1 Recursie

In dit OPO bestuderen we twee speciale datastructuren: bomen en grafen. Omdat er hiervoor nogal veel recursief geprogrammeerd moet worden, is het zinvol om aan dit principe wat extra aandacht te besteden. Hieronder vind je de oefeningen van de eerste twee lesweken.

Oefening 1.1

Implementeer een recursieve methode `fibonacci(int getal)` die het `getal`-de fibonacci-getal berekent (zie `https://nl.wikipedia.org/wiki/Rij_van_Fibonacci`). Vermoedelijk vind je een eenvoudige en korte versie die echter heel inefficiënt is. Kan je uitleggen waarom?

Oefening 1.2

Schrijf een recursieve methode `somCijfers(int getal)`. Uitvoer is de som van de cijfers van `getal`.

Voorbeeld: $234 \rightarrow 2+3+4=9$

Oefening 1.3

Schrijf een recursieve methode `keerOm(String str)`. Uitvoer is een string waarbij alle karakters van `str` in omgekeerde volgorde voorkomen.

Voorbeeld: `abcd` → `dcba`

Uitbreiding: Vind twee versies van het algoritme. In de eerste versie wordt de omgekeerde string van links naar rechts opgebouwd; in de tweede versie van rechts naar links.

Oefening 1.4

Implementeer een recursieve methode `countX(String str)`. Invoer is de string `str`. Uitvoer is het aantal keer dat de letter `x` voorkomt in `str`.

Zie ook http://codingbat.com/prob/p170371.

Oefening 1.5

Implementeer een recursieve methode `countHi(String str)`. Invoer is de string `str`. Uitvoer is het aantal keer dat de combinatie `hi` voorkomt in str.

Zie ook http://codingbat.com/prob/p184029.

Oefening 1.6

Implementeer een recursieve methode `changeXY(String str)`. Invoer is de string `str`. Uitvoer is een nieuwe string waarin elk voorkomen van `x` vervangen wordt door `y`.

Voorbeelden:

- `changeXY("codex")` → `"codey"`
- `changeXY("xxhixx")` → `"yyhiyy"`
- `changeXY("xhixhix")`→ `"yhiyhiy"`

Zie ook http://codingbat.com/prob/p101372

Oefening 1.7

Implementeer een recursieve methode `changePi(String s)`. Invoer is de string `s`. Uitvoer is een nieuwe string waarin elke deelstring `pi` vervangen wordt door '`3.14`'.

Voorbeelden:

- `changePi("xpix")` → `"x3.14x"`
- `changePi("pipi")` → `"3.143.14"`
- `changePi("pip")`→ `"3.14p"`

Zie ook http://codingbat.com/prob/p170924.

Oefening 1.8

We zeggen dat de logaritme met grondtal 2 van het getal 8 gelijk is aan 3 omdat $2^3 = 2 \cdot 2 \cdot 2 = 8$. De tweelog van 256 is 8 want $2^8 = 256$. Schrijf een recursieve functie `tweelog(int x)`. Je mag uitgaan van de veronderstelling dat `x` een macht van 2 is.

Oefening 1.9

Schrijf een recursieve methode `findMaximum(List<double> lijst)` die het grootste getal van `lijst` teruggeeft.

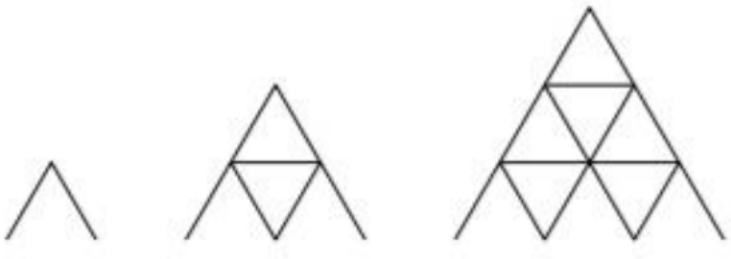
Oefening 1.10

Schrijf een recursieve methode `findSubstrings(String string)` die een lijst teruggeeft met alle mogelijke combinaties van de letters van `string`. Let op: Je hoeft geen rekening te houden met de volgorde van de letters. De combinatie `abc` beschouwen we gelijk aan de combinatie `cab`. Voorbeeld:

- Mogelijke combinaties van de letters abc zijn: [a, b, c, bc, ab, ac, abc]

Oefening 1.11

Schrijf een recursieve functie `aantalKaarten(int n)` die berekent hoeveel kaarten er nodig zijn voor een kaartenhuisje van `n` verdiepingen. Een kaartenhuisje wordt opgebouwd zoals getoond in figuur 1.1



Figuur 1.1 Tekening bij opgave 1.11

Voorbeeld:

- een kaartenhuisje van 1 verdieping = 2 kaarten
- een kaartenhuisje van 2 verdiepingen = 7 kaarten
- een kaartenhuisje van 3 verdiepingen = 15 kaarten
- een kaartenhuisje van 12 verdiepingen = 222 kaarten
- een kaartenhuisje van 20 verdiepingen = 610 kaarten

Meer oefenmateriaal nodig?
http://codingbat.com/java/Recursion-1
en http://codingbat.com/java/Recursion-2.

2

Binaire bomen

Inleiding

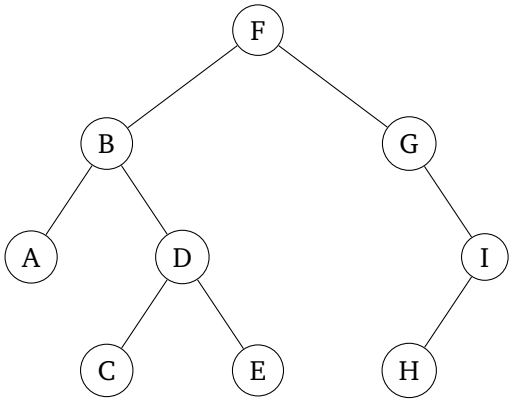
Download het bestand `BG_les3.zip` van Toledo. Importeer dit bestand in Eclipse als volgt:

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

In de `src` folder van het Java project `BG_Les3` vind je in de package domain de code van de klasse: `binaryTree` die alle code bevat die je zal nodig hebben bij het oplossen van onderstaande oefeningen en de opdracht van deze week.

Oefening 2.1

Bekijk de boom uit figuur 2.1



Figuur 2.1 Een (binaire?) boom

a) Is dit een binaire boom?

b) Wat is de wortel van deze boom?

c) Wat is de diepte van deze boom?

d) Is deze boom compleet?

e) Geef alle bladeren van deze boom.

f) Geef alle interne knopen van deze boom.

g) Hoeveel knopen bevat de linkersubboom? Teken deze subboom.

h) Schrijf de knopen van de gegeven boom (zie vorige bladzijde) op in de volgorde waarin ze bezocht worden bij een pre-order wandeling.

i) Schrijf de knopen van de gegeven boom (zie vorige bladzijde) op in de volgorde waarin ze bezocht worden bij een in-order wandeling.

j) Schrijf de knopen van de gegeven boom (zie vorige bladzijde) op in de volgorde waarin ze bezocht worden bij een post-order wandeling.

Oefening 2.2

Voor deze oefening duiken we in de code die je in Eclipse geïmporteerd hebt.

a) Bestudeer de implementatie van `BinaryTree<E>`. Stel vragen als er onduidelijkheden zijn.

b) Bestudeer de `main` functie in de `BinaryTreeDriver` klasse. Hierin wordt een boom aangemaakt waarmee de functie `printPreorder` wordt geïllustreerd. Deze functie is een recursieve implementatie om de waarden van de knopen in pre-order volgorde af te printen.

c) Teken de boom uit de driver klasse op papier.

d) Schrijf op papier de verwacht output bij het doorlopen van de boom in pre-order.

e) Test je verwachte output door de diver-klasse te runnen.

f) Vervang de code in de driver-klasse door de constructie van de boom uit oefening 1.

g) Run de code en controleer op die manier je antwoord op vraag h) van oefening 1.

Oefening 2.3

Van pre-order naar in-order ...

a) Implementeer volledig analoog aan `printPreorder` een recursieve implementatie `printInorder` om de waarden van de knopen in in-order volgorde af te printen.

b) Controleer je implementatie met behulp van je testvoorbeeld. Ga ook na of de uitvoer overeenkomt met je antwoord op vraag i) van oefening 1.

