UNIVERSITEIT GENT

Algoritmen en Datastructuren 2

Project Dynamisch Programmeren

Deadlines: 28 oktober en 25 november 2018

1 Inleiding

Dit jaar gaat het programma over dynamisch programmeren. We zullen werken met gerichte grafen waarbij de toppen gelabeld zijn met symbolen en gaan in deze gerichte grafen op zoek naar paden met speciale eigenschappen. 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. Bij een gerichte graaf bestaat de tweede verzameling uit geordende paren waardoor de bogen een richting hebben – ze hebben een expliciet beginpunt en een expliciet eindpunt. In afbeeldingen zullen we de gerichte bogen voorstellen door een pijl.

Tijdens de loop van jouw algoritme ga je normaal gezien verzamelingen van deze symbolen moeten bijhouden. Een deel van de opdracht is om hiervoor bitvectoren te gebruiken.

2 Opgave

Het project gaat over het zoeken van *maximaal palindromische paden* in *gerichte grafen*. Alle nodige definities worden hieronder gegeven.

2.1 Gerichte grafen

Een *gerichte graaf* $G = (V, E)$ bestaat uit een verzameling toppen V en een verzameling gerichte bogen $E \subset V \times V$. Een boog is dus een koppel (u, v) met een richting van u naar v, waarbij $u = v$ is toegelaten.

Een *pad* van lengte k in een gerichte graaf is een reeks toppen $v_1, \ldots, v_k$ (die mogelijks dubbels bevat) zodat er voor elke $1 \leq i < k$ een boog (v_i, v_{i+1}) bestaat. Een pad waarvoor $v_1 = v_k$ noemen we een *(gerichte) cykel*.

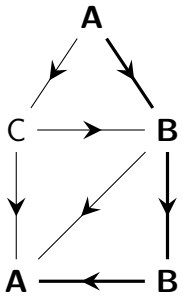
Een *acyclische graaf* is een gerichte graaf die geen gerichte cykels bevat.

2.2 Palindromische paden

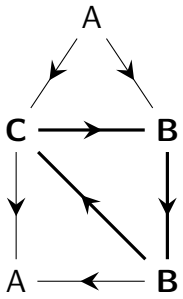
In een gerichte graaf G waarvan elke top gelabeld is met een symbool, is een *palindromisch pad* een pad waarvan de symbolen op de toppen een palindroom vormen. Dat betekent dus dat je dezelfde reeks symbolen krijgt als je het pad in tegengestelde richting volgt.

Een *palindromische cykel* is een palindromisch pad dat begint en eindigt in dezelfde top.

Als er voor elke $k \in \mathbb{N}$ een palindromisch pad van lengte $n > k$ bestaat, dan noemen we G *palindromisch onbegrensd*.



(a) Een acyclische graaf met een maximaal palindromisch pad



(b) Een palindromisch onbegrensde graaf met een palindromische cykel

Figuur 1: Voorbeelden van palindromische paden in gerichte grafen.

Als G niet palindromisch onbegrensd is, dan noemen we de graaf *palindromisch begrensd* en is een *maximaal palindromisch pad* een palindromisch pad van maximale lengte. Er kunnen meerdere maximaal palindromische paden zijn in een graaf.

2.3 Opdracht

Gegeven een gelabelde gerichte graaf G, willen we alle maximaal palindromische paden vinden.

Voor het eerste deel van de opgave veronderstellen we dat G acyclisch is. Bedenk en implementeer een efficiënt dynamisch programmeren algoritme dat een maximaal palindromische pad vindt. Bereken vervolgens ook de doorsnede van de symbolen op alle maximaal palindromische paden. Dit zijn de symbolen die in elk maximaal palindromisch pad voorkomen. Als er geen maximaal palindromische paden zijn, is deze doorsnede leeg.

Er zijn 62 toegelaten symbolen. Dit zijn de hoofdletters A-Z, de kleine letters a-z, en de cijfers 0-9. Gebruik bitvectoren om de verzameling van gebruikte symbolen bij te houden.

In het tweede deel laten we cykels toe in G. De rest van de opgave blijft ongewijzigd.

3 Theoretische vragen

Het theoretische gedeelte is even belangrijk als de implementatie, dus besteed er voldoende tijd aan. Geef een duidelijke beschrijving met pseudocode van jouw algoritme. Implementatiedetails en Java-specifieke dingen horen hier niet thuis. Zorg ervoor dat jouw bewijzen volledig en correct zijn. Je mag gebruik maken van resultaten uit de cursus. Vermeld altijd welke eigenschappen je gebruikt.

1. Bewijs dat jouw algoritme voor acyclische grafen correct is.

2. Bepaal en bewijs de complexiteit van jouw algoritme voor acyclische grafen. Bereken de complexiteit zowel in functie van het aantal toppen n, als in functie van het aantal toppen n en het aantal bogen m.

3. Geef een voorbeeld van de volgende situaties en leg het uit, of bewijs dat het onmogelijk is.

(a) een acyclische graaf die palindromisch onbegrensd is

(b) een gerichte graaf met cykel die palindromisch begrensd is

(c) een maximaal palindromisch pad dat een cykel bevat

(d) een palindromisch onbegrensde graaf zonder palindromische cykels

4. Het maximale aantal verschillende paden in een gerichte graaf is exponentieel in functie van het aantal toppen. De maximaal palindromische paden zijn daar natuurlijk maar een deel van. Is het maximale aantal maximaal palindromische paden exponentieel?

5. Geef en bewijs een polynomiale bovengrens voor de lengte van een maximaal palindromisch pad in functie van het aantal toppen. Dit hoeft geen scherpe bovengrens te zijn.

6. Bewijs dat jouw algoritme voor algemene gerichte grafen correct is.

7. Bepaal en bewijs de complexiteit van jouw algoritme voor algemene gerichte grafen.

4 Implementatie

We verwachten de volgende code in de package `palindrome`:

- Een klasse `DAG` die een `main` methode heeft en uitgevoerd kan worden zonder een argument mee te geven. Deze implementeert het algoritme voor acyclische gerichte grafen.
- Een klasse `DG` die een `main` methode heeft en uitgevoerd kan worden zonder een argument mee te geven. Deze implementeert het algoritme voor algemene gerichte grafen.

Voorzie jouw 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.

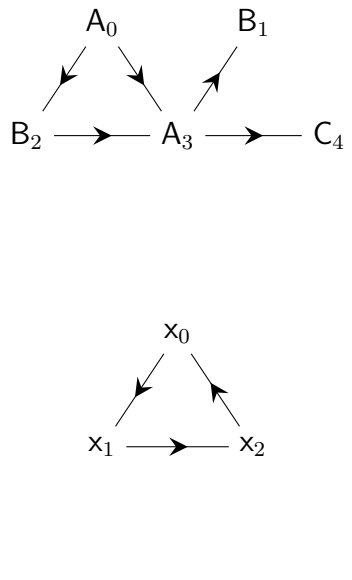
4.1 Input

De input van jouw programma is een gerichte graaf met gelabelde toppen. De graaf wordt gelezen van standaard input (`stdin`). De eerste regel bestaat uit één getal n dat het aantal toppen geeft. Daarna volgen voor elke top, genummerd van 0 tot $(n-1)$, twee regels. De eerste regel geeft het label van top i en het aantal uitgaande bogen (i, j), gescheiden door een spatie. De tweede regel geeft de nummers van alle toppen j waarvoor er een gerichte boog (i, j) bestaat, gescheiden door spaties.

De input kan uit meerdere grafen achter elkaar bestaan.

4.2 Output

Jouw programma moet voor elke graaf in de input de doorsnede van de symbolen in de maximaal palindromische paden geven, en één maximaal palindromisch pad als dat bestaat. Dit moet geschreven worden naar de standaard output (`stdout`). Voor elke graaf schrijf je een regel naar de standaard output (`stdout`) met (gescheiden door spaties):




```
5
A 2
2 3
B 0

B 1
3
A 2
1 4
C 0

3
x 1
1
x 1
2
x 1
0
```

Figuur 2: Een voorbeeld van twee gerichte grafen met bijbehorende input. De index bij de symbolen geeft de nummer van de top, en hoort dus niet bij het symbool.

