UNIVERSITEIT GENT

Algoritmen en Datastructuren 2

Project Gretige Algoritmen

Deadlines: 30 oktober en 27 november 2016

1 Inleiding

Dit jaar gaat het project over gretige algoritmen voor graafinvarianten. Zoals je gezien hebt in Algoritmen en Datastructuren 1 is een graaf een combinatorische structuur die bestaat uit twee verzamelingen: een eerste verzameling die we de toppen van de graaf noemen, en een tweede verzameling die uit paren van elementen van de eerste verzameling bestaat en die we de bogen van de graaf noemen. De grafen waarover wij het hier gaan hebben, gaan allemaal simpele, samenhangende, ongerichte en ongewogen grafen zijn.

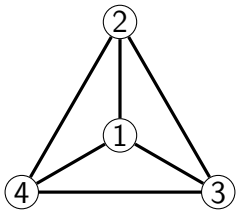
Een graafinvariant is een waarde of een eigenschap van een graaf die niet afhangt van de inbedding of de labelling van de graaf en zijn toppen. Enkele eenvoudige voorbeelden van graafinvarianten zijn o.a. het aantal toppen, de maximale graad, de minimale graad en het samenhangend zijn.

2 Opgave

2.1 Vlakke grafen

Een *vlakke graaf* is een graaf die ingebed is in het vlak zodat geen van de bogen snijden. Die gebieden die afgebakend worden door de bogen en toppen noemen we de vlakken van de graaf. Je kan bewijzen dat een vlakke graaf met n toppen ten hoogste $3n - 6$ bogen en ten hoogste $2n - 4$ vlakken heeft. Het aantal bogen en het aantal vlakken zijn dus beide lineair in het aantal toppen.

Het is meestal niet zo handig om met coördinaten voor toppen te werken. De eigenschappen van een vlakke graaf worden echter ook volledig bepaald door de volgorde van de buren rond een top vast te leggen. Dit noemen we een *combinatorische inbedding* van de graaf. We moeten nog wel bepalen wanneer een graaf samen met een combinatorische inbedding correspondeert met een vlakke graaf. Hiervoor moeten we eerst vastleggen hoe je de vlakken bepaalt van een graaf die combinatorisch is ingebed. Dit zullen we doen aan de hand van het volgende voorbeeld.



Als graaf hebben we dus een complete graaf op 4 toppen, en de volgorde is zoals aangegeven in de tabel hieronder:

Top	Buren
1	2,3,4
2	3,1,4
3	4,1,2
4	2,1,3

Merk alvast op dat de volgorde een cyclische lijst is en we dus op gelijk welke positie hadden kunnen beginnen. Om de vlakken te bepalen beginnen we met een gerichte boog, bijvoorbeeld de boog $(1, 2)$. We vertrekken dus in top 1 en gaan eerst naar top 2. We zoeken dan in de lijst van buren van top 2 naar de top die na top 1 staat. Dit is top 4. We gaan dus naar top 4 en zoeken daar in de lijst van buren naar de top die na top 2 staat. Dit is top 1. We gaan naar top 1 en zoeken daar in de lijst van buren naar de top die na top 4 staat. Dit is top 2 en aangezien we de gerichte boog $(1, 2)$ reeds hebben gebruikt stoppen we. We hebben nu een vlak gevonden: $1 \to 2 \to 4 \to 1$.

Vervolgens herhalen we de stappen hierboven totdat er geen gerichte bogen meer zijn die we nog niet gebruikt hebben in een vlak. De vlakken die we in dit geval vinden zijn: $1 \to 2 \to 4 \to 1$, $1 \to 3 \to 2 \to 1$, $1 \to 4 \to 3 \to 1$, en $2 \to 3 \to 4 \to 2$.

Eenmaal we de vlakken bepaald hebben van een combinatorisch ingebedde graaf, kunnen we snel nagaan of deze ingebedde graaf correspondeert met een vlakke graaf. Een combinatorisch ingebedde graaf is een vlakke graaf als $v - e + f = 2$, met v het aantal toppen, e het aantal bogen en f het aantal vlakken.