Oefening 2.4

. . . en naar post-order.

a) Implementeer volledig analoog aan `printPreorder` en `printInorder` een recursieve implementatie `printPostorder` om de waarden van de knopen in post-order volgorde af te printen.

b) Controleer je implementatie met behulp van je testvoorbeeld. Ga op die manier ook na of de uitvoer overeenkomt met je antwoord op vraag j) van oefening 1.

Oefening 2.5

Het doel van deze opdracht bestaat erin om een recursieve implementatie te schrijven voor het bepalen van het aantal knopen van een boom.

a) Ga voor de boom uit oefening 1 na dat het aantal knopen kan gevonden worden als volgt: 1 + als er een linkersubboom is: het aantal knopen van de linkersubboom + als er een rechtersubboom is: het aantal knopen van de rechterSubboom.

b) Gebruik het vorige idee om een recursieve implementatie `countNodes` te schrijven om het aantal knopen van een binaire boom te bepalen.

c) Controleer je implementatie van `countNodes` met behulp van je testvoorbeeld in de `main` functie.

Oefening 2.6

Het doel van deze opdracht bestaat erin om een recursieve implementatie te schrijven voor het bepalen van de diepte van een gegeven binaire boom.

a) Ga voor de boom uit oefening 1 na dat zijn diepte kan gevonden worden als 1 + het maximum van de diepte van de linker- en rechtersubboom van de boom.

b) Gebruik het vorige idee om een recursieve implementatie `getDepth` te schrijven om de diepte van een binaire boom te bepalen.

c) Controleer je implementatie van `getDepth` met behulp van je testvoorbeeld in de `main` functie.

Oefening 2.7

Schrijf een functie `isLeaf` om na te gaan of een boom een blad is. Dit wil zeggen dat de linkerdeelboom en de rechterdeelboom leeg zijn.

Oefening 2.8

Het doel van deze opdracht bestaat erin om een recursieve implementatie te schrijven voor het bepalen van het aantal bladeren van een boom.

a) Ga voor de boom uit oefening 1 na dat het aantal bladeren kan gevonden worden als volgt: 1 voor een boom die enkel 1 blad bevat en anders de som van het aantal bladeren van de linkersubboom als er een linkersubboom is en het aantal bladeren van de rechtersubboom als er een rechtersubboom is.

b) Gebruik het vorige idee om een recursieve implementatie `countLeaves` te schrijven om het aantal bladeren van een binaire boom te bepalen.

c) Controleer je implementatie van `countLeaves` met behulp van je testvoorbeeld in de `main` functie.

Oefening 2.9

Het doel van deze opdracht bestaat erin om een recursieve implementatie `getDataLeaves` te schrijven voor het bepalen van een lijst van datavelden van de bladeren van een boom. Controleer je implementatie met behulp van je testvoorbeeld en vergelijk de uitvoer met je antwoord op vraag e) uit oefening 1. De volgorde zal natuurlijk afhangen van het soort wandeling dat je gebruikt.

Oefening 2.10

Het doel van deze opdracht bestaat erin om een recursieve implementatie te schrijven voor het bepalen of een gegeven data-veld in de binaire boom voorkomt. Schrijf een methode `contains` die gegeven een data-veld true teruggeeft indien de boom een knoop bevat met het gegeven data-veld en false anders. Probeer in je `main` deze functionaliteit uit door vier keer de functie op te roepen met respectievelijk “D”, “H”, “F” en “Q” als parameter.

3

Binaire zoekbomen BST

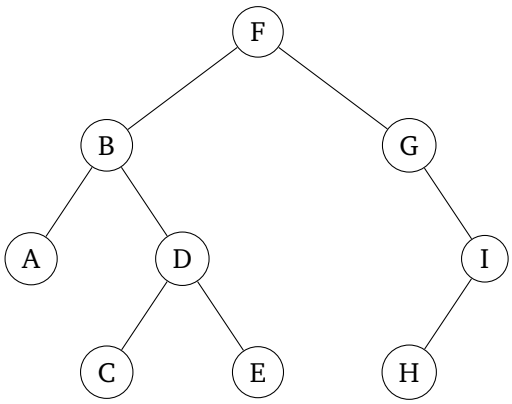
Inleiding

Download het bestand `BG_Les4.zip` van Toledo. Importeer dit bestand in Eclipse alsvolgt:

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

Oefening 3.1

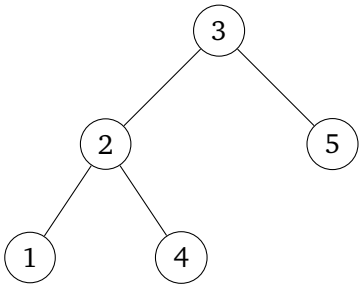
Gegeven twee binaire bomen in figuren 3.1 en 3.2. Ga voor elk van beide na of het een binaire zoekboom is (BST).



Figuur 3.1 Een binaire boom, maar is het ook een BST?

Oefening 3.2

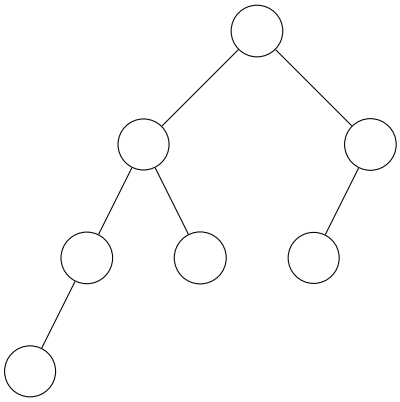
Kan je een BST doorlopen met 1 van de 3 strategieën besproken in les 3 (pre-order, in-order of post-order) zodanig dat de knopen worden bezocht van klein naar groot?



Figuur 3.2 Een binare boom met getallen in de knopen

Oefening 3.3

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.



Figuur 3.3 Getallen van 3 t.e.m. 9 invullen in de knopen

Oefening 3.4

Bestudeer de klasse `BinaryTree` in de package `domain` in de `src` folder. Een gebruiker kan enkel operaties uitvoeren op een BST via enkele publieke methoden. Een eerste dergelijke methode is `lookup` waarmee snel een bepaalde waarde in de BST kan opgezocht worden (zie de slides van deze les). Een tweede methode is `addNode` waarmee gegeven data aan de BST kan toegevoegd worden.

a) Teken op papier eerst de binaire zoekboom zoals die door de `main` methode in de klasse `BinarySearchTreeDriver` zal worden toegevoegd.

b) Implementeer de `addNode` methode in de `BinarySearchTree` klasse.

c) Om je implementatie te controleren run je de `BinarySearchDriver` klasse uit de `ui` package. Verwachte uitvoer: 3 4 5 6 7 8 9

Oefening 3.5

Het doel van deze oefening is een implementatie te maken van een methode die de grootste waarde uit de BST teruggeeft.

a) Implementeer de `searchGreatest` methode in de `BinarySearchTree` klasse.

b) Om je implementatie te controleren run je de `BinarySearchDriver` klasse uit de `ui` package. Verwachte uitvoer: 9

Oefening 3.6

Het doel van deze oefening is een implementatie te maken van een methode die de kleinste waarde uit de BST teruggeeft.

a) Implementeer de `searchSmallest` methode in de `BinarySearchTree` klasse.

b) Om je implementatie te controleren run je de `BinarySearchDriver` klasse uit de `ui` package.Verwachte uitvoer: 3

Oefening 3.7

Een volgende methode is `removeNode` waarmee gegeven data uit de BST zal verwijderd worden indien mogelijk. Eerst breiden we de boom een beetje uit met twee extra knopen.

a) Breid in de `main` methode van de klasse `BinarySearchTreeDriver` de gegeven boom uit met twee extra datavelden: 10 en 11. Teken de nieuwe boom.

b) Implementeer de `removeNode` methode in de `BinarySearchTree` klasse.

c) Verwijder de knoop met data-veld 9.

Oefening 3.8

Een probleem dat weggesnoeid moet worden . . .

a) Gebruik de `countNodes()` methode die je in vorige les maakte om het aantal knopen voor en na de verwijdering van de knoop met data-veld 9 te tellen. Je merkt ongetwijfeld iets eigenaardigs op: het verwijderen van de knoop heeft geen effect op het aantal knopen. We onderzoeken dat in de volgende deelvragen.

