

1 Graaf Isomorphisme

1.1 Inleiding

In dit hoofdstuk kijken we naar het begrip *isomorphisme* tussen grafen.

Definitie 1.1. Twee grafen $G = (V, E)$ en $H = (W, F)$ zijn isomorph, dan en slechts dan als er een bijectieve functie $f : V \rightarrow W$ bestaat, zodat voor alle $v, w \in V$: $(v, w) \in E \Leftrightarrow (f(v), f(w)) \in F$.

Twee grafen zijn dus isomorph, als ze, op hernaamgeving van de knopen na, hetzelfde zijn.

Een belangrijk probleem is het volgende:

Graaf Isomorphisme

Gegeven: Grafen $G = (V, E)$, $H = (W, F)$.

Gevraagd: Zijn G en H isomorph?

Dit is een probleem met verschillende belangrijke toepassingen, o.a., is het probleem een klassieke vraag uit de chemie, waar men wil kijken of twee moleculen hetzelfde zijn.

Het graaf isomorphisme probleem is een probleem, dat een merkwaardige status inneemt in het vakgebied van de analyse en complexiteit van algoritmen: het is namelijk niet bekend of het probleem in polynomiale tijd op te lossen is, maar het is ook niet bekend of het probleem NP-volledig is.

Het naieve algoritme: bekijk alle permutaties van de knopen van een van de grafen, en check voor elk of die een graaf isomorphisme geeft, is al voor kleine grafen snel te duur. Er zijn heuristische methoden, die in veel gevallen goede resultaten geven, en er zijn ook algoritmen die het probleem snel oplossen op speciale grafen. We zullen o.a. een algoritme voor graaf isomorphisme op bomen bekijken.

1.2 Het reconstructie-vermoeden

Een beroemd open probleem uit de wiskunde is het reconstructie vermoeden, ook wel genoemd *Ulam's vermoeden* (zo genoemd omdat het vermoeden als eerste geuit werd door P. J. Kelly, een student van Ulam.)

Dit is als volgt:

Vermoeden 1 Laat $G = (V, E)$ en $H = (W, F)$ grafen zijn, met elk tenminste 3 knopen. Stel er is een bijectie $f : V \rightarrow W$, zodat voor elke $v \in V$, $G[V - \{v\}]$ isomorph is met $H[W - \{f(v)\}]$. Dan zijn G en H isomorph.

Het reconstructie-vermoeden blijkt waar te zijn voor veel soorten grafen, maar is nog niet bewezen. Echter, een bewijs van dit vermoeden lijkt niet te helpen kunnen voor het vinden van een polynomiaal algoritme voor graaf isomorphisme.

1.3 Problemen, even moeilijk als graaf isomorfisme

Van verschillende problemen kan worden aangetoond dat ze 'even moeilijk' zijn als graaf isomorfisme, d.w.z., dat er een polynomiaal algoritme voor het probleem bestaat, dan en slechts dan als er een polynomiaal algoritme voor graaf isomorfisme bestaat. Een voorbeeld van zo'n probleem is het vinden van een isomorfisme functie f .

Voor beslissingsproblemen kennen we het begrip *isomorfisme-volledig*.

Definitie 1.2. *Een probleem Q is isomorfisme volledig, als er een polynomiale transformatie van Q naar GRAAF ISOMORPHISME bestaat en er een polynomiale transformatie van GRAAF ISOMORPHISME naar Q bestaat.*

Voorbeelden van isomorfisme-volledige problemen zijn:

- Isomorfisme voor semi-groepen
- Isomorfisme voor eindige automaten
- Isomorfisme voor eindige algebra's
- Isomorfisme voor samenhangende grafen
- Isomorfisme voor gerichte grafen
- Isomorfisme voor bipartite grafen
- Isomorfisme voor reguliere grafen
- Isomorfisme voor perfecte grafen
- Isomorfisme voor chordale grafen
- Herkennen van grafen die isomorph met hun complement zijn

Soms wordt het begrip isomorfisme-volledig in een ruimere zin gebruikt: in plaats van de twee transformaties is het voldoende om algoritmen te geven die het probleem oplossen, gebruik makend van een subroutine (die hooguit polynomiaal vaak wordt aangeroepen) voor het andere probleem.

