Project Gretige Algoritmen

Jarre Knockaert

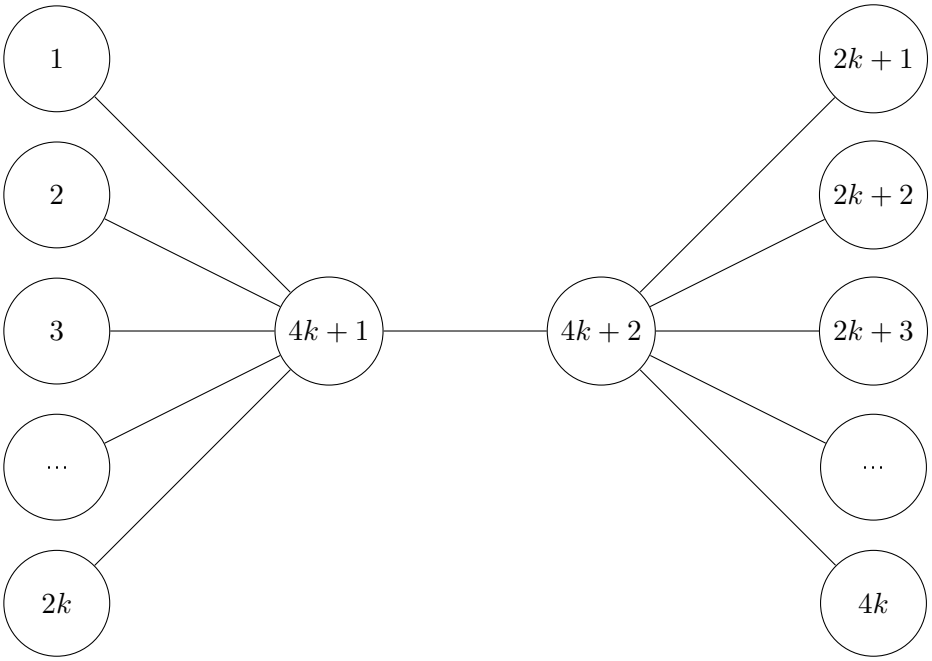
30 oktober 2016

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 buren 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 buren (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 buren.*
- *De werkelijke bedekking/coverage van een top is gelijk aan het aantal onbezochte buren.*
- *De coverage van een verzameling van toppen is een som van de coverage van alle buren.*
- *De totalCoverage is de de som van de coverage van alle buren. 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.*

Algorithm 1 Dominante verzameling van vlakke grafen

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** ▷ Optimalisatie 1
4: Neem $w \mid wv \in E(G)$ ▷ 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** $minimum \in \{6, 5, ..., 0\}$ **do** ▷ Optimalisatie 2
23: $i \leftarrow 0$
24: **while** $starttotalCoverage < |V(G)| \wedge i < |V(G)|$ **do**
25: Neem v, de i-de top, $\in$ V(G)
26: **if** coverage(v) > 0 **then**
27: Neem max $\mid (\forall u \in N_G(v)) \wedge (max \neq u) \Rightarrow coverage(max) > coverage(u))$
28: **if** coverage(v) $>$ coverage(max) **then**
29: max $\leftarrow$ v
30: **end if**
31: **if** coverage(max) $>$ minimum **then**
32: actualCoverage $\leftarrow \{w \mid w \in N_G(max) \wedge w \text{ niet bezocht}\}|$
33: **if** actualCoverage $>$ minimum **then**
34: totalCoverage $\leftarrow$ totalCoverage + actualCoverage
35: $coverage(max) \leftarrow 0$
36: $D \cup \{max\}$
37: Bezoek max en zijn buren
38: **end if**
39: **end if**
40: **end if**
41: $i \leftarrow i + 1$
42: **end while**
43: **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* $\rightarrow$ *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* $O(7*(\textit{complexiteit van elke iteratie}))$

- *Lijn 24* $\rightarrow$ *42 : De complexiteitsanalyse van deze lus is gelijkaardig aan de complexiteitsanalyse van lijn 3* $\rightarrow$ *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 27. 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 32 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 37 nogmaals over dezelfde bogen als op lijn 32 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