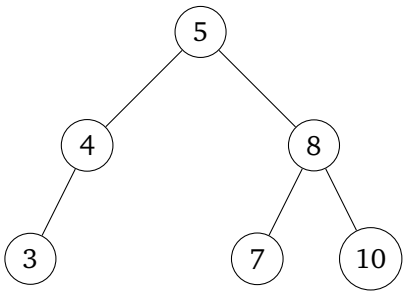
b) Teken de boom die je na het verwijderen van de knoop met waarde 9 kreeg. Eén van de bladeren van de boom heeft nu als data de waarde `null` gekregen. Met de in-orderwandeling kan je deze bladeren niet visualiseren. Eventueel maak je een alternatieve `printInOrderNull()` methode die de string “null” print als het dataveld als waarde `null` heeft.

c) Blaadjes met als datawaarde `null` laten we niet aan de boom staan. Tijd om te snoeien! Schrijf een methode `ruimOp()` die deze 'verdorde' blaadjes verwijdert. Pas ze toe op de boom na het verwijderen van de knoop met waarde 9 en laat zien dat het aantal knopen na de snoeibeurt wel degelijk ééntje verminderd is.

d) Verwijder tenslotte uit de boom data-veld 11 en data-veld 6 en ruim lege blaadjes op. De verwachte uitvoer van een in-order wandeling is dan: 3 4 5 7 8 10.

Oefening 3.9

De bedoeling van deze oefening is een methode te schrijven die gegeven een data-veld een pad teruggeeft van de wortel van de boom tot het dataveld indien mogelijk. We passen dit toe op de boom die je als resultaat na vorige oefening zou moeten bekomen. Bij wijze van controle: figuur 3.4 toont deze boom met zes knopen.



Figuur 3.4 BST na uitbreiding met knopen 10 en 11 en verwijdering van 9, 11 en 6

a) Implementeer de `getPath` methode in de `BinarySearchTree` klasse.

b) Als controle pas je de klasse `BinarySearchTreeDriver` aan zodat je drie keer de methode `getPath` oproept met als parameter respectievelijk 7, 4 en 8. De uitvoer moet dan respectievelijk zijn: bij 7: [5,8,7] ; bij 4: [5, 4] en bij 8: [5, 8]. Probeer de methode ook uit met als parameter 22. In dit geval moet de methode `null` als returnwaarde hebben.

4

Binary min-heap

Inleiding

Download het bestand `BG_Les6.zip` van Toledo. Importeer dit bestand in Eclipse alsvolgt:

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

Oefening 4.1

Gegeven twee binaire bomen in figuren 4.1 en 4.2 op pagina 14. Stel dat je intern een arrayimplementatie voor bomen gebruikt. Welke array zou je dan bekomen voor deze beide bomen?

Oefening 4.2

Een gegeven binaire min-heap bevat 1000 knopen. Intern wordt deze met een array voorgesteld.

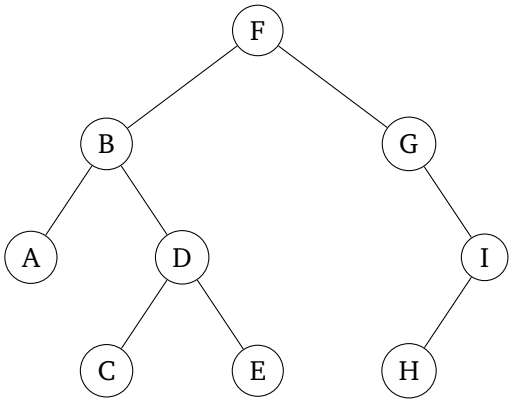
a) Waar vind je de ouder van de knoop die zich in de array op index 256 bevindt?

b) Waar vind je de rechterbuur (op hetzelfde niveau) van de knoop die zich in de array op index 72 bevindt?

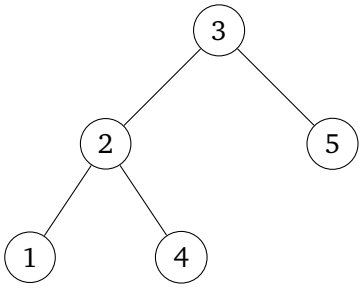
c) Heeft de knoop die zich in de array op index 249 bevindt kleinkinderen?

Oefening 4.3

De `BinaryMinHeap` klasse bevat 1 instantievariabele: `values`, een arraylist die de waarden van de min-heap bevat. Hierbij zal in het eerste element van de arraylist steeds de kleinste waarde van de values zitten.



Figuur 4.1 Welke arrayimplementatie voor deze binaire boom?



Figuur 4.2 Hoe deze boom voorstellen met een array?

Voeg een methode `getMin` aan de `BinaryMinHeap` klasse toe die de minimale waarde opgeslagen in de min-heap teruggeeft. De methode gooit een `IllegalStateException` indien de heap geen elementen bevat. Voor het testen van deze methode moet je even wachten tot na de implementatie van de `addValue` methode in de volgende oefening.

Oefening 4.4

In deze oefening bekijken we het toevoegen van een gegeven waarde aan een min-heap.

a) Bestudeer de methode `addValue` van de `BinaryMinHeap` klasse die een gegeven waarde aan de min-heap toevoegt. Zie de slides van deze les voor meer uitleg.

b) Implementeer de `bubbleUp` functie in de `BinaryMinHeap` klasse.

c) Run de `main` functie in de `BinaryMinHeapDriver` klasse en controleer hiermee de implementatie van de `addValue` en `getMin` methode.
Verwachte uitvoer :
[-7, -4, -2, 0, 1, 2, -1, 3, 2]
Kleinste waarde = -7

Oefening 4.5

Deze oefening gaat over het verwijderen van het kleinste getal uit een binary min-heap.

a) Bestudeer de methode `removeSmallest` van de `BinaryMinHeap` klasse die een gegeven waarde uit de min-heap verwijdert. Zie de slides van deze les voor meer uitleg.

b) Implementeer de `bubbleDown` functie in de `BinaryMinHeap` klasse.

c) Run de `main` functie in de `BinaryMinHeapDriver` klasse en controleer hiermee de implementatie van de `removeSmallest` methode.
Verwachte uitvoer:
-7
[-4, 0, -2, 2, 1, 2, -1, 3]
-4
[-2, 0, -1, 2, 1, 2, 3]
-2
[-1, 0, 2, 2, 1, 3]
-1
[0, 1, 2, 2, 3]
0
[1, 2, 2, 3]

Oefening 4.6

In een binary min-heap is elk pad van de kleinste naar een ander element gesorteerd.

a) Implementeer een `getPath` functie in de `BinaryMinHeap` klasse die het pad van de kleinste naar een gegeven dataveld geeft indien mogelijk.

b) Run de `main` functie in de `BinaryMinHeapDriver` klasse en controleer hiermee de implementatie van de `getPath` methode.
Verwachte uitvoer :
[1, 2, 3]
[1, 2]
null

5

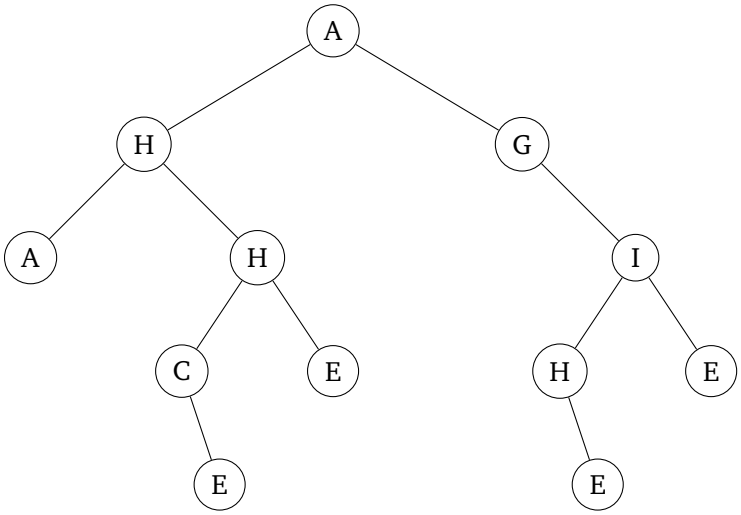
Herhalingsoefeningen binaire bomen

Oefening 5.1

Gegeven de implementatie van een binaire boom zoals in les 3 gegeven. De bedoeling van deze oefening is het aantal voorkomens van een gegeven data-veld in een binaire boom te tellen.

a) Implementeer een methode `count` in de `BinaryTree` klasse die het aantal voorkomens van een gegeven data-veld telt in een boom.

b) Maak een nieuwe `BinaryTreeDriver` klasse waarin je de boom overeenkomstig figuur 5.1 maakt.



