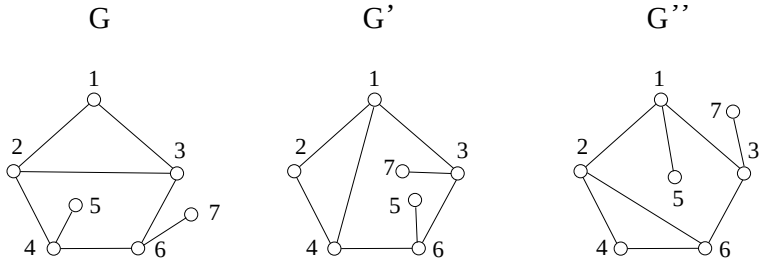
171 Oefening *Stel dat $G = (V, E, opv())$ en $G' = (V, E, opv()^{-1})$.*
Is er een isomorfisme van $VL(G)$ naar $VL(G')$?

172 Oefening *OK – geen echt realistisch voorbeeld uit de praktijk maar het helpt om iets te leren:*
In een bedrijf wordt onderzocht op hoeveel verschillende manieren je 16 transistoren met elkaar op een printplaat kan verbinden. Elke transistor heeft 3 verschillende aansluitingen. Dat zijn in wijzerzin als je van boven op de transistor kijkt a, b, c.
De aanzet is nu naar vlakke 3-reguliere grafen te kijken (hier gebruiken wij wel dat onze definitie van vlak *hetzelfde is als* in het vlak te tekenen*) en de toppen als transistoren te zien.*
Maar wat betekent in dit model verschillende *mogelijkheden? Is één van de definities van isomorfie toepasbaar?*
Als ja: welke definitie is het – geef uitleg.
Als neen: formuleer een nieuwe definitie die wel toepasbaar is en geef uitleg.

173 Oefening *Toon aan: als $f()$ een isomorfisme van $G = (V, E, opv)$ en $G' = (V', E', opv')$ (als ingebedde grafen) is, dan volgt uit $f()$ een bijectieve afbeel-*

ding $F(G) \to F(G')$ die vlakken met hetzelfde aantal georiënteerde bogen op elkaar afbeeldt.
Tip: kijk ook naar Oefening 171.

174 Oefening *Schrijf voor elk van de verzamelingen $\{G, G\}, \{G, G'\}, \{G, G''\}$, $\{G', G'\}, \{G', G''\}, \{G'', G''\}$ van grafen in Figuur 33 of de bevatte ingebedde grafen oriëntatiebewarend isomorf, oriëntatie omkerend isomorf, gewoon isomorf of geen van deze opties zijn.*



Figuur 33: De tekeningen van deze grafen zijn natuurlijk alleen maar één manier om de toppen, de bogen en de opvolgerfuncties voor te stellen...

Nu zullen wij een string beschrijven die een isomorfiebepalende invariant is voor de definitie van $\cong_b$. Wij hebben het begrip *isomorfiebepalende invariant* alleen voor $\cong$ gedefinieerd maar het is duidelijk wat het in deze (en analoge) contexten betekent. Het algoritme is in principe alleen maar een BFS algoritme dat nummers geeft en waarbij de volgorde gegeven is door de opvolgerfunctie. Bovendien wordt ook de eerste te beschouwen buur voor elke top vastgelegd.

175 Algoritme Get_string(a,b)
Input: een ingebedde graaf $G = (V, E, opv())$ en een georiënteerde boog $[a, b]$ met $\{a, b\} \in E$.
Wij hebben een array van getallen nummer[] *van lengte $|V|$ (waarbij wij stellen dat $V = \{0, 1, \ldots, |V| - 1\}$) met nummer$[v] = \infty$ voor alle $v \in V$.*
Wij hebben een initieel lege string string[]*, een array* eersteboog[] *van georiënteerde bogen, een initieel lege FIFO-lijst* toppenlijst *van toppen die wij op een BFS manier zullen gebruiken en een teller* nextnummer *die in het begin* 1 *is.*

Initialisatie:
Plaats a in toppenlijst *en definieer eersteboog$[a] = [a, b]$ en nummer$[a] = 0$*
Dan herhaal de volgende stappen tot de toppenlijst *leeg is:*

- *Verwijder een top uit de lijst. Stel dat dat de top v is.*
- *Dan zij $[v, y]$ de boog eersteboog$[v]$ (de eerste top is altijd v)*
- *(Als* deg(v) *niet gekend is, kan je het volgende herhalen tot je terug bent aan eersteboog$[v]$ – anders:) Herhaal* deg(v) *keer:*
  - *Als nummer$[y] = \infty$:*
    - *voeg y toe aan lijst.*
    - *Definieer nummer$[y] = nextnummer$*
    - *verhoog* nextnummer *met* 1
    - *Definieer eersteboog$[y] = [y, v]$*
  - *voeg nummer$[y]$ toe aan* string.
  - *als opv$([v, y]) = [v, z]$ dan zet $y = z$.*
- *voeg het teken „|“ toe aan* string.

176 Oefening *De beste manier om dit algoritme te verstaan, is het gewoon toe te passen op een kleine graaf.*
Pas het toe op de eerste graaf in Figuur 33 met beginboog $[a, b] = [5, 4]$

Het is gemakkelijk om te zien dat Algoritme 175 in lineaire tijd $O(|V| + |E|)$ draait – tus in tijd $O(|V|)$ voor vlakke grafen – en dat als er twee grafen met georiënteerde bogen zijn die dezelfde string opleveren je een isomorfisme kan bouwen door toppen die tijdens de berekening hetzelfde nummer krijgen op elkaar af te beelden. En het is even gemakkelijk om te zien dat als er een oriëntatiebewarend isomorfisme is, je vanuit twee georiënteerde bogen die op elkaar worden afgebeeld ook identieke strings krijgt. Een algoritme dat een isomorfiebepalende invariant berekent is nu gemakkelijk om te vinden:

177 Algoritme Minstring_b(G)
Input: een ingebedde graaf $G = (V, E, opv())$.
Dan bereken Get_string(a,b) voor elke georiënteerde boog $[a, b]$ en bereken de lexicografisch minimale string van al deze strings.

Het is duidelijk dat dit algoritme in het algemeen in tijd $O((|V| + |E|)^2)$ en voor vlakke grafen in tijd $O(|V|^2)$ draait en dat je voor twee grafen dezelfde string vindt als en slechts als er een oriëntatiebewarend isomorfisme bestaat. Minstring_b(G) is dus een isomorfiebepalende invariant voor $\cong_b$.
Analoog met Get_string(a,b) in Algoritme 175 kan je ook een functie Get_invers_string(a,b) definiëren door in het algoritme de functie opv() te vervangen door de functie $\text{opv}^{-1}()$.

En dan kan je een functie Minstring_o() (of Min_invers_string(G)) definiëren analoog met Minstring_b(G) in Algoritme 177 – alleen dat je Get_invers_string(a,b) in plaats van Get_string(a,b) gebruikt.
Dan bestaat er een oriëntatie-omkerend isomorfisme tussen G en G' als en slechts als Minstring_o(G) = Minstring_b(G').
Als wij nu Minstring(G) definiëren als de lexicografisch kleinere van Minstring_o(G) en Minstring_b(G') dan is dat een isomorfiebepalende invariant voor $\cong_i$ die je in tijd $O((|V| + |E|)^2)$ ($O(|V|^2)$ voor vlakke grafen) kan berekenen.
Je kan in de literatuur ook een $O(|V|)$ algoritme van Hopcroft en Wong (1974) terugvinden om isomorfie van vlakke grafen te testen. Dus waarom wordt hier een $O(|V|^2)$ algoritme voorgesteld? Inderdaad is er tot nu toe nog niemand in geslaagd het lineaire algoritme te implementeren. Het is niet voldoende precies beschreven om geïmplementeerd te kunnen worden – en inderdaad kan je dus ook niet echt zeker zijn dat het lineair en juist is.... Een $O(|V| \log |V|)$ algoritme is beter beschreven, maar ook dit algoritme is ingewikkelder dan het hier beschreven algoritme en de constanten in de complexiteit zijn gewoon slechter.

5.2.3 Optimalisaties

Maar wij kunnen aan dit voorbeeld ook sommige trucjes zien om de prestatie in het algemeen te verbeteren (hoewel de asymptotische complexiteit dezelfde blijft). Wij zullen deze trucjes alleen beschrijven voor het geval dat wij een isomorfiebepalende invariant voor $\cong_b$ willen berekenen – voor $\cong_i$ zal het dan een oefening zijn.
Het is natuurlijk duidelijk dat je niet altijd Get_string(a,b) moet berekenen. Zodra duidelijk wordt dat je geen kleinere string kan verkrijgen, kan je de berekening stoppen. Een routine die zo werkt als Get_string(a,b) maar altijd met een andere string vergelijkt en stopt zodra de nieuwe string groter is, noemen wij Get_string'(a,b) – waarbij wij stellen dat er een string om te vergelijken aan de routine wordt doorgegeven. Deze routine berekent dezelfde string als die kleiner dan of gelijk aan de gegeven string is en anders gewoon *een string* (die misschien zelfs niet aan de regels van de isomorfiebepalende invariant voldoet) die groter is.
In plaats van de string van Get_string(a,b) gebruiken wij een string die eerst met sommige gemakkelijk te berekenen getallen voor de boog $[a, b]$ begint. Wij kunnen dus bv. eerst aan elke georiënteerde boog e de getallen $f_r(e)$ en $f_l(e)$ toekennen die de grootte van het vlak rechts en links van e beschrijven. Dat kan voor alle georiënteerde bogen samen gemakkelijk in lineaire tijd gebeuren. Dan kunnen wij de string voor de georiënteerde boog $[a, b]$

bv. beginnen met $\deg(a), \deg(b), f_l(a,b), f_r(a,b)$ en eerst daarna het resultaat van Get_string(a,b) toevoegen. Als wij nu de kleinste string willen berekenen, kunnen wij eerst alle mogelijke startpunten met lexicografisch kleinste string $\deg(a), \deg(b), f_l(a,b), f_r(a,b)$ berekenen en dan alleen voor deze Get_string(a,b) toepassen. Of je kan nog voor $\deg(a), \deg(b), f_l(a,b), f_r(a,b)$ het aantal georiënteerde bogen met deze string plaatsen en daardoor van zo weinig mogelijk georiënteerde bogen moeten vertrekken. Dergelijke gemakkelijke trucjes zijn voldoende om dit algoritme zo snel te maken dat je gemakkelijk een toevallige graaf met meerdere honderdduizenden toppen in weinige seconden kan afwerken.

178 Oefening *Beschrijf een algoritme dat met input een ingebedde graaf in lineaire tijd alle georiënteerde bogen* $[a,b]$ *berekent waarvoor* $\deg(a), \deg(b), f_l(a,b), f_r(a,b)$ *minimaal is.*

Stel dat e, e' georiënteerde bogen zijn. Wij definiëren $e \equiv_= e'$ als en slechts als je vertrekkend vanuit e en e' dezelfde string krijgt. Dan is $e \equiv_= e'$ een equivalentierelatie. De equivalentieklassen zijn precies de orbits van de automorfismegroep (de groep van isomorfismen die de graaf op zichzelf afbeeldt) op de georiënteerde bogen – dus de maximale verzamelingen van georiënteerde bogen zodat voor elk paar van bogen e, e' in deze verzameling een automorfisme $f()$ bestaat met $f(e) = e'$.
Als je twee keer dezelfde string hebt gevonden – één keer vertrekkend van $[a,b]$ en één keer vertrekkend van $[x,y]$, dan heb je een automorfisme $f()$ gevonden. Nu kan je union-find algoritmen toepassen: als $f(e) = e'$ dan kan je de verzameling die e bevat en de verzameling die e' bevat samenvoegen. Deze verzamelingen zijn deelverzamelingen van de equivalentieklassen en natuurlijk moet je in elke equivalentieklasse maar één boog testen.

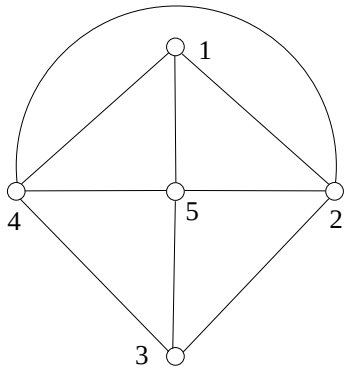
179 Algoritme Minstring_b_2(G)
Input: een ingebedde graaf $G = (V, E, opv())$.
In het begin bevat de optimale string str_opt[] alleen maar het getal ∞ *op positie* 0.

- *Bereken alle georiënteerde bogen* $[a,b]$ *waarvoor* $\deg(a), \deg(b), f_l(a,b), f_r(a,b)$ *minimaal is.*
- *Stop deze bogen in aparte verzamelingen* $M_1, \ldots, M_k$.
- *Herhaal de volgende lus totdat voor elke verzameling een string werd bepaald:*

- *Kies een verzameling M waarvoor nog geen string werd bepaald en een boog $[a, b] \in M$.*
- *Bereken newstring[] door Get_string'(a,b).*
    - *Als newstring[] < str_opt[] vervang str_opt[] door newstring[]*
    - *als newstring[] = str_opt[], bereken het automorfisme en verenig verzamelingen die bogen bevatten die door dit algoritme op elkaar worden afgebeeld.*

De beste manier om dit algoritme en de trucjes te verstaan is het algoritme en de trucjes gewoon toe te passen:

180 Oefening *Bereken de isomorfiebepalende string voor $\cong_b$ van de graaf in Figuur 34. Begin de string voor een boog $[a, b]$ met $\deg(a), \deg(b), f_l(a, b), f_r(a, b)$.*



Figuur 34: Bereken de isomorfiebepalende string voor $\cong_b$.

181 Oefening *Werk het geoptimaliseerde algoritme uit voor $\cong_i$. Let vooral op automorfismen die de oriëntatie niet bewaren!*

182 Oefening *Een Fullereen is een koolstofmolecule met de formule C_{2*n} voor een zekere n waarbij de bindingen tussen de atomen een vlakke 3-reguliere graaf beschrijven waarvan alle vlakken 5- of 6-hoeken zijn.*
Toon aan dat het aantal 5-hoeken een constante is en dat de algoritmen om een isomorfiebepalende string te berekenen die een string gebruiken die met $\deg(a), \deg(b), f_l(a, b), f_r(a, b)$ begint in lineaire tijd draaien.

5.2.4 Ingebedde grafen die als grafen isomorf zijn

Het is dus blijkbaar veel gemakkelijker te bepalen of twee ingebedde grafen isomorf zijn als **ingebedde** grafen (dus volgens $\cong_i$) dan of ze isomorf zijn als grafen ($\cong$). Het is wel duidelijk dat $G \cong_i G' \Rightarrow G \cong G'$ maar jammer genoeg geldt de andere richting niet (zie bv. Figuur 33).
Maar een stelling van Whitney zegt dat voor een belangrijke klasse van grafen wel beide richtingen geldig zijn:

183 Stelling *(Whitney)*
Gegeven twee vlakke 3-samenhangende grafen G, G'. Dan geldt
$G \cong_i G' \Leftrightarrow G \cong G'$

Bewijs: De richting $G \cong_i G' \Rightarrow G \cong G'$ is triviaal, stel dus dat $G \cong G'$ vermiddels het isomorfisme $f()$.

