Project Zeepbelbomen

Rien Maertens
2de Bachelor Informatica

10 november 2015

Theoretische Vragen

Opgave 1. *Geen top in een bladzeepbel heeft een kind in een andere zeepbel.*

Bewijs. Stel we hebben een bladzeepbel α met een top T die een kind K heeft in zeepbel β. Neem n het aantal zeepbellen dat het pad van zeepbel α naar de wortel bevat. We kunnen de volgende twee gevallen onderscheiden:

Geval 1 Zeepbel β bevat een top zonder kinderen en is dus een bladzeepbel. Het pad van de wortel van de boom naar β gaat door α. Dit pad bevat dus $n+1$ zeepbellen, wat in tegenstrijd is met het gegeven dat het pad van de wortel naar alle bladzeepbellen evenveel zeepbellen bevat. Het gestelde is dus vals. Het is onmogelijk voor een bladzeepbel om een kind te hebben in een andere zeepbel.

Geval 2 Zeepbel β is geen bladzeepbel, maar bevat wel een pad naar een top zonder kinderen. Deze top is onderdeel van bladzeepbel γ. Het pad van γ naar de wortel van de zeepbelboom bevat $n+k$ zeepbellen, met k het aantal zeepbellen in het pad tussen β en γ. Dit is opnieuw in tegenstrijd met het gegeven dat het pad van de wortel naar alle bladzeepbellen hetzelfde aantal zeepbellen bevat. Ook in dit geval kan een bladzeepbel geen kinderen in andere zeepbellen hebben.

□

Opgave 2. *Wat is de maximale diepte van een k-zeepbelboom met n sleutels?*

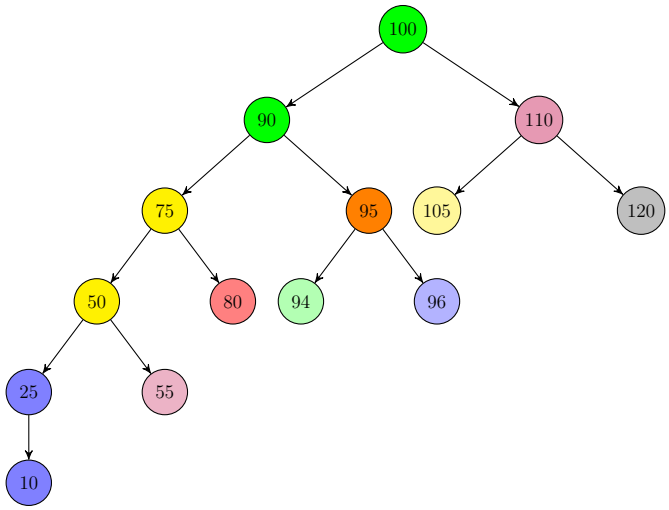
a. Gemeten in zeepbellen.

Stel d_z de diepte in zeepbellen van een zeepbelboom. d_z bereikt een maximum wanneer iedere zeepbel maar één top omvat, m.a.w. wanneer het pad van de wortel naar een blad evenveel zeepbellen als toppen bevat. De maximale diepte d_z in zeepbellen van een k-zeepbelboom met n sleutels is dus gelijk aan de maximale diepte in toppen van een complete binaire boom met n toppen. Dit is dus:

$$d_z \leq \lfloor \log_2 n \rfloor$$

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

Stel d_t de diepte in toppen van een zeepbelboom. d_t bereikt een maximum wanneer we één lang pad van zeepbellen maken waar elke top van een zeepbel maximum één kind in dezelfde zeepbel bevat. De andere zeepbellen die kinderen zijn van dit lange pad zijn terug zeepbellen met grootte één. Dit ziet er dan als volgt uit:



Het valt op dat wanneer we een niveau zeepbellen toevoegen, we het dubbele aantal toppen moeten toevoegen van de vorige zeepbelboom met een niveau lager. Voor n het aantal toppen en z het aantal niveaus in zeepbellen geldt:

$$n \geq \sum_{i=1}^{z} k2^{z-1} = k(2^z - 1)$$

We lossen dit op naar z:

$$z \leq \log_2 (\frac{n+k}{k})$$

De maximale diepte d_t van deze zeepbelboom in toppen is gelijk aan het aantal niveaus zeepbellen vermenigvuldigd met het maximale aantal toppen per zeepbel. Dus krijgen we dat $d_t = k \cdot z$. Wanneer voegen dit alles samen tot één formule:

$$d_t \leq k \cdot \log_2 (\frac{n+k}{k})$$

Waar d_t gelijk is aan de maximale diepte in toppen.

Implementatie

Hieronder volgt er een korte uitleg over de interne structuur en algoritmes die ik heb gebruikt voor de implementaties van de zeepbelbomen.

1 Binaire boom

1.1 Top

Mijn binaire boom bestaat intern uit toppen die hun ouder, linker en rechterkind bijhouden. Om problemen met deze links te vermijden kan de verwijzing naar de ouder van een top enkel ingesteld worden wanneer deze top als kind wordt aangeduid van een andere top met de methodes `setRightChild()` en `setLeftChild()`. Omdat de wortel van de binaire boom natuurlijk geen ouder heeft is er ook een methode die deze verwijzing op `null` zet. Een top bevat ook een booleaanse waarde `removed` die functioneert als grafsteen. Daarnaast bevat iedere top ook een verwijzing naar de zeepbel waartoe die top behoort.