Figuur 5.1 Binaire boom met herhaalde datavelden

c) Run de main functie in de `BinaryTreeDriver` klasse en controleer hiermee de implementatie van de `count` methode met respectievelijke parameters: I, A, H, E en Q. Verwachte uitvoer:
Aantal voorkomen van I = 1
Aantal voorkomens van A = 2
Aantal voorkomens van H = 3
Aantal voorkomens van E = 4

Aantal voorkomens van Q = 0

Oefening 5.2

a) Schrijf een methode `getNodesAtDistance(k)` in de `BinaryTree` klasse die een lijst teruggeeft van de datavelden die op een afstand k van de wortel van de binaire boom verwijderd zijn. In de boom uit figuur 5.1 zijn A, H en I de datavelden van knopen die op een afstand 2 van de root verwijderd zijn.

b) Test je implementatie uit door in de klasse `BBDriver` de methode `getNodesAtDistance` op te roepen met parameters 0, 1, 2, 3 en 4. Verwachte uitvoer:
Datavelden van knopen verwijderd op een afstand van 0 van de root = [A]
Datavelden van knopen verwijderd op een afstand van 1 van de root = [H, G]
Datavelden van knopen verwijderd op een afstand van 2 van de root = [A, H, I]
Datavelden van knopen verwijderd op een afstand van 3 van de root = [C, E, H, E]
Datavelden van knopen verwijderd op een afstand van 4 van de root = [E, E]

Oefening 5.3

Gegeven volgende code:

Listing 1 Mystery methode

```
public ArrayList<E> mystery() {
  ArrayList<E> lijst = new ArrayList<>();
  if (this.leftTree != null) lijst.add(this.leftTree.data);
  if (this.rightTree != null) lijst.add(this.rightTree.data);
  return lijst;
}

public ArrayList<E> mystery(int g) {
  if (g == 1) {
    return this.mystery();
  } else {
    ArrayList<E> links = new ArrayList<>();
    if (this.leftTree != null) links = this.leftTree.mystery(g - 1);
    ArrayList<E> rechts = new ArrayList<>();
    if (this.rightTree != null) rechts = this.rightTree.mystery(g - 1);
    links.addAll(rechts);
    return links;
  }
}
```

Geef nu de uitvoer van volgende oproepen van deze methode voor de boom uit figuur 5.1:

a) `boom.mystery(1)`

b) `boom.mystery(2)`

c) `boom.mystery(3)`

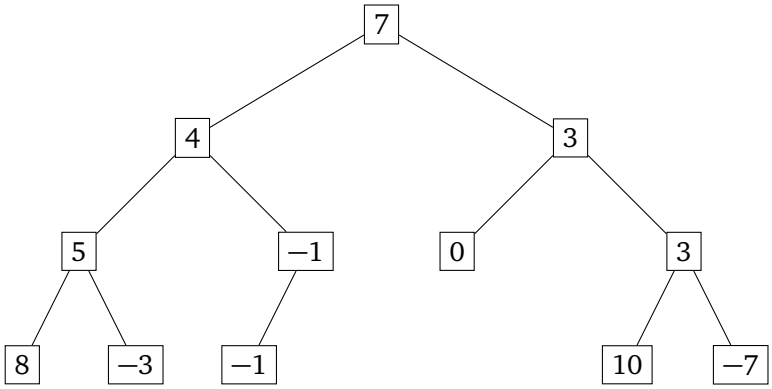
d) `boom.mystery(4)`

e) `boom.mystery(5)`

Oefening 5.4

Schrijf een methode `kinderSom()`. Het resultaat is een boolean. De methode geeft true terug als de boom waarop ze toegepast wordt voldoet aan de eigenschap dat elke knoop als waarde de som van zijn kinderen heeft (dat aantal kinderen kan natuurlijk 2, 1 of 0 zijn). Figuur 5.2 toont een boom die aan deze eigenschap voldoet.

Je zal voor deze methode een nieuwe klasse moeten schrijven. Je kan immers niet zomaar de waarde van twee knopen van een nog te bepalen type `E` optellen, bvb. als je voor `E` het type `String` kiest! Beperk je dus voor deze oefening tot een nieuw soort binaire boom, nl. één waarbij het veld `data` een `int` is. Test je methode uit door de boom van figuur 5.2 te implementeren in een mainmethode en het resultaat van de toepassing van deze methode op deze boom naar de console te schrijven.



Figuur 5.2 Binaire boom voldoet aan kinderSom

6

Herhalingsoefeningen BST

Oefening 6.1

De `preorder` methode van een BST geeft volgende uitvoer: 30, 20, 10, 15, 25, 23, 39, 35, 42. Geef de `postorder` uitvoer.

Oefening 6.2

We wensen een BST te bouwen bestaande uit de gehele getallen 1 tot en met 10. In welke volgorde moeten de getallen worden toegevoegd opdat de resulterende boom compleet is?

Oefening 6.3

Stel dat 7, 5, 1, 8, 3, 6, 0, 9, 4 en 2 worden toegevoegd aan een lege BST. Wat is de diepte van de resulterende boom?

Oefening 6.4

Stel dat we een BST maken door vertrekkende van een lege BST vervolgens de getallen 71, 65, 84, 69, 67 en 83 toe te voegen. Welke zijn de data-velden van de bladeren van de boom?

Oefening 6.5

Stel dat boom een BST is bestaande uit gehele getallen groter dan of gelijk aan 1 en kleiner dan of gelijk aan 100. Welke van de volgende paden kan (kunnen) niet?

a) 10, 75, 64, 43, 60, 57, 55

b) 90, 12, 68, 34, 62, 45, 55

c) 9, 85, 47, 68, 43, 57, 55

d) 79, 14, 72, 56, 16, 53, 55

7

Herhalingsoefeningen binaire heap

Oefening 7.1

Stel dat we een binaire max-heap hebben geconstrueerd met de code uit dat hoofdstuk. Welke van de volgende lijst van getallen is mogelijk?

a) $[25, 12, 16, 13, 10, 8, 14]$

b) $[25, 14, 16, 13, 10, 8, 12]$

c) $[25, 14, 12, 13, 10, 8, 16]$

Oefening 7.2

De getallen 32, 15, 20, 30, 12, 25 en 16 worden *in die volgorde* toegevoegd aan een binaire max-heap. Welke van volgende getallenlijst is de juiste volgorde in de arraylist?

a) $[32, 30, 25, 15, 12, 20, 16]$

b) $[32, 25, 30, 12, 15, 20, 16]$

c) $[32, 30, 25, 15, 12, 16, 20]$

d) $[32, 25, 30, 12, 15, 16, 20]$

Oplossingen

Oplossing 1.1 De defintie op Wikipedia klopt volgens ons niet helemaal. Het eerste Fibonaccigetal is niet 0, maar 1. Het tweede Fibonaccigetal is ook 1. Daarna is het n-de Fibonaccigetal gelijk aan de som van de twee vorige. De code in listing 2 is kort, maar nogal onefficiënt. Om het 100ste getal te berekenen, moeten we eerst het 99ste en het 98ste getal berekenen en die optellen. Om het 99ste te berekenen moeten we echter opnieuw eerst het 98ste berekenen. We doen dus veel te veel werk.