Wij gebruiken hier Lemma 159. Wij zullen eerst bewijzen dat als $[v, w]$ en $[w, x]$ en hoek vormen in $G = (V, E, \text{opv})$ dat dan ofwel $[v, w]$ en $[w, x]$ (oriëntatiebewarend) ofwel $[x, w]$ en $[w, v]$ (oriëntatie-omkerend) een hoek vormen in $G'(V', E', \text{opv}')$. Dan zullen wij bewijzen dat zich voor alle hoeken hetzelfde geval voordoet.

Inderdaad zullen wij bewijzen dat het beeld van een vlak een vlak is of een vlak is als je de richtingen van de georiënteerde bogen verwisselt.

Een cykel van georiënteerde bogen vormt een vlak als en slechts als er geen bogen links van de cykel liggen. Als (en slechts als) er geen bogen rechts van de cykel liggen, dan vormt de cykel met verwisselde richtingen een vlak.

Wij willen dus aantonen dat voor het beeld van een vlak geldt dat er ofwel rechts ofwel links geen bogen liggen.

Stel dus dat F een vlak is. Stel eerst dat er links en rechts van $f(F)$ ook toppen zijn – bv. x links en y rechts. Volgens Stelling 45 en Lemma 159 is er dan een pad van $f^{-1}(x)$ naar $f^{-1}(y)$ dat geen top van F bevat. Maar dan heb je een $B_{V(f(F))}(e)$ die bogen rechts en links van $f(F)$ bevat – in tegenspraak met Stelling 154.

De laatste mogelijkheid als noch rechts noch links een vlak is, is dus dat er aan één kant geen top maar alleen een boog $\{v, w\}$ is die (bv.) links van $f(F)$ zit en waarvoor dus $v, w \in f(F)$ – dat is dus een koorde. In G ligt de boog $\{f^{-1}(v), f^{-1}(w)\}$ rechts van F (omdat er links geen bogen zijn). Als wij nu in het midden van F een top x toevoegen en met $f^{-1}(v), f^{-1}(w)$ verbinden, dan hebben wij een cykel C' met lengte 3.

Stel dat P_1, P_2 de twee paden langs F van $f^{-1}(v)$ naar $f^{-1}(w)$ zijn. Er is ten minste één top $v_1 \in P_1$ en één top $v_2 \in P_2$ die niet in $\{f^{-1}(v), f^{-1}(w)\}$ zitten (anders zou je een dubbele boog hebben). Stel nu dat er een pad v_1 naar v_2 is. Omdat v_1 en v_2 aan verschillende kanten van C' liggen, moet elk pad van v_1 naar v_2 dus één van $x, f^{-1}(v), f^{-1}(w)$ bevatten – dus één van $f^{-1}(v), f^{-1}(w)$ omdat $x \notin G$. Omdat elk pad van v_1 naar v_2 één van deze twee toppen moet bevatten, kan de graaf niet 3-samenhangend zijn – het geval van een koorde kan zich dus niet voordoen.

Dus geldt inderdaad dat als $e' = \text{opv}(e)$ dan ook (a) $f(e') = \text{opv}'(f(e))$ of (b) $f(e) = \text{opv}'(f(e'))$. Het feit dat voor dezelfde $f()$ nooit één keer (a) en één keer (b) geldt volgt uit het feit dat voor één vlak altijd hetzelfde van de twee geldt (dat hebben wij net getoond) en dat rond een top ook altijd hetzelfde geval van toepassing moet zijn – anders zou je een geval tegenkomen waar voor twee verschillende gerichte bogen e, e'' zou gelden $\text{opv}'(e') = e$ **en** $\text{opv}'(e'') = e$ (met e een derde boog) – en dat kan volgens de vereisten aan $\text{opv}'()$ niet.

■

Deze stelling kan natuurlijk gebruikt worden om een efficiënt algoritme te ontwikkelen om ook gewone isomorfie van 3-samenhangende planaire grafen te testen: je berekent een vlakke inbedding voor elke graaf en test dan of de grafen als ingebedde grafen isomorf zijn. Voor het berekenen van een inbedding in het vlak bestaan er meerdere lineaire algoritmen (Hopcroft, Tarjan 1974 was de eerste en Boyer, Myrvold 2003 misschien de laatste tot nu toe) en ook een relatief gemakkelijk $O(|V|^2)$ algoritme van Demoucron, Malgrange und Pertuisetan dat wij in het volgende hoofdstuk zullen zien. Door 3-samenhangende componenten apart te behandelen (ook die kunnen in lineaire tijd berekend worden (Hopcroft, Tarjan 1973 – maar er waren meerdere verbeteringen van dit algoritme nodig)) kan dit algoritme (in zekere zin) uitgebreid worden op grafen die niet 3-samenhangend zijn.

5.3 Het testen van planariteit

Omdat de lineaire algoritmen toch iets ingewikkelder en technischer zijn, zullen wij hier het algoritme van Demoucron, Malgrange en Pertuisetan voorstellen.

184 Oefening *Toon aan dat als je een 2-samenhangende graaf $G = (V, E)$ hebt en een pad P waarvan de eindpunten in V liggen, $G \cup P$ ook 2-samenhangend is.*

Een belangrijke rol spelen hier opnieuw de verzamelingen $B_W(e)$ voor een zekere $W \subset V$, maar deze keer zijn sommige van de verzamelingen niet interessant – dat zullen de triviale verzamelingen zijn die uit maar één boog bestaan die in feite een boog is die wij al hebben ingebed. Of precies:

185 Definitie *Gegeven een graaf $G = (V, E)$ en $G' = (V', E') \subseteq G$. Dan noemen wij de grafen $B_{V'}(e)$ met $e \notin E'$ de restcomponenten van G en G'.*

186 Oefening *Toon aan dat voor een gegeven graaf $G = (V, E)$ en $W \subseteq V$ de relatie $e \equiv e'$ op de boogverzameling die gedefinieerd is door $e \equiv e'$ als en slechts als er een $e_0 \in E$ bestaat zodat $e, e' \in B_W(e_0)$ een equivalentierelatie is.*

187 Algoritme *(Demoucron, Malgrange en Pertuisetan)*
Stel dat G 2-samenhangend is – anders kunnen de 2-samenhangscomponenten apart ingebed worden en ten slotte kunnen deze componenten (zoals wij gezien hebben) zelfs in lineaire tijd gevonden worden.

- *Zoek een arbitraire cykel C van G en kies voor deze graaf G_1 de inbedding vast (dus de rotatievolgorde). Voor een cykel is er natuurlijk maar één mogelijke volgorde.*

(a) *G_i zij de subgraaf waarvoor er al een inbedding vastgelegd is – (in het geval $G_i = G$ zijn we dus klaar).*
Als $G_i \neq G$ dan bereken alle restcomponenten van G, G_i en bereken bovendien voor elke restcomponent B_j de verzameling $F(B_j, G_i)$ die bestaat uit alle vlakken met de eigenschap dat alle toppen van $B_j \cap V(G_i)$ in de rand van dit vlak liggen. Dat is in feite de verzameling van alle vlakken waarin B_j misschien nog geplaatst kan worden.

(b) *Als er een B_k is met $\#F(B_k, G_i) = 0$, output dat G niet planair is.*
Anders: Als er een B_k met $\#F(B_k, G_i) = 1$ is, neem f als het unieke vlak $f \in F(B_k, G_i)$,
anders kies een arbitraire B_k en een arbitrair vlak $f \in F(B_k, G_i)$.

(c) *Kies een pad P in B_k tussen twee toppen in $B_k \cap G_i$ en leg de inbedding (rotatievolgorde) vast zodat het pad in f terechtkomt. Dat betekent dat de eindbogen in de hoeken geplaatst worden die in het vlak f liggen. De toppen in het midden van het pad hebben graad 2 – daar bestaat dus maar 1 rotatievolgorde.*

Stel nu $G_{i+1} := G_i \cup P$
en begin opnieuw met a) en $i+1$ *in plaats van* i.

Dit algoritme is duidelijk polynomiaal (als je de stappen goed implementeert zal het in het slechtste geval tijd $O(|V|^2)$ vragen) omdat in elke doorloop van de lus ten minste één boog geplaatst wordt. Je moet je dus enkel ervan overtuigen dat één doorloop van de lus in polynomiale tijd ($O(|V|$ als goed geïmplementeerd) kan gebeuren. Het is duidelijk dat als het algoritme alle bogen inbed, de inbedding vlak is. In feite hebben wij in Lemma 144 heel gelijkaardige operaties gezien en bewezen dat voor het geval dat de twee hoeken in hetzelfde vlak liggen de Euler karakteristiek niet verandert. Het blijft dus bij de Euler karakteristiek van de oorspronkelijke graaf – en dat was een cykel, de Euler karakteristiek was dus gelijk aan 2.
Wat wij nog moeten bewijzen, is dat als het algoritme zegt dat een graaf niet planair is dat inderdaad ook het geval is.

188 Oefening *Gegeven een op de manier van Algoritme 187 verkregen deelinbedding* G_i. *Wat is dan het maximale* $\#F(B, G_i)$ *voor één van de restcomponenten* B.

189 Definitie *Een deelgraaf* G_i *van* G *die al planair ingebed is, heet* $\underline{G\text{-goed}}$ *als er een planaire inbedding van* G *bestaat zodat de inbedding van* $\overline{G_i}$ *een deelinbedding is.*

Het is duidelijk dat als G een vlakke graaf is de inbedding van G_1 G-goed is omdat er voor de toppen van een cykel maar één rotatievolgorde bestaat. Nu zullen wij met inductie bewijzen dat als G planair is alle G_i G-goed zijn wat betekent dat er een inbedding gevonden wordt omdat elke inbedding die nog niet de hele graaf bevat uitgebreid kan worden.

Schets van het bewijs:
In feite is de schets voldoende precies om de details gemakkelijk te kunnen invullen – dus bv. de hier gegeven tekeningen te vervangen en expliciet de rotaties rond de toppen en de hoeken van de vlakken te gebruiken.
Stel dat G planair is.
Inductie: wij veronderstellen dat G_i G-goed is. Wij moeten bewijzen dat dan ook het opgebouwde G_{i+1} G-goed is.
$\tilde{G}$ zij de graaf G die zo ingebed is dat de ingebedde G_i een ingebedde deelgraaf is. B zij de voor de volgende stap gekozen restcomponent.

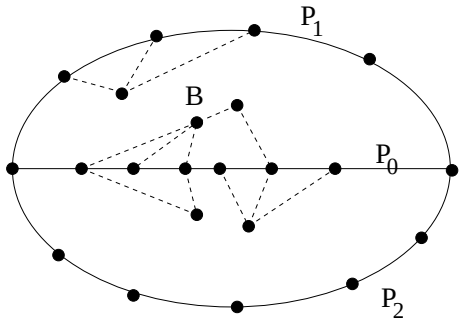
$\#F(B, G_i) = 1$**:** In dit geval moet het pad dat toegevoegd wordt ook in $\tilde{G}$ in hetzelfde vlak van G_i liggen. De eindbogen worden dus in dezelfde

hoeken toegevoegd en G_{i+1} is dus ook een ingebedde deelgraaf van $\tilde{G}$ dus G-goed.

$\#F(B, G_i) > 1$ **voor alle** B**:** Als het nieuw toegevoegde pad P in $\tilde{G}$ in hetzelfde vlak ligt (in dezelfde hoeken begint en eindigt) als in G_{i+1} is dat precies de definitie van G-goed – wij zijn dus klaar.

Stel dus dat P in een ander vlak f' ligt. Omdat de grafen G_i allemaal 2-samenhangend zijn (zie Oefening 184) zijn de randen van vlakken allemaal cykels (zie Stelling 157). Omdat G 2-samenhangend is kan geen restcomponent maar één top met de rest gemeen hebben. Wij kijken nu naar 2 gevallen:

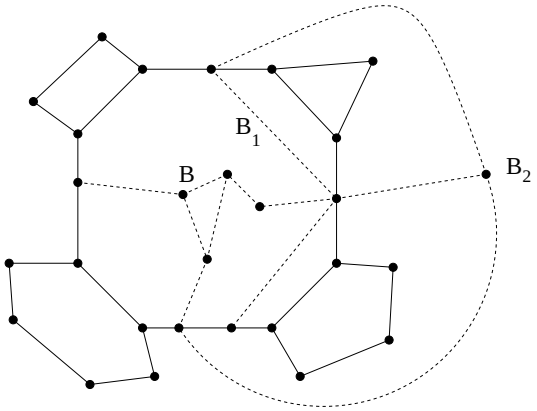
(a) $f \cap f'$ is samenhangend – dus (behalve in het eerste geval G_1 waar het een cykel is en de situatie nog altijd symmetrisch) een pad P_0. Stel dat P_1 en P_2 de twee overblijvende delen van f en f' zonder P_0 zijn. Er is dus geen restcomponent die interne toppen van P_1 of P_2 **en** P_0 bevat – omdat voor deze component B zou gelden dat $\#F(B, G_i) = 1$.



Restcomponenten die geen interne toppen van P_0 bevatten, kunnen *onafhankelijk* van B ingebed worden. Dat betekent dat om het even welke hoeken van P_0 gekozen worden om P in te bedden deze componenten op dezelfde manier ingebed kunnen worden.

Als wij dus in $\tilde{G}$ **alle** restcomponenten die toppen uit P_0 bevatten aan P gespiegeld in het andere vlak inbededden (dat wil zeggen dat voor alle hoeken die op P liggen het andere hoek wordt gekozen en de rotatievolgorde aan de toppen in de restcomponenten wordt omgekeerd) dan is het resultaat een ingebedde graaf $\tilde{G}'$ die toont dat G_{i+1} G-goed is.

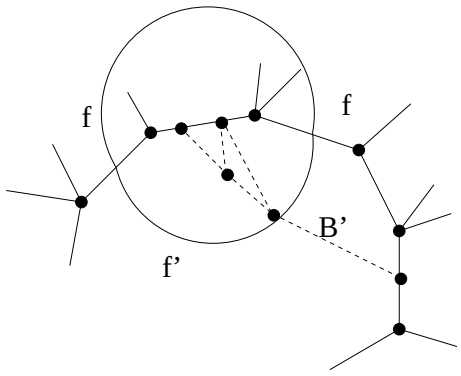
(b) $f \cap f'$ is niet samenhangend



Ook hier geldt dat geen restcomponenten B bestaan die toppen in $f \cap f'$ en $f - f'$ kunnen bevatten omdat anders $\#F(B, G_i) = 1$. Maar er zijn ook geen restcomponenten met toppen in verschillende componenten C_1, C_2 van $f - f'$:

Stel dat een dergelijke restcomponent B' wel bestaat en dat A_1, A_2 componenten van $f \cap f'$ zijn zodat de volgorde op de rand van f gelijk is aan A_1, C_1, A_2, C_2 (goed nadenken dat dat verondersteld mag worden).

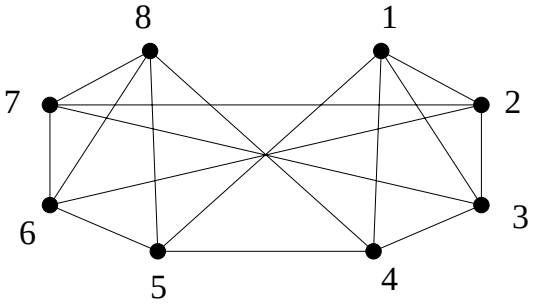
Als nu een intern punt a van A_1 en b van A_2 één keer door een pad in f en één keer door een pad in f' worden verbonden, is het resultaat een Jordaanse kromme J die door G_i alleen in a en b gekruisd wordt en waar C_1 en C_2 aan verschillende kanten liggen. De vlakken die niet f' zijn en waaraan C_1, C_2 liggen, zijn dus verschillend en daarom $\#F(B', G_i) = 1$ (een tegenstrijdigheid).