Met deze ruimere betekenis wordt het begrip isomorfisme-volledig wordt ook gebruikt voor problemen, die geen beslissingsproblemen zijn.

Van graaf isomorfisme naar automorfisme partitie Stel we hebben een algoritme voor graaf isomorfisme. We kunnen testen of $v \sim w$, door te kijken of er een isomorfisme is tussen de grafen, verkregen uit $G = (V, E)$ door aan v een kliek met $|V| + 1$ knopen te hangen en aan w een kliek met $|V| + 1$ knopen te hangen. (Als zo'n isomorfisme bestaat, dan moeten de kliks op elkaar worden afgebeeld vanwege het formaat, en moet dus v op w worden afgebeeld.)

Dit kan voor elk paar v, w gedaan worden. Hiermee wordt de gevraagde partitie verkregen.

Van automorfisme partitie naar graaf isomorfisme Stel we hebben een algoritme dat automorfisme partities vindt. En, stel dat we willen bepalen of samenhangende grafen $G = (V, E)$ en $H = (W, F)$ isomorph zijn. Bepaal nu de automorfisme partitie van $G \cup H = (V \cup W, E \cup F)$, de disjuncte vereniging van G en H . Er is een isomorfisme van G naar H , dan en slechts dan als er een cel is, waarin zowel minstens één knoop uit V zit, als minstens één knoop uit W .

Dus, maak de automorfisme partitie van $G \cup H$, en test het bovenstaande criterium.

Definitie 1.4. *Een graaf $G = (V, E)$ heet transitief, als alle knopen van de automorfisme partitie in dezelfde cel zitten.*

Met andere woorden, voor elk paar knopen v, w is er een automorfisme dat v op w afbeeldt. Voorbeelden van transitieve grafen zijn de volledige grafen, en de cycles.

1.5 Iteratieve vertex partitie

Een veel gebruikte methode voor het oplossen van graaf isomorfisme is de volgende. Het idee is dat we zo goed mogelijk de automorfisme partitie willen proberen te benaderen.

Het algoritme kent een aantal rondes. We hebben steeds een partitie van de knopen $V = V_1 \cup V_2 \cup \dots \cup V_r$, zodat als invariant geldt: de automorfisme partitie is een onderverdeling van deze partitie, met andere woorden:

Voor alle $v, w \in V$: als er een automorfisme f is met $f(v) = w$, dan zitten v en w in dezelfde verzameling V_i .

(Wanneer f een automorfisme is, dan geldt het omgekeerde van de bovenstaande eigenschap ook, nu hoeft f dus geen automorfisme te zijn. Een voorbeeld van zo'n partitie is altijd die in maar één verzameling; echter, hoe verfijnder de partitie, hoe beter.)

1.4 Graaf Automorfisme

Een automorfisme van een graaf $G = (V, E)$ is een bijectieve functie $f : V \rightarrow V$, zodat voor alle $v, w \in V$: $(v, w) \in E \Leftrightarrow (f(v), f(w)) \in E$.

Met andere woorden, een automorfisme is een isomorfisme van G naar zichzelf. Iedere graaf heeft minstens 1 automorfisme: de identiteit. Sommige grafen hebben er meerdere of zelfs heel veel (een volledige graaf met n knopen heeft $n!$ automorphismen), sommige grafen hebben geen ander automorfisme als de identiteit (neem bijvoorbeeld een pad met 6 knopen, en maak een extra knoop grenzend aan de 3e knoop op het pad.)

