

Project Gretige Algoritmen

Jarre Knockaert

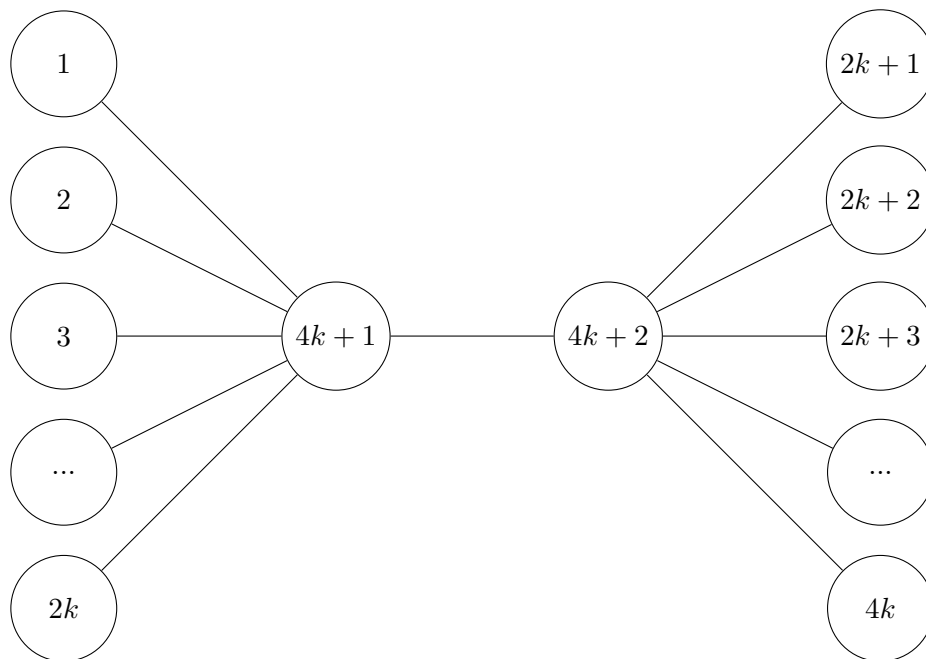
3 november 2016

1 Theoretische vragen

Opgave 1. Indien er 2 naburige toppen v en w een grote bedekking leveren, zal de top v met de hoogste graad worden toegevoegd aan de dominante verzameling en de burenen zullen worden verwijderd. Het zou echter beter zijn om zowel v en w toe te voegen aan de dominante verzameling, want w leidt ook tot een hoge toename van de bedekking van de dominante verzameling. Een voorbeeld van dergelijk geval, een graaf waarbij het algoritme zeer slecht presteert, zie je op figuur 1. Als we het algoritme uitvoeren op de graaf gebeurt het volgende:

- Neem top $v = 4k + 1$ met hoogste graad: $2k - 1$. (Dit kon evengoed top $4k + 2$ zijn, aangezien zijn graad gelijk is.)
- Voeg v toe aan de dominante lijst D .
- Verwijder v en al zijn burenen (top 1 tot en met top $2k$) uit de graaf G . Nu bevat de G nog $2k - 1$ toppen (top $2k + 1$ tot en met top $4k$).
- De resterende toppen uit de graaf hebben elk graad 0 en zijn geïsoleerde toppen. (Aangezien hun enige buur werd verwijderd uit de graaf.) Voor elke top w van $2k + 1$ tot en met top $4k$ gebeurt nu het volgende:
 - Neem de top w . Aangezien elke top graad 0 heeft maakt het niet uit welke top we uit de graaf kiezen. Hun graad is (en blijft) gelijk aan elkaar.
 - Voeg w toe aan D .
 - Verwijder w uit G . De top w is geïsoleerd en dus er zullen ook geen extra toppen uit de graaf verwijderd worden.

In totaal werden $2k$ toppen toegevoegd aan de dominante lijst: top $4k + 1$ en top $2k + 1$ tot en met top $4k$. De minimale dominante verzameling bevat echter enkel de 2 toppen $4k + 1$ en $4k + 2$. Het resultaat van het algoritme is een dominante verzameling met $\frac{2k}{2} = k$ keer meer toppen meer dan de optimale dominante verzameling.