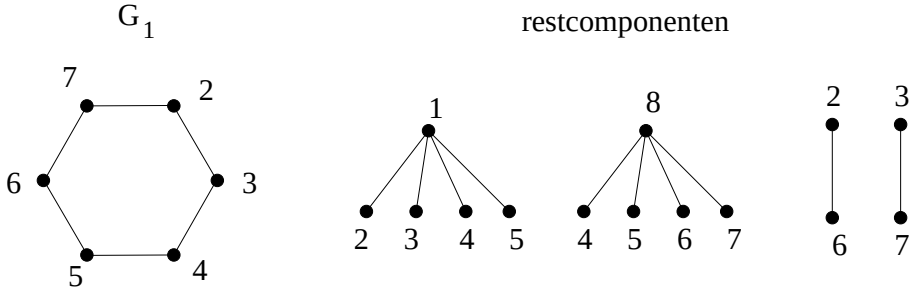
Maar restcomponenten die alle eindtoppen in dezelfde component van $f - f'$ hebben, kunnen opnieuw onafhankelijk van die met

eindpunten in $f \cap f'$ ingebed worden. Wij krijgen dus opnieuw een inbedding $\tilde{G}'$ die bewijst dat G_{i+1} G-goed is als wij voor alle restcomponenten met eindpunten in $f \cap f'$ de vlakken f en f' ruilen. Ook dit kan je als spiegelen aan een lijn of een cykel op een bol interpreteren om het beter te verstaan. Maar je kan het ook formeel uitdrukken als *toekennen van hoeken* en het is zeker een goede oefening dat iets preciezer uit te werken!

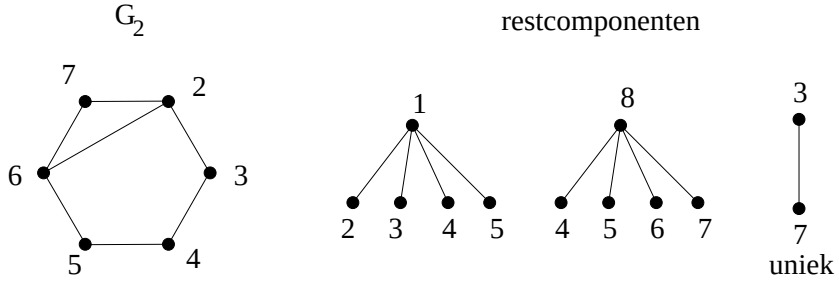
Voorbeeld:

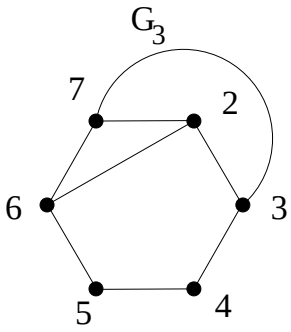


Kies een cykel – als dat snel kan natuurlijk een lange cykel. Optimaal zou hier $1, 2, 3, 4, 8, 7, 6, 5, 1$ zijn, maar dan zie je niet zo goed wat er gebeurt, dus:

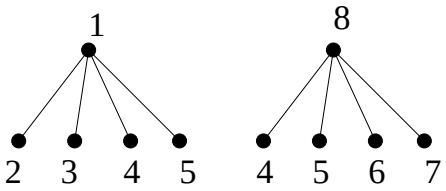


Natuurlijk zijn alle restcomponenten in beide vlakken in te bedden ($\#F(B, G_i) = 2$ voor alle B). Kies de restcomponent met de boog $\{2, 6\}$.

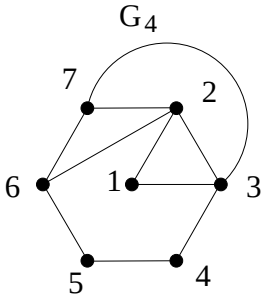




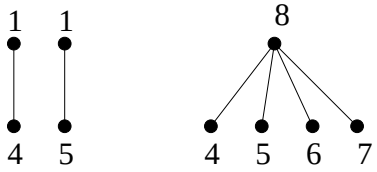
restcomponenten



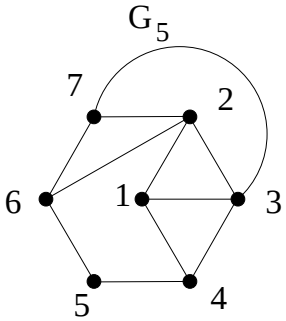
allebei uniek −− kies pad 2−1−3



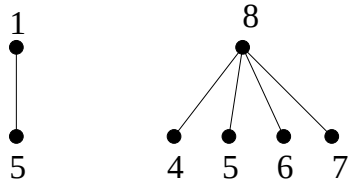
restcomponenten



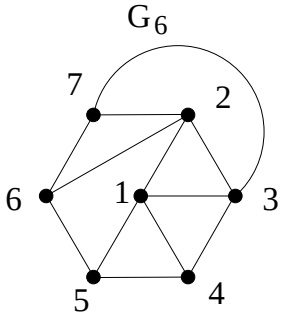
allemaal uniek − kies 1−4



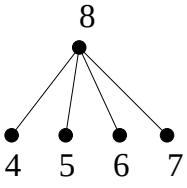
restcomponenten

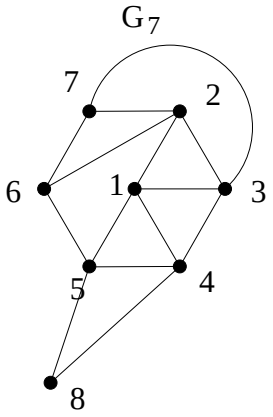


allemaal uniek − kies 1−5

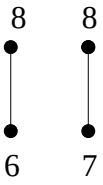


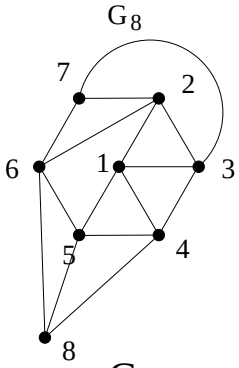
restcomponenten



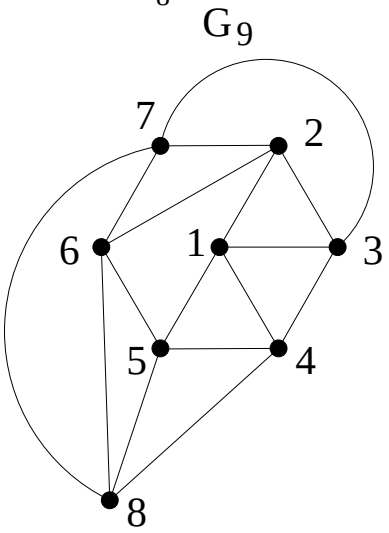


restcomponenten





restcomponenten



de graaf is dus planair

Dat was nu met heel kleine stappen. Een efficiënt algoritme zou natuurlijk niet altijd alle restcomponenten en $\#F(.,G_i)$ berekenen als dat niet nodig blijkt.

190 Oefening *Gebruik bv. het algoritme om voor de volgende grafen ofwel een vlakke inbedding te vinden ofwel te bewijzen dat een dergelijke inbedding niet bestaat.*

```
|20| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|
|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|
|  |14| 3| 2| 8|10|16| 9| 4| 7| 5|13|13|11| 1| 1| 6| 1| 3| 6| 5|
|  |15|14|14|14|15|19|17|18|18|19|16|15|12| 2| 5|11| 2| 4|10| 6|
|  |17|17|18|18|20|20|20|20|20|20|19|16|16| 3|12|12| 7| 8|11| 7|
|  |  |  |  |  |  |  |  |  |  |  |  |19|19| 4|14|13|15| 9|12| 8|
|  |  |  |  |  |  |  |  |  |  |  |  |  |15|17|17|16|17|13| 9|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |18|20|20|10|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |18|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |19|
```

```
|24| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|
|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|
|  | 2| 1| 1| 1| 1| 1| 2| 2| 2| 3| 4| 4| 5| 6| 7| 8| 8| 9|11|11|13|14|16|19|
|  | 3| 3| 2| 3| 4| 2| 6| 7| 3| 4|10| 5| 6| 7| 8|15| 9|10|17|12|14|15|17|20|
|  | 4| 6| 4| 5| 6| 5| 8| 9| 8| 9|12|11|12|13|14|17|16|11|18|13|20|16|19|21|
|  | 5| 7| 9|10|12| 7|14|15|10|11|18|13|14|15|16|22|18|17|20|19|22|21|21|22|
|  | 6| 8|10|11|13|13|15|16|17|18|19|20|20|21|22|23|19|19|23|21|23|23|22|23|
|  |  | 9|  |12|  |14|  |17|18|  |20|  |21|22|  |  |23|  |24|24|24|24|24|  |
```

```
|16| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|
|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|==|
|  | 9|10|10|13|11|11|11|11| 1| 1| 2| 5| 1| 2| 1| 1|
|  |10|11|13|14|12|12|12|12|  | 2| 5| 6| 3| 3| 2| 4|
|  |13|14|14|15|13|  |14|16|  | 3| 6| 7| 4| 4| 4| 8|
|  |15|15|  |16|14|  |  |  |  |  | 7| 8| 5| 5|  |  |
|  |16|  |  |  |  |  |  |  |  | 8|  |  | 7|  |  |
```

6 Moeilijke problemen

Bijzonder mooi en interessant zijn natuurlijk polynomiale grafenalgoritmen die een leuk idee gebruiken om efficiënt te werken. Maar het zou een helemaal foute indruk van grafenalgoritmen geven als wij het niet ten minste een beetje ook over problemen zouden hebben waarvoor er geen dergelijk algoritme is gekend. Jammer genoeg is dat voor heel veel problemen op grafen het geval...

Dus moeten wij het ten minste een beetje ook over NP-complete grafenproblemen hebben. Een heel goed boek op dit vlak is van M.R.Garey en

D.S.Johnson: Computers and Intractability. Sommige van de bewijzen in deze lesnota's zijn uit dit boek afkomstig.
Een heel korte en informele beschrijving van de klasse NP is als volgt (voor details volg het best de les *Complexiteit en Berekenbaarheid*):

191 Definitie *Wij schrijven A^* voor de verzameling van alle eindige strings met tekens uit een (eindig) alfabet A.*
Een acceptor is een computer die als input strings heeft (elementen uit A^* voor een A) en als output altijd alleen maar ja *of* neen.
Een niet-deterministische acceptor is een acceptor waarvan de stappen gerandomiseerd mogen zijn – bv. door gerandomiseerd te beslissen of iets gedaan wordt of niet. Een niet-deterministische acceptor mag voor dezelfde input soms ja en soms nee zeggen – hij definieert dus geen functie.
De aanvaarde taal van een acceptor is de verzameling van strings waarvoor het mogelijk is dat het resultaat ja *is – waarvoor er dus een aanvaardende berekening bestaat en om het even of er ook niet-aanvaardende berekeningen zijn.*

Een probleem identificeren wij nu met een taal: één mogelijke taal is bv. de verzameling van alle strings die een graaf beschrijven die een Hamiltoniaanse cykel heeft. Als wij een acceptor voor deze taal hebben en wij nemen als input een graaf dan weten wij als de output *ja* is dat de graaf een Hamiltoniaanse cykel heeft. Als de output *nee* is en de acceptor is niet-deterministisch weten wij niets...

192 Definitie *De klasse P is de klasse van talen (problemen) waarvoor er een (deterministische) acceptor bestaat, die precies deze taal aanvaardt en waarbij het aantal stappen begrensd is door een polynoom in de grootte van de input.*
De klasse NP is de klasse van talen (problemen) waarvoor er een niet-deterministische acceptor bestaat die precies deze taal aanvaardt en waarbij het aantal stappen begrensd is door een polynoom in de grootte van de input.

Inderdaad doet het er niet toe of je in de definitie van de klasse NP voor de berekening van de tijd rekening houdt met alle mogelijke berekeningen (ten slotte is het mogelijk dat er voor dezelfde input berekeningen van verschillende lengte zijn), alleen de aanvaardende berekeningen of alleen de snelste aanvaardende berekening. Als je alleen zou eisen dat de tijd van de snelste aanvaardende berekening begrensd is door een polynoom in de grootte van de input en geen eisen stellen voor alle andere berekeningen, zou je nog altijd dezelfde klasse NP krijgen – maar dat zullen wij hier niet bewijzen.

193 Definitie *Wij gaan ervanuit dat getallen in de invoer en uitvoer altijd op de gewone manier binair (misschien met een deel dat na de komma komt) voorgesteld worden – reële getallen die op deze manier geen eindige voorstelling hebben, kunnen dus niet voorgesteld worden.*
Een functie $f()$ die als origineel een adjacentiematrix van een graaf heeft en als beeld een string, noemen wij een codering van een graaf als $f()$ berekend kan worden en als er een berekenbare functie f^{-1} bestaat zodat als A de adjacentiematrix van een graaf is $f^{-1}(f(A))$ de adjacenciematrix van een isomorfe graaf is. Bovendien moet voor elke string beslist kunnen worden of hij het beeld van een adjacentiematrix van een graaf is. Voor een adjacentiematrix kan je gemakkelijk beslissen of die bij een graaf behoort.
Een codering waarvoor er algoritmen bestaan zodat $f()$ en f^{-1} in polynomiale tijd berekend kunnen worden en waarvoor ook in polynomiale tijd beslist kan worden of een string het beeld van een adjacentiematrix van een graaf is, noemen wij een polynomiale codering.

Hierbij was het inderdaad niet belangrijk dat wij van een adjacentiematrix vertrokken – andere (gewone) manieren om grafen in computers voor te stellen kunnen altijd op een polynomiale manier naar een adjacentiematrix vertaald worden – die zouden dus even geschikt geweest zijn om in deze definitie te gebruiken.

Voorbeeld: Het Hamiltoniaanse cykel probleem
Gegeven een polynomiale codering van grafen. Voor een gegeven input werk als volgt

- test of de input een graaf codeert. Als nee stop en output *nee*.
- kies elke boog in de graaf met kans bv. $\frac{1}{2}$.
- test of de gekozen bogen een Hamiltoniaanse cykel vormen:
  - als ja: stop en output *ja*
  - als nee: stop en output *nee*

Het is duidelijk dat je dit kan implementeren zodat het aantal stappen dat de computer doet polynomiaal begrensd is – het Hamiltoniaanse cykel probleem is dus zeker in NP .

194 Oefening *Beschrijf een manier om grafen als strings te coderen zodat de lengte van de code $O((|V| + |E|) \log |V|)$ is en zodat je in lineaire tijd (van de inputlengte) kan testen of de input een graaf codeert.*

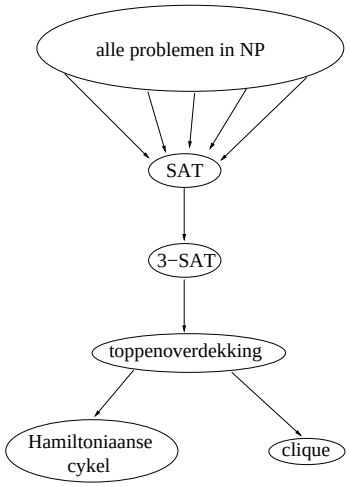
Wij zeggen dat je een taal T polynomiaal kan reduceren naar een taal T' als er een (deterministisch) programma bestaat dat opgestart op een input i (een arbitraire string) een output $f(i)$ (ook een string) bouwt zodat $f(i) \in T' \Leftrightarrow i \in T$. Formeel gesproken is $f()$ dus een functie van A^* naar A^* met $f(i) \in T' \Leftrightarrow i \in T$ die in polynomiale tijd berekenbaar is.
Als je dus de taal T' in polynomiale tijd kan herkennen, dan ook de taal T – je moet een input gewoon vertalen en dan bepalen of het resultaat in T' zit. Dit wordt soms geschreven als $T \leq_p T'$ en dat geeft een goede indruk ervan welk probleem *ten minste even moeilijk als het andere* is.
Het verrassende is nu dat de klasse NP enkele talen bevat zodat je **elk** probleem in NP polynomiaal kan reduceren naar elke van deze talen. Eén van deze talen is b.v het Hamiltoniaanse cykel probleem HC. Een andere taal in NP is bv. de verzameling van alle inputs die een graaf G en een getal k coderen waarvoor geldt dat G een verzameling M van k toppen bevat die onafhankelijk zijn – waarvoor dus geen boog bestaat waarvan beide eindpunten in M zijn.