Merk op:

Lemma 1.1. *Als f een automorfisme van G is, en g is een automorfisme van G , dan is $f \circ g$ ook een automorfisme van G .*

De verzameling automorphismen op G vormt dus een groep: de identiteit is het neutrale element, en compositie is de operatie van deze groep. De automorphismengroep van een graaf G is de verzameling van alle mogelijke automorphismen van G ; deze groep wordt aangegeven als $aut(G)$. Hoewel de groep $|V|!$ elementen kan hebben is er altijd een stel generatoren, die alle elementen kunnen genereren, van formaat hooguit $|V|$. (Dit kan door groepentheorie bewezen worden, of met een direct algoritme geconstrueerd worden.)

Lemma 1.2. *Het vinden van een stel generatoren van $aut(G)$ is isomorfismevolledig.*

Een gevolg hiervan is dat ook het vinden van het aantal isomorphismen tussen twee gegeven grafen isomorfismevolledig is: immers, als G en H niet isomorph zijn, dan is het aantal isomorphismen 0, en als ze wel isomorph zijn, is het aantal isomorphismen gelijk aan het aantal automorphismen van G .

In plaats van naar $aut(G)$ te kijken, is het ook nuttig om te kijken naar verzamelingen knopen die onder een automorfisme op elkaar worden afgebeeld.

Stel dat $v \sim w$, als er een automorfisme is, dat v op w afbeeldt. Merk op dat \sim een equivalentie-relatie is. Een opsplitsing van de knopen in de equivalentieklassen van deze relatie noemen we de *automorfisme partitie van G* .

Definitie 1.3. *De automorfisme partitie van een graaf $G = (V, E)$, is de opsplitsing van G in disjuncte cellen, zodat knopen v en w in dezelfde cel komen, dan en slechts dan als er een automorfisme is dat v op w afbeeldt.*

Het vinden van een automorfisme partitie van een gegeven graaf is weer isomorfismevolledig. Omdat we in sommige gevallen liever het vinden van een automorfisme partitie oplossen als graaf isomorfisme, geven we hier de transformaties:

Wat we doen is twee knopen in verschillende verzamelingen stoppen, wanneer we weten dat er geen automorfisme tussen hen is. Hierbij splitsen we alleen verzamelingen, en voegen nooit verzamelingen samen (immers, knopen in verschillende verzamelingen hebben geen automorfisme tussen hen.)

Een veel gebruikte methode gaat met behulp van de graden van knopen, en het aantal burens dat ze in andere klassen hebben. De volgende subroutine wordt herhaald, tot er geen verdere splitsingen door ontstaan:

- Stel we hebben partitie V_1, \dots, V_r . We bekijken alle knopen in een verzameling V_i .
- Voor elke knoop $v \in V_i$ bekijken we de lijst $a_v = (a_1, a_2, \dots, a_r)$, waarbij a_j het aantal kanten van v is naar knopen in V_j .
- Sorteert de knopen in V_i op lexicaal volgorde van hun lijsten a_v .
- Als niet alle knopen in V_i dezelfde lijst a_v hebben, dan splitsen we V_i in nieuwe verzamelingen, waarbij elk van deze verzamelingen precies alle knopen met een bepaalde lijst a_v bevat.

De onderliggende gedachte van dit algoritme is de volgende: als $v \sim w$, dan moet $a_v = a_w$, immers, voor elk automorfisme f : als $v \in V_j$, dan $f(v) \in V_j$, en dus beeldt f alle burens van v in V_j af op burens van w in V_j . Dus, als $a_v \neq a_w$, dan kunnen v en w in verschillende klassen gezet worden.

Met enig geluk, en met meerdere soortgelijke trucs kan de partitie heel fijn gemaakt worden. Tenslotte wordt dan met bijvoorbeeld 'backtracking' de uiteindelijke automorfisme partitie bepaald.

Dit algoritme werkt heel goed op random grafen: die hebben i.h.a. zo'n onregelmatige structuur, dat het snel een heel fijne partitie geeft. Echter, soms werkt het algoritme ook heel slecht, bijvoorbeeld op transitieve grafen.