1.2 Zeepbel

Om de toppen in groepjes in te delen bestaat de klasse `Zeepbel`. Deze houdt enkel zijn wortel bij en het aantal toppen dat deze bezit. Een zeepbel heeft

ook de methodes `topAdded()` en `topsRemoved()` die de zeepbel waarschuwen wanneer er toppen bijgekomen of weggevallen zijn. De methode `topAdded()` heeft ook een boolean terug die zegt of de zeepbel te groot is en verkleind moet worden.

1.3 Zeepbelboom

In de abstracte klasse Zeepbelboom zit de gemeenschappelijke functionaliteit die alle zeepbelboom-implementaties delen. Het is enkel de methode `shrinkBubble()` die verschilt tussen de verschillende zeepbelbomen. Deze methode zorgt ervoor dat een overvolle zeepbel gesplitst wordt.

2 Implementaties zeepbelboom

2.1 Zeepbelboom1

De eerste zeepbelboom lost een overvolle zeepbel op door zo weinig mogelijk veranderingen te brengen aan de interne structuur van de boom. Wanneer de wortel van de overvolle zeepbel beide kinderen in diezelfde zeepbel heeft dan worden er geen wijzigingen aangepast aan de interne structuur: deze twee kinderen vormen de wortels voor twee nieuwe zeepbellen en de top wordt toegevoegd aan de bovenliggende zeepbel. Als de wortel van de zeepbel maar één kind in dezelfde zeepbel bevat worden de eerste drie toppen geroteerd zodat de wortel wel twee kinderen in dezelfde zeepbel heeft en er kan gesplitst worden zoals normaal.

2.2 BalancingBubbleTree

Dit is een superklasse voor alle zeepbelbomen die een zeepbel moeten kunnen balanceren. Deze balancering gebeurt door de methode `balanceBubble()`. Dit gebeurt door eerst alle toppen in inorde te overlopen en in een lijst te plaatsen. Vervolgens wordt van deze lijst een nieuwe binaire boom opgebouwd door de recursieve methode `listToTree()`. Dit balanceren gebeurt in $\Theta(n)$ tijd als we het aantal te balanceren toppen als n kiezen. Maar er wordt enkel in één zeepbel gebalanceerd en een zeepbel heeft een maximaal aantal toppen, dus werkt deze methode in constante tijd.

2.3 Zeepbelboom2

Voor de zeepbel gesplitst wordt wordt deze eerst gebalanceerd met de methode uit `BalancingBubbleTree`. Daarna wordt op dezelfde manier gesplitst als `Zeepbelboom1`.

2.4 Zeepbelboom3

Deze zeepbel werkt op dezelfde manier als `Zeepbelboom2` echter worden er hier zoveel mogelijk toppen aan de bovenliggende zeepbel toegevoegd. Deze zeepbelboom kan geconfigureerd worden: er kan gekozen worden om een maximum te zetten op het aantal omhoog te duwen toppen en er kan ingesteld worden dat er geprobeerd wordt om net genoeg toppen toe te voegen aan de bovenliggende zeepbel dat die verplicht wordt om te splitsen.

3 Functies

3.1 Iterator

Een top heeft een methode `traverseInorder()` waarmee er in inorde kan geïtereerd worden over zijn kinderen. Aan deze methode worden twee functionele interfaces meegegeven: een `consumer` waaraan het volgende element in de iteratie wordt aan doorgegeven en een `predicate` die extra restricties kan opleggen over welke toppen er geïtereerd mag worden (bijvoorbeeld enkel over de toppen van een bepaalde zeepbel. Alle iterators maken van deze methode gebruik.

3.2 `Zeepbelboom.find()`

De drie basisoperaties: `add()`, `contains()` en `remove()` maken gebruik van de methode `find()` om een top op te zoeken. Deze methode werkt door binair te zoeken naar een `Top` tot de juiste top gevonden is of tot er `null` werd gevonden, waarna de top aan de juiste consumer wordt doorgegeven.

3.3 `Zeepbelboom.add()`

Er wordt eerst met `find()` gezocht naar de juiste top, als de top werd gevonden dan gebeurt er niets en wordt `false` teruggegeven. Als de top niet werd

gevonden wordt top toegevoegd op de plaats waar `null` werd tegengekomen en wordt de betreffende zeepbel verwittigd van deze toevoeging, waarna er eventueel kan gesplitst worden als deze zeepbel overvol is.

3.4 `Zeepbelboom.contains()`

Hier wordt enkel met `find()` gezocht en teruggegeven of de top gevonden werd of niet.

3.5 `Zeepbelboom.remove()`

De te verwijderen top wordt opnieuw gezocht met `find()`. Wanneer de top gevonden werd wordt er een grafsteen geplaatst door de `removed` waarde op `true` te zetten. Als er een kritiek aantal grafstenen wordt bereikt (standaard 50%) wordt de boom opnieuw opgebouwd zonder de grafstenen.

Ik was begonnen met een implentatie die de toppen echt uit de zeepbelboom verwijdert, maar helaas zitten er nog wat bugs in de code en had ik geen tijd meer om deze te debuggen. Dus heb ik een werkende implementatie met grafstenen gemaakt.