Listing 2 Recursieve methode om Fibonacci-getallen te berekenen

```
public static int fibonacci(int getal) {
  if (getal <= 0)
    throw new IllegalArgumentException();
  if (getal <= 2)
    return 1;
  else
    return fibonacci(getal - 1) + fibonacci(getal - 2);
}
```

Oplossing 1.2

Listing 3 Recursieve methode om de som van de cijfers van een getal te berekenen

```
public static int somVanCijfers(int getal) {
  getal = Math.abs(getal);
  if (getal < 10) //basisgeval, slechts 1 cijfer
    return getal;
  else //getal bestaat uit meer dan 1 cijfer, recursieve oproep van de functie nodig
    return getal % 10 + somVanCijfers(getal / 10);
}
```

Oplossing 1.3 Een vraag die we geregeld krijgen in de oefenzitting is waarom het niet werkt als de test of de gegeven string null is, later in de code staat, bvb. na de test of de lengte van de string kleiner of gelijk is aan 1. Wat gebeurt er als je de lengte vraagt van een niet bestaande string? In plaats van `charAt` kan je vanzelfsprekend ook gebruik maken van `substring`.

Listing 4 Recursieve methode om een string om te keren

```
public static String keerOm(String s) {
  if (s == null)
    throw new IllegalArgumentException();
  else if (s.length() <= 1)
    return s;
  else
    return s.charAt(s.length() - 1) + Recursie.keerOm(s.substring(0, s.length() - 1));
}
```

Oplossing 1.4 Deze methode is eigenlijk veel eenvoudiger als een lus te programmeren, maar het kan ook recursief. Merk de ternaire operator `( ? : )` in de laatste return op. Je kan die altijd vervangen door een toekenning in een `if else`, maar deze ternaire operator is een stuk korter.

Listing 5 Recursieve methode om het aantal keer te tellen dat de letter x in een string voorkomt

```
public static int countX(String s) {
  if (s == null)
    throw new IllegalArgumentException();
  else if (s.length() == 0)
    return 0;
  else
    return ((s.charAt(0) == 'x') ? 1 : 0) + countX(s.substring(1));
}
```

Oplossing 1.5 Een aandachtspunt (wat je ongetwijfeld weet vanuit het eerste semester, maar misschien wel even vergeten was) is het vergelijken van strings. Kom niet in de verleiding om == te gebruiken, maar doe dit altijd met de methode `equals`. Weet je nog waarom?

Listing 6 Recursieve methode om het aantal keer te tellen dat de string hi in een string voorkomt

```
public static int countHi(String s) {
  if (s == null)
    throw new IllegalArgumentException();
  else if (s.length() <= 1)
    return 0;
  else {
    if ((s.substring(0, 2)).equals("hi")) {
      return 1 + countHi(s.substring(2));
    } else
      return countHi(s.substring(1));
  }
}
```

Oplossing 1.6

Listing 7 Vervang in een string elke letter x door een y

```
public static String changeXY(String s) {
  if (s == null)
    throw new IllegalArgumentException();
  if (s.length() == 0)
    return s;
  else
    return ((s.charAt(0) == 'x' ? "y" : s.charAt(0)) + changeXY(s.substring(1)));
}
```

Oplossing 1.7

Listing 8 Vervang in een string elke pi door een 3.14

```
public static String changePi(String s) {
  if (s == null)
    throw new IllegalArgumentException();
  if (s.length() <= 1)
    return s;
  else {
    if ((s.substring(0, 2)).equals("pi")) {
      return "3.14" + Recursie.changePi(s.substring(2));
    } else
      return s.charAt(0) + Recursie.changePi(s.substring(1));
  }
}
```

Oplossing 1.8

Listing 9 Tweelog van een macht van twee

```
public static int tweelog(int getal) {
  if (getal <= 0)
    throw new IllegalArgumentException();
  if (getal == 1)
    return 0;
  else
    return 1 + tweelog(getal / 2);
}
```

Oplossing 1.9 Ook voor deze oefening ligt het meer voor de hand om een lus te gebruiken, maar het kan dus ook recursief.

Listing 10 Maximum van een lijst getallen

```
public static double findMaximum(List<Double> lijst) {
  if (lijst == null || lijst.size() == 0)
    throw new IllegalArgumentException();
  else if (lijst.size() == 1)
    return lijst.get(0);
  else {
    double g = findMaximum(lijst.subList(1, lijst.size()));
    if (lijst.get(0) > g)
      return lijst.get(0);
    else
      return g;
  }
}
```

Oplossing 1.10 Ongetwijfeld de moeilijkste oefening, ook al omdat je nu moet werken met lijsten.

Listing 11 Alle substrings van een gegeven string

```
public static ArrayList<String> findSubstrings(String string) {
  if (string == null)
    throw new IllegalArgumentException();
  ArrayList<String> res = new ArrayList<String>();
  if (string.length() <= 1) { //ook de lege string telt mee!
    res.add(string);
    return res;
  }
  else {
    res.add(string.substring(0, 1));
    ArrayList<String> res2 = findSubstrings(string.substring(1));
    res.addAll(res2);
    for (String s : res2) {
      res.add(string.charAt(0) + s);
    }
    return res;
  }
}
```

Oplossing 1.11

Listing 12 Aantal kaarten nodig om een kaartenhuisje te maken

```
public class Kaartenhuisje {

  public static void main(String[] args) {
    for (int n = 1 ; n <= 20 ; n++)
    System.out.println("Aantal Kaarten nodig voor een kaarten huisje van " + n +
      " verdieping" + (n == 1 ? " = ": "en = ") + aantalKaarten(n));
  }

  private static int aantalKaarten(int n){
    if (n < 1) throw new IllegalArgumentException();
    if (n == 1) return 2;
    else return aantalKaarten(n-1) + (n-1) + 2 * n ;
  }

}
```

Oplossing 2.1

a) Ja want elke knoop in deze boom heeft maximaal 2 kinderen.

b) Knoop F.

c) Het maximaal aantal knopen van een pad van de wortel naar een blad is vier, dus de diepte van deze boom is vier.

d) Neen, niet compleet. Alle niveaus behalve eventueel de laatste zijn niet volledig gevuld. Zo ontbreekt er een knoop op niveau 3: het linkerkind van knoop G. Verder zijn alle knopen op het laatste niveau ook niet aan de linkerzijde: de twee kinderen van knoop A ontbreken.

e) Knopen A, C, E en H.

f) Knopen F, B, D, G en I.

g) Deze subboom bevat 5 knopen.

h) F B A D C E G I H

i) A B C D E F G H I

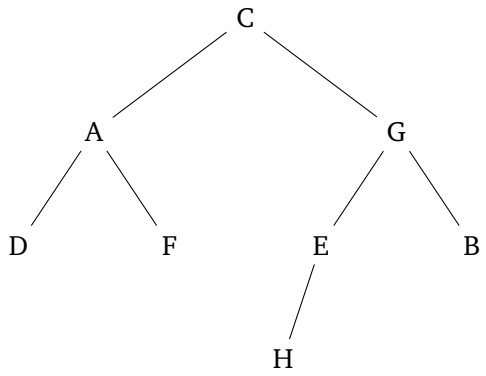
j) A C E D B H I G F

Oplossing 2.2

a)

b)

c) Figuur 1 toont een grafische voorstelling van de gegeven boom.



Figuur 1 Boom uit oefening 2

d) Een preorder doorloop geeft: C A D F G E H B.

e)

f) De code van de boom uit oefening 1 kan er als volgt uitzien:

Listing 13 Binaire boom uit oefening 1

```
package ui;

import domain.BinaryTree;

public class BinaryTreeDriver2 {

  public static void main(String[] args) {
    //begin bij de bladeren ...
    BinaryTree<String> nodeA = new BinaryTree<>("A");
    BinaryTree<String> nodeC = new BinaryTree<>("C");
    BinaryTree<String> nodeE = new BinaryTree<>("E");
    BinaryTree<String> nodeH = new BinaryTree<>("H");

    // ... ga vervolgens naar alle interne knopen ...
    // knoop I heeft links H en rechts niets
    BinaryTree<String> nodeI = new BinaryTree<>("I",nodeH, null);
    // knoop G heeft links niets en rechts I
    BinaryTree<String> nodeG = new BinaryTree<>("G",null, nodeI);
    // knoop D heeft links C en rechts E
    BinaryTree<String> nodeD = new BinaryTree<>("D",nodeC,nodeE);
    // knoop B heeft links A en rechts D
    BinaryTree<String> nodeB = new BinaryTree<>("B",nodeA, nodeD);

    // ... en eindig met de wortel
    // boom heeft root F en heeft links B en rechts G
    BinaryTree<String> boom = new BinaryTree<>("F",nodeB, nodeG);
    boom.printPreorder();
  }

}
```

g) F B A D C E G I H

Oplossing 2.3 In het `domain` package voeg je in `BinaryTree.java` volgende methode in:

Listing 14 In-order doorloop van een binaire boom

```
public void printInorder(){
  if (this.leftTree != null) {
    this.leftTree.printInorder();
  }
  System.out.print(this.data + " ");
  if (this.rightTree != null) {
    this.rightTree.printInorder();
  }
}
```

Oplossing 2.4 In het `domain` package voeg je in `BinaryTree.java` volgende methode in:

Listing 15 Post-order doorloop van een binaire boom

```
public void printPostorder(){
  if (this.leftTree != null) {
    this.leftTree.printPostorder();
  }
  if (this.rightTree != null) {
    this.rightTree.printPostorder();
  }
  System.out.print(this.data + " ");
}
```

Oplossing 2.5 Je vindt het correcte aantal knopen (9) bvb. met de methode `countNodes` die je implementeert in `BinaryTree.java`. Vanzelfsprekend kan je met `if` constructies werken, maar de *ternaire operator* van java maakt de code wel een stuk kernachtiger (listing 16).

Listing 16 Tel het aantal knopen in een binaire boom

```
public int countNodes() {
  return 1 + (this.leftTree == null ? 0 : this.leftTree.countNodes())
           + (this.rightTree == null ? 0 : this.rightTree.countNodes());
}
```

Oplossing 2.6 Je vindt de maximale diepte van een boom met de methode `getMaxDepth` die je implementeert in `BinaryTree.java`. Ook hier weer een oplossing met de *ternaire operator* van java (listing 17).

Listing 17 De diepte van een binaire boom

```
public int getDepth() {
  return 1 + Math.max((this.leftTree == null ? 0 : this.leftTree.getDepth()),
           (this.rightTree == null ? 0 : this.rightTree.getDepth()));
}
```

Oplossing 2.7 Listing 18 toont een eenvoudige implementatie voor deze functie.

Listing 18 Is een knoop een blad?

```
public boolean isLeaf() {
  return this.leftTree == null && this.rightTree == null;
}
```

Oplossing 2.8 Listing 19 toont een eenvoudige implementatie voor deze functie.

Listing 19 Hoeveel bladeren bevat deze boom?

```
public int countLeaves() {
  if (this.isLeaf()) {
```

```
    return 1;
  } else {
    return (this.leftTree == null ? 0 : this.leftTree.countLeaves())
          + (this.rightTree == null ? 0 : this.rightTree.countLeaves());
  }
}
```

Oplossing 2.9 Deze methode is wat moeilijker dan de vorige omdat je nu met lijsten moet werken. Listing 20 toont een mogelijke implementatie. Met een lus in de `main` methode kan je als test de data van alle bladeren uitschrijven.

Listing 20 Genereer een lijst met de data van alle bladeren

```
public ArrayList<E> getDataLeaves() {
  ArrayList<E> res = new ArrayList<>();
  if (this.isLeaf()) {
    res.add(this.data);
  } else {
    res = (this.leftTree == null ? new ArrayList<>() : this.leftTree.getDataLeaves());
    ArrayList<E> rightLeaves =
      (this.rightTree == null ? new ArrayList<>() : this.rightTree.getDataLeaves());
    res.addAll(rightLeaves);
  }
  return res;
}
```

Oplossing 2.10 Een mogelijke implementatie vind je in listing 21.

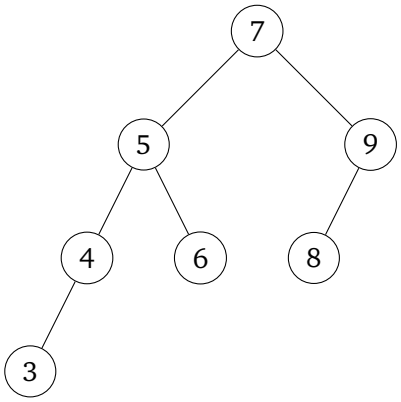
Listing 21 Komt een bepaalde waarde voor in een boom?

```
public boolean contains(E s) {
  if (s == null) {
    return false;
  }
  if (s.equals(this.data)) {
    return true;
  } else {
    return (this.leftTree == null ? false : this.leftTree.contains(s)) ||
        (this.rightTree == null ? false : this.rightTree.contains(s));
  }
}
```

Oplossing 3.1 De eerste boom (figuur 3.1) is een BST omdat elke knoop (alfabetisch) groter is dan alle knopen in de linkersubboom en kleiner dan alle knopen in de rechtersubboom. De tweede boom (figuur 3.2) is echter geen BST omdat 3 niet groter is dan 4 (terwijl de knoop 4 toch in de linkersubboom zit).

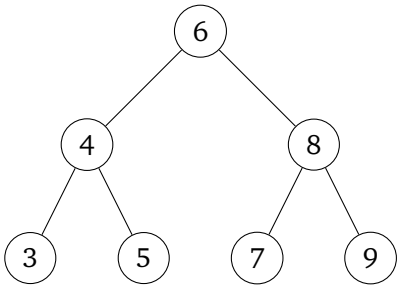
Oplossing 3.2 We bekijken nog eens de definitie van binaire zoekboom: voor elke knoop in de boom geldt dat zijn waarde strikt groter is dan alle waarden in zijn linkersubboom en strikt kleiner dan alle waarden in zijn rechtersubboom. Deze volgorde is ook wat een in-order doorloop doet.

Oplossing 3.3 Er is maar één mogelijke oplossing, nl. figuur 2.



Figuur 2 Getallen van 3 t.e.m. 9 invullen in de knopen

Oplossing 3.4 De `main` methode construeert de (gebalanceerde) BST uit figuur 3.



Figuur 3 ResultaatBST van de `main` methode

Listing 22 addNode(data) methode

```
public boolean addNode(E data){
  if (data == null) return false;
  else if (this.data == null) {
    this.data = data;
    return true;
  }
  else {
    if (data.compareTo(this.data) == 0) return false;
    else if (data.compareTo(this.data) < 0){
      if (this.leftTree == null) {
```

```
        this.leftTree = new BinarySearchTree<>(data);
        return true;
      }
      else return this.leftTree.addNode(data);
    }
    else {
      if (this.rightTree == null) {
        this.rightTree = new BinarySearchTree<>(data);
        return true;
      }
      else return this.rightTree.addNode(data);
    }
  }
}
```

Oplossing 3.5

Listing 23 searchGreatest methode

```
public E searchGreatest() {
  if (this.rightTree == null) {
    return this.data;
  } else {
    return this.rightTree.searchGreatest();
  }
}
```

Oplossing 3.6

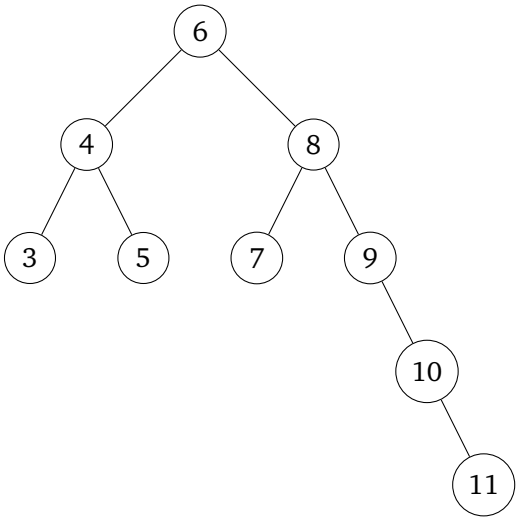
Listing 24 searchSmallest methode

```
public E searchSmallest() {
  if (this.leftTree == null) {
    return this.data;
  } else {
    return this.leftTree.searchSmallest();
  }
}
```

Oplossing 3.7 Figuur 4 toont de uitgebreide BST

Listing 25 removeNode methode

```
public boolean removeNode(E data){
  if (data == null || this.data == null) return false;
  else if (data.compareTo(this.data) == 0){
    //gevonden
    if (this.isLeaf()){
      this.data = null;
    }
```