195 Definitie *Een taal (probleem) heet NP-compleet als het in NP zit en* **elk** *ander probleem in NP polynomiaal op dit probleem gereduceerd kan worden.*

Het feit dat HC NP-compleet is, betekent dus dat er een algoritme bestaat dat een input i in polynomiale tijd zo verandert dat het resultaat een codering van een graaf met een Hamiltoniaanse cykel is als en slechts als i een graaf G en een getal k codeert zodat G een verzameling M van k onafhankelijke toppen bevat.
Voor het eerste probleem waarvan getoond werd dat het NP-compleet is (SAT, Cook 1971) was het natuurlijk bijzonder moeilijk omdat je moest bewijzen dat er voor **elke** taal een reductie bestaat – en dat zonder de taal echt te kennen. Voor de latere talen was het veel gemakkelijker: als $T, T' \in$ NP en je weet dat T NP-compleet is dan moet je alleen bewijzen dat je T naar T' kan reduceren om te tonen dat T' ook NP-compleet is: je weet dat je elke taal $T'' \in$ NP naar T kan reduceren en als je het resultaat daarna naar T' reduceert dan is de opeenschakeling van de reducties nog altijd polynomiaal begrensd en je hebt T'' naar T' gereduceerd.

Voor deze NP-complete problemen is geen polynomiaal algoritme gekend – en men denkt dat er ook geen bestaat. Maar als je voor één enkele van de problemen een dergelijk algoritme hebt, dan bestaat er ook een algoritme voor alle andere problemen (gebruik makend van de reductie).
Als je van een probleem kan bewijzen dat het NP-compleet is dan betekent dat dat er vermoedelijk geen polynomiaal tijdbegrensd algoritme bestaat dat het probleem oplost...

Als je van een probleem wil bewijzen dat het NP-compleet is, moet je dus een ander probleem waarvan al gekend is dat het NP-compleet is, naar dit probleem reduceren. Vaak is dat een beetje teleurstellend: als je noch van probleem A noch van probleem B het bewijs hebt gezien dat het NP-compleet is, helpt het niet echt dat probleem A op probleem B wordt gereduceerd. Als je van probleem A gewoon moet geloven dat het NP-compleet is, kan je het ook gewoon van probleem B geloven...
Maar toch: je moet natuurlijk ergens beginnen. Inderdaad is er een *natuurlijk* begin. Er is maar één probleem waarvan direct werd bewezen dat het NP-compleet is en dat is SAT. Sommigen van jullie hebben het bewijs voor dit probleem gezien en voor de anderen is dit misschien het probleem waarvoor het het gemakkelijkst is om te aanvaarden dat het NP-compleet is. Ons doel zal nu zijn voor sommige van de problemen waarvan jullie al vaak gehoord hebben dat die NP-compleet zijn, dat echt te bewijzen – vertrekkend van het oorspronkelijke probleem SAT.



Figuur 35: Deze „boom“ toont aan welk probleem op welk ander probleem wordt gereduceerd. Natuurlijk is het niet echt een boom omdat de problemen allemaal tot de bovenste top behoren (in NP zitten).

196 Definitie *De taal SAT is gedefinieerd als de verzameling van alle strings die met tekens uit* $A := \{0, 1, (,), x, \vee, \wedge, \neg\}$ *een vervulbare Booleaanse uitdrukking in conjunctieve normaalvorm coderen. Daarbij zijn de variabelen de (maximale) deelstrings van de vorm* xs *waarbij* s *een string bestaande uit* 0 *en* 1 *is.*

Het gaat om uitdrukkingen van de vorm $K_1 \wedge \cdots \wedge K_t$ waarbij elke K_i van de vorm $(y_1 \vee y_2 \vee \cdots \vee y_s)$ is, waarbij geldt dat y_i een variabele is of in $\{0, 1\}$ of $y_i = \neg z_i$ met z_i een variabele. Zo'n K_i heet een clause en zo'n y_i heet een literaal.

197 Voorbeeld

- $(x11 \vee x01) \wedge (x10 \vee \neg x01)$ *is in SAT omdat het in conjunctieve normaalvorm en vervulbaar is – bv. door* $x11 = x01 = x10 = 1$.
- $(x11 \vee x01) \wedge (x10 \vee x01) \wedge (\neg x10 \vee \neg x11) \wedge (0 \vee \neg x01)$ *is niet in SAT omdat het wel in conjunctieve normaalvorm maar niet vervulbaar is.*
- $(x11 \vee x01 \wedge x001) \wedge (\neg x10 \vee x01)$ *is niet in SAT omdat het niet in conjunctieve normaalvorm is. Het is wel vervulbaar.*
- $x \vee 11$ *is niet in SAT omdat het gewoon geen Booleaanse uitdrukking codeert.*

198 Stelling *(Cook 1971)*
SAT is NP-compleet.

Bewijs: Het bewijs kunnen wij hier onmogelijk geven. Het is vrij ingewikkeld en de details zijn alles behalve triviaal. Als jullie de les „Complexiteit en Berekenbaarheid" volgen, zullen jullie het bewijs natuurlijk zien...

Hier alleen een **heel grove schets** van het idee achter het bewijs:

Als een probleem in NP zit, dan bestaat er een computerprogramma en een polynoom $p()$ zodat met input x het programma de input in $p(|x|)$ stappen aanvaardt – of met andere woorden: er bestaat een aanvaardende berekening met lengte ten hoogste $p(|x|)$. Het bewijs neemt nu het programma en bouwt (in polynomiale tijd) een Booleaanse uitdrukking in conjunctieve normaalvorm waarvoor een vervullende belegging *overeenkomt* met een aanvaardende berekening van het programma opgestart op deze input. Maar de details zijn vrij ingewikkeld...

■

Inderdaad is dit probleem het begin van alle NP-complete problemen en het werd onmiddellijk vertaald naar een probleem dat iets gemakkelijker *lijkt* en geschikter is om te reduceren. Dit probleem – 3-SAT en de reductie van SAT naar 3-SAT heeft niets te maken met grafentheorie, maar omdat wij het hele

pad vanaf SAT willen kennen voor sommige grafentheoretische problemen zullen wij het (gemakkelijke) bewijs hier toch geven. Omdat sommigen van jullie het bewijs al kennen, zullen wij het in de les er niet over hebben.

199 Definitie *De taal 3-SAT is gedefinieerd als de deelverzameling van SAT met de eigenschap dat alle clauses 3 literalen bevatten.*

200 Stelling

3-SAT is NP-compleet.

Bewijs: Het is gemakkelijk (zoals voor de meeste NP-complete problemen) te zien dat 3-SAT $\in$ NP:

- test of de input een Booleaanse uitdrukking in conjunctieve normaalvorm codeert met 3 literalen in elke clause. Als nee: output *nee* en stop.
- kies toevallig waarden 0 en 1 voor de variabelen en vul ze in
- als de zo gevormde uitdrukking de waarde 1 heeft, output *ja* anders output *nee*

Wij kunnen 3-SAT gemakkelijk naar SAT reduceren: test gewoon of een input in conjunctieve normaalvorm is en elke clause 3 literalen bevat. Als ja: wijzig de input niet. Als nee: output een arbitraire string die niet in SAT is (bv. „$\wedge \wedge \wedge$“).

Jammer genoeg is dat niet de richting die wij nodig hebben...

Om aan te tonen dat inderdaad elk probleem uit NP naar 3-SAT gereduceerd kan worden, zullen wij SAT op 3-SAT reduceren – als dat lukt dan kan elk probleem uit NP via SAT polynomiaal op 3-SAT gereduceerd worden.

Dat zullen wij doen door voor elke Booleaanse uitdrukking b in conjunctieve normaalvorm een equivalente uitdrukking b' in conjunctieve normaalvorm aan te geven waarin elke clause 3 literalen bevat. Daarbij zal het mogelijk zijn b' door een computer in polynomiale tijd (een polynoom afhankelijk van de lengte van de inputstring) op te bouwen.

Wij zullen elke clause $(x_1 \vee x_2 \vee \cdots \vee x_n)$ van b vervangen door een uitdrukking in conjunctieve normaalvorm waarin elke clause 3 literalen heeft. Daarvoor hebben wij sommige hulpvariabelen y_i nodig die voor

de verschillende clauses van b verschillend zijn – maar om het gemakkelijker te kunnen lezen zullen wij de extra index j voor een clause die nodig is niet altijd mee opschrijven.

Wij kijken naar verschillende gevallen voor het aantal n van variabelen in de clause $(x_1 \vee x_2 \vee \cdots \vee x_n)$:

$n = 1$**:** extra variabelen: y_1, y_2.
Vervang (x) door $(x \vee 0 \vee 0)$ of $(x \vee x \vee x)$. Maar zelfs dan als je wilt dat alle literalen verschillend zijn en niet constant, kan het:
Met de extra variabelen: y_1, y_2:
$(x \vee y_1 \vee y_2) \wedge (x \vee \bar{y}_1 \vee y_2) \wedge (x \vee y_1 \vee \bar{y}_2) \wedge (x \vee \bar{y}_1 \vee \bar{y}_2)$

$n = 2$**:** Vervang $(x_1 \vee x_2)$ door $(x_1 \vee x_2 \vee 0)$ of $(x_1 \vee x_2 \vee x_1)$
of met de extra variabele: y_1:
$(x_1 \vee x_2 \vee y_1) \wedge (x_1 \vee x_2 \vee \bar{y}_1)$

$n = 3$**:** Deze clauses kunnen zo blijven.

$n \geq 4$**:** extra variabelen: $y_1, \ldots, y_{n-3}$.
Vervang $(x_1 \vee \cdots \vee x_n)$ door $(x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee y_3) \wedge \cdots \wedge (\bar{y}_{n-4} \vee x_{n-2} \vee y_{n-3}) \wedge (\bar{y}_{n-3} \vee x_{n-1} \vee x_n)$

Voor de gevallen $n \in \{1, 2, 3\}$ is het onmiddellijk duidelijk dat een vervullende belegging voor de oorspronkelijke clause in een vervullende belegging voor de nieuwe uitdrukking vertaald kan worden die voor de x_i die in beide uitdrukkingen voorkomen dezelfde waarden heeft. En ook de andere richting is duidelijk.

Maar het geval $n \geq 4$ heeft misschien een beetje uitleg nodig:

Als $(x_1 \vee \cdots \vee x_n)$ vervuld is dan bestaat er een literaal x_j met waarde 1.

Als $j \in \{1, 2\}$ kan je alle y_i als 0 kiezen en als $j \in \{n-1, n\}$ kan je alle y_i als 1 kiezen om een vervullende belegging te krijgen.

Stel dus dat $3 \leq j \leq n-2$. De variabelen $y_{j-1}, \bar{y}_{j-2}$ staan dus in één clause met x_j.

Beleg alle y_k met $k \geq j-1$ met 0 en alle y_k met $k \leq j-2$ met 1. Dan staat er in elke clause (behalve $(\bar{y}_{j-2} \vee x_j \vee y_{j-1})$ waar dat ook niet nodig is) een bij een y behorend literaal met waarde 1. Dus levert deze belegging de waarde 1 voor de hele uitdrukking.

Nu de andere richting:

Als er een vervullende belegging van $(x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee y_3) \wedge \cdots \wedge (\bar{y}_{n-4} \vee x_{n-2} \vee y_{n-3}) \wedge (\bar{y}_{n-3} \vee x_{n-1} \vee x_n)$ is gegeven dan

moeten wij aantonen dat één van de literalen x_i de waarde 1 oplevert en dus ook $(x_1 \vee \cdots \vee x_n)$ vervuld is. Stel dat alle x_i de waarde 0 opleveren. Dan moet y_1 met 1 belegd zijn om de eerste clause te vervullen en y_{n-3} met 0 belegd zijn om de laatste clause te vervullen. Er bestaat dus een k zodat y_k met 1 belegd is en y_{k+1} met 0. Maar dan is de waarde van de clause $(\bar{y}_k \vee x_{k+2} \vee y_{k+1})$ en dus van de hele uitdrukking ook 0 (een tegenstrijdigheid). Dus moet één van de literalen x_i $1 \leq i \leq n$ de waarde 1 hebben en dus is ook de lange clause vervuld.

Van het feit dat de geziene vertaling in polynomiale tijd kan gebeuren kunnen jullie jullie gemakkelijk overtuigen.

■

201 Oefening *Toon aan dat 3-SAT in polynomiale tijd gereduceerd kan worden op het deelprobleem (de deelverzameling van 3-SAT) waar er alleen maar variabelen en geen constanten in clauses opduiken.*

Nu hebben wij een basis en kunnen wij ons bezig houden met problemen die wij (in deze les) interessanter vinden: grafentheoretische problemen!
Eerst zullen wij het over het toppenoverdekkingsprobleem hebben. Een korte herhaling (zie Definitie 86):

202 Definitie *Gegeven een graaf $G = (V, E)$. Een verzameling $C \subseteq V$ heet een toppenoverdekking (vertex cover) van G als voor alle $e \in E$ geldt dat $e \cap C \neq \emptyset$ (dus: als voor elke boog ten minste één van de eindtoppen in C zit).*

203 Definitie *De taal VC is de verzameling van alle strings van de vorm $s\#s'$ waarbij s een graaf G codeert en s' de binaire voorstelling van een getal k is en waarvoor geldt dat G een toppenoverdekking met k of minder toppen heeft.*
De codering kan bv. de codering uit Oefening 194 zijn of een andere – vaste – polynomiale codering.

204 Stelling *VC is NP-compleet.*

Bewijs: Het is opnieuw gemakkelijk om te zien dat VC $\in$ NP:

- test of de input een graaf $G = (V, E)$ en een getal k codeert. Als nee: zeg *nee* en stop.
- als $k > |V|$ zeg *ja* en stop.

- kies k toevallige toppen en test of het een toppenoverdekking is.
  - als ja: zeg *ja* en stop.
  - als nee: zeg *nee* en stop.

Het is duidelijk dat dit algoritme precies VC aanvaardt en dat het in polynomiale tijd draait.

Nu nog het moeilijkere gedeelte: aantonen dat elk probleem in NP polynomiaal op VC gereduceerd kan worden...

Wij reduceren 3-SAT naar VC. Wij moeten dus een polynomiaal algoritme beschrijven dat een input i omzet naar een output o zodat i een vervulbare Booleaanse uitdrukking in conjunctieve normaalvorm met 3 literalen in elke clause codeert als en slechts als o van de vorm $o1\#o2$ is waarbij $o2$ een getal k codeert en $o1$ een graaf G waarvoor er een toppenoverdekking met k of minder toppen bestaat.