2.2 Dominantie in vlakke grafen

Een *dominerende verzameling* D van een graaf is een verzameling van toppen zodat voor elke top v van de graaf geldt dat $v \in D$ of dat v adjacent is met een top in D. Het *dominantiegetal* van een graaf is de grootte van de kleinste dominerende verzameling van die graaf. Het beslissingsprobleem of er een dominerende verzameling bestaat kleiner dan een gegeven getal is NP-compleet – ook voor de klasse van vlakke grafen, en dus is er weinig hoop op een efficiënt algoritme om het dominantiegetal te bepalen. Dat is dus ook niet de opgave voor dit project. Wel vragen we je om een deterministisch, gretig algoritme te ontwikkelen en te implementeren dat een zo klein mogelijke dominerende verzameling opbouwt voor een gegeven vlakke graaf in een tijd die lineair is in het aantal toppen.

2.3 Hamiltoniaanse cykels in vlakke triangulaties

Een *vlakke triangulatie* is een vlakke graaf waarin alle vlakken (ook het buitenste, onbegrensde vlak) driehoekig zijn. Ze worden dus afgebakend door drie toppen en drie bogen. Een combinatorisch ingebedde graaf met n toppen is een vlakke triangulatie als het aantal bogen gelijk is aan $3n - 6$ en het aantal vlakken gelijk is aan $2n - 4$. Een *hamiltoniaanse cykel* is – zoals jullie gezien hebben bij Algoritmen en Datastructuren 1 – een cykel die alle toppen van de graaf exact één keer aandoet.

Het beslissingsprobleem of een graaf een hamiltoniaanse cykel heeft, is NP-compleet, zelfs als we ons beperken tot de klasse van vlakke triangulaties. Een efficiënt algortime schrijven dat bepaalt of een vlakke triangulatie een hamiltoniaanse cykel heeft, zou dus geen eerlijke opgave zijn voor een project in 2e bachelor Informatica. Daarom is de opgave om een deterministisch,

gretig algoritme te schrijven dat probeert om een hamiltoniaanse cykel op te bouwen. We eisen dat de looptijd van dit algoritme lineair is in het aantal toppen.

3 Opdracht

Beantwoord onderstaande theoretische vragen. Implementeer de verschillende algoritmen zoals beschreven in sectie 5. Let daarbij heel goed op complexiteit en efficiëntie. Voer experimenten uit volgens sectie 6 en schrijf een verslag volgens sectie 7.

4 Theoretische vragen

Opgave 1: Beschouw het volgende gretige algoritme om een dominerende verzameling op te bouwen.

```
D = lege verzameling

while(G bevat een top){
    v = top met grootste graad in G
    voeg v toe aan D
    verwijder v en al zijn buren uit G
}

schrijf D uit
```

Construeer voor elke k een vlakke graaf waarvoor het bovenstaande algoritme een dominerende verzameling vindt die minstens k keer groter is dan een optimale dominerende verzameling voor dezelfde graaf.

Opgave 2: Bewijs dat voor een hamiltoniaanse cykel in een vlakke triangulatie er evenveel vlakken binnen als buiten de cykel liggen.

Opgave 3: Geef je algoritmes in pseudocode en analyseer de verschillende stappen om duidelijk te maken dat het algoritme inderdaad in lineaire tijd loopt. Als je theoretische resultaten hebt gebruikt of bewezen om een bepaalde heuristiek te implementeren, dan voeg je die ook toe. Leg ook uit waarom je algoritmen gretig algoritmen zijn.

5 Implementatie

We verwachten de volgende Java code in package `gretig`:

- Een klasse `Hamilton` die een `main` methode heeft en uitgevoerd kan worden zonder een argument mee te geven.
- Een klasse `Dominantie` die een `main` methode heeft en uitgevoerd kan worden zonder een argument mee te geven.

Voorzie je code van voldoende commentaar en geef elke methode een correcte javadoc header. Geef bij elke niet-triviale methode in de javadoc header aan wat de bedoeling is van deze methode en wat eventuele neveneffecten zijn. Implementeer JUnit tests om jouw implementatie op correctheid te testen.

5.1 SIMPLE EDGE CODE

Je programma zal de grafen moeten inlezen in een binair formaat dat SIMPLE EDGE CODE heet. Dit formaat begint steeds met de volgende header: >>SEC<<. Na de header volgen de grafen. Elke graaf start met een byte die aangeeft hoeveel bytes elk volgend getal zal innemen. Als deze byte 1 is, dan bestaat elk volgend getal voor die graaf uit één byte; als deze byte 2 is, dan bestaat elk volgend getal voor die graaf uit twee bytes; enz. Als een getal uit meerdere bytes bestaat, dan wordt litte endian gebruikt. Stel bv. dat elk getal uit drie bytes bestaat: we lezen het volgende getal in en krijgen deze drie bytes:

$$116, 201, 5.$$

Deze stellen dan het volgende getal voor:

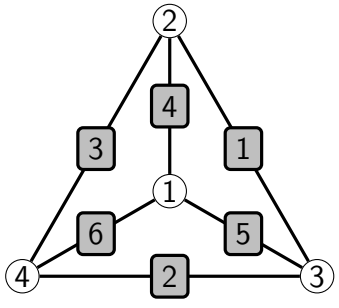
$$116 + 256 * 201 + 256^2 * 5 = 379252.$$

Na deze eerste byte volgt dan een eerste getal dat aangeeft hoeveel toppen de graaf heeft. Het tweede getal geeft aan hoeveel bogen de graaf heeft.

Na deze twee getallen volgt nu voor elke top een lijst die in wijzerzin de bogen rond die top geeft. Hiervoor krijgt elke boog een ander willekeurig nummer van 1 en tot en met het aantal bogen. Elk van deze lijsten wordt afgesloten met een 0.

Na de 0 achter de bogenlijst voor de laatste top, kan dan een nieuwe graaf volgen die opnieuw begint met één byte die aangeeft uit hoeveel bytes elk getal bestaat voor deze graaf.

Als we in de graaf hieronder de nummers in de vierkante vakjes als nummers voor de bogen nemen, dan krijgen we de bogenlijst zoals hieronder.



Top	Bogenlijst
1	4, 5, 6
2	1, 4, 3
3	1, 2, 5
4	2, 3, 6

Een mogelijke SIMPLE EDGE CODE voor deze graaf is dan

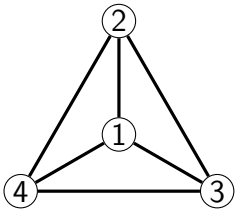
$$62, 62, 83, 69, 67, 60, 60, 1, 4, 6, 4, 5, 6, 0, 1, 4, 3, 0, 1, 2, 5, 0, 2, 3, 6, 0.$$

De eerste zeven getallen in deze code corresponderen met de header >>SEC<<.

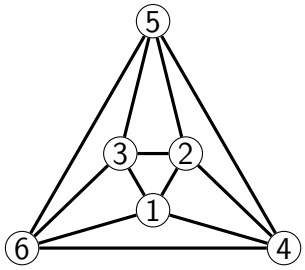
5.2 Uitvoeren en resultaten

Je programma zal de grafen moeten inlezen vanuit 'standard in'. Het resultaat van je programma wordt achteraf uitgeschreven naar 'standard out' als een lijst van getallen gescheiden door spaties met voor elke graaf een nieuwe regel. In het geval van de dominerende verzameling zijn de nummers de labels van de toppen. Voor de hamiltoniaanse cykel geef je de toppen in een volgorde waarin je ze tegenkomt op de cykel. In beide gevallen labelen we de toppen van 1 tot en met n. Als je programma geen hamiltoniaanse cykel heeft gevonden, dan druk je de tekst "geen cykel gevonden" af.

Hieronder zie je enkele voorbeelden van hoe wij je programma gaan testen. We zullen je programma compileren en in een jar met de naam `project.jar` steken.



`graaf1.sec`



`graaf2.sec`

```
$ java -cp project.jar gretig.Dominantie < graaf1.sec
1
$ java -cp project.jar gretig.Dominantie < graaf2.sec
1 5
$ java -cp project.jar gretig.Hamilton < graaf1.sec
1 2 3 4
$ java -cp project.jar gretig.Hamilton < graaf2.sec
geen cykel gevonden
```

Je programma moet ook om kunnen gaan met bestanden die meerdere grafen bevatten. Stel bv. dat het bestand `grafen.sec` de twee grafen van hierboven bevat en je programma gaf de uitvoer die hierboven getoond wordt, dan zou dit de uitvoer moeten zijn voor het bestand `grafen.sec`:

```
$ java -cp project.jar gretig.Dominantie < grafen.sec
1
1 5
$ java -cp project.jar gretig.Hamilton < grafen.sec
1 2 3 4
geen cykel gevonden
```

We hebben de volgende eisen voor de uitvoer:

- De uitvoer van `gretig.Dominantie` moet steeds een dominerende verzameling zijn. Het moet niet de kleinste dominerende verzameling zijn, maar het doel is dus wel om in lineaire tijd een zo klein mogelijke dominerende verzameling op te bouwen.

- De uitvoer van `gretig.Hamilton` moet ofwel een hamiltoniaanse cykel, ofwel de tekst "geen cykel gevonden" zijn.

6 Experimenten

Beschrijf welke experimenten je hebt uitgevoerd om verschillende ideeën die je had te vergelijken. Voeg ook de resultaten van deze experimenten toe.

7 Verslag

Schrijf een verslag van dit project. Beantwoord eerst de theoretische vragen en beschrijf dan jouw implementaties, waaronder mogelijke optimalisaties die je hebt doorgevoerd. Gebruik daarbij steeds de juiste wiskundige notatie. Wat kun je op basis van de experimenten concluderen? Komen je resultaten overeen met je verwachtingen? Probeer steeds om al je resultaten te verklaren.

8 Beoordeling

De beoordeling van het ingeleverde werk zal gebaseerd zijn op de volgende onderdelen:

- Werden de theoretische vragen goed beantwoord? Dit weegt zwaar door.
- Is de implementatie volledig? Is zij correct?
- Is er zelf nagedacht? Is alles eigen werk?
- Hoe goed presteert je algoritme? Het is gemakkelijk om een lineair gretig algoritme te schrijven dat heel slecht presteert. Daarom zal er ook rekening gehouden worden met de prestatie van je algoritme op de testset.
- Hoe is er getest op correctheid?
- Zijn de experimenten goed opgezet? Zijn de parameters voor de datasets goed gekozen?
- Is het verslag toegankelijk, duidelijk, helder, verzorgd, ter zake en objectief?

Projecten die geen lineair gretig algoritme implementeren, zullen niet beoordeeld worden. Je krijgt eerder punten voor een lineair gretig algoritme waar nog een kleine fout in zit, dan voor een algoritme dat totaal de opgave niet volgt.

8.1 Bonus

Met dit project is er naast de gewone punten ook nog een bonus te verdienen. We zullen je programma's gebruiken om dominerende verzamelingen te zoeken voor een verzameling S_d van vlakke grafen en hamiltoniaanse cykels voor een verzameling S_h van vlakke triangulaties. Aan de hand van de resultaten op deze grafen zullen we een score geven. Deze score staat los van je punten op het project.

De dominantiescore voor een vlakke graaf G is

$$s_d(G) = \begin{cases} \frac{|D(G)|-m_d(G)}{|V(G)|-m_d(G)} & \text{als } D(G) \text{ een dominerende verzameling is} \\ 2 & \text{als } D(G) \text{ geen dominerende verzameling is} \end{cases}$$

waarbij $D(G)$ de verzameling is die door jouw programma gevonden wordt, en $m_d(G)$ de orde is van de kleinste dominerende verzameling die gevonden is voor de graaf G.