Figuur 4 BST uitgebreid met de getallen 10 en 11

```
    else
    if (this.leftTree != null){
      E grootsteLinks = leftTree.searchGreatest();
      this.data = grootsteLinks;
      this.leftTree.removeNode(grootsteLinks);
    }
    else{
      E kleinsteRechts = rightTree.searchSmallest();
      this.data = kleinsteRechts;
      this.rightTree.removeNode(kleinsteRechts);
    }
    return true;
  }
  else if (data.compareTo(this.data) < 0) {
    if (this.leftTree == null) return false;
    else return this.leftTree.removeNode(data);
  }
  else if (this.rightTree == null) return false;
      else return this.rightTree.removeNode(data);
}
```

Oplossing 3.8

Listing 26 ruimOp methode

```
public void ruimOp() {
  if (this.leftTree != null) {
    if (this.leftTree.data == null) {
      //ouder heeft linkerkind met data null
      this.leftTree = null; //verwijder het linkerkind
    } else {
```

```
      //ruim de linkerboom verder op
      this.leftTree.ruimOp();
    }
  }
  if (this.rightTree != null) {
    if (this.rightTree.data == null) {
    //ouder heeft rechterkind met data null
      this.rightTree = null; //verwijder het rechterkind
    } else {
      //ruim de rechterboom verder op
      this.rightTree.ruimOp();
    }
  }
}
```

Oplossing 3.9

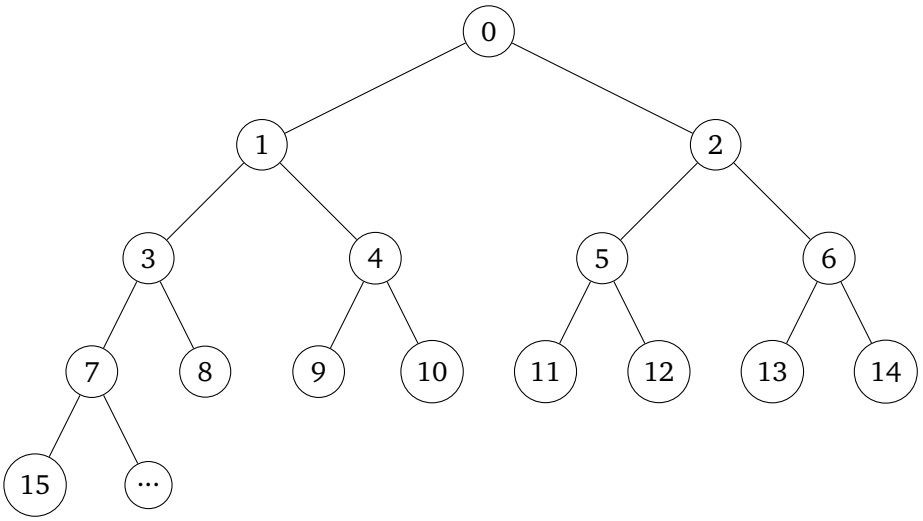
Listing 27 getPath methode

```
public ArrayList<E> getPath(E waarde) {
  if (data == null) throw new IllegalArgumentException();
  ArrayList<E> pad = new ArrayList<>();
  pad.add(this.data);
  if (this.data.compareTo(waarde) == 0) {
    //knoop met waarde gevonden
    return pad;
  } else {
    ArrayList<E> deelpad = new ArrayList<>();
    if (this.data.compareTo(waarde) > 0) {
      //als er een pad is, zit het in de linkerboom
      if (this.leftTree == null) {
        //onderaan de boom, geen pad gevonden
        return null;
      }
      deelpad = this.leftTree.getPath(waarde);
    } else {
      //als er een pad is, zit het in de rechterboom
      if (this.rightTree == null) {
        //onderaan de boom, geen pad gevonden
        return null;
      }
      deelpad = this.rightTree.getPath(waarde);
    }
    if (deelpad == null) {
      return null;
    }
    pad.addAll(deelpad);
    return pad;
  }
}
```

Oplossing 4.1 Zoals we in de theorie zagen, moet je een bepaalde waarde afspreken om aan te geven dat een bepaald kind ontbreekt. Laten we dat voor de eenvoud hier gewoon voorstellen door het getal `0`. Een mogelijke arrayimplementatie voor figuur 4.1 is:
`[F, B, G, A, D, 0, I, 0, 0, C, E, 0, 0, H]`
Voor figuur 4.2: `[3, 2, 5, 1, 4]`. Aangezien dit een complete binaire boom is, bevat de array geen "nullen".

Oplossing 4.2 Een goed startpunt om deze oefening op te lossen is een tekening maken. Het lukt natuurlijk niet om heel de boom te tekenen, maar we kunnen wel de eerste niveau's tekenen en dan de kracht van abstractie toepassen op zoek naar een patroon.

Bekijk figuur 5. Op niveau 1 is er één knoop met nummer 0. Op niveau 2 zijn er twee knopen, nummer 1 en 2. Als we deze getallen in een tabel zetten (tabel 1) valt er één en ander op.



Figuur 5 Op weg naar 1000 knooppunten, genummerd van 0 tot 999

Op niveau 9 zijn er dus 256 knooppunten, genummerd van 255 tot 510. In totaal heeft deze binaire boom op de eerste 9 niveau's samen 511 elementen. Op niveau 10 zou er plaats zijn voor 512 elementen, maar dat hoeft niet meer. Als er in totaal 1000 knooppunten moeten zijn, zal het laagste niveau nog $1000 - 511 = 489$ knopen tellen.

Met deze informatie kunnen we nu de vragen beantwoorden:

a) *Waar vind je de ouder van de knoop die zich in de array op index 256 bevindt?* De knoop met nummer 256 staat op niveau 9 als tweede van links. Het is het rechterkind van de knoop op niveau 8 met nummer 127. Dat is de eerste knoop links op het achtste niveau.

b) *Waar vind je de rechterbuur (op hetzelfde niveau) van de knoop die zich in de array op index 72 bevindt?* Knoop 72 bevindt zich op niveau 7, meerbepaald de tiende knoop op dit niveau. De rechterbuur van 72 is natuurlijk knoop 73. Beide knopen hebben een verschillende ouder (welke?).

Tabel 1 Overzicht van alle knopen

niveau	aantal	nummers	totaal
1	1	0	1
2	2	$1 \rightarrow 2$	3
3	4	$3 \rightarrow 6$	7
4	8	$7 \rightarrow 14$	15
5	16	$15 \rightarrow 30$	31
...	...	...	...
i	2^{i-1}	$2^{i-1}-1 \rightarrow 2^i-2$	2^i-1
...	...	...	...
9	256	$255 \rightarrow 510$	511

c) *Heeft de knoop die zich in de array op index 249 bevindt kleinkinderen?* De kinderen van knoop i zijn knopen $2i+1$ en $2i+2$. Knoop 249 heeft dus twee kinderen: 499 en 500. De kinderen van 499 zijn nummer 999 en 1000, die van 500 zijn 1001 en 1002. Vermits deze boom maar duizend knopen heeft, heeft de laatste knoop het nummer 999. Daaruit volgt dat knoop 249 wel degelijk één kleinkind heeft, nl. de allerlaatste knoop op het tiende niveau rechts, met nummer 999.

