Les 2: Binaire zoekbomen

Toegepaste wiskunde 3

Inleiding

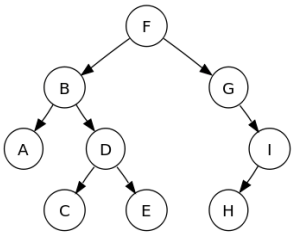
Download het bestand `TW3_les2.jar` van Toledo. Importeer dit bestand in Eclipse alsvolgt:

```
File > Import... > General > Existing Projects into Workspace
            > Next > Select archive file > Finish
```

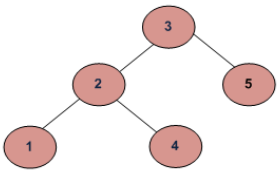
Oefening 1

Ga voor elk van onderstaande bomen na of ze een binaire zoekboom zijn.

a)



b)

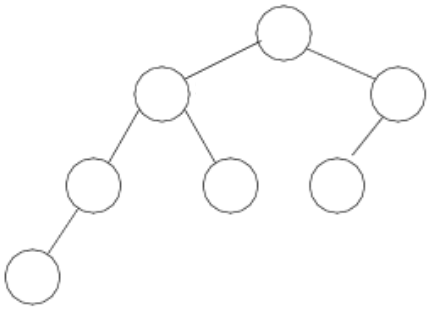


Oefening 2

Kan je een BST doorlopen met 1 van de 4 strategieën besproken in les 1 (pre-order, in-order, post-order of level-order) zodanig dat de knopen worden bezocht van klein naar groot?

Oefening 3

a) Op hoeveel verschillende manieren kan je de getallen 3 tot en met 9 in onderstaande knopen invullen met als resultaat een BST? Teken deze verschillende mogelijkheden.



b) Kan je de getallen 3 tot en met 9 opslaan in een andere BST waarvoor de worst-case tijdscomplexiteit van de `lookup` methode kleiner is? Zo ja, geef deze BST. Zo nee, leg uit waarom niet.

Oefening 4

a) Bestudeer de klassen `BinaryNode` en `BinaryTree` in het tweede package (`binaryTree`) in de `src` folder.

Deze klassen zijn quasi identiek aan de implementatie die je tijdens de eerste les gekregen hebt. Een eerste verschil is dat er nu vanuit gegaan wordt dat de knopen van de boom integers zullen bevatten. Hierdoor kan generics achterwege worden gelaten en wordt de code eenvoudiger. Een tweede verschil is de toevoeging van een methode `isBST` aan de `BinaryTree` klasse. Deze methode gaat na of een binaire boom een binaire zoekboom is.

b) Schrijf een correcte en efficiënte implementatie voor de `isBST` methode.

c) Test je implementatie aan de hand van de 2 testvoorbeelden die je vindt in de `main` functie van de `BinaryTreeDriver` klasse.

Professionele software

De tot hiertoe ter beschikking gestelde code werd om didactische redenen zo beknopt mogelijk gehouden. Wees je er wel van bewust dat de gegeven code hierdoor niet van professioneel niveau is.

- Het is nogal simplistisch om de verantwoordelijkheid om een binaire boom te construeren volledig aan de gebruiker over te laten (zie de twee testvoorbeelden in `BinaryTreeDriver`). Een gebruiker kan gemakkelijk een lus creëren waardoor het geheel van knopen geen boom vormt. Hij zou ook perfect de ouder van een knoop kind kunnen maken van diezelfde knoop. Met data encapsulation is dus onvoldoende rekening gehouden.
- Het is ook niet de verantwoordelijkheid van een `main` functie om een binaire boom te construeren. (Schending van het Single Responsibility Principe (SRP) — zie OOO). Het is beter om dit over te laten aan een andere (Factory) klasse.
- In professionele code zou men ook niet verschillende implementaties van een bepaalde functie in 1 klasse stoppen en deze met cijfers nummeren zoals we deden in les 1 (bijvoorbeeld `printPreorder` en `printPreorder2`). In feite gaat het hier om verschillende strategieën om een probleem (het printen van de waarden in pre-order volgorde) op te lossen. Deze kan je beter in aparte klassen stoppen (Strategy patroon — zie OOO).

Nu je echter al wat meer vertrouwd bent met bomen, zullen we geleidelijk aan de complexiteit opvoeren. In de hierna ter beschikking gestelde code zal wel meer rekening worden gehouden met goed OO design.

Oefening 5

a) Bestudeer de `BinarySearchTreeFactory` klasse in het eerste package (`binarySearchTree`) in de `src` folder.

Deze klasse is verantwoordelijk voor het construeren van nieuwe objecten van de `BinarySearchTree` klasse. Voorlopig bevat de Factory klasse slechts 1 eenvoudige methode waarmee een klein testvoorbeeld kan worden gegenereerd.

b) Bestudeer de implementatie van de `BinarySearchTree` klasse.

Merk op dat `BinaryNode` nu als een private klasse wordt geïmplementeerd binnen de klasse `BinarySearchTree`. Dit wordt een geneste klasse genoemd. De klasse `BinaryNode` is een "helper klasse"die enkel relevant is voor de `BinarySearchTree` klasse dus is het logisch om de twee samen te houden. Bovendien wordt deze geneste klasse private gemaakt omdat het voor een

gebruiker niet relevant is om te weten hoe een BST binnenin precies is opgebouwd. Dit is dus een goed voorbeeld van data encapsulation. Om data encapsulation verder te kunnen garanderen wordt er ook geen publieke `setRoot` methode voorzien.

Een gebruiker kan enkel operaties uitvoeren op een BST via enkele publieke methoden. Een eerste dergelijke methode is `lookup` waarmee snel een bepaald getal in de BST kan opgezocht worden (zie de slides van deze les). Een tweede methode is `insert` waarmee een gegeven getal aan de BST kan toegevoegd worden.

c) Implementeer de `insert` methode in de `BinarySearchTree` klasse. (Hint: een nieuwe waarde moet toegevoegd worden waar je hem zou gevonden hebben als je de `lookup` methode zou uitvoeren. Als je het niet op die plaats toevoegt, zal je het ook later niet kunnen terugvinden.)

d) Om je implementatie te controleren, kan je gebruik maken van de gegeven `BinarySearchTreeDriver` klasse. Je hoeft enkel nog drie maal eenzelfde regel code toe te voegen aan de `main` methode waarmee je de waarden van een BST in oplopende volgorde kan afprinten. Maak gebruik van je antwoord op oefening 2 en kopieer de bijhorende implementatie van les 1 naar de `BinarySearchTree` klasse.

Opdracht van de week

Indieninstructies

Stuur enkel je bestand `BinarySearchTree.java` (niet je volledige project!) met daarin o.a. je implementatie van de methode `insert` (zie oefening 5) voor aanstaande zondag, 20u, per email naar `mario.ausseloos@khleuven.be`. Vermeld als onderwerp van je email “TW3 opdracht2”.