Gegevensstructuren en Algoritmen: Practicum 3

Academiejaar 2020-2021

1 Communicatie

Indien je bij deze opgave vragen zou hebben, stel dan je vragen eerst via het Toledo forum. Voor zaken die je niet publiekelijk kan communiceren, mail je in dat geval naar `maxwell.szymanski@kuleuven.be`. Zorg dat het onderwerp van je e-mail begint met `[GnA]`.

2 Gedragscode

(Laatste update gedragscode: 12 maart 2019)

De practica worden gequoteerd, en het onderwijs- en examenreglement is dan ook van toepassing. Soms is er echter wat onduidelijkheid over wat toegestaan is en niet inzake samenwerking bij opdrachten zoals deze.

De oplossing en/of verslag en/of programmacode die ingediend wordt moet volledig het resultaat zijn van werk dat je zelf gepresteerd hebt. Je mag je werk uiteraard bespreken met andere studenten, in de zin dat je praat over algemene oplossingsmethoden of algoritmen, maar de bespreking mag niet gaan over specieke code of verslagtekst die je aan het schrijven bent, noch over specieke resultaten die je wenst in te dienen. Als je het met anderen over je practicum hebt, mag dit er dus NOOIT toe leiden, dat je op om het even welk moment in het bezit bent van een geheel of gedeeltelijke kopie van het opgeloste practicum of verslag van anderen, onafhankelijk van of die code of verslag nu op papier staat of in elektronische vorm beschikbaar is, en onafhankelijk van wie de code of het verslag geschreven heeft (mede-studenten, eventueel uit andere studiejaren, volledige buitenstaanders, internet-bronnen, e.d.). Dit houdt tevens ook in dat er geen enkele geldige reden is om je code of verslag door te geven aan mede-studenten, noch om dit beschikbaar te stellen via publiek bereikbare directories of websites. Elke student is verantwoordelijk voor de code en het werk dat hij of zij indient. Als tijdens de beoordeling van het practicum er twijfels zijn over het feit of het practicum zelf gemaakt is (bvb. gelijkaardige code, grafieken, of oplossingen met andere practica), zal de student gevraagd

worden hiervoor een verklaring te geven. Indien dit de twijfels niet wegwerkt, zal er worden overgegaan tot het melden van een onregelmatigheid, zoals voorzien in het onderwijs- en examenreglement (zie `http://www.kuleuven.be/onderwijs/oer/`).

3 Wat is de bedoeling van dit practicum?

3.1 Probleem

In dit practicum ga je een grid-pathfinder implementeren dat, gegeven een start- en eindtegel, een mogelijk pad vindt tussen de twee. Hieraan kunnen nog enkele constraints (beperkingen) verbonden zijn waar de pathfinder rekening mee moet houden. Voor dit practicum werken wij met de volgende context: gegeven een $n \times m$ bord, moet een hardloper vanuit startpositiepositie `P` in 0×0 (wij gebruiken de conventie linksboven) de finish of goal `G` in $n-1 \times m-1$ (conventie rechtsonder) bereiken. De loper begint met een bepaald energieniveau, en kan ofwel naar rechts of naar onder bewegen. Er zijn echter verschillende soorten tegels waarlangs de loper kan passeren: een gewone tegel (aangeduid met `.`) kost één energiepunt, een modderpoel (aangeduid met `M`) twee punten, een heuvel (aangeduid met `H`) drie punten, en bij een energietegel `E` *krijgt* de loper netto één energiepunt (i.e. -1 punt voor de verplaatsing maar +2 energiepunten = +1). Door een muur `X` kan de loper niet passeren.

Het doel van de loper is om een geldig pad uit te stippelen zodanig dat hij zo veel mogelijk energie overhoudt bij het bereiken van de goal. Om een pad als een geldige oplossing te beschouwen, moet de energie van de loper in elke tegel (inclusief de finish) strikt groter zijn dan nul. Hier een voorbeeld:

```
. . M . . .
H M . . X X
H . E . E E
E E X . . .
```

Indien de hardloper (beginnend bij de tegel linksboven) met een energieniveau van 5 naar de finishline (goal rechtsonder) rent, zorgt de volgende sequentie:

```
[RIGHT, RIGHT, DOWN, DOWN, RIGHT, RIGHT, RIGHT, DOWN]
```

voor een zo hoog mogelijk energie-niveau van 2 wanneer de loper aankomt bij de finish. Merk op dat er meerdere optimale paden zijn die tot dezelfde oplossing leiden, zo is

```
[RIGHT, DOWN, DOWN, RIGHT, RIGHT, RIGHT, RIGHT, DOWN]
```

ook een optimaal pad. Wanneer de loper echter begint met een energieniveau van 4, is er geen geldig pad tussen de start- en eindpositie: hoewel de loper met een positief energieniveau zou kunnen eindigen (bijvoorbeeld met bovenstaande sequentie), zou zijn energieniveau op een bepaald moment tijdens het lopen kleiner of gelijk zijn aan 0. Volgens onze definitie is dit geen geldig pad, en is er dus geen geldige oplossing.

