Algoritme en complexiteit: opgaves deel 2

Sander van Veen & Richard Torenvliet
Taddeüs Kroes & Jayke Meijer

December 9, 2010

Contents

1 Langste pad in een boom

We kiezen een blad, dus een knoop met slechts 1 connectie naar een andere knoop, en stellen dit als wortel. We krijgen nu dus een boom die helemaal aan 1 kant van de wortel 'hangt'. Van deze boom bepalen we het langste pad, dus de diepte, door een Breadth-First-Search uit te voeren. De knoop die we hierin als laatste tegen komen, is de diepste knoop, en het pad hierheen is nu te bepalen door omhoog te gaan naar de wortel. Noteer dit pad en de lengte van dit pad. Nu kiezen we het volgende blad, en voeren hetzelfde principe uit. Als we alle bladeren hebben gehad, kijken we welk pad het langste is. Dit is het langste pad in de boom.

Het algoritme is in de orde $\mathcal{O}(n^2)$. Het zoeken van het langste pad in de boom is in de orde $\mathcal{O}(n)$. Dit omdat elke knoop langs gegaan moet worden. Dit moeten we uitvoeren op maximaal $n-1$ knopen, in het geval dat alle knopen met elkaar zijn verbonden via 1 centraal punt. Daarom voeren we worst-case $(n-1)*n$ operaties uit, dus n^2-n, en dat is in de orde van $\mathcal{O}(n^2)$.

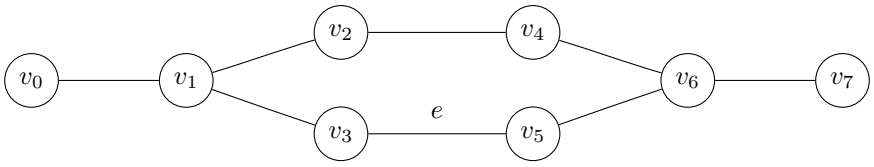
2 Ski's en kinderen

Greedy algoritme: zoek de combinatie $(l_i,\ h_j)$, waarbij $|l_i - h_j|$ minimaal is. Haal l_i en h_j nu uit de verzamelingen, en doe dit opnieuw.

Dit algoritme is in de orde van n^3, en wel omdat je n paar ski's en n kinderen hebt. Per keer zoek je per kind door de verzameling ski's heen en noteer je de waarde voor $|l_i - h_j|$, dus je doet n maal n zoekopdrachten per keer. Vervolgens, nadat je deze n^2 zoekopdrachten hebt uitgevoerd, begin je opnieuw met de verzamelingen van $n-1$ ski's en $n-1$ kinderen. Dit is wederom in de orde van n^2. Dit blijf je herhalen tot er geen kinderen en ski's meer over zijn. Omdat er n kinderen waren, is dit n maal. We voeren dus n maal een n^2 operatie uit, dus de orde van dit algoritme is $\mathcal{O}(n^3)$.

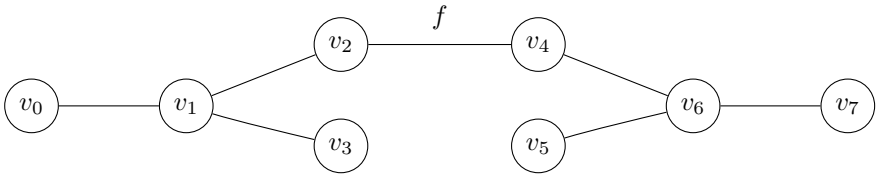
3 Minimum Spanning Tree

(a) Gegeven graaf G met een cycle en een kant e met maximaal gewicht:



Het bewijs dat er is een Minimum Spanning Tree is voor graaf G, waarin kant e niet zit, volgt uit het bewijs van het ongerijmde:

Stel dat e in een MST zit, dan kunnen er twee subtrees worden gemaakt door e uit de tree te halen:



De knopen die verbonden werden met e (v_3 en v_5) zitten nu elk in andere subtree. De twee subtrees worden verbonden met elkaar (kant f), doordat e onderdeel was van een cycle. Omdat er een kant f bestaat die een kleiner gewicht heeft dan e (want e was het maximum), is er een MST in graaf G die de kant e niet bevat. De MST van kant f heeft immers een kleiner totaal gewicht dan de MST met kant e.

(b) Het "Reverse-delete algorithm"[1] beschrijft het bovenstaande principe. Het algoritme beschreven in pseudocode:

```
function ReverseDelete(edges[] E):

    sort E in decreasing order
    i := 0

    for( i := 0; i < size(E); i := i + 1 ):
        if temp.v1 is connected to temp.v2:
            delete E[i]

    return E
```

Op wikipedia[2] is de bron van de bovenstaande pseudocode. Echter wordt daar een delete operatie uitgevoerd voor elke kant en als het niet mogelijk is om de kant te verwijderen, dan wordt de kant teruggeplaatst. Ondanks dat deze verwijder operatie in $\mathcal{O}(1)$ zit, is de pseudocode aangepast zodat

[1] Reverse-delete algorithm: `http://en.wikipedia.org/wiki/Reverse-Delete_algorithm`

[2] Pseudocode Reverse-delete algorithm: `http://en.wikipedia.org/wiki/Reverse-Delete_algorithm#Pseudocode`

de verwijder operatie pas uitgevoerd wordt, als de knopen van een kant nog wel verbonden zijn.