De hamiltonscore voor een vlakke graaf G is

$$s_h(G) = \begin{cases} 0 & \text{als een hamiltoniaanse cykel gevonden wordt} \\ 0 & \text{als de output "geen cykel gevonden" is} \\ & \text{en niemand heeft een cykel gevonden} \\ 1 & \text{als geen hamiltoniaanse cykel gevonden wordt,} \\ & \text{maar andere studenten hebben wel een cykel gevonden} \\ 2 & \text{als de output ongeldig is} \end{cases}$$

De score voor jouw project is dan gelijk aan

$$s = \sum_{G \in S_d} s_d(G) + \sum_{G \in S_h} s_h(G).$$

Vervolgens kijken we naar het project met de *laagste* score, en bij een gelijkstand naar dat waarvoor de totale looptijd het kleinst was, en dit project krijgt een bonus van 2 punten voor dit vak.

De grafen die gebruikt zullen worden om deze bonus te bepalen zijn random vlakke grafen (maar wel voor iedereen dezelfde grafen), die gelijkaardig zijn aan de testset die jullie via Minerva kunnen downloaden.

Als je project geen gretig algoritme implementeert, of als de looptijd niet lineair is, dan zal je project gediskwalificeerd worden voor deze bonus. Ook als je project een incorrect geformateerde uitvoer heeft (t.t.z. je uitvoer is niet zoals hierboven geformateerd, of naast de gevraagde uitvoer schrijft je programma nog extra zaken naar 'standard out'), zal je gediskwalificeerd worden voor de bonus.

9 Deadlines

Er zijn twee indienmomenten:

1. Zondagavond 30 oktober 2016 om 23 uur 59 moet een implementatie van het programma om dominerende verzamelingen te vinden ingeleverd worden en een verslag.pdf met daarin de antwoorden op de theoretische opgaven 1 en 3 (voor het eerste algoritme).

2. We verwachten een volledig project tegen zondagavond 27 november 2016 om 23 uur 59.

Code en verslag worden elektronisch ingediend. Van het verslag van het **tweede** indienmoment verwachten we ook een **papieren versie**. Nadien zal je mondeling je werk verdedigen. Het tijdstip hiervoor wordt later bekendgemaakt.

Om de bonus te bepalen zullen we gebruik maken van de versies die bij het tweede indienmoment ingediend worden. Je mag dus na de eerste deadline nog verder werken aan het programma om dominerende verzamelingen te vinden, al is het misschien beter om je op dat moment te focussen op de hamiltoniaanse cykels.

10 Elektronisch indienen

Op https://indianio.ugent.be/ kan elektronisch ingediend worden. Maak daartoe een ZIP-bestand en hanteer daarbij de volgende structuur:

- verslag.pdf is de elektronische versie van je verslag in PDF-formaat.
- De package `gretig` bevat minstens de klassen gespecifieerd hierboven. Alle code bevindt zich onder deze package.
- De subdirectory test bevat alle JUnit-tests.

11 Algemene richtlijnen

- Zorg ervoor dat alle code compileert met Oracle Java 8 update 60 of hoger.
- Extra externe libraries zijn niet toegestaan. De JDK en JUnit mogen wel.
- Niet-compileerbare code en incorrect verpakte projecten **worden niet beoordeeld**.
- Het project wordt gequoteerd op **4** van de 20 te behalen punten voor dit vak, en deze punten worden ongewijzigd overgenomen naar de tweede examenperiode.
- Projecten die ons niet bereiken voor beide deadlines worden niet meer verbeterd: dit betekent het verlies van alle te behalen punten voor het project.
- Dit is een individueel project en dient dus door jou persoonlijk gemaakt te worden. Het is niet toegestaan om code uit te wisselen of over te nemen van het internet. Wij gebruiken geavanceerde plagiaatdetectiesoftware om ongewenste samenwerking te detecteren. Zowel het gebruik van andermans werk, als het delen van werk met anderen, zal als examenfraude worden beschouwd.