Figuur 1: Een graaf waarbij het algoritme slecht presteert.

Opgave 2.

Opgave 3. Hier volgt een uitvoerige bespreking van de algoritmes.

Enkele belangrijke vermeldingen bij algoritme 1.

- Een buur is bezocht indien de top element is van de dominante verzameling of indien een buur van deze top element is van de dominante verzameling.
- De bedekking/coverage van een top is de bovengrens van het aantal onbezochte burens.
- De werkelijke bedekking/coverage van een top is gelijk aan het aantal onbezochte burens.
- De coverage van een verzameling van toppen is een som van de coverage van alle burens.
- De totalCoverage is de de som van de coverage van alle burens. Indien de coverage gelijk is aan $|V(G)|$, dan is de dominante verzameling klaar.
- De variabele, minimum, stelt een ondergrens voor van de werkelijke bedekking die een top moet bieden aan de graaf voor de top kan toegevoegd worden aan de dominante verzameling.

Algoritme 1 Dominante verzameling van vlakke grafen (met optimalisaties)

Input: Een planaire graaf $G(V(G), E(G))$ **Output:** Een dominante verzameling D

```
1:  $totalCoverage \leftarrow 0$ 
2: Count-sort  $V(G)$  op basis van de graad van de toppen
3: for all  $v \mid v \in V(G) \wedge deg(v) = 1$  do                                 $\triangleright$  Optimalisatie 1
4:   Neem  $w \mid vw \in E(G)$                                                $\triangleright v$  heeft één buur
5:   if  $w$  nog niet bezocht  $\wedge coverage(w) > 0$  then
6:      $D \leftarrow D \cup \{w\}$ 
7:     bezoek  $w$ 
8:      $totalCoverage \leftarrow totalCoverage + 1$ 
9:      $coverage(w) \leftarrow 0$ 
10:  end if
11:  for all  $u \mid u \in V(G) \wedge uw \in E(G)$  do
12:     $coverage(u) \leftarrow coverage(u) - 1$ 
13:    if  $u$  niet bezocht then
14:      bezoek  $u$ 
15:       $totalCoverage \leftarrow totalCoverage + 1$ 
16:    end if
17:  end for
18:  if  $totalCoverage = |V(G)|$  then
19:    Stop de foreach loop
20:  end if
21: end for
22: for all  $minimum \in \{6, 5, \dots, 0\}$  do                                 $\triangleright$  Optimalisatie 2
23:    $i \leftarrow 0$ 
24:   while  $totalCoverage < |V(G)| \wedge i < |V(G)|$  do
25:      $v \leftarrow v_i \in V(G)$ 
26:     if  $coverage(v) > 0$  then
27:       Neem  $max \mid (\forall u \in \{v\} \cup N_G(v)) \wedge (max \neq u) \Rightarrow coverage(max) > coverage(u)$ 
28:       if  $coverage(max) > minimum$  then
29:          $actualCoverage \leftarrow \{w \mid w \in N_G(max) \wedge w \text{ niet bezocht}\}$ 
30:         if  $actualCoverage > minimum$  then
31:            $totalCoverage \leftarrow totalCoverage + actualCoverage$ 
32:            $coverage(max) \leftarrow 0$ 
33:            $D \cup \{max\}$ 
34:           Bezoek  $max$  en zijn burens
35:         end if
36:       end if
37:     end if
38:      $i \leftarrow i + 1$ 
39:   end while
40: end for
```

Algoritme 1 loopt in lineaire tijd. Ik zal de verschillende delen van het algoritme bespreken om de complexiteit te verklaren. Beschouw bij deze analyse n als het aantal toppen. De lijnen die niet besproken worden zijn triviaal en hebben constante tijd $\Theta(1)$.

- Lijn 2: Hier wordt counting sort toegepast op de toppenverzameling om deze te ordenen op basis van hun graad. De complexiteit van counting sort is $O(n+k)$. Hier is k (de graad) begrensd door $n-1$. De totale complexiteit is hier $O(2n - 1)$.
- Lijn 3 → 21: Er wordt geïtereerd over de toppen met graad 1. Dit zijn er hoogstens $n-1$. Stel v de top met graad 1 en w de enige buur van deze top. Er wordt geïtereerd over alle aanliggende bogen van w . Elke boog wordt hoogstens 2 keer bekeken. De reden hiervan is dat de top w maar één keer wordt genomen als naburige top van top v met graad 1. Er wordt eenmalig geïtereerd over al zijn aanliggende bogen. De aanliggende bogen kunnen echter nog maar één keer bekeken worden bij het itereren over de bogen van een buur van een andere top met graad 1. Elke boog kan dus hoogstens 2 keer bekeken worden bij het itereren over de bogen van toppen die een buur zijn van een top met graad 1. De boog van een top met graad 1 wordt ook maar twee keer bekeken, eens om zijn buur te bepalen, nogmaals tijdens het itereren over de bogen van zijn buur. In het geheel wordt dus elke boog hoogstens 2 keer bekeken. In een planaire graaf zijn er hoogstens $3n - 6$ bogen. De volledige lus is dus lineair in het aantal toppen. De totale complexiteit is hier $O(2 * (3n - 6)) = O(6n - 12)$
- Lijn 22: De for lus wordt precies 7 keer uitgevoerd. De totale complexiteit is hier $O(7 * (\text{complexiteit van elke iteratie}))$
- Lijn 24 → 39 : De complexiteitsanalyse van deze lus is gelijkaardig aan de complexiteitsanalyse van lijn 3 → 21. De buitenste lus gebeurt hoogstens n keer. (i stelt de index voor van de huidige top.) Eerst wordt lokaal gezocht naar het maximum via de aanliggende bogen van top v . Elke aanliggende boog wordt nu hoogstens één keer bekeken op lijn ???. Dit wordt hoogstens 1 keer uitgevoerd bij elke top, een boog heeft 2 eindpunten, dus wordt hoogstens 2 keer vanuit elk eindpunt bekeken. Vervolgens op lijn 29 wordt opnieuw geïtereerd over de bogen van de node max . Een node kan hoogstens één keer als max genomen worden. $n - 2$ nodes kunnen als maximum gekozen worden. Elke boog kan op deze lijn opnieuw hoogstens 2 keer overlopen worden, 1 keer vanuit zijn beide eindpunten, indien beide eindpunten eens de max node zijn. Als laatste wordt ook op lijn 34 nogmaals over dezelfde bogen als op lijn 29 geïtereerd. Elke boog wordt op deze lijn dus ook hoogstens 2 keer bekeken. In totaal wordt deze boog dus hoogstens 4

keer bekeken. Nu kan dus elke boog hoogstens 6 keer bekeken worden in de volledige while-lus. In een planaire graaf zijn er hoogstens $3n - 6$ bogen. De volledige lus is dus lineair in het aantal toppen. De totale complexiteit is hier $(O(6 * (3n - 6))) = O(18n - 36)$.

De volledige complexiteit is nu:

$$(2n - 1) + (6n - 12) + (7 * (18n - 36)) = 134n - 256 (= O(n))$$

Het algoritme is gretig aangezien we itereren over de toppen en telkens lokaal zoeken naar de best toe te voegen top. De top die het beste lijkt wordt vervolgens toegevoegd aan de lijst en zo wordt (het grootste deel van de) verzameling opgebouwd. Het algoritme is nu dus per definitie gretig aangezien het bij elk stadium naar het lokale optimum zoekt. Het gretig algoritme heeft 5 onderdelen: (De 5 onderdelen van een gretig algoritme volgens wikipedia.)

- Een kandidaatverzameling: de verzameling D
- Een selectie functie: het lokaal zoeken naar de optimale top
- Een haalbaarheidsfunctie het vergelijken van de eigenlijke bedekking van een top met 0.
- Een objectieve functie: het toevoegen van het maximum aan de verzameling D
- Een oplossing functie: het vergelijken van de actualCoverage met $—V(G)—$

Vooraf worden ook toppen toegevoegd die buur zijn van een top met graad 1. Hier wordt niet gekeken naar de bedekking van de top. Het is een optimalisatie en wordt voorafgaand aan het werkelijk gretig algoritme uitgevoerd.

2 Implementatie

2.1 Dominantie in vlakke grafen

2.1.1 Algoritme en implementatie

Opslag van de graaf Een graaf is een aparte klas en bevat de volgende 3 datastructuren:

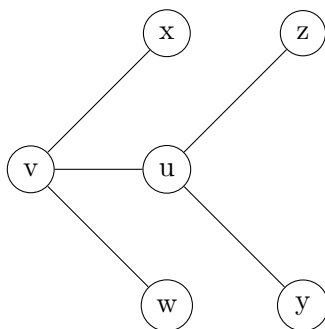
- Een array met bogen
- Een array met toppen
- Een SortedNodeArray (eigen klasse)

De eerste 2 datastructuren zijn redelijk voor de hand liggend. Aangezien initieel telkens het aantal bogen en toppen geweten is uit een graaf is het eenvoudig om deze op te slaan in een array waarbij de index van elke boog en top gelijk is aan zijn nummer min één. Hiernaast bevat de graaf ook een zelf geïmplementeerde datastructuur, deze bevat (na het sorteren) alle toppen opgeslagen met stijgende graad (laagste graad eerst). Aangezien het aantal toppen in een begrens interval ligt, en dus ook de graad van een top begrensd is, is het mogelijk om deze in lineaire tijd met counting sort te sorteren na het toevoegen van alle toppen. Experimenteel blijkt dat het algoritme het best presteert waarvan de toppen opgeslagen zijn met stijgende graad. Een andere optie was om de toppen op te slaan in een binomiale wachtlijn aangezien deze ook in lineaire tijd kan opgebouwd worden. Het probleem hierbij is dat enkel de top bovenaan de boom kan geraadpleegd worden, om de volgende top te verkrijgen moet de bovenste verwijderd worden, wat gebeurt in logaritmische tijd, indien men dit voor alle toppen echter uitvoert, is dit te duur.

Algoritme en optimalisaties Mijn algoritme bestaat uit 3 belangrijke stappen:

1. Sorteert $V(G)$ met counting sort zodat $(\forall v_i, w_j \in V(G))(i < j \rightarrow \deg(v) \leq \deg(w))$
2. $D := D \cup \{v \in V(G) | vw \in E(G) \wedge \deg(w) = 1\}$
3. Voer 7 keer het eigenlijke algoritme uit met dalende minimum graad.

Belangrijk om te vermelden bij de bespreking van de implementatie bij mijn algoritme is het volgende. Telkens een top toegevoegd wordt aan de dominante lijst, wordt de coverage van deze top op 0 geplaatst en de coverage van zijn burens gedecrementeert. Deze coverage wordt gebruikt om te bepalen welke van de toppen uit $\{v\} \cup N_G(v)$ lokaal optimaal zijn om toe te voegen. De top w met de hoogste coverage uit $\{v\} \cup N_G(v)$ wordt telkens toegevoegd. De top v is een top met $coverage > 0$. Na het toevoegen van de top wordt $\forall u \in N_G(w)$ de coverage gedecrementeerd, en de coverage van w op 0 geplaatst. De variabele coverage stelt hier dus eigenlijk een bovengrens voor van de werkelijke bedekking een top kan leveren. De bedekking van een top stelt de bijdrage voor dat een top kan leveren aan de dominante lijst. De bedekking van top v is $\deg(v) + 1$ omdat het toevoegen van top v aan de dominante lijst ervoor zorgt dat de dominante lijst een bereik heeft van 2 toppen, zodra de dominante lijst een bereik heeft van $|V(G)|$ toppen, bereikt hij de hele graaf, dus is de lijst dominant. De dominante lijst D bereikt een top $w \iff w \in D \vee (v \in D \wedge vw \in E(G))$. In wiskundige notatie kan de werkelijke bedekking van een top v als volgt voorgesteld worden: $|\{w \in \{v \cup N_G(v)\} \wedge w \notin D\}|$ Neem bijvoorbeeld figuur 2.1.1. Initieel geldt $(\forall v \in V(G))(coverage(v) = \deg(v) + 1)$,



Figuur 2: Coverage

dus $\text{coverage}(v) = 4$, $\text{coverage}(u) = 4$, $\text{coverage}(w) = \text{coverage}(x) = \text{coverage}(z) = \text{coverage}(y) = 2$. Stel we voegen v toe aan de dominante lijst, dan plaatsen we de coverage van v op 0 en decrementeren we de burens zodat: $\text{coverage}(v) = 0$, $\text{coverage}(x) = \text{coverage}(w) = 1$ en $\text{coverage}(u) = 3$ en $\text{coverage}(z) = \text{coverage}(y) = 2$. Nu wordt het duidelijk dat de coverage niet meer overeen komt met de graad maar een bovengrens voorstelt van de bedekking die een top kan leveren aan de dominante lijst. Top u levert nog hoogstens 3 bedekking aan de dominante lijst. De coverage van z en y is 2, zij leveren echter geen bedekking aangezien hun buur reeds bedekt is door het toevoegen van v . Desondanks dat er met een bovengrens wordt gewerkt voor dit getal, zal het algoritme nog steeds goed presteren met deze benadering. Het is nodig om met dergelijke bovengrens te werken aangezien het onmogelijk is om het algoritme lineair te houden als bij elke iteratie over de burens van de burens van een top moet geïtereerd worden om de exacte graad te bepalen van elke buur. Nu volgt de bespreking van de implementatie van elke stap in het algoritme.

Stap 1: Het eigenlijke sorteren van de SortedNodeArray gebeurt tijdens deze stap. De toppen worden gesorteerd in stijgende graad. Dit is een optimalisatie op het eigenlijke algoritme zodat het algoritme beter en sneller presteert. Deze SortedNodeArray heeft 3 methodes: toppen toevoegen, sorteren en de gesorteerde toppen opvragen. Dit zijn de enige vereisten. Het toevoegen van de toppen gebeurt tijdens het inlezen van de data en het sorteren van de toppen gebeurt éénmalig bij het begin van het algoritme. Vervolgens kunnen de gesorteerde toppen altijd opgevraagd worden verder in het algoritme, deze array verandert niet tijdens het algoritme. Enkel de coverage van de elementen is variabel. Als laatste wil ik de nadruk leggen op het feit dat deze stap een optimalisatie is. Het algoritme zou perfect werken, maar met iets slechtere prestaties zonder deze stap.

Stap 2: Dit is ook een optimalisatie op het eigenlijke algoritme. Indien een top graad 1 heeft, weten we zeker dat de buur van deze top moet toegevoegd worden aan de dominante lijst aangezien de top enkel op deze manier bereikt

kan worden.

Stap 3: Deze stap voegt telkens toppen toe met een werkelijke bedekking groter dan een zeker getal. Dit getal start met 6 en decrementeert na elke iteratie van deze stap. Het algoritme zou ook werken indien er één iteratie is met minimum bedekking 1. Echter, het algoritme levert betere prestaties door eerst alle toppen toe te voegen die een hogere bedekking hebben, en vervolgens toppen met een kleinere bedekking. Voor de duidelijkheid wordt het minimale algoritme, m.a.w. het algoritme die een dominante lijst produceert zonder de voorgaande optimalisaties, beschreven in algoritme 2. Enkele belangrijke optimalisaties in het algoritme:

- Lijn 5: Als de coverage van de top gelijk is aan 0. Is het beter om bij een andere top lokaal te zoeken.
- Lijn 7: Als de bedekking (bovengrens van de werkelijke bedekking) van een top 0 is, heeft het geen zin om deze top verder te bekijken.
- Lijn 9: Als de werkelijke bedekking van een top gelijk is aan 0, heeft het ook geen zin om deze toe te voegen.

Al deze optimalisaties verhinderen dat slechte toppen zouden toegevoegd worden. In het eigenlijke algoritme, wordt vergeleken in deze stappen met een minimum. Dit is echter niet nodig, maar zorgt wel voor betere prestaties. De enige vereiste van het algoritme is, dat de stap die beschreven wordt door algoritme 2 uitgevoerd wordt. In de implementatie wordt bij het bekijken van de nabuurschap van een top over de bogen geïtereerd van de top. De top zelf bevat namelijk een List datastructuur die alle adjecente bogen bevat van de top. Ook wordt een variabele bijgehouden die het bereik van de dominante lijst bijhoudt. Zodra dit gelijk is aan $|V(G)|$ kan het algoritme eindigen. Als laatste wordt ook een boolean visited bijgehouden bij elke top. Een top wordt bezocht zodra hij of een buur wordt toegevoegd aan de dominante lijst. Eens een buur bezocht is, kan hij geen verdere bijdrage leveren tot het vergroten van het bereik van de dominante lijst. De implementatie van het bezoeken van een top verzekerd dat, indien een top nog niet bezocht werd, zijn coverage decrementeerd.

Algoritme 2 Dominante verzameling van vlakke grafen

Input: Een planaire graaf $G(V(G), E(G))$

Output: Een dominante verzameling D

$totalCoverage \leftarrow 0$

$i \leftarrow 0$

while $totalCoverage < |V(G)|$ **do**

$v \leftarrow v_i \in V(G)$

if $coverage(v) > 0$ **then**

 Neem $max \mid (\forall u \in \{v\} \cup N_G(v)) \wedge (max \neq u) \Rightarrow coverage(max) > coverage(u)$

if $coverage(max) > 0$ **then**

$actualCoverage \leftarrow \{w \mid w \in \{max\} \cup N_G(max) \wedge w \text{ niet bezocht}\}$

if $actualCoverage > 0$ **then**

$totalCoverage \leftarrow totalCoverage + actualCoverage$

$coverage(max) \leftarrow 0$

$D \cup \{max\}$

 Bezoek max en zijn burens

end if

end if

end if

$i \leftarrow i + 1$

end while

2.1.2 Experimenten

Optimalisaties Bij de experimenten omtrent optimalisaties gebruik ik steeds de gegeven testsets om het nut van een optimalisatie te testen. Er wordt (onder andere) vergeleken tussen een uitvoering zonder de optimalisatie en met de optimalisatie. De tabellen zijn telkens als volgt geformatteerd:

Sorteren De variabele is hier de ordening van de toppen $V(G)$ gebaseerd op hun graad. Er zijn 3 mogelijkheden: stijgende volgorde, dalende volgorde of willekeurige volgorde.

	Ordering					
Graaf	Geen ordening		Stijgende volgorde		Dalende volgorde	
graaf1.sec	2047/10000	20,47%	2062/10000	20,62%	1998/10000	19,98%
graaf2.sec	9573/50000	19,15%	9544/50000	19,09%	9320/50000	18,64%
graaf3.sec	2158/10000	21,58%	2145/10000	21,45%	2087/10000	20,87%
graaf4.sec	10064/50000	20,13%	10034/50000	20,07%	9741/50000	19,48%
graaf5.sec	2931/7500	39,08%	2945/7500	39,27%	2500/7500	33,33%
graaf6.sec	11691/30000	38,97%	11730/30000	39,10%	10000/30000	33,33%
graaf7.sec	4663/12000	38,86%	4695/12000	39,13%	4000/12000	33,33%
graaf8.sec	17498/45000	38,88%	17610/45000	39,13%	15000/45000	33,33%
triang1.sec	1716/10000	17,16%	1709/10000	17,09%	1672/10000	16,72%
triang2.sec	8451/50000	16,90%	8387/50000	16,77%	8291/50000	16,58%

Minimum bedekking Hye

Toppen met graad 1

Verschillende grafen