De bedoeling van dit practicum is om in deze context telkens het pad te vinden waarbij de loper toekomt met de hoogst mogelijke resterende energie, alsook een mogelijk pad te geven dat tot deze oplossing leidt.

3.2 Recursieve methode

Je kan het probleem aanpakken door vanuit een tegel recursief te kijken welke paden er mogelijk zijn vanuit de naburige tegels, beginnend bij de startpositie. Bespreek de theoretische tijds- en geheugencomplexiteit van deze recursieve methode. Hoeveel berekeningen zou je nodig hebben voor een 100×100 bord? En voor een 1000×1000 bord? Wat maakt de recursieve methode zo inëfficient, en hoe zou dynamic programming dit kunnen oplossen?

Noot: de recursieve methode moet je niet implementeren, je wordt enkel gequoteerd op je theoretische bespreking ervan.

3.3 Dynamic programming

Je kan dit probleem ook oplossen met behulp van dynamisch programmeren. In de meegeleverde code moet je de volgende functies implementeren in de `Board`-klasse:

- `boolean isSolvable()`: geeft `true` indien er een geldig pad is van de startpositie tot de finish.
- `int getFinishTileEnergy()`: geeft de hoogst mogelijke energieniveau dat de loper zou hebben als hij/zij aankomt in de finishtegel, na het volgen van het ideale pad.
- `String[] getPathSequence()`: geeft een string-array bestaande uit de elementen `RIGHT` en `DOWN`, die een pad aanduiden vanuit de startpositie tot finish zodoende dat de loper een zo hoog mogelijk energieniveau heeft in de finish.

Bespreek in detail hoe je dit zou aanpakken: welke tussenberekeningen sla je op, hoe kom je tot een oplossing, e.d. Je mag hiervoor extra hulpfuncties schrijven maar *bestaande functies moet je laten zoals ze zijn*. Welke veronderstellingen maak je bij het opstellen van je tabel? Zorgt dit altijd voor een optimale oplossing? Zo ja, in welke zin is de oplossing optimaal? Zo niet, waarom niet? Bespreek ook de tijd- en geheugencomplexiteit van deze aanpak.

3.4 Uitbreiding: twee hardlopers

Voor de uitbreiding bekijken we het volgend probleem: we werken nog steeds in een `n x m` bord, waarbij hardloper `P1` linksboven met zoveel mogelijk energie zijn goal-positie `G1` rechtsonder bereikt door naar rechts (`RIGHT`) en onder (`DOWN`) te lopen. Tijdens deze race zit er echter ook een tweede hardloper `P2` in het team. Deze moet een weg afleggen van linksonder tot goal-positie `G2` rechtsboven, en kan enkel naar rechts (`RIGHT`) en boven (`UP`) lopen. Tijdens het afleggen van hun paden moeten de spelers een baton doorgeven. Dit doen ze door elkaars afgelegd pad exact één keer te kruisen. Dit wilt zeggen, hun paden moeten in één, en hoogstens één tegel $i \times j$ kruisen (met $0 < i < n-1$, $0 < j < m-1$), zodanig dat beide lopers `P1` en `P2` hun goal-positie met maximale energie bereiken: `G1` en `G2` respectievelijk. De spelers kunnen dit slim aanpakken door de baton te laten liggen op de tegel waar hun pad kruist. Hierdoor moeten ze elkaar niet op hetzelfde moment kruisen.

Implementeer je oplossing hiervoor in de klasse `Board2P`: de functies `getFinishTileEnergy()` en `getPathSequence()` besproken in sectie 3.3 hebben nu elk een `P1` en `P2` suffix, waarbij ze voor `P1` de oplossing geven van de loper die linksboven begint en rechtsonder eindigt, en voor `P2` de oplossing geven van de loper die linksonder begint, en rechtsboven begint. Bespreek je implementatie en hoe je dit probleem hebt aangepakt met dynamisch programmeren. Welke aanpassingen heb je moeten maken t.o.v. je oplossing voor sectie 3.3? Bespreek ook de tijdscomplexiteit van je oplossing.

4 Technische uitwerking

1. Voorbereiding
   (a) Installeer het programma Ant en voer het eens uit. Doe dit in het begin, zo kom je vlak voor de deadline niet voor verrassingen te staan. Bekijk hiervoor de appendices onderaan het document.
   (b) Implementeer de functies in `Board` en `Board2P` zodat ze voldoen aan de beschrijving in sectie 3.3 en 3.4.
   (c) Schrijf JUnit tests in UnitTests.java die automatisch de hulpfuncties in `Board` en `Board2P`, zoals `Board.isSolvable`, `Board.getFinishTileEnergy` en `Board.getPathSequence`, uitvoeren en testen. We raden je ook aan om enkele tests te schrijven die automatisch puzzels oplost en controleert of de oplossing optimaal en correct is. Schrijf testen zodat ze zo veel mogelijk randgevallen behandelen. Zo voorkom je fouten in je implementatie die je makkelijk over het hoofd kan zien.
   (d) Met `ant run` kan je de code uitvoeren die in de `Main.java` staat.
2. Verslag

(a) Schrijf je verslag, zoals besproken in sectie 5. Wees volledig, bespreek je implementatie, en ga er van uit dat de lezer je code niet door heeft genomen.

3. Releasen

(a) Voer `ant test` uit om te controleren op een aantal veelvoorkomende fouten. Kijk zelf ook geregeld of je code werkt door zelf testen te schrijven en ze uit te voeren.

(b) Controleer of op je verslag je naam en studentennummer staat zodat we bij het printen weten welk verslag van wie is.

(c) Maak een zipfile met Ant (met het commando ant release). Deze opgave bevat een appendix dat wat uitleg boedt over hoe je Ant moet gebruiken. Ant is verplicht. Zo heeft iedere zip-file dezelfde directory structuur: ervaring leert dat dit niet lukt als studenten de zip-file met de hand maken. Hierdoor kunnen we sneller verbeteren en dus sneller feedback geven. Als je .zip-file duidelijk niet met Ant is gemaakt, zullen we hem niet verbeteren.

(d) Ant maakt een zipfile build/firstname lastname studentnumber.zip. Vul hier je voornaam, achternaam en studentennummer (inclusief letters 'r', 's' of 'm', merk op dat dit geen hoofdletters zijn) in in de bestandsnaam. Verwissel niet je voornaam en je achternaam.

i. Fout: `Janssens_Jan_r0123456.zip`: verkeerde volgorde

ii. Fout: `Jan_Janssens_0123456.zip`: geen letter bij studentennummer

iii. Fout: `Jan_Janssens_R0123456.zip`: hoofdletter bij studentennummer,

iv. Juist: `Jan_Janssens_r0123456.zip`

(e) Open je zip-file en controleer of alles er in zit.

(f) Upload je zip-file op Toledo. Je hoeft geen papieren verslag in te dienen. Indien Toledo niet bereikbaar is, mail dan een screenshot hiervan en je zip naar de verantwoordelijke van dit practicum (zie sectie Communicatie).

5 Verslag

Naast het implementeren van van je oplossing in `Board` en `Board2P`, vragen we om de vragen in bovenstaande secties te beantwoorden. Wees uitgebreid in je bespreking van je implementatie: **je wordt in de eerste plaats gequoteerd op je bespreking van de oplossing en implementatie.** Plaats ook nu weer je verslag in de vorm van een PDF-bestand met de naam `report.pdf` in de map report. Structureer je verslag volgens de vragen: bespreek eerst sectie 3.2, vervolgens sectie 3.3, en ten slotte 3.4 . Vermijd het door elkaar beantwoorden van vragen.

Je wordt in de eerste plaats beoordeeld op je verslag. We doelen naar je kennis van dynamisch programmeren, en je bekwaamheid om dit correct toe te passen in de bovenstaande context. Hiernaast moet je algoritme een optimale oplossing geven zoals gevraagd in de opgave.

6 Deadline

De deadline is **donderdag 27 mei 2021 om 14u**. Laattijdige inzendingen worden niet aanvaard.

A Hoe Ant installeren

Ant is niet standaard bijgeleverd bij Java en ook niet bij Windows. Je moet Ant dus eerst installeren.

Windows thuis: Het tweede google-resultaat over hoe je Ant installeert onder Windows levert `https://code.google.com/archive/p/winant/downloads` op. Dit is een heel eenvoudige installer. Deze installer vraagt wat de directory is waar JDK is geïnstalleerd: dit is typisch zoiets als `C:\Program Files\Java\jdk1.7.0 17\bin` (afhankelijk van welke JDK je precies hebt).

Linux thuis: Voor Ubuntu en Debian: de installatie is eenvoudigweg "
`sudo apt-get install ant`" intypen in een terminalvenster. Voor andere distributies: gebruik je

package manager.

`PC-labo computerwetenschappen` (gebouw 200A): Ant is reeds geïnstalleerd.
`Mac OS X:`

1. Open een terminal
2. Type volgende commando's:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

brew install ant
```

Ant is nu geinstalleerd. Als je bij het uitvoeren van `ant` de error krijgt dat je JDK moet installeren, dan moet je dat doen. Je hebt JDK (Java Development Kit) nodig om Java programma's te compileren in het algemeen, dus ook als je Java programma's compileert via Ant.

B Hoe Ant gebruiken

1. Start een terminalvenster (dit werkt ook onder Windows: menu start, dan execute, dan `"cmd"` intypen: zie anders `http://www.google.com/search?q=how+to+open+windows+command)`
2. Navigeer naar de directory waar je bestanden voor dit practicum staan, meer bepaald de directory waar zich `build.xml` in bevindt. Met `"cd"` verander je van directory en met `"ls"` (Unix) of `"dir"` (Windows) toon je de bestanden en directories in de huidige directory.
3. Typ `"ant release"`. Ant doet een aantal checks om je tegen een aantal fouten te beschermen. Check dus of Ant geen error gaf.