Algoritmen en Datastructuren II

Project Zeepbelbomen

Deadlines: 1 en 29 november 2015

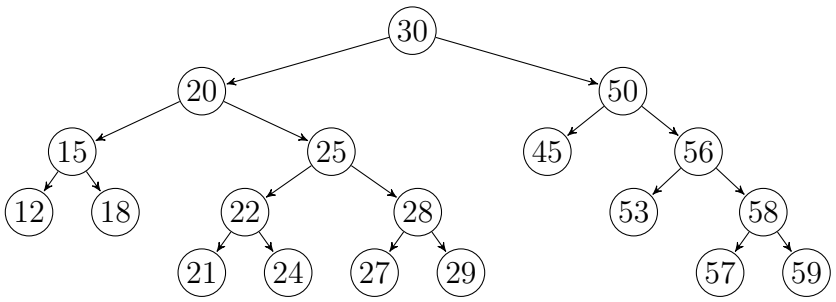
UNIVERSITEIT GENT

1 Inleiding

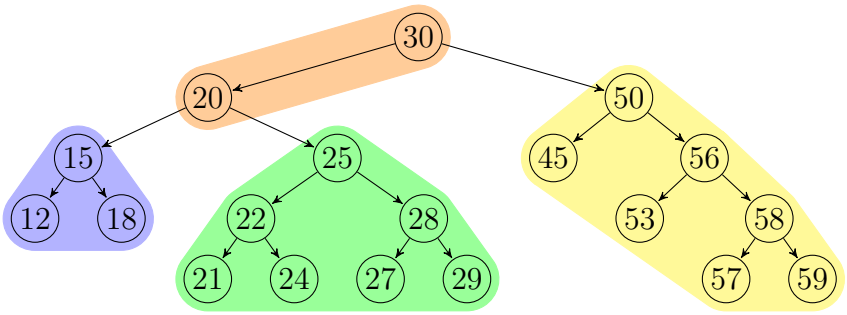
Tot dusver hebben wij boomstructuren gezien waarbij een boom gedefinieerd is als een worteltop, die een of meer sleutels bevat met bijbehorende verwijzingen naar kindtoppen, waarbij de boom al of niet onderhevig kan zijn aan balanceringsvoorwaarden. In dit project gaan wij de eigenschappen onderzoeken van bomen die (denkbeeldig, virtueel) gepartitioneerd zijn in zeepbellen, waarbij elke zeepbel een niet-lege binaire boom bevat van tenminste 1 en ten hoogste k sleutels, waarbij $k \geq 2$ een op te geven constante is. Wij noemen deze structuren *zeepbelbomen*. Voor een zeepbelboom zijn de voor binaire bomen gebruikelijke bewerkingen als toevoegen, opzoeken en verwijderen te definiëren.

2 Gebalanceerde Zeepbelbomen

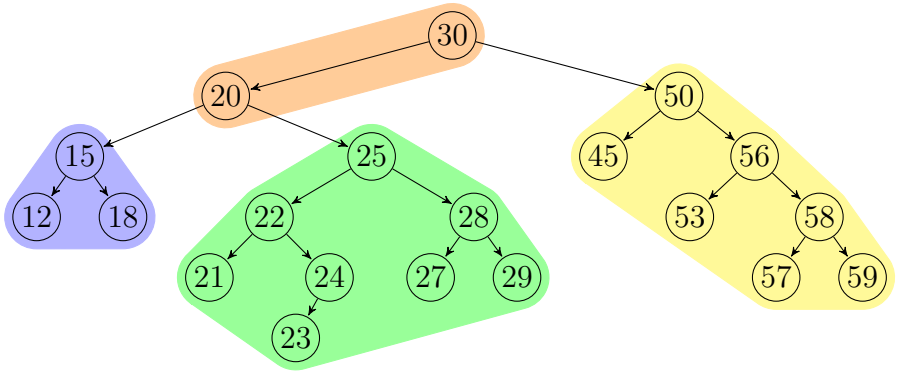
Een *wortelzeepbel* is de enige zeepbel die de wortel van de binaire boom bevat. Een *bladzeepbel* is een zeepbel die een top van de binaire boom bevat die tenminste één leeg kind heeft. Bekeken op het niveau van de zeepbellen, is elke zeepbelstructuur altijd perfect gebalanceerd. Meer precies: elk pad van de wortelzeepbel naar een bladzeepbel bevat evenveel zeepbellen. De binaire bomen binnen een zeepbel kunnen wel ongebalanceerd zijn. We illustreren zeepbelbomen uitgaande van de volgende binaire boom:



Op deze binaire boom kunnen wij verschillende zeepbelbomen voorstellen. Laten wij bijvoorbeeld eens een zeepbelboom voor $k = 7$ nemen. Voor bovenstaande boom zijn daarvoor vele (hoeveel?) configuraties mogelijk, bijvoorbeeld de volgende:



Als we een sleutel toevoegen dan proberen we die toe te voegen aan een bestaande zeepbel. Laten wij bijvoorbeeld eens kijken wat er gebeurt als we sleutel 23 toevoegen. Deze wordt volgens de definitie van de binaire boom toegevoegd als linker kind van sleutel 24 in de groene zeepbel. Nu is er echter een probleem ontstaan, omdat daardoor de zeepbelcapaciteit van $k = 7$ is overschreden:

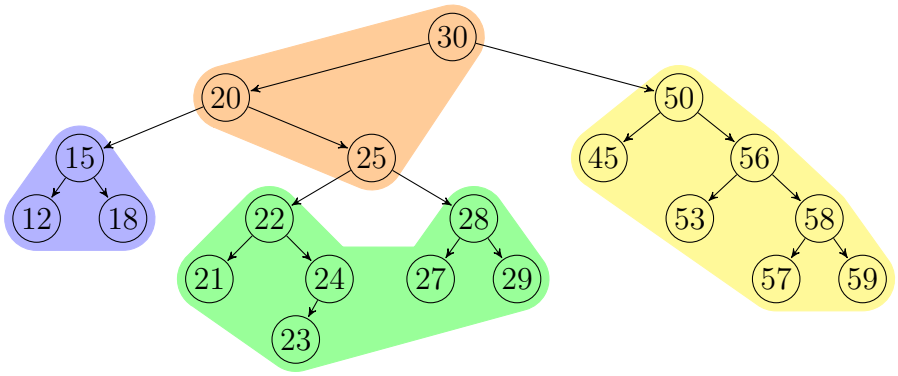


3 Balanceringsvarianten

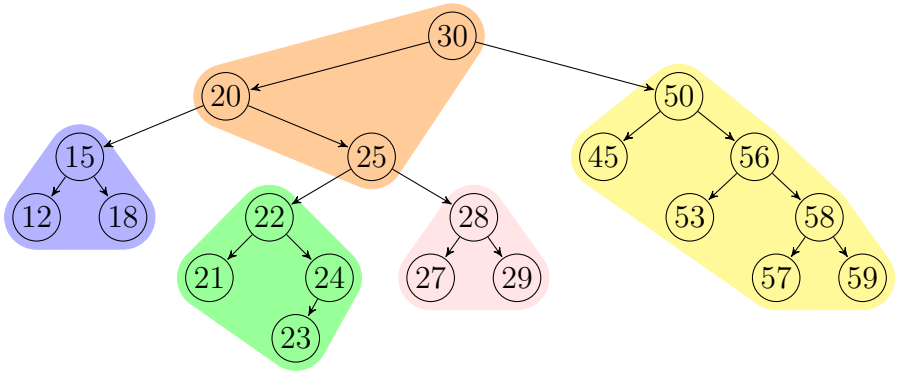
Als een zeepbel te vol zit, dan zijn er verschillende manieren om te herbalanceren:

1. Altijd zo weinig mogelijk wijzigingen aanbrengen in de onderliggende binaire boom.
2. Het tegendeel van 1.: de middelste sleutel gaat omhoog. De twee helften worden daarna zo goed mogelijk gebalanceerd. Een vraag kan zijn: wat is de middelste sleutel bij een even aantal sleutels? Als k even is dan is *na* toevoeging de middelste sleutel duidelijk. Als k oneven is dan is er *voor* toevoeging een middelste sleutel.
3. Duw meer dan één sleutel omhoog. Hiervoor zijn meerdere subvarianten denkbaar.