1.6 Graaf Isomorfisme op gewortelde bomen

We beschouwen nu de volgende vraag: gegeven twee bomen $T_1 = (V_1, e_1)$, $T_2 = (V_2, E_2)$ met wortel $r_1 \in V_1$, $r_2 \in V_2$, is er een graaf isomorfisme f van T_1 op T_2 , met $f(r_1) = r_2$? (Equivalent: neem de bomen gericht, met alle kanten in de richting naar de wortel, of met alle kanten in de richting naar de bladeren, en vraag dan om een graaf-isomorfisme).

We kunnen dit probleem in lineaire tijd oplossen. Dit kan zelfs, wanneer we knopen labelen met een label $\in L$, met L een eindige verzameling van constante grootte. D.w.z., we hebben functies $l_1 : V_1 \rightarrow L$, $l_2 : V_2 \rightarrow L$, en eisen dat voor alle $v \in V_1$: $l_1(v) = l_2(f(v))$. Met andere woorden: knopen moeten op knopen

met hetzelfde label worden afgebeeld. Voor het gemak noteren we het label van een knoop v met $l(v)$.

Neem een willekeurige ordening op L aan. We nemen verder aan, dat werken met elementen uit L $O(1)$ tijd per operatie kost.

De *diepte* van een knoop is z'n afstand naar de wortel. De diepte van een boom is de maximale diepte van een knoop in de boom.

Als T_1 en T_2 verschillende diepte hebben, dan kunnen we direct stoppen. Neem dus aan dat ze dezelfde diepte hebben.

In het algoritme krijgen alle knopen een i-nummer (een integer), en een li-paar (een paar, gevormd door een label, en een gesorteerde rij i-nummers) geassocieerd.

Knopen worden behandeld 'laag voor laag', beginnend met de knopen met maximale diepte, daarna de knopen met 1 kleinere afstand tot de wortel, etc. (Een *laag* is een verzameling knopen met dezelfde diepte tot de wortel. We behandelen steeds een laag, in volgorde van dalende diepte.)

Een laag wordt als volgt behandeld:

- Assigneer aan elke knoop v in de laag het li-paar $(l(v), S)$, waar S de gesorteerde rij i-nummers van de kinderen van v is.
- Sorteert alle knopen in de laag op li-paar, byv. op lexicographische volgorde.
- Het i-nummer van een knoop in de laag is nu de rang van zijn li-paar t.o.v. de andere knopen in de laag. Hierbij zorgen we ervoor dat knopen met hetzelfde li-paar hetzelfde li-nummer krijgen.

Correctheid Wanneer de beide wortels r_1 en r_2 hetzelfde li-paar, dus i-nummer krijgen, dan zijn T_1 en T_2 isomorf, anders niet. We bewijzen dit nu.

Notatie. Met T_v , ($v \in V_1$ of $v \in V_2$), noteren we de deelboom van T_1 of T_2 , geïnduceerd (gevormd) door v en alle afstammelingen van v .

Lemma. Voor alle $x, y \in V_1 \cup V_2$, x, y dezelfde diepte: T_x en T_y zijn isomorf met een isomorfisme dat x op y afbeeldt, dan en slechts dan als x en y hetzelfde i-nummer hebben.

Bewijs. Knopen hebben natuurlijk hetzelfde i-nummer, d.e.s.d. als ze hetzelfde li-paar hebben.

We bewijzen dit met inductie naar dalende diepte.

Als x en y maximale diepte hebben, zijn ze bladeren en is de stelling natuurlijk waar. Stel stelling is waar met knopen met diepte α en stel x en y hebben diepte $\alpha - 1$.

\Rightarrow : Stel f is een isomorfisme van T_x naar T_y met $f(x) = y$. Voor elk kind z_i van x geldt dat $f(z_i)$ een kind van y is, en bovendien dat $T(z_i)$ isomorf is met $T(f(z_i))$. Dus hebben z_i en $f(z_i)$ hetzelfde i-nummer (IH). Omdat dit geldt voor

alle kinderen van x , en ieder kind van y beeld is van precies een kind van x , geldt dat x en y hetzelfde li-paar, dus hetzelfde i-nummer hebben.