Het algoritme werkt als volgt:

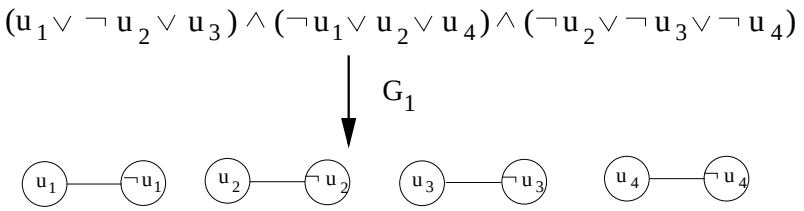
- test of i een Booleaanse uitdrukking in conjunctieve normaalvorm met 3 literalen in elke clause codeert
  - als nee: output bv. een string die de complete graaf met 2 toppen K_2 codeert en het getal 0 en stop. Je kan ook een string uitvoeren die helemaal geen graaf codeert...
  - als ja: vertaal de Booleaanse uitdrukking naar een graaf G en een getal k zodat G een toppenoverdekking met k of minder toppen heeft als en slechts als de uitdrukking in conjunctieve normaalvorm vervulbaar is.

Wij zullen het hier niet over het gedeelte hebben dat test of i een Booleaanse uitdrukking in conjunctieve normaalvorm met 3 literalen in elke clause codeert – dat is gemakkelijk – maar onmiddellijk stellen dat dat zo is.

Het probleem is natuurlijk het vertalen... Wij stellen daarbij dat de clauses alleen variabelen bevatten – anders kunnen wij de input zoals in Oefening 201 omvormen.

Stel dat de variabelen $u_1, \ldots, u_n$ zijn en dat de clauses $c_1, \ldots, c_m$ zijn.

Een eerste deelgraaf van de graaf die wij nu opbouwen is gegeven door $G_1 = (V_1, E_1)$ waarbij $V_1 = \{u_1, \ldots, u_n\} \cup \{\bar{u}_1, \ldots, \bar{u}_n\}$ en $E_1 = \{\{u_i, \bar{u}_i\} | 1 \leq i \leq n\}$.



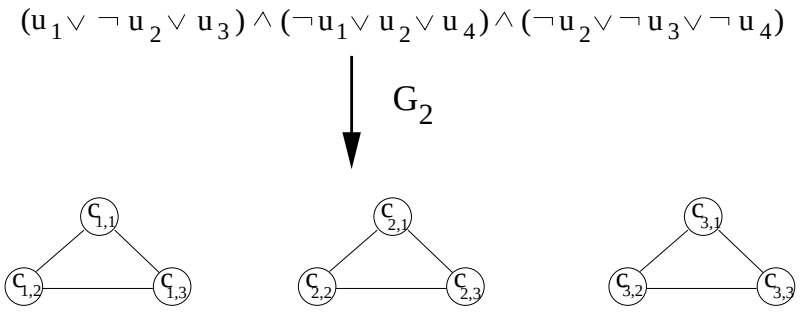
Figuur 36: Het vertalen van een Booleaanse uitdrukking naar de deelgraaf G_1. Die hangt inderdaad alleen maar af van de variabelen.

De interpretatie van deze deelgraaf zal later de belegging van de variabelen zijn: je hebt ten minste n toppen voor een overdekking nodig. Als je een overdekking C met **precies** n toppen hiervoor hebt dan kan je die als volgt als belegging van de variabelen van de formule interpreteren: de waarde van een variabele u_i is 1 als $u_i \in C$ en 0 als $\bar{u}_i \in C$.

Nu zullen wij een tweede deelgraaf $G_2 = (V_2, E_2)$ opbouwen die iets te maken heeft met het vervullen van de clauses:

$V_2 = \{c_{i,j} | 1 \leq i \leq m, \quad 1 \leq j \leq 3\}$ en $E_2 = \{\{c_{i,j}, c_{i,j'} | 1 \leq i \leq m, \quad 1 \leq j, j' \leq 3, \quad j \neq j'\}$

Wij maken dus voor elke clause een complete graaf K_3 met 3 toppen aan. Een toppenoverdekking C van G_2 heeft dus ten minste $2 * m$ toppen nodig en als het precies $2 * m$ toppen zijn dan zitten in elke kopie van K_3 precies twee toppen van C.



Figuur 37: Het vertalen van een Booleaanse uitdrukking naar de deelgraaf G_2. Die hangt inderdaad alleen maar af van het aantal clauses.

Een toppenoverdekking van de graaf $G_1 \cup G_2$ heeft dus ten minste

$n + 2 * m$ toppen nodig. Als wij nu nog bogen toevoegen dan kan dit getal dus alleen maar stijgen.

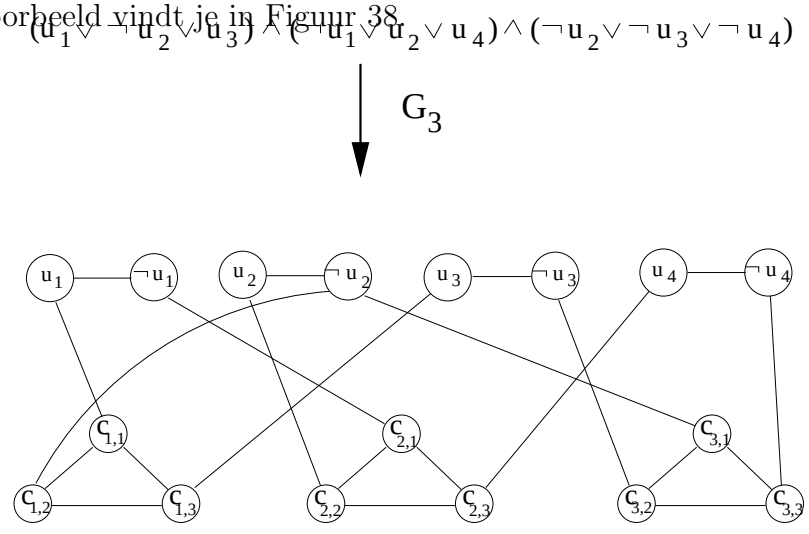
Tot nu toe hebben wij nog geen rekening gehouden met de echte structuur van de Booleaanse uitdrukking – dus welke variabele in welke clause zit en of er het teken $\neg$ staat. Dat zal nu gebeuren door meer bogen toe te voegen. Dat zal op een manier gebeuren dat de Booleaanse uitdrukking vervulbaar is als en slechts als deze bogen al door een toppenoverdekking van $G_1 \cup G_2$ met $n + 2 * m$ toppen mee overdekt kunnen worden:

In elke clause c_i zitten 3 literalen $c'_{i,1}, c'_{i,2}, c'_{i,3}$ (wij schrijven hier $c'_{i,j}$ in plaats van $c_{i,j}$ om ze niet te verwisselen met de toppen van de graaf). Deze literalen zijn allemaal variabelen of variabelen met een $\neg$ ervoor – dus van de vorm u_i of $\neg u_i$.

De graaf in de volledig vertaalde Booleaanse uitdrukking wordt beschreven door $G_3 = (V_3, E_3)$ waarbij

- $V_3 = V_1 \cup V_2$.
- $E_3 = E_1 \cup E_2 \cup \{\{c_{i,j}, u_k\} | c'_{i,j} = u_k, c_{i,j} \in V_2, u_k \in V_1\}$
  $\cup \{\{c_{i,j}, \neg u_k\} | c'_{i,j} = \neg u_k, c_{i,j} \in V_2, \neg u_k \in V_1\}$

Wij voegen tot de al beschreven bogen dus nog een boog toe tussen elk literaal en de posities waar het literaal in een clause voorkomt. Een voorbeeld vindt je in Figuur 38.



Figuur 38: Het vertalen van een Booleaanse uitdrukking naar de graaf G_3. Dat is nu de complete vertaling....

De volledige vertaling van de input is nu eerst een codering van G_3, dan een # en dan de binaire voorstelling van $k = n + 2 * m$. Deze vertaling kan zeker (wel, let op de details...) als programma geïmplementeerd worden dat in polynomiale tijd draait. Maar wij moeten nog tonen dat voor de output o en de input i geldt dat $i \in$ 3-SAT $\Leftrightarrow o \in VC$.

$\Rightarrow$: Stel dat een string $s \in$ 3-SAT gegeven is waarbij $U = \{u_1, \ldots, u_n\}$ de verzameling van variabelen is. Dan bestaat er een belegging $f : U \to \{0, 1\}$ die de gecodeerde uitdrukking vervult.

Kies nu een toppenverzameling als volgt: Als $f(u_i) = 1$ kies u_i en anders kies $\neg u_i$. Op deze manier kies je n toppen.

Stel nu dat $c_{i,1}, c_{i,2}, c_{i,3}$ de toppen zijn die bij een clause c_i behoren. Dan is er ten minste één j zodat literaal $c'_{i,j}$ de waarde één heeft. Kies de andere twee. Zo kies je nog eens $2m$ toppen.

Wij moeten nu tonen dat dit een toppenoverdekking is. Het is duidelijk dat alle bogen in $E_1 \cup E_2$ overdekt zijn. Stel nu dat een boog $\{c_{i,j}, u\}$ gegeven is met $u \in \{u_k, \neg u_k\}$ voor een zekere k. Als u gekozen werd is deze boog overdekt. Stel dus dat u niet gekozen werd. Dan is de waarde van het literaal $c'_{i,j}$ dus 0. Dat betekent dat ten minste één van de andere literalen in de clause waarde één heeft. Maar om het even welk ander literaal dat is, $c_{i,j}$ werd zeker gekozen en de boog is dus overdekt.

$\Leftarrow$: Stel nu dat een string $s \in VC$ is gegeven die de output van de beschreven routine is. Er bestaat dus een toppenoverdekking C met $n+2m$ toppen en de structuur van de deelgrafen G_1, G_2 toont dat voor elke verzameling $\{u_k, \neg u_k\}$ precies één element in C moet zitten en voor elke verzameling $c_{i,1}, c_{i,2}, c_{i,3}$ precies 2. Definieer nu voor de formule die vertaald werd de belegging

$$f(u_k) := \begin{cases} 1 & \text{als } u_k \in C \\ 0 & \text{als } \neg u_k \in C \end{cases}$$

Stel dat c_i een clause is. Dan is er één literaal $c'_{i,j}$ zodat $c_{i,j} \notin C$. Maar dan moet de top uit V_1 waarmee $c_{i,j}$ verbonden is in C zitten (toppenoverdekking). Dat betekent dat de waarde van het literaal 1 is. De waarde van elke clause is dus 1 en dus ook de waarde van de hele uitdrukking.

■

Het mooie aan dit bewijs is dat jullie direct het verband zien tussen zo'n Booleaanse formule en het probleem op een dergelijke graaf...

Soms is het gemakkelijker deelproblemen op te lossen (anders geformuleerd: deelverzamelingen van een taal kunnen soms gemakkelijker herkend worden). Definieer bv.

205 Definitie *Gegeven een vaste polynomiale codering van grafen.*
De taal $\underline{VC_i}$ is de verzameling van alle strings van de vorm $s\#k$ waarbij s een i-reguliere graaf G codeert en k de binaire voorstelling van een getal is en waarvoor geldt dat G een toppenoverdekking met k of minder toppen heeft.

Dan is de taal VC_2 zeker **niet** NP-compleet (behalve als P = NP) omdat je voor 2-reguliere grafen (verzamelingen van cykels) gemakkelijk de grootte van een kleinste toppenoverdekking zelfs in lineaire tijd kan berekenen!

206 Oefening *Toon aan dat VC_3 wel NP-compleet is.*

Nu dat wij weten dat VC NP-compleet is kunnen wij dat voor nog twee problemen heel gemakkelik aantonen:

207 Definitie *Gegeven een vaste polynomiale codering van grafen.*
De taal $\underline{CL}$ ($\underline{\text{clique}}$, $\underline{\text{kliek}}$) is de verzameling van alle strings van de vorm $s\#s'$ waarbij s een graaf G codeert en s' de binaire voorstelling van een getal k is en waarvoor geldt dat G een clique – dat is een complete deelgraaf – met ten minste k toppen bevat.
De taal $\underline{IS}$ ($\underline{\text{independent set}}$, $\underline{\text{onafhankelijke verzameling}}$) is de verzameling van alle strings van de vorm $s\#s'$ waarbij s een graaf G codeert en s' de binaire voorstelling van een getal k is en waarvoor geldt dat G een onafhankelijke verzameling met ten minste k toppen bevat (dus een verzameling van ten minste k toppen zodat er geen boog bestaat met beide eindpunten in de verzameling).

Wij zullen het in de volgende stellingen niet meer over het *preprocessing* hebben – het testen of het een code is die een graaf of een Booleaanse formule, etc is. Dat is wel een belangrijke stap, maar omdat het niet moeilijk is en omdat wij weten dat het gedaan moet worden, moeten wij het niet altijd expliciet zeggen.

208 Stelling *De talen IS en CL zijn NP-compleet.*

Bewijs: Dat de talen in NP zitten is duidelijk.

Inderdaad kan VC gemakkelijk naar IS gereduceerd worden: als C een toppenoverdekking met grootte k in een graaf G is, dan is $V \setminus C$ een onafhankelijke verzameling met groote $|V| - k$ (en omgekeerd). G heeft

dus een toppenoverdekking met ten hoogste k toppen als en slechts als G een onafhankelijke verzameling met ten minste $|V| - k$ toppen heeft.

De reductie is dus gewoon het vervangen van de binaire voorstelling van k in de input door de binaire voorstelling van $|V| - k$ (en de codering van de graaf houden).

Maar nu kunnen wij ook gemakkelijk aantonen dat CL NP-compleet is: wij reduceren IS naar CL: een clique in een graaf $G = (V, E)$ is een onafhankelijke verzameling in het complement G^c (dus de graaf $G = (V, \binom{V}{2} \setminus E)$). Wij moeten dus gewoon de codering van de inputgraaf G vervangen door de codering van G^c (en de codering van het getal k houden).

■

Dat waren inderdaad twee gemakkelijke gevallen. Nu zullen wij nog één probleem zien dat heel belangrijk is en waarvoor het bewijs dat het NP-compleet is niet zo triviaal is (dus twee redenen om ernaar te kijken):

209 Definitie *Gegeven een vaste polynomiale codering van grafen.*
De taal HC (Hamiltoniaanse cykel) is de verzameling van alle strings s die een graaf G coderen die een Hamiltoniaanse cykel bevat – een cykel die elke top bevat.

210 Stelling *HC is NP-compleet.*

Bewijs: Dat HC in NP is, is duidelijk. Nu moeten wij nog een ander NP-compleet probleem op HC reduceren.

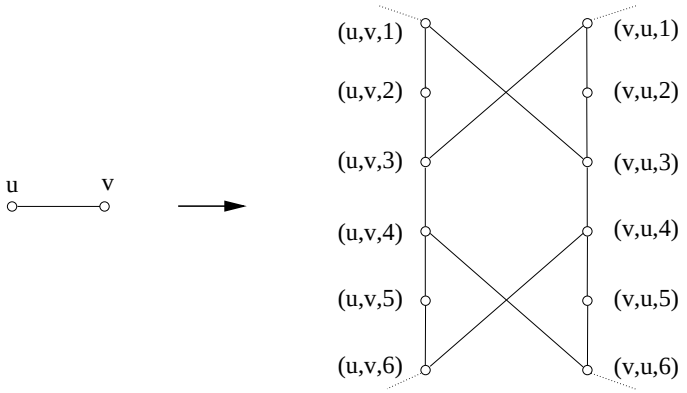
Wij reduceren VC naar HC:

Input is dus een graaf $G = (V, E)$ en een getal k en wij moeten in polynomiale tijd een graaf $G' = (V', E')$ opbouwen die een Hamiltoniaanse cykel bevat als en slechts als G een toppenoverdekking met k toppen bevat.