Oplossing 4.3

Listing 28 kleinste waarde van min-heap

```
public E getMin() {
  if (this.values.size() == 0) {
    throw new IllegalStateException();
  } else {
    return this.values.get(0);
  }
}
```

Oplossing 4.4

Listing 29 bubbleUp methode

```
private void bubbleUp() {
  int index = this.values.size() - 1; //start met laatste element

  while (heeftOuder(index) && ouder(index).compareTo(values.get(index)) > 0) {
    //ouder en kind staan in verkeerde volgorde, wissel ze om
    this.wisselOm(index, ouderIndex(index));
    index = ouderIndex(index);
  }
}
```

```
private boolean heeftOuder(int i) {
  return i >= 1;
}

private E ouder(int i) {
  return values.get(ouderIndex(i));
}

private int ouderIndex(int i) {
  return (i - 1)/2;
}

private void wisselOm(int i, int j) {
  //wissel i-de en j-de element in de ArrayList om
  E hulp = this.values.get(i);
  this.values.set(i, this.values.get(j));
  this.values.set(j, hulp);
}
```

Oplossing 4.5

Listing 30 bubbleDown methode

```
private void bubbleDown() {
  int index = 0; //start met de wortel

  boolean wisselOK = true;
  while (heeftLinkerKind(index) && wisselOK) {
    //welk kind is het kleinste?
    int indexKleinsteKind = indexLinkerKind(index);
    if (heeftRechterKind(index)
      && values.get(indexKleinsteKind).compareTo(values.get(indexRechterKind(index))) > 0) {
      indexKleinsteKind = indexRechterKind(index);
    }
    //vergelijk ouderwaarde met waarde van kleinste kind
    if (values.get(index).compareTo(values.get(indexKleinsteKind)) > 0) {
      //foute volgorde, wissel om
      this.wisselOm(index, indexKleinsteKind);
    } else {
      //volgorde OK, while lus mag stoppen
      wisselOK = false;
    }

    //vertrek nu vanuit de index van het kleinste kind
    index = indexKleinsteKind;
  }
}

private int indexLinkerKind(int i) {
  return 2 * i + 1;
}

private int indexRechterKind(int i) {
```

```
  return 2 * i + 2;
}

private boolean heeftLinkerKind(int i) {
  return indexLinkerKind(i) < values.size();
}

private boolean heeftRechterKind(int i) {
  return indexRechterKind(i) < values.size();
}
```

Oplossing 4.6

Listing 31 getPath methode

```
public ArrayList<E> getPath(E value) {
  int index = this.values.indexOf(value);
  if (index == -1) {
    //value komt niet voor in de heap
    return null;
  } else {
    //value zit in heap, index = plaats van eerste voorkomen
    ArrayList<E> pad = new ArrayList<>();
    pad.add(value);
    while (index > 0) {
      //we zijn nog niet aan de wortel
      index = (index - 1)/2; //ouder
      pad.add(0, this.values.get(index)); //voeg vooraan toe
    }
  return pad;
  }
}
```

Oplossing 5.1

Listing 32 count methode

```
public int count(E geg) {
  if (this.data == null || geg == null) {
    return 0;
  }
  return (this.data.equals(geg) ? 1 : 0)
      + (this.leftTree != null ? this.leftTree.count(geg) : 0)
      + (this.rightTree != null ? this.rightTree.count(geg) : 0);
}
```

Oplossing 5.2

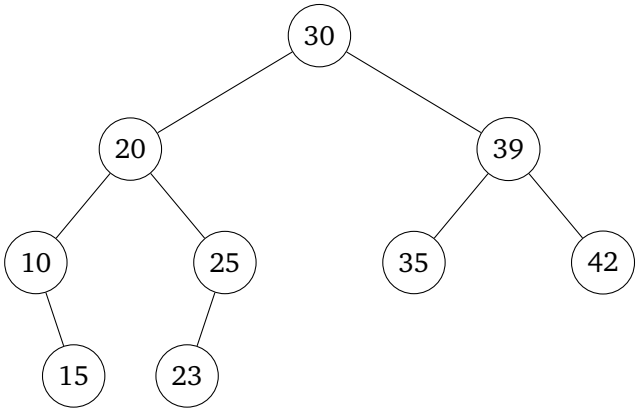
Listing 33 getNodesAtDistance(k) methode

```
public ArrayList<E> getNodesAtDistance(int k) {
  if (k < 0) {
    throw new IllegalArgumentException("Foute waarde voor afstand!");
  }
  ArrayList<E> res = new ArrayList<>();
  if (k == 0) {
    res.add(this.data);
  } else {
    if (this.leftTree != null) {
      res = this.leftTree.getNodesAtDistance(k - 1);
    }
    if (this.rightTree != null) {
      ArrayList<E> rechtsteLijst = this.rightTree.getNodesAtDistance(k - 1);
      res.addAll(rechtsteLijst);
    }
  }
  return res;
}
```

Oplossing 5.3 Je had al door dat dit een andere manier is om de vorige oefening in een opgave te gieten, niet?

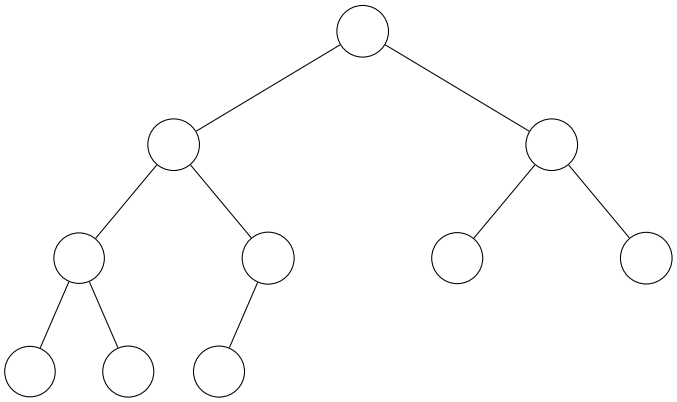
Oplossing 5.4

Oplossing 6.1 Steun op de eigenschappen van een BST: een binaire boom, met alle knopen links van een knooppunt kleiner dan dit knooppunt en in de rechterboom van het knooppunt allemaal waarden die groter zijn. Je bekomt dan de boom in figuur 6. Kijk na dat die inderdaad de gegeven pre-order volgorde geeft. Als je nu op deze BST de post-order wandeling doet bekom als uitvoer: 15, 10, 23, 25, 20, 35, 42, 39, 30.



Figuur 6 BST horend bij de gegeven pre-order uitvoer

Oplossing 6.2 Een goede manier om aan deze oefening te beginnen is een complete binaire boom tekenen met 10 knooppunten (figuur 7). Logisch redeneren met behulp van de ba-

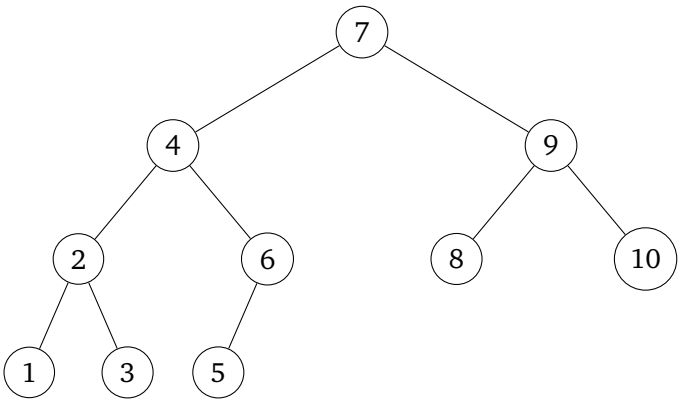


Figuur 7 complete BST voor 10 getallen

siskenmerken van een BST op deze blanco figuur levert in een aantal stappen de gewenste oplossing. Je kan bvb. volgende info gemakkelijk bekomen:

- links onderaan moet 1 staan (kleinste getal)
- rechts onderaan moet 10 staan (grootste getal)
- er zijn drie getallen groter dan de wortel, dus moet de wortel een 7 zijn
- ...

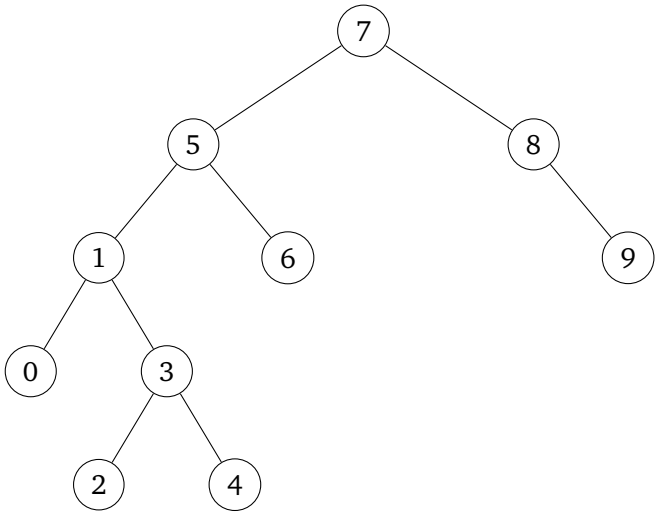
Je bekomt uiteindelijