UNIVERSITEIT GENT

Algoritmen en Datastructuren 2

Semi-splay – all the way

Deadlines: 27 oktober en 24 november 2019

1 Inleiding

In dit project gaan we een geparametriseerde variant op de semi-splaybomen uit de cursus implementeren en bekijken wat de theoretische en experimentele performantie van deze variant is in functie van de parameter.

2 Opgave

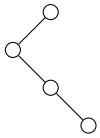
Bij de semi-splaybomen zoals ze in de cursus beschreven staan, worden op het pad naar de wortel steeds drie toppen genomen, en dit deelpad wordt vervangen door een volledige binaire boom met drie toppen. De bedoeling van dit project is om een variant te implementeren, waarbij de splaygrootte k (het aantal toppen dat wordt samengenomen in één splaystap) kan aangepast worden, maar we stellen wel dat dit steeds minstens drie moet zijn. Vervolgens vervangen we dit deelpad door een perfect gebalanceerde boom, dus met diepte $\lfloor \log(k) \rfloor$. Als de parameter k van de vorm $2^n - 1$ is met n een natuurlijk getal, dan is dit een volledige binaire boom. Als k echter niet van deze vorm is, dan zijn er meerdere mogelijke bomen. Je bent hier vrij om zelf een keuze te maken (eventueel gebaseerd op experimenten), maar kies wel een boom die zo goed mogelijk gebalanceerd is.

3 Theoretische vragen

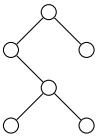
Het theoretische gedeelte is even belangrijk als de implementatie, dus besteed er voldoende tijd aan. Geef een duidelijke beschrijving met pseudocode van je algoritmes. Implementatiedetails en Java-specifieke dingen horen hier niet thuis. Zorg ervoor dat je bewijzen volledig en correct zijn. Je mag gebruik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

1. Geef een (exacte) uitdrukking voor het aantal mogelijke vormen dat het pad met k toppen kan hebben dat tijdens een splaystap vervangen wordt door een boom.

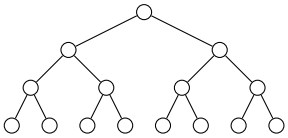
2. Geef voor elk van de onderstaande bomen en splaygroottes een reeks van toevoeg- en opzoekbewerkingen die resulteren in een boom isomorf aan deze boom, of bewijs dat dit niet kan. Let op: er mogen geen verwijderbewerkingen gebruikt worden.



splaygrootte 3



splaygrootte 4



splaygrootte 7

3. Bepaal en bewijs de gemiddelde complexiteit van een bewerking in een reeks van n bewerkingen op een initieel lege semi-splayboom met splaygrootte 7.

4. Geef een zo efficiënt mogelijk algoritme om een splaypad met k toppen te vervangen door een perfect gebalanceerde deelboom.

5. Bepaal en bewijs de complexiteit van het algoritme uit de vorige vraag.

4 Implementatie

We verwachten een klasse `SemiSplayTree` in de package `semisplay` die de onderstaande interface `SearchTree` implementeert. Deze klasse moet een constructor hebben die één `int` k als argument neemt en dan een lege zoekboom aanmaakt met splaygrootte k.

```
1  package semisplay;
2
3  public interface SearchTree<E extends Comparable<E>> extends Iterable<E> {
4
5      /** Voeg de gegeven sleutel toe aan de boom als deze er nog niet in zit.
6        * @return true als de sleutel effectief toegevoegd werd. */
7      boolean add(E e);
8
9      /** Zoek de gegeven sleutel op in de boom.
10       * @return true als de sleutel gevonden werd. */
11     boolean contains(E e);
12
13     /** Verwijder de gegeven sleutel uit de boom.
14       * @return true als de sleutel gevonden en verwijderd werd. */
15     boolean remove(E e);
16
17     /** @return het aantal sleutels in de boom. */
18     int size();
19
20     /** @return de diepte van de boom. */
21     int depth();
22
23 }
```

Listing 1: SearchTree.java specificeert de interface voor zoekbomen.

De implementatie van de methode `depth` mag deze gewoon berekenen op het moment dat de methode wordt aangeroepen. Dit mag dus lineair zijn in het aantal sleutels in de boom.

De iterator moet de sleutels doorlopen in stijgende volgorde. Bij het oproepen van `depth` en het gebruik van de iterator wordt er niet gesplayed.

Je kan de klasse `SemiSplayTree` dus als volgt gebruiken:

```java
SearchTree<Integer> tree = new SemiSplayTree(7);
tree.add(5); // returns true
tree.remove(2); // returns false
tree.size(); // returns 1
tree.depth(); // returns 0
for (Integer value : tree) {
    System.out.println(); // prints 5
}
```

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.

4.1 Testen en optimalisatie

Implementeer JUnit 4 tests in de package `test` om je implementatie op correctheid te testen en zorg dat je alle onderdelen van je implementatie goed test. Schrijf je testen zo vroeg mogelijk tijdens het project zodat je geen tijd verliest met het optimaliseren van een foute implementatie. Houd er rekening mee dat je veel punten kan verliezen als er nog fouten in je implementatie zitten, dus test uitvoerig.

Als je programma correct is, kan je het beginnen optimaliseren. Voer experimenten uit om kritieke punten te vinden, en probeer die efficiënter te implementeren. Controleer ook of je optimalisaties effect hebben.

Vergelijk de prestaties van semi-splaybomen met verschillende splaygroottes op verschillende reeksen bewerkingen. Probeer de resultaten te verklaren.

5 Verslag

Schrijf een verslag van dit project. Beantwoord eerst de theoretische vragen en beschrijf dan je implementaties, waaronder mogelijke optimalisaties die je hebt doorgevoerd. Beschrijf hoe je de verschillende experimenten hebt gedaan. Voeg de resultaten van je experimenten op overzichtelijke wijze toe. Wat kun je op basis van de experimenten concluderen? Komen de resultaten overeen met je verwachtingen?

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 je 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 27 oktober 2019 om 23u59 moet je implementatie in staat zijn om te werken als zoekboom, maar hoeft het splayen nog niet noodzakelijk geïmplementeerd te zijn. De code hiervoor en de antwoorden op vraag 1 en 2 moeten ingediend zijn tegen deze deadline.
2. We verwachten een volledig project tegen zondagavond 24 november 2019 om 23u59.

Code en verslag worden elektronisch ingediend. Van het verslag van het **tweede** indienmoment verwachten we ook een **papieren versie**. Deze bezorg je aan de studenten ten laatste op vrijdag 29 november 2019. Bij voorkeur gebeurt dit tijdens de oefeningenles, maar het kan ook afgegeven worden in de respectievelijke bureau's van de assistenten.

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 je verslag in PDF-formaat.
- De package `semisplay` bevat minstens de klassen die hierboven gespecificeerd zijn. Alle code behalve de testen bevindt zich in deze package.
- De package test bevat alle JUnit-tests en experimenten.

Bij het indienen wordt gecontroleerd of je 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 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** en resulteren dus in **nul punten voor het project**.

- 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.
- Als een van de twee deadlines gemist wordt, zal het project niet meer verbeterd worden: 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. **De minimumstraf hiervoor is een nul voor het volledige vak en zwaardere straffen zijn mogelijk en gebruikelijk.**