\Leftarrow : Omdat de gesorteerde rij i-nummers van kinderen van x hetzelfde is als de gesorteerde rij i-nummers van kinderen van y kunnen we bijectief ieder kind van x afbeelden op een kind van y met hetzelfde i-nummer. IH: kinderen van x en y hebben paarsgewijs isomorfe bomen onder zich. Omdat x en y hetzelfde label hebben, zijn T_x en T_y dus isomorf met een isomorfisme dat x of y afbeeldt.

QED

Tijd. Hoe zorgen we ervoor dat het algoritme lineaire tijd gebruikt? Dit vergt een precies gebruik van radix-sort achtige technieken.

1. Sorteren van rijen i-nummers van kinderen . We mogen aannemen dat we alle knopen in de vorige laag in volgorde van i-nummer hebben. Alle knopen op de nu te bekijken laag beginnen een gelinkte lijst. We lopen nu langs de knopen in de vorige laag in volgorde van stijgend i-nummer: bij iedere knoop plakken we zijn i-nummer achteraan de lijst van z'n ouder.

2. Sorteren van knopen in laag op li-paar . Het idee hier is om radix-sort te gebruiken. Echter, dit kan teveel tijd kosten. (Ga na.)

(Kijk in Cormen, Leiserson, Rivest de paragraaf over radix sort na.)

1. Sorteert de knopen in de laag op aantal kinderen. (Lineair met counting-sort (radix sort).)
2. Maak alle paren (j, i) , waarbij i het j -de i-nummer is in de gesorteerde lijst i-nummers van kinderen van een knoop in de laag.
3. Sorteert deze verzameling van paren (met radix sort).
4. Nu gaan we radix-sort toepassen om de verzameling te sorteren, maar we gebruiken twee modificaties van het gewone radix-sort algoritme:
 - In de eerste ronde doen alleen de knopen met maximum aantal kinderen mee en kijken we naar het laatste i-nummer van kind, in de tweede rond alleen knopen met maximum of maximum min 1 aantal kinderen en kijken we naar het 1-na laatste i-nummer van kind, etc. (Hierom hebben we in stap 1 de knopen op aantal kinderen gesorteerd.)
 - Bij een radix-sort kan het gebeuren dat sommige 'buckets' leeg zijn. We voorkomen dat we naar die buckets moeten kijken, en kunnen dat doen met de in stap 3 verkregen lijst: die vertelt precies welke buckets

wel knopen zullen bevatten (als (j, i) in de lijst, dan is de i -de bucket vol als we het j -de i -nummer van kind bekijken van de li-paren.)

- De laatste ronde van radix-sort sorteert op labels.

Een preciese analyse van een stap leert dat elke laag gedaan kan worden, lineair in het aantal knopen in die laag plus het aantal knopen in de daaronder liggende laag. Gesommeerd over alle lagen is dit $O(V_1 + V_2)$.

1.7 Bomen zonder wortel

Definieer de straal van een knoop v in een graaf G als $r(v) = \max_{w \in V} d_G(v, w)$, waarbij $d_G(v, w)$ de afstand van v naar w in G aangeeft. Het *centrum* van een graaf $G = (V, E)$ is de verzameling knopen in G met minimum radius.

Lemma. Het centrum van een boom bevat hooguit twee knopen.

Bewijs dit zelf.

Het gevolg van dit lemma is dat ook isomorfisme van ongewortelde bomen in lineaire tijd kan. Bereken in lineaire tijd het centrum van de bomen. (Opgave: ga na hoe dat kan.) We weten dan dat knopen in het centrum altijd op knopen in het centrum moeten worden afgebeeld, en hoeven maar een of twee verschillende gevallen te bekijken, die we elk afhandelen als in het algoritme hiervoor.