Wij zullen het opnieuw niet over het preprocessing hebben, maar dat moet natuurlijk gebeuren...

Wij zullen eerst testen of $k \leq |V|$. Als dat niet zo is dan heeft G zeker een toppenoverdekking met k of minder toppen en wij schrijven bv. een cykel C_3 als uitvoer. Deze stap is belangrijk omdat wij later k toppen $a_1, \ldots, a_k$ zullen aanmaken en dat zou natuurlijk exponentieel veel tijd kunnen vragen als k groot is en dan zou de reductie niet meer polynomiaal zijn.

Onze graaf wordt opnieuw uit verschillende delen samengebouwd. Eerst bouwen wij een deelgraaf $G_1 = (V_1, E_1)$ die voor elke boog $\{u, v\}$ een component met 12 toppen zoals in Figuur 39 bevat.



Figuur 39: De deelgraaf die voor elke boog $\{u, v\}$ wordt opgebouwt. De vier maar aangeduide halve bogen zullen dit deel later met andere delen verbinden.

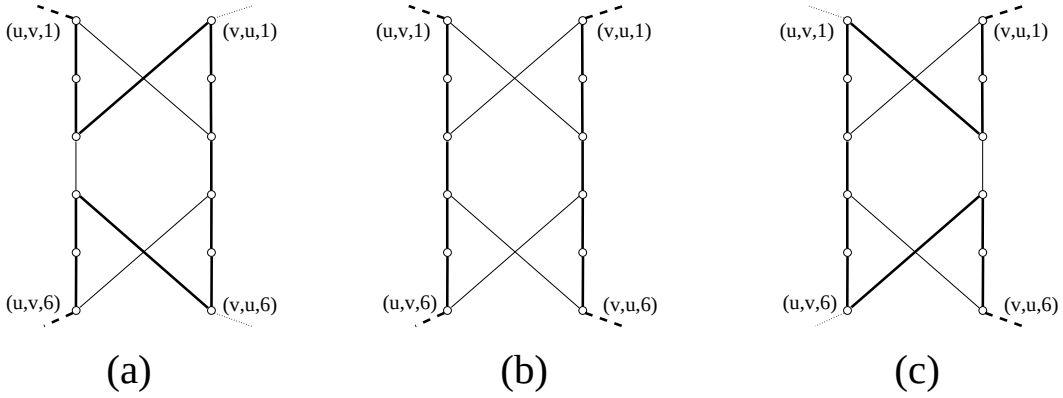
Of precies:

$V_1 = \{(u, v, i) | \{u, v\} \in E, \quad 1 \leq i \leq 6\}$

Let op, dat voor $\{u, v\} \in E$ de toppen (u, v, i) en (v, u, i) verschillend zijn, omdat in een 3-tal de volgorde wel belangrijk is.

$$\begin{aligned} E_1 = &\{\{(u, v, i), (u, v, i + 1)\} | \{u, v\} \in E, \quad 1 \leq i < 6\} \\ &\cup \{\{(u, v, 1), (v, u, 3)\} | \{u, v\} \in E\} \\ &\cup \{\{(u, v, 4), (v, u, 6)\} | \{u, v\} \in E\} \end{aligned}$$

Later zullen deze delen door bogen naar één van de toppen (u, v, i) met $i \in \{1, 6\}$ met de rest van de graaf verbonden worden. Maar nu al kunnen wij zien hoe een Hamiltoniaanse cykel – als die bestaat – door deze delen zou moeten lopen. De drie mogelijkheden zien jullie in Figuur 40. Wat wij later zullen gebruiken, is dat een deelpad van een Hamiltoniaanse cykel een dergelijk deel altijd aan een top verlaat die dezelfde eerste index heeft – alleen de laatste index verschilt (1 als het pad met 6 begon en 6 als het pad met 1 begon).



Figuur 40: De drie mogelijkheden voor een Hamiltoniaanse cykel om de toppen van deze deelgraaf te doorlopen. Een deelpad van een Hamiltoniaanse cykel verlaat een deelgraaf altijd aan een top met dezelfde eerste index.

Nu maken wij nog k toppen $a_1, \ldots, a_k$ aan die later zullen aantonen welke toppen in G in een overdekking moeten zitten. Dus

$V' = V_1 \cup \{a_1, \ldots, a_k\}$

Om de componenten met elkaar te verbinden kiezen wij rond elke top u een vaste volgorde $v_1(u), v_2(u), \ldots, v_{\deg(u)}(u)$ van zijn buren. Hier heb je dus een begin- en een eindtop in de reeks van buren, maar elke manier om deze volgorde te kiezen zal een geldige reductie opleveren. Wij verbinden de componenten die bij met u incidente bogen behoren zoals in Figuur 41. Of precies:

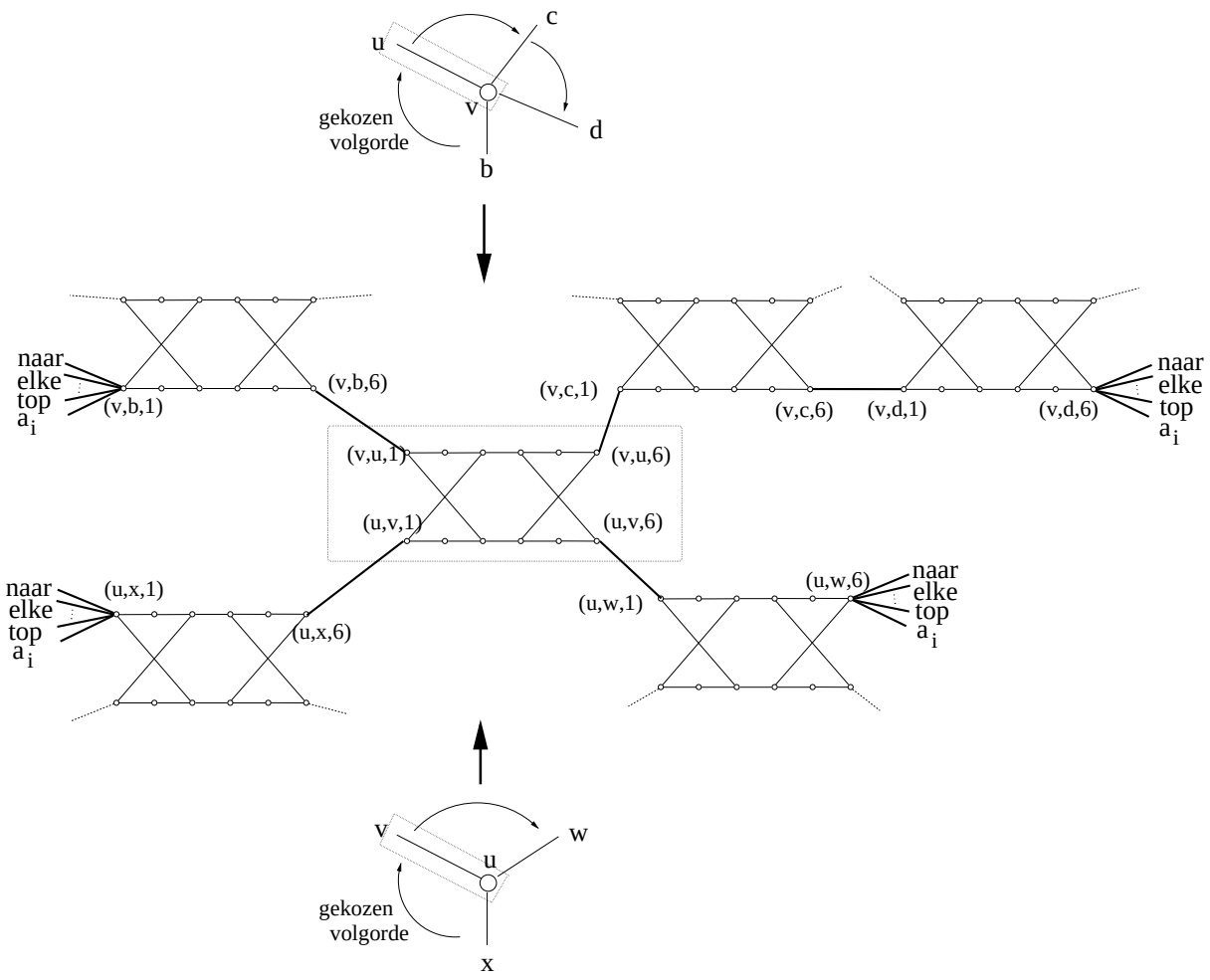
$$\begin{aligned} E' = E_1 \cup &\{\{(u, v_1(u), 1), a_i\} | u \in V, 1 \leq i \leq k\} \\ &\cup \{\{(u, v_{\deg(u)}(u), 6), a_i\} | u \in V, 1 \leq i \leq k\} \\ &\cup \{\{(u, v_i(u), 6), (u, v_{i+1}(u), 1)\} | u \in V, 1 \leq i < \deg(u)\} \end{aligned}$$

Wat wij later zullen gebruiken is dat de bogen tussen verschillende componenten van G_1 altijd tussen toppen met dezelfde eerste index lopen.

Nu is de hele graaf G' beschreven. Het is *duidelijk* dat die in polynomiale tijd opgebouwd kan worden.

Nu moeten wij nog tonen dat G' een Hamiltoniaanse cykel bevat als en slechts als G een toppenoverdekking met k of minder toppen heeft.

Eerst zullen wij aantonen dat een Hamiltoniaanse cykel in G' een toppenoverdekking met k toppen impliceert. Stel dat C een Hamiltoni-



Figuur 41: Voor elke top z wordt een volgorde van de buren gekozen en de $\deg(z)$ deelgrafen die tot een met z incidente boog behoren worden langs het z-gedeelte aan elkaar geschakeld. De eindpunten worden met alle toppen a_i, $1 \leq i \leq k$ verbonden. De figuur toont dat voor de toppen u en v.

aanse cykel is en $a_i, v_1, v_2, \ldots, v_l, a_j$ een deelpad in C met $\{v_1, v_2, \ldots, v_l\} \cap \{a_1, \ldots, a_k\} = \emptyset$.

De manier waarop een dergelijk pad de componenten van G_1 kan doorlopen (zie Figuur 40) en de manier waarop deze componenten aan elkaar zijn geschakeld, zorgen ervoor dat de deelpaden $v_1, v_2, \ldots, v_l$ altijd precies alle componenten doorlopen die bij bogen behoren die met één ze-

kere top incident zijn. Ze zijn altijd van de vorm $(u, v_1(u), 1), \ldots, (u, v_{\deg(u)}(u), 6)$ of $(u, v_{\deg(u)}(u), 6), \ldots, (u, v_1(u), 1)$. In het midden van dit deelpad kunnen ook toppen van de vorm (v_i, u, j) zijn – waar u dus in de tweede component staat, maar dat doet er niet toe.

Omdat de $a_1, \ldots, a_k$ de cykel in k deelpaden splitst, zijn er dus k verschillende $u_1, \ldots, u_k$ die de eerste index van deze deelpaden vormen. En die vormen een toppenoverdekking van G: voor elke boog $\{x, y\}$ zit de component die de toppen (x, y, i) en (y, x, i) bevat in ten minste één van de deelpaden – maar dan werd ten minste één van de twee toppen x, y ook gekozen – de (arbitrair gekozen) boog is dus overdekt.

Stel nu dat G een toppenoverdekking T met k of minder toppen heeft. Wij mogen stellen dat $|T| = k$ – als er minder toppen inzitten, kunnen wij gewoon overbodige toppen toevoegen. Dus $T = \{u_1, u_2, \ldots, u_k\}$. Omdat het een toppenoverdekking is, geldt dus voor elke boog $e = \{x, y\}$ dat $e \cap T \in \{\{x\}, \{x, y\}, \{y\}\}$. Nu kiezen wij voor elke component van G_1 hoe die van de Hamiltoniaanse cykel doorlopen moet worden. Stel dat de component bij de boog $\{u, v\}$ behoort.

als $\{u, v\} \cap T = \{u\}$**:** Kies de bogen zoals in Figuur 40 (a).

als $\{u, v\} \cap T = \{u, v\}$**:** Kies de bogen zoals in Figuur 40 (b).

als $\{u, v\} \cap T = \{v\}$**:** Kies de bogen zoals in Figuur 40 (c).

Omdat zo als (en slechts als) $u \in T$ de toppen $(u, v, 1)$ en $(u, v, 6)$ (voor alle buren v) met maar 1 van de gekozen bogen incident zijn (anders met 2 bogen) kunnen wij nu nog de bogen ertussen toevoegen:

Wij kiezen voor elke $u \in T$ de bogen $\{(u, v_i(u), 6), (u, v_{i+1}(u), 1)\}$ met $1 \leq i < \deg(u)$ voor onze Hamiltoniaanse cykel.

Nu hebben wij k paden – voor elk $1 \leq j \leq k$ een pad van $(u_j, v_1(u_j), 1)$ naar $(u_j, v_{\deg(u_j)}(u_j), 6)$. In deze paden zijn alle toppen – behalve $a_1, \ldots, a_k$ al bevat.

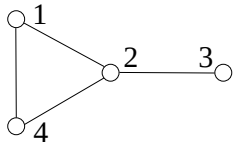
Nu verbinden wij deze paden nog over de a_i om een Hamiltoniaanse cykel te vormen:

Voor elke a_i, $1 \leq i \leq k$ kies de bogen
$\{a_i, (u_i, v_1(u_i), 1)\}$ en $\{a_i, (u_{i'}, v_{\deg(u_{i'})}(u_{i'}), 6)\}$ waarbij $i' = i - 1$ als $i > 1$ en $i' = k$ als $i = 1$.

Op deze manier hebben wij de paden tot een Hamiltoniaanse cykel samengevoegd. G' bevat dus een Hamiltoniaanse cykel als er voor G een toppenoverdekking met k of minder toppen bestaat.

■

211 Oefening *Pas de reductie in het bewijs van Stelling 210 op de graaf in Figuur 42 en $k = 2$ toe.*



Figuur 42: Een kleine graaf die een toppenoverdekking met twee toppen heeft.

212 Oefening *Wij hebben altijd een polynomiale codering van grafen geëist. Zijn de stellingen niet juist als je gewoon* een *codering hebt? Voor deze oefening veronderstel dat $P \neq NP$ en je mag hier ook stellen dat er talen zijn waarvoor er wel acceptoren bestaan die altijd stoppen maar die zo moeilijk zijn, dat ze zelfs niet in NP zitten.*

- *Waar kan er een probleem in het bewijs van Stelling 204 zijn als de codering niet noodzakelijk polynomiaal is.*
- *Is Stelling 210 misschien fout als de codering niet polynomiaal is omdat de gemodificeerde taal HC met deze codering in P zou kunnen zitten?*
- *Is Stelling 210 misschien fout als de codering niet polynomiaal is omdat de gemodificeerde taal HC met deze codering zo moeilijk zou kunnen zijn dat ze niet meer in NP zit?*

213 Oefening

- *Definieer precies wat het handelsreizigersprobleem (travelling salesman problem) TS voor gewogen complete grafen met gehele getallen als gewichten is. De vraag is: gegeven een getal k en een gewogen complete graaf G: is er een rondreis (Hamiltoniaanse cykel) in G met gewicht ten hoogste k?*
- *Bewijs dat TS NP-compleet is.*

214 Definitie *Twee functies heten polynomiaal equivalent als in het geval dat er voor de ene een polynomiaal algoritme bestaat om de functie te berekenen er dan ook voor de andere zo'n algoritme bestaat (en omgekeerd).*

215 Oefening *Het probleem de kortste rondreis in een gewogen complete graaf te vinden (de functie kr() is dus de functie die een inputstring die een gewogen complete graaf voorstelt met gehele gewichten een outputstring toekent die een kortste rondreis in deze graaf voorstelt – en misschien* „nee“ *als de invoer geen gewogen graaf met gehele gewichten codeert) is natuurlijk niet NP-compleet omdat het geen beslissingsprobleem is. Maar het hangt er nauw mee samen – als je het ene in polynomiale tijd kan oplossen dan ook het andere:*

- *Bewijs: de characteristieke functie van het handelsreizigersprobleem TS is polynomiaal equivalent met de functie kr().*
- *Kan je als invoer voor kr() ook reële getallen als gewichten nemen?*

Zoals al eerder gezegd, betekent het feit dat iets NP-compleet is of polynomiaal equivalent met een NP-compleet probleem niet noodzakelijk dat het voor praktische toepassingen hopeloos is een oplossing te vinden. Vaak bestaan er heuristieken die in de meeste gevallen heel snel kunnen beslissen en de moeilijke gevallen zijn heel zeldzaam.
Maar ook als je naar het slechtste geval kijkt, kan je natuurlijk *iets* doen. Als je bv. in de realiteit een probleem zoals het vinden van een kortste rondreis hebt, dan is het beter een benaderende oplossing te vinden waarmee je kan werken en die niet **te** slecht is dan heel lang te wachten op een optimale oplossing. De vraag is dus hoe goed je problemen kan benaderen die polynomiaal equivalent zijn met NP-complete problemen.
Sommigen van jullie hebben al benaderende algoritmen voor problemen die polynomiaal equivalent zijn met NP-complete problemen gezien – bv. inpakheuristieken in *Datastructuren en Algoritmen 2* of misschien een heuristiek voor TP als gesteld wordt dat de driehoeksvergelijking geldt in *Wiskundige Optimalisatie*. Beide heuristieken konden het optimum in die zin benaderen dat een gevonden oplossing ten hoogste een constante factor slechter was dan het optimum.
Hier zullen wij nu nog alleen maar zien dat sommige problemen zelfs niet op een manier benaderd kunnen worden die garandeert dat het resultaat ten hoogste met een constante factor van het optimum verschilt.

216 Stelling *Als $P \neq NP$ bestaat er geen $r \in \mathbb{R}$ zodat er een polynomiaal algoritme bestaat dat een rondreis in een gewogen complete graaf berekent die gegarandeerd ten hoogste r keer de lengte van een optimale rondreis heeft – ook dan niet als alle gewichten ten minste 1 zijn.*

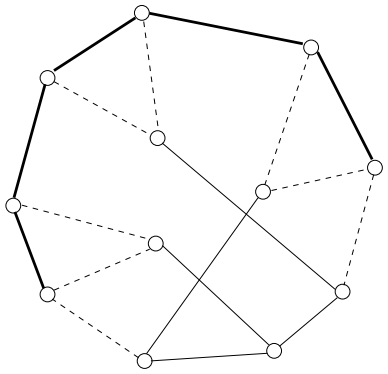
Bewijs: Wij zullen aantonen dat als een dergelijk algoritme zou bestaan, wij ook HC in polynomiale tijd zouden kunnen oplossen. Stel dat $r \in \mathbb{N}$ – anders neem gewoon $\lceil r \rceil$ – en dat een polynomiaal algoritme bestaat dat altijd een rondreis met lengte ten hoogste r keer het optimum vindt.

Stel dat een graaf $G = (V, E)$ (met ten minste 3 toppen) gegeven is waarvan wij willen beslissen of hij een Hamiltoniaanse cykel heeft. Dan bouw de gewogen complete graaf $G' = (V, E')$ met:

$$g(e) := \begin{cases} 1 & \text{als } e \in E \\ r * |V| & \text{als } e \in E' \setminus E \end{cases}$$

Het is duidelijk dat G een Hamiltoniaanse cykel heeft als en slechts als een kortste rondreis in G' lengte $|V|$ heeft. Maar er is geen rondreis met lengte meer dan $|V|$ en minder dan $r * |V|$. Het benaderend algoritme zal dus altijd een rondreis met lengte $|V|$ vinden als die bestaat – dus een Hamiltoniaanse cykel in G. Maar dan is HC $\in$ P en dus P = NP – een contradictie.

■



Figuur 43: Een Yutsis-decompositie van een 3-reguliere graaf. De twee geïnduceerd bomen zijn getekend met volle lijnen in verschillende diktes en de bogen tussen de bomen met stippellijnen.

Dat klinkt alles heel erg en je zou kunnen denken dat het hopeloos is een NP-compleet probleem in de praktijk op te lossen. Maar als het om complexiteit gaat, gaat het altijd om *het slechtste geval* en dat zegt niets daarover hoe groot de kans is dat dit slechtste geval zich voordoet als je het probleem voor een zekere invoer (graaf) moet oplossen. Afsluitend zullen wij nog een voorbeeld zien van een NP-compleet probleem dat zelfs voor heel grote grafen bijna altijd snel op te lossen is.

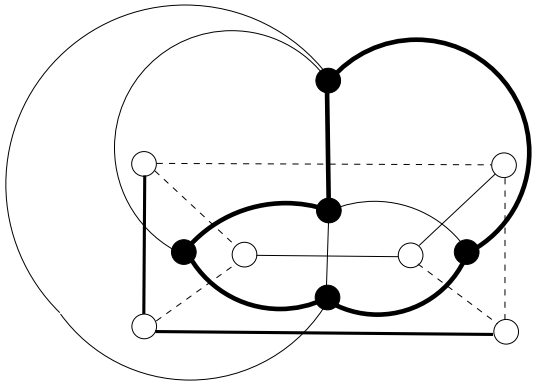
217 Definitie *Gegeven een 3-reguliere graaf $G = (V, E)$. Een Yutsis-decompositie is een paar V_1, V_2 met $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ en waarvoor $G[V_1]$ en $G[V_2]$ bomen zijn.*
Een 3-reguliere graaf met Yutsis-decompositie heet Yutsis-graaf.
Een voorbeeld van een Yutsis-graaf en een decompositie zien jullie in Figuur 43.

De taal YG is de verzameling van alle strings s die een Yutsis-graaf coderen.

Dat YG in NP zit, is onmiddellijk duidelijk. Maar je kan ook bewijzen dat een planaire 3-samenhangende 3-reguliere graaf een Yutsis-decompositie heeft als en slechts als de duale graaf (die is een triangulatie) een Hamiltoniaanse cykel heeft. Maar het Hamiltoniaanse cykel probleem is zelfs dan nog NP-compleet als het op planaire triangulaties beperkt wordt. En dit beperkte Hamiltoniaanse cykel probleem kan je dus gemakkelijk op YG reduceren, wat bewijst dat YG NP-compleet is.

218 Oefening *Bewijs: een planaire 3-samenhangende, 3-reguliere graaf heeft een Yutsis-decompositie als en slechts als de duale graaf een Hamiltoniaanse cykel heeft.*
Je mag hier gebruiken dat de duale graaf van een 3-samenhangende vlakke graaf geen dubbele bogen heeft, de duale graaf dus bestaat.

Als wij het over de *kans dat een berekening lang duurt* hebben dan moeten wij natuurlijk weten welke waarschijnlijkheidsverdeling we hebben. Wij veronderstellen hier dat je een gelijke verdeling over alle mogelijke invoeren hebt. Maar zelfs dan hebben *kansen* er nog altijd iets mee te maken welke mogelijke invoeren je beschouwt. Het is zeker duidelijk dat je als je met een toevallige, lange string als invoer te maken hebt, zeker heel snel kan beslissen of die invoer in YG zit: het is bijna zeker dat het geen graaf is. En zelfs als je de codering van een toevallige, grote graaf moet testen, is dat zeker gemakkelijk: de kans is extreem groot dat de graaf niet 3-regulier is. Maar dat is niet echt interessant. Als je echt met dit probleem te maken hebt, dan is dat zeker voor een 3-reguliere graaf. De vraag is dus: hoe groot is de kans het snel te kunnen oplossen als je de codering van een toevallige, grote 3-reguliere graaf hebt.
In een artikel werden de resultaten van een local search metaheuristiek gepubliceerd die tijd $O(|V|^4)$ voor één toepassing vroeg. Voor grafen met 10 toppen kon die met twee pogingen voor 97% van de grafen die een Yutsis-decompositie hadden een Yutsis-decompositie vinden. Voor 200 toppen waren dat nog maar 50% waarbij niet duidelijk was of de andere random grafen een decompositie hadden of niet. De gemiddelde tijd was in het geval van



Figuur 44: Een Yutsis-decompositie van een planaire 3-reguliere graaf (witte toppen) en de duale graaf (zwarte toppen) met een Hamiltoniaanse cykel. De twee geïnduceerd bomen zijn getekend met volle lijnen in verschillende diktes en de bogen tussen de bomen met stippellijnen. In de duale graaf vormen de dikkere lijnen de Hamiltoniaanse cykel.

200 toppen ongeveer 100 seconden voor elke graaf waarvoor na ten hoogste 2 pogingen een decompositie werd gevonden. De tijd voor de grafen waar er geen werd gevonden is niet gegeven. Om een relevante gemiddelde waarde te krijgen, werden 2000 grafen met 200 toppen getest. Nog grotere grafen werden niet getest.
Je zou kunnen denken dat dat er niet zo slecht uitziet. Je weet niet of de andere 50% van de grafen misschien geen decompositie hadden en bovendien is het een NP-compleet probleem...
Maar nu zullen wij een veel eenvoudigere gretige heuristiek voorstellen die zeer veel beter presteert en toont dat ook NP-complete problemen niet noodzakelijk hopeloos zijn.
Wij gebruiken de volgende eenvoudige observaties:

219 Oefening *Bewijs:*

- *In een Yutsis-decompositie van een 3-reguliere graaf met n toppen heeft elke boom precies $n/2$ toppen.*
- *Als T een geïnduceerde boom met $n/2$ toppen in een 3-reguliere graaf met n toppen is en de door het complement geïnduceerde graaf is samenhangend, dan is de graaf een Yutsis-graaf.*

Wij werken nu met één boom B waaraan wij elke keer een top toevoegen en één lijst L van toppen die *in principe* aan B toegevoegd kunnen worden.

220 Algoritme *Gegeven een 3-reguliere graaf $G = (V, E)$.*

- *Neem B als een toevallig gekozen top uit V en L als de lijst van zijn buren.*
- *Herhaal de volgende stappen totdat L leeg is:*

  (X) Kies een top v uit L die zo veel mogelijk buren in $G \setminus B$ heeft die nog nooit in L zaten.

  (a) *Verwijder v uit L.*

  (b) *Als v twee buren in B heeft of een cutvertex in $G \setminus B$ is, doe niets.*

  (c) *Anders voeg v toe aan B en voeg de buren van v in $G \setminus B$ die nog nooit in L zaten toe aan L.*

- *Als B precies $|V|/2$ toppen bevat, geef B en $G \setminus B$ terug als Yutsis-decompositie en anders geef* geen gevonden *terug.*

Als $|V|/2$ toppen in B bereikt zijn, kan je natuurlijk ook stoppen. Je weet al op voorhand dat de toppen die dan nog in L zitten allemaal 2 buren in B hebben of cutvertices van $G \setminus B$ zijn.
Stap (b) zorgt ervoor dat B altijd een boom blijft (geen twee buren in B) en ook dat $G \setminus B$ samenhangend blijft. Als B op het einde $|V|/2$ toppen bevat, hebben wij dus inderdaad een Yutsis-decompositie gevonden.
Het algoritme heeft geen decompositie gevonden als de lijst L te vroeg – dus voordat $|V|/2$ toppen in B zitten – leeg is. Lokaal lijkt het dus een goede beslissing, in stap (X) een top te kiezen die ervoor zorgt, dat L zo veel mogelijk groeit. Deze lokaal goede beslissing is dus typisch voor een gretig algoritme.
Hoe goed dit eenvoudig algoritme presteert, is verrassend:
Het kan geïmplementeerd worden zodat het in tijd $O(n^2)$ draait en in de praktijk meestal zelfs nog beter. Voor 350.000 toevallig gekozen 3-reguliere grafen met 100 tot 300.000 toppen was er maar één waarvoor de heuristiek geen decompositie vond. Deze graaf bleek een brug te hebben, dus was dat zeker geen Yutsis-graaf. Voor alle andere grafen waren maximaal 8 toepassingen van de heuristiek nodig om een decompositie te vinden. Gemiddeld werd die al naar 1.2 toepassingen van de heuristiek gevonden. Voor elke grootte werden 25.000 toevallig gekozen grafen getest. Het lijkt dus zo alsof bijna

alle 3-reguliere grafen Yutsis-grafen zijn en alsof dat in bijna alle gevallen ook heel snel beslist kan worden – ook al is het probleem NP-compleet.
Inderdaad zijn er ook criteria die het toelaten voor bijna alle kleine 3-reguliere grafen die geen Yutsis grafen zijn dat te beslissen, maar jammer genoeg is het niet mogelijk *toevallige, grote niet-Yutsis grafen* te genereren om de routines daarop te testen.
Maar dit voorbeeld is hopelijk voldoende om te illustreren dat het niet hopeloos is een probleem op te lossen **alleen** omdat bewezen werd dat het NP-compleet is.

7 Alles fout?

Ik heb heel vaak beklemtoond hoe voorzichtig je moet zijn met jouw argumentatie – misschien zo vaak dat jullie dat ondertussen al beu zijn... Dus zal ik hier ten slotte nog bewijzen geven waarvan sommigen (maar niet allemaal) fout zijn, maar waar dat niet onmiddellijk duidelijk is. Ook de "stellingen" die hier "bewezen" worden, zijn deels juist en deels fout en ik zal niet verklappen welke wel juist zijn. Maar de bewijzen hier zijn niet artificieel – het gaat allemaal om bewijzen die in de overtuiging gegeven werden dat ze juist zijn en die deels ook in tijdschriften of boeken gepubliceerd zijn!

221 Oefening *Welke van de bewijzen in dit hoofdstuk zijn juist en welke zijn fout?*

222 Definitie *Een toppenkleuring van een graaf $G = (V, E)$ met kleuren $1, \ldots, k$ is een afbeelding $c : V \to \{1, \ldots, k\}$ zodat voor alle $\{x, y\} \in E$ geldt dat $c(x) \neq c(y)$. Het chromatische getal $\chi(G)$ is het kleinste getal k' zodat er een toppenkleuring met kleuren $1, \ldots, k'$ bestaat.*

223 Stelling *Elke vlakke graaf $G = (V, E)$ heeft $\chi(G) \leq 4$.*

Bewijs: Wij bewijzen de stelling alleen voor triangulaties. Elke vlakke graaf G is deelgraaf van een triangulatie T met dezelfde toppen (dat hebben wij al gezien) en als T met 4 kleuren gekleurd kan worden dan is deze kleuring ook geldig voor G omdat die alleen maar minder bogen bevat.

Wij gebruiken inductie in het aantal toppen $|V|$. Als $|V| \leq 4$ is het duidelijk omdat zelfs elke top een andere kleur kan krijgen. Veronderstel dus dat de stelling geldt tot en met $|V| - 1$ en dat G een triangulatie is met $|V|$ toppen. Wij gebruiken de operaties van Eberhard (Stelling 164). Stel dat G' de graaf is waaruit G verkregen kan worden. G' kan dus volgens inductie met 4 kleuren gekleurd worden.

Als de laatste operatie het toevoegen van een top met graad 3 was, hebben de buren van de laatste top dus maar 3 verschillende kleuren. Wij kunnen dus de vierde kleur voor de nieuwe top gebruiken en zijn klaar.

Als de laatste operatie het toevoegen van een top met graad 4 was, zijn we om dezelfde reden klaar als er maar 3 kleuren voor de buren gebruikt worden. Stel dus dat alle buren verschillende kleuren hebben.

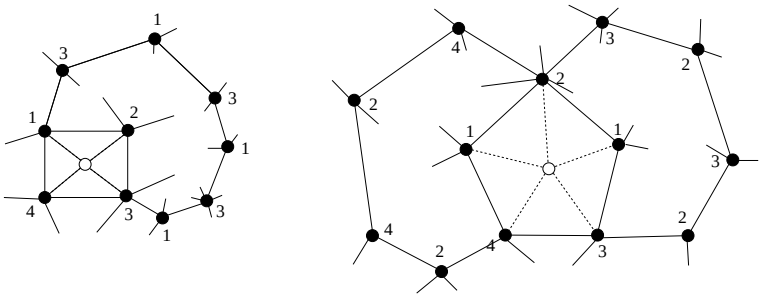
Wij schrijven G'' voor de graaf $G' - \{e\}$ waarbij e de diagonaal is die bij de operatie verwijderd wordt – die dus in G niet aanwezig is. Voor $i \neq j$ schrijven wij $H_{i,j}^{G''}$ voor de door de toppen met kleur i en j geïnduceerde subgraaf van G''. Binnen een component van een $H_{i,j}^{G''}$ kan je de kleuren i en j gewoon wisselen en je hebt een nieuwe geldige kleuring: bogen binnen de component hebben nog altijd de verschillende kleuren i en j op de twee einden en bogen die naar buiten gaan hebben op het andere einde toch al een kleur die noch i noch j is – anders zouden ze tot dezelfde component behoren.

Stel nu dat $1, 2, 3, 4$ de volgorde van kleuren rond het vierhoek in G'' is (anders hernoem de kleuren gewoon). Als de top v_1 met kleur 1 en de top v_3 met kleur 3 **niet** in dezelfde component van $H_{1,3}^{G''}$ zitten, kunnen wij de kleuren in de component van $H_{1,3}^{G''}$ waarin v_1 zit wisselen. Dan hebben we twee keer een 3 in het vierhoek en kunnen kleur 1 voor de nieuwe top gebruiken.

Als die wel in dezelfde component zitten, is er een pad van v_1 naar v_3 waarop alle toppen kleur 1 of kleur 3 hebben en dat samen met een pad door het vierhoek een Jordaanse kromme vormt. Er is dus (Stelling 154) zeker geen pad van de met 2 gekleurde top v_2 naar de met 4 gekleurde top. Wij kunnen dus de kleuren 2 en 4 in de component van $H_{2,4}^{G''}$ wisselen die v_2 bevat en kleur 2 gebruiken voor de nieuwe top.

Het laatste geval is dus dat de laatste top graad 5 heeft. Wij schrijven G'' voor de graaf $G' - \{e, e'\}$ waarbij e, e' de diagonalen zijn die bij de operatie verwijderd worden – die dus in G niet aanwezig zijn. G'' is dus met 4 kleuren kleurbaar en opnieuw kunnen wij de nieuwe top gemakkelijk kleuren als er maar 3 kleuren in de rand van het vijfhoek gebruikt worden. Veronderstel dus dat alle 4 kleuren gebruikt worden. De volgorde van toppen in de rand is v_1, v_2, v_1', v_3, v_4 en de kleuren zijn – op hernoemen na – $c(v_1) = 1, c(v_2) = 2, c(v_1') = 1, c(v_3) = 3, c(v_4) = 4$: omdat alle 4 kleuren aanwezig moeten zijn, moet dus één kleur twee keer voorkomen en dat is in een 5-hoek op afstand 2.

Als nu de component van $H_{2,3}^{G''}$ die v_2 bevat niet v_3 bevat kunnen wij de



Figuur 45: Paden met maar twee kleuren van de toppen.

kleuren in deze component wisselen en zijn klaar, omdat de kleur 2 dan in de rand ontbreekt en voor de nieuwe top gebruikt kan worden. Als dat wel zo is maar de component van $H_{2,4}^{G''}$ die v_2 bevat niet v_4 bevat, kunnen wij opnieuw wisselen en zijn klaar.

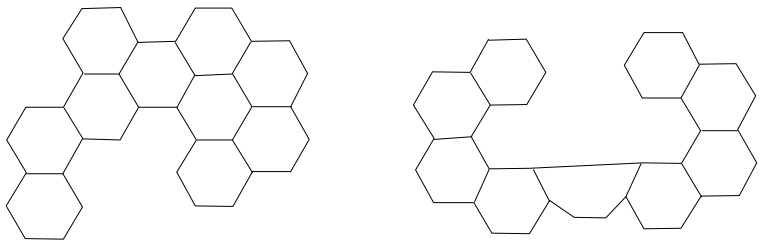
Als geen van deze mogelijkheden bestaat, is er een pad van v_2 naar v_3 dat alleen toppen met kleur 2 of 3 bevat en een pad van van v_2 naar v_4 dat alleen toppen met kleur 2 of 4 bevat. Maar dan kunnen wij opnieuw Stelling 154 toepassen: v_3 en v_1' zitten niet in dezelfde component C van $H_{1,3}^{G''}$ als v_1 en de toppen v_4 en v_1 zitten niet in dezelfde component C' van $H_{1,4}^{G''}$ als v_1'. Wij kunnen dus in C en C' de kleuren wisselen. Het resultaat is een kleuring die kleur 1 niet voor de rand van het vijfhoek gebruikt, zodat wij de nieuwe top met 1 kunnen kleuren.

■

224 Definitie *Een heliceen is een 2-samenhangende vlakke graaf $G = (V, E)$ met één speciaal vlak – het* buitenvlak. *Alle vlakken – behalve het buitenvlak – zijn zeshoeken. De grootte van het buitenvlak is niet vastgelegd. Alle toppen hebben graad 2 of 3 maar de toppen met graad 2 moeten allemaal in de rand van het buitenvlak liggen. De zeshoeken die niet het buitenvlak zijn, noemen wij interne zeshoeken.*

Deze helicenen zijn interessant in de scheikunde. Het is een klasse van moleculen waarin ook de al geziene benzenoïden bevat zijn. In feite kan je alleen een zeshoek als buitenvlak hebben, als de graaf de cykel van lengte 6 is, maar dat zal hier niet bewezen worden en dus ook niet gebruikt.

225 Lemma *In een heliceen met ten minste 2 interne zeshoeken bestaat een intern zeshoek die ten minste één boog in de rand van het buitenvlak heeft*



Figuur 46: Eén heliceen dat je met regelmatige zeshoeken in het vlak kan tekenen en één waar dat niet mogelijk is.

zodat als de snede met het buitenvlak uit de graaf verwijderd wordt het resultaat een kleiner heliceen is.

Dit *verwijderen van de snede met het buitenvlak* kan je je ook voorstellen als het verwijderen van een zeshoek zonder dat het heliceen uit elkaar valt.

Bewijs: Gegeven een heliceen G. Teken nu in het centrum van elk intern zeshoek een nieuwe top en verbind die met elke top in de rand van dat zeshoek. Noem deze graaf G'. Voor 2 interne zeshoeken S_1, S_2 die allebei niet het buitenvlak zijn, definieren wij de afstand $d(S_1, S_2)$ als de lengte van een kortste pad tussen de twee centrale toppen van deze zeshoeken in G', waarbij het pad alleen bogen bevat die niet in G zitten. Als je het concept van een *binnenduaal* kent, kan dat iets gemakkelijker, maar deze definitie komt op hetzelfde neer.

Kies nu een arbitrair intern zeshoek S en een intern zeshoek S' dat een boog in de rand heeft en waarvoor $d(S, S')$ maximaal is (onder alle andere zeshoeken met deze eigenschap). Als je de snede van S' met het buitenvlak verwijdert en het resultaat blijft samenhangend, heb je het gezochte zeshoek. Veronderstel dus dat dat niet zo is.

Dan heeft het resultaat ten minste twee samenhangscomponenten, waarvan ten minste één – noem die C – het zeshoek S niet bevat. Omdat G 2-samenhangend is, verlaten in G twee bogen van S' de component C. *In principe* kunnen het ook meer dan twee zijn, maar het is gemakkelijk om te zien dat dat met zeshoeken niet kan. Als je nu een pad in S' en in het buitenvlak tussen de middenpunten van deze twee bogen tekent krijg je een Jordaanse kromme met C aan de ene kant en de component die S bevat aan de andere. Dat betekent dat elk pad dat we voor de berekening van de lengten beschouwen van S naar een intern zeshoek

in C (dus ook naar interne zeshoeken met bogen in de rand van het buitenvlak) door S' moet gaan – dus langer is dan het pad naar S'. Dat is een tegenstrijdigheid omdat S' een maximale afstand had en dus moet het resultaat na het verwijderen van S' samenhangend zijn.

■

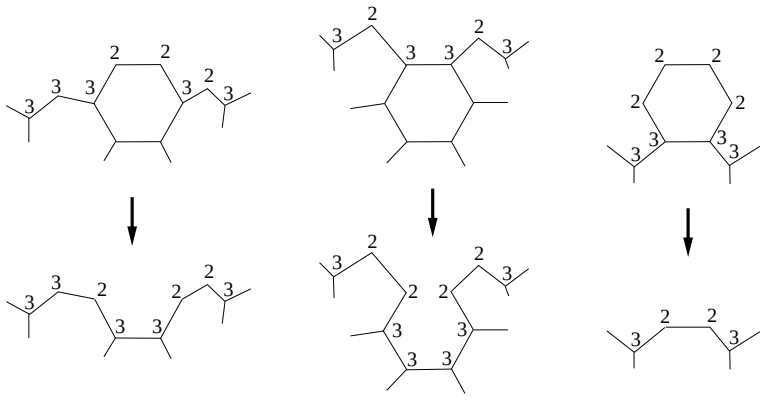
226 Stelling *Als twee helicenen dezelfde cyklische volgorde van graden in het buitenvlak (wij noemen dat de randcodering) hebben, dan zijn ze ook isomorf.*

Bewijs: Wij bewijzen dat met inductie in het aantal interne zeshoeken van het kleinste heliceen voor een gegeven randcodering. Wij mogen nog niet veronderstellen dat dit aantal voor alle helicenen met dezelfde randcodering hetzelfde is (en zullen dat ook niet doen) – maar dat is wel zo. Als er maar 1 of 2 interne zeshoeken zijn, kan je gemakkelijk nagaan dat voor de cyklische volgorden van graden $(2,2,2,2,2,2)$ of $(2,2,2,2,3,2,2,2,2,3)$ op isomorfie na maar 1 heliceen bestaat. Stel dus dat een randcodering $r = r_1, \ldots, r_k$ gegeven is waarvoor een heliceen G met deze randcodering en n zeshoeken bestaat en dat de stelling voor alle randcoderingen waarvoor er een heliceen met deze randcodering en ten hoogste $n-1$ zeshoeken bestaat al bewezen is.

Vogens Lemma 225 bestaat er een zeshoek in de rand die verwijderd kan worden zonder dat het heliceen uit elkaar valt. Maar de randcodering r' na het verwijderen van het zeshoek kan uit de randcodering r berekend worden en is uniek bepaald en dus dezelfde – om het even of je het zeshoek in G of G' verwijdert. Als de snede van het verwijderde zeshoek met het buitenvlak bv. de reeks $3,2,2,3$ in de randcodering bevatte, dan heeft de randcodering van het heliceen na het verwijderen op deze plaats $2,3,3,2$. Dit en andere gevallen zijn ook te zien in afbeelding 47.

Maar voor de randcodering r' bestaat er een heliceen met $n-1$ zeshoeken en dat is – volgens inductie – uniek bepaald. Voor de twee helicenen met deze randcodering die je uit G en G' door het verwijderen hebt gekregen, bestaat er dus een isomorfisme dat het ene heliceen op het andere afbeeldt. Maar deze isomorfismen kunnen nu uitgebreid worden tot isomorfismen van de helicenen voordat het zeshoek verwijderd werd – er worden alleen paden van dezelfde lengte tussen paren van toppen die door het isomorfisme op elkaar worden afgebeeld, toegevoegd. Ook de helicenen met randcodering r moeten dus isomorf zijn.

■



Figuur 47: Voorbeelden voor het effect van het verwijderen van een zeshoek op de randcodering. Omdat een zeshoek verwijderd wordt waarbij het heliceen samenhangend blijft, is de snede van het zeshoek met het buitenvlak samenhangend en hebben alle toppen die niet in deze snede zitten graad 3, waardoor de randcodering van het gereduceerde heliceen uniek bepaald is.

227 Stelling *Elke 2-samenhangende graaf $G = (V, E)$ die niet compleet is en waar alle toppen graad ten minste 3 hebben (dus $\delta(G) \geq 3$) hebben een geïnduceerd pad $P = v_1, v_2, v_3$ zodat de door $V \setminus \{v_1, v_3\}$ geïnduceerde graaf $G[V \setminus \{v_1, v_3\}]$ samenhangend is.*

Bewijs: Als G 3-samenhangend is, is het resultaat duidelijk: omdat G niet compleet is, zijn er toppen die niet adjacent zijn. Kies v_1, v_3 als twee niet adjacente toppen met minimale afstand. Deze minimale afstand is dan automatisch 2 en je hebt het pad $P = v_1, v_2, v_3$, zodat $\{v_1, v_3\} \notin E$. Omdat G 3-samenhangend is, is $G[V \setminus \{v_1, v_3\}]$ natuurlijk samenhangend – in feite zelfs om het even hoe v_1, v_3 gekozen zijn.

Stel dus dat het samenhangsgetal $\kappa(G) = 2$ en dat $\{v, w\}$ een 2-cut in G is. De componenten van $G[V \setminus \{v, w\}]$ zijn $G_1, \ldots, G_k$ met $k \geq 2$. Tussen de G_i heb je geen bogen (anders zouden het geen componenten zijn) en elke van de toppen v, w heeft buren in alle componenten omdat anders de andere top alleen een cutvertex was en de graaf niet 2-samenhangend. Kies nu v_1 als buur van v in G_1, $v_2 = v$ en v_3 als buur van v in G_2. Dan is het pad $P = v_1, v_2, v_3$ geïnduceerd omdat v_1 en v_3 in verschillende componenten van $G[V \setminus \{v, w\}]$ liggen – er dus geen boog tussen v_1 en v_3 bestaat. In elke component $G_1, \ldots, G_k$ wordt ten hoogste 1 top verwijderd. Omdat G 2-samenhangend is, blijven de componenten dus samenhangend. De top w heeft een buur in elke component, waaruit met transitiviteit volgt dat alle toppen in de com-

ponenten tot dezelfde component behoren als w. De enige top behalve w die niet in één van de componenten zit is v. Omdat $\delta(G) \geq 3$ heeft v nog ten minste één buur die niet verwijderd werd in $G[V \setminus \{v_1, v_3\}]$ en behoort dus tot dezelfde component als deze buur. In feite behoren dus alle toppen uit $V \setminus \{v, w\}$ tot één component en is $G[V \setminus \{v_1, v_3\}]$ dus samenhangend.

■

Index