1. de maximale lengte van een palindromisch pad, of `0` als dat niet bestaat;
2. de symbolen in de doorsnede, gesorteerd in de volgorde A-Z a-z 0-9 en zonder spaties, of / als de doorsnede leeg is;
3. de opeenvolgende nummers van één maximaal palindromisch pad, als dit bestaat.

De eerste graaf van Figuur 2 heeft maximaal palindromische paden $\mathtt{A}_0\mathtt{B}_2\mathtt{A}_3$ en $\mathtt{B}_2\mathtt{A}_3\mathtt{B}_1$. De doorsnede daarvan zijn de symbolen A en B.

De tweede graaf heeft geen maximaal palindromisch pad, dus de doorsnede is leeg.

Een mogelijke output – als er verschillende maximaal palindromische paden zijn, kan je kiezen – wordt gegeven in Figuur 3.

```
3 AB 0 2 3
0 /
```

Figuur 3: De output voor de grafen van Figuur 2

4.3 Bitvectoren

Gebruik bitvectoren om verzamelingen van symbolen bij te houden. Hiervoor mag je de ingebouwde klasse `BitSet` *niet* gebruiken.

Naar eigen inschatten mag je op andere plaatsen ook bitvectoren gebruiken als dit de efficiëntie ten goede komt.

4.4 Testen en optimalisatie

Implementeer JUnit 4 tests in de package `test` om jouw implementatie op correctheid te testen en zorg dat je alle onderdelen van jouw implementatie goed test. Schrijf jouw testen zo vroeg mogelijk tijdens het project zodat je geen tijd verliest met het optimaliseren van een foute implementatie.

Als jouw programma correct is, kan je het beginnen optimaliseren. Ga op zoek naar de kritieke punten, en probeer die efficiënter te implementeren. Controleer ook of jouw optimalisaties effect hebben.

4.5 Uitvoeren

We zullen jouw programma compileren en in een jar met de naam project.jar steken. Vervolgens voeren we het uit met

```
$ java -Xmx512m -cp project.jar palindrome.DAG < input.txt
$ java -Xmx512m -cp project.jar palindrome.DG < input.txt
```

5 Verslag

Schrijf een verslag van dit project. Beantwoord eerst de theoretische vragen en beschrijf dan jouw implementaties, waaronder mogelijke optimalisaties die je hebt doorgevoerd. Voeg de resultaten van jouw experimenten op overzichtelijke wijze toe. Wat kun je op basis van de experimenten concluderen? Komen jouw resultaten overeen met jouw verwachtingen? Probeer steeds om al jouw resultaten te verklaren. Leg ook uit waarvoor je bitvectoren hebt gebruikt. Waarom is dit (g)een goede keuze? Wat zijn de alternatieven?

6 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 ze correct?
- Is er zelf nagedacht? Is alles eigen werk?
- Hoe goed presteert jouw algoritme?
- Hoe is er getest op correctheid?
- Zijn de experimenten goed opgezet?
- Is het verslag toegankelijk, duidelijk, helder, verzorgd, ter zake en objectief?

7 Deadlines

Er zijn twee indienmomenten:

1. Zondagavond 28 oktober 2018 om 23u59 moet de code voor acyclische grafen en antwoorden op vragen 1, 2 en 3 ingediend worden. Je mag hieraan nog wel wijzigingen doen tegen het volgende indienmoment.

2. We verwachten een volledig project tegen zondagavond 25 november 2018 om 23u59.

Code en verslag worden elektronisch ingediend. Van het verslag van het **tweede** indienmoment verwachten we ook een **papieren versie**. Nadien zal elk project individueel besproken worden met een van de assistenten. Het tijdstip hiervoor wordt later bekendgemaakt.

8 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 jouw verslag in PDF-formaat.
- De package `palindrome` bevat minstens de klassen gespecifieerd hierboven. Alle code bevindt zich onder deze package.
- De directory test bevat alle JUnit-tests.

Bij het indienen wordt gecontroleerd of jouw code voldoet aan de gevraagde structuur en interfaces, en of ze uitgevoerd kan worden. Wacht dus niet tot het laatste moment om een eerste keer in te dienen, enkel de laatste versie wordt verbeterd.

9 Algemene richtlijnen

- Zorg ervoor dat alle code compileert met Oracle Java 8.
- Extra externe libraries zijn niet toegestaan. De JDK en JUnit 4 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. We 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.