(c) De *running time complexity* van het Reverse-delete algoritme is in $\mathcal{O}(E\ logE\ (log\ log\ E)^3)$, waar E staat voor het aantal kanten in de graaf. Dit volgt uit:

1. Het sorteren van de kanten op aflopend gewicht is van de orde $\mathcal{O}(E\ log\ E)$.
2. Het aflopen van de kanten is van de orde $\mathcal{O}(E)$.
3. Kijken of twee knopen zijn verbonden is in $\mathcal{O}(logV\ (log\ log\ V)^3)$.[3] Hier is V het aantal knopen in de graaf.

Hieruit volgt dat de running time complexity van het Reverse algoritme neerkomt op $\mathcal{O}(E \times logV\ (log\ log\ V)^3)$, want voor elke kant moet gecontroleerd worden of de twee verbonden punten van de kant nog via een ander pad verbonden zijn.

4 Eenheidsintervallen

(a) Om aan te tonen dat het greedy algoritme gegeven in de opdracht niet de optimale indeling geeft, wordt bewezen door het volgende tegenvoorbeeld:

Gegeven de verzameling rationele getallen $A = \{0, \frac{2}{10}, \frac{3}{10}, \frac{11}{10}, \frac{12}{10}, \frac{14}{10}\}$. Het greedy algoritme zou de eenheidsintervallen $\{[-1, 0], [\frac{2}{10}, \frac{12}{10}], [\frac{14}{10}, \frac{24}{10}]\}$ geven. Dat zijn drie intervallen. Het is mogelijk om met twee intervallen de zelfde getallen te omvatten: $\{[0, 1], [\frac{11}{10}, \frac{21}{10}]\}$. Hierdoor is aangetoond dat er een optimalere methode is om het probleem op te lossen. De definitie van een greedy algoritme is dat het de meest optimale methode moet zijn, daar voldoet de gegeven algoritme niet aan.

(b) Pseudocode:

```
quicksort list in ascending order

min := i := first element from the list
V := {}

do:
    i := next element from the list

    if( i > min + 1 ):
        add [min, min+1] to V
        min := i := next element from the list

while( end of the list is not reached )

return V
```

[3] Near-optimal algorithm to check if vertices are connected: `http://portal.acm.org/citation.cfm?doid=335305.335345`

Bewijs dat het werkt:
Als we een willekeurig interval uit V verwijderen, kunnen we geen combinatie van intervalgrenzen in V bedenken zodat alle elementen uit de reeks in een interval in V zijn in te delen. Dit werkt alleen met twee of meer elementen in de lijst.

(c) Het quicksort algoritme zit in $\mathcal{O}(n \log n)$. De iteratie wordt n maal uitgevoerd, dit valt weg tegen de tijdscomplexiteit van de quicksort. Het totale algoritme zit dus in $\mathcal{O}(n \log n)$.

5 Eén na kortste pad

Zoek alle paden van A naar B door alles uit te proberen, met behulp van Dijkstra's algoritme[4], onthoudt alle paden en sorteer deze op lengte van klein naar groot. Retourneer het op één na korste pad, dat is de tweede in de rij.

6 Dijkstra's algoritme

De complexiteit van Dijkstra's algoritme is in de orde van n^2. Dit betekent dat het vinden van het korste pad in de graaf van orde n^2 is. Dit komt omdat je in het ergste geval (de *worst-case*), tussen elke knoop een kant hebt. In dat geval moet je tussen vanaf elke knoop de afstand tot die knoop optellen bij de afstand tot al zijn buren. Dat betekent dat je n maal n keer moet gaan optellen, dus $\mathcal{O}(n^2)$.

In het programma voeren we dit n maal uit, omdat we in de graaf van elk punt de afstand naar alle andere punten berekenen, in plaats van van 1 punt naar alle andere punten. Dit levert ons dus een functie in de orde van $\mathcal{O}(n * n^2 = n^3)$ op.

De complexiteit van het naïve algoritme is voor het berekenen van een enkel punt naar alle andere punten in de orde van $\mathcal{O}(n!)$, omdat je van elke knoop die je tegen komt, in de worst case naar alle andere knopen toe kunt gaan. Dit voeren we ook weer n maal uit, omdat we van elk punt naar elk punt het korste pad zoeken. Dan krijg je dus een functie in de orde van $\mathcal{O}(n * n!)$.

Dit betekent dat als n groter wordt, het verschil steeds meer toe neemt. Dit zien we terug in de test resultaten. We hebben het programma de ratio laten berekenen, dus hoeveel sneller Dijkstra's algoritme is ten opzichte van het naïve algoritme. De resultaten hiervan staan in tabel 1.

[4]Reverse-delete algorithm: `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`

n	ratio
4	1.3
5	2.0
6	4.1
7	9.7
8	25
9	78
10	$2.5 * 10^2$
11	$6.0 * 10^2$
12	$3.0 * 10^3$
13	$2.0 * 10^5$

Table 1: De tabel met daarin de ratio tegenover het aantal punten in de graaf

Bij deze tabel moet wel de kanttekening worden gemaakt dat bij grotere waarden, we over minder grafen hebben getest, omdat het anders te veel tijd in beslag nam. Daardoor waren de testresultaten erg verschillend, en is de weergegeven ratio een benadering.

Hieruit blijkt dus dat zelfs de ratio extreem snel toe neemt. Daaruit blijkt dat Dijkstra's algoritme veel efficiënter is.