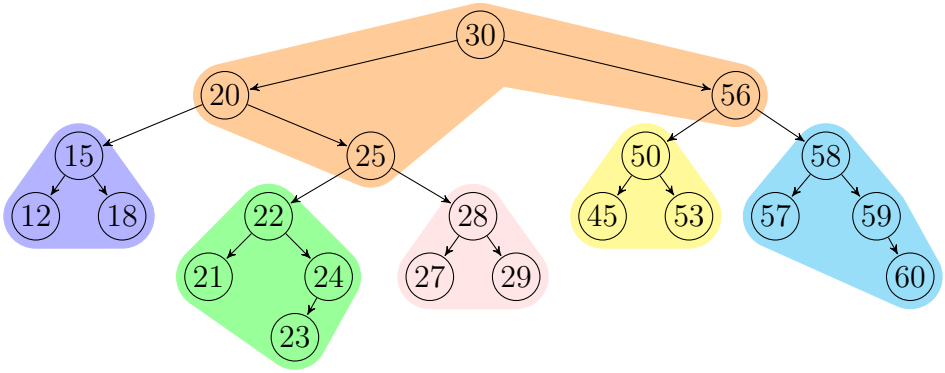
Als wij in ons voorbeeld balanceringsvariant 1 willen toepassen moeten wij een top uit de overvolle groene zeepbel halen en deze toevoegen aan zijn ouderzeepbel. De enige top die daarvoor in aanmerking komt is de worteltop van de groene zeepbel:



Hierdoor ontstaan er in de groene zeepbel twee worteltoppen. Dit is niet toegestaan. Daarom moeten wij de groene zeepbel in tweeën splitsen:



Als wij aan deze boom sleutel 60 toevoegen volgens balanceringsmethode 2 dan resulteert dat in de volgende boom:



4 Semi-splay Zeepbelbomen

Voor *semi-splay zeepbelbomen* gebruiken wij een iets gewijzigde definitie ten opzichte van de gebalanceerde zeepbelbomen uit sectie 2:

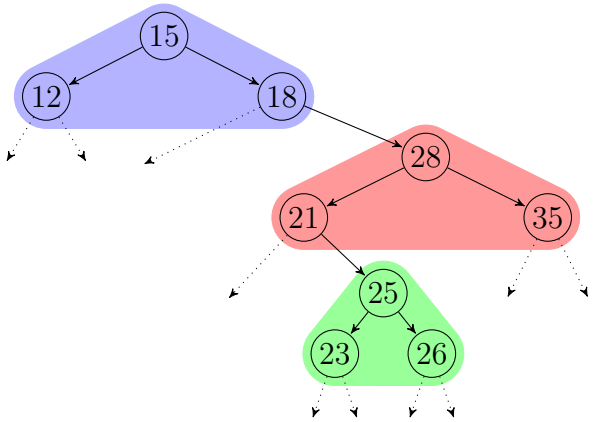
- Een zeepbel met precies k sleutels is intern altijd gebalanceerd en noemen wij een *complete zeepbel*. De diepte van de boom in een complete zeepbel is dus $\lfloor log\ k \rfloor$.

- Een zeepbel die kinderen in andere zeepbellen heeft, is een *interne zeepbel*. Interne zeepbellen zijn altijd compleet.
- Een *eindzeepbel* heeft geen kinderen in andere zeepbellen en heeft $1..k$ sleutels. Als een eindzeepbel minder dan k sleutels heeft dan is deze *niet-compleet* en hoeft deze intern dus niet gebalanceerd te zijn.

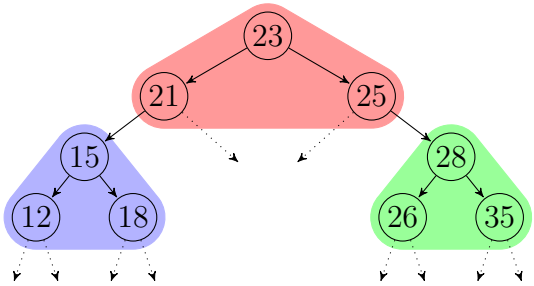
Voor semi-splay zeepbelbomen definiëren wij twee operaties:

Toevoegen: Als er bij toevoeging van een sleutel aan een eindzeepbel met $k-1$ sleutels een complete zeepbel ontstaat dan moet deze gebalanceerd worden. Daarna blijft deze zeepbel voorgoed compleet en gebalanceerd.

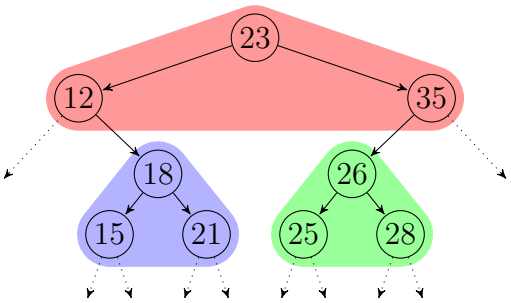
Opzoeken: Bij het opzoeken van een sleutel wordt het pad waarlangs wordt gezocht gebalanceerd volgens de regels van semi-splay op het niveau van de zeepbellen, *voor zover ze compleet zijn*. Zeepbellen met minder dan k sleutels worden nooit gesplayd. Wij splayen steeds volgens de *bottom-up* methode uit de cursusnota's. Het semi-splayen kan niet domweg uitgevoerd worden op de zeepbellen zonder naar de sleutels in de zeepbellen te kijken. Beschouw bijvoorbeeld de splay-stap waarbij de volgende drie zeepbellen betrokken zijn:



De splay-stap voor zeepbelbomen is als volgt gedefinieerd. We vervangen de deelboom gevormd door alle toppen in deze 3 zeepbellen door een boom die gepartitionieerd is in 3 zeepbellen en op niveau van de zeepbellen diepte 1 heeft. Hierbij mogen de sleutels van zeepbel veranderen, mits deze zeepbellen opnieuw compleet en gebalanceerd zijn. Er zijn verschillende manieren mogelijk om dit te doen. Maak zelf een goed beredeneerde keuze en documenteer deze in het verslag. Een eerste mogelijke vervanging voor het voorbeeld hierboven zou kunnen zijn:



Het volgende voorbeeld is echter ook een mogelijke vervanging.



Voor semi-splay zeepbelbomen is er geen verwijderoperatie gedefinieerd.

5 Opdracht

Beantwoord onderstaande theoretische vragen. Implementeer de verschillende varianten zeepbelbomen volgens sectie 7. Let daarbij heel goed op complexiteit en efficiëntie. Voer experimenten uit volgens sectie 8 en schrijf een verslag volgens sectie 9.

6 Theoretische vragen

Opgave 1: Toon aan voor gebalanceerde zeepbelbomen: geen top in een bladzeepbel heeft een kind in een andere zeepbel.

Opgave 2: Wat is voor gebalanceerde zeepbelbomen de maximale diepte van een k-zeepbelboom met n sleutels? (Gebruik geen O-notatie, maar wees concreet.) Deze vraag kan op twee niveau's beantwoord worden:

a. Gemeten in zeepbellen.

b. Gemeten in toppen van de ten gronde liggende binaire boom.

Opgave 3: Analyseer voor gebalanceerde zeepbelbomen de verschillende mogelijkheden om te herbalanceren en beargumenteer daarbij zorgvuldig wat de complexiteit is:

a. Voor toevoegen.

b. Voor opzoeken.

Opgave 4: Bepaal voor semi-splay zeepbelbomen zowel de slechtste-geval complexiteit als de geamortiseerde complexiteit.

a. Voor toevoegen.

b. Voor opzoeken.

7 Implementatie

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

- Een `Zeepbelboom` is een `Collection`. Dit stelt ons onder andere in staat om in *inorde*[1] te itereren over alle sleutels van een zeepbelboom. Overweeg hierbij dat een iteratie ook vroegtijdig afgebroken kan worden; dan hoeft er dus slechts een beperkt deel van de boom bezocht te worden. Bij semi-splay zeepbelbomen hoeft er tijdens het itereren niet gesplayd te worden.

- `Zeepbelboom` bevat tenminste een methode `zeepbelIterator`, die een iterator teruggeeft, waarmee we in inorde kunnen itereren over de zeepbellen. Bijvoorbeeld:

```
System.out.println("boom met wortel "
                   + zeepbelboom.getWortelSleutel());
Iterator<Zeepbel<Key>> iter = zeepbelboom.zeepbelIterator();
for (int zeepbelCount = 0; iter.hasNext(); ++zeepbelCount) {
    Zeepbel<Key> zeepbel = iter.next();
    System.out.println("zeepbel " + zeepbelCount +
                       " met wortel " + zeepbel.getWortelSleutel());
    for (Key key : zeepbel) {
        System.out.println("sleutel " + key.toString());
    }
}
```

  Het is essentieel voor onze correctheidstesten dat deze code probleemloos werkt. De methode `getWortelSleutel` geeft de waarde van de sleutel in de top van een zeepbelboom, respectievelijk van een zeepbel.

- Een `Zeepbel` erft van `Iterable` waardoor we in inorde kunnen itereren over de sleutels in een zeepbel. Misschien is voor jouw implementatie `Zeepbel` overbodig of ongewenst (bijvoorbeeld om redenen van efficiëntie!), maar in dat geval kan deze tijdens het itereren wel geconstrueerd worden door de zeepbel-iterator.

- Voor de verschillende balanceringsvarianten verwachten we klassen waarbij de naam `Zeepbelboom` gebruikt wordt onmiddelijk gevolgd door het nummer (1, 2 of 3) van de balanceringsvariant: `Zeepbelboom1, Zeepbelboom2`, etc. Als er subvarianten zijn geef die dan een extra suffixletter: `Zeepbelboom3a, Zeepbelboom3b`, etc. Gebruik voor semi-splay zeepbelbomen de naam `Zeepbelboom4`.

- Zeepbelboomvarianten erven van `Zeepbelboom`.

- Elke constructor van een zeepbelboom verwacht een integer parameter $k \geq 2$, die de capaciteit van de zeepbellen bepaalt.

Voorzie je code van voldoende commentaar. Elke methode behoeft een javadoc header. Geef bij elke niet-triviale methode in de javadoc header expliciet aan wat daarvan de complexiteit is en maak duidelijk waarom dat zo is. Implementeer JUnit testen om jouw implementatie op correctheid te testen.

[1] Zie het cursusboek van AD1, sectie 9.2.2, pagina 184 of https://en.wikipedia.org/wiki/Traversal

8 Experimenten

Bedenk en beschrijf zelf relevante experimenten om de verschillende zeepbelboomvarianten met elkaar te vergelijken. Voer deze experimenten ook uit.

9 Verslag

Schrijf een verslag van dit project. Het verslag begint met jouw naam, de naam van dit project en de datum. Beantwoord daarna eerst de theoretische vragen. Beschrijf dan jouw implementaties, waaronder mogelijke optimalisaties die je hebt doorgevoerd. Analyseer en vergelijk de varianten. Bespreek de mogelijke voor- en nadelen van deze varianten. Gebruik daarbij steeds de juiste wiskundige notatie.

Beschrijf de opzet en parameters van de experimenten. Geef de meetresultaten in overzichtelijke grafieken weer en bespreek ze grondig. Wat kun je op basis van de experimenten concluderen? Komen je resultaten overeen met je verwachtingen? Probeer steeds om al je resultaten te verklaren.

10 Beoordeling

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

- Worden de theoretische vragen goed beantwoord? Dit weegt zwaar.
- Is de implementatie volledig? Is zij correct?
- Is er zelf nagedacht? Is alles eigen werk?
- Hoe is er getest op correctheid?
- Zijn de experimenten goed opgezet? Tonen zij de verschillen in performantie tussen de varianten goed aan? Zijn de parameters voor de datasets goed gekozen?
- Is het verslag toegankelijk, duidelijk, helder, verzorgd, ter zake en objectief?

11 Deadlines

Er zijn twee indienmomenten:

1. Zondagavond 1 november 2015 om 23 uur 59 moet een implementatie van de varianten voor gebalanceerde zeepbelbomen ingeleverd worden en een verslag.pdf met daarin de antwoorden op de theoretische opgaven 1, 2a en 2b.
2. Een volledig project zondagavond 29 november 2015 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.

12 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 subdirectory zeepbelboom bevat de Java broncode voor zeepbelbomen in package `zeepbelboom`. Maak *geen* andere subpackages of subdirectories aan.
- De subdirectory test bevat alle JUnit-testen en alle meetcode.

13 Algemene richtlijnen

- Zorg ervoor dat alle code compileert met Oracle Java 8 update 60 of hoger.
- Compilatie met de compiler-optie -Xlint:all mag geen warnings geven.
- 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 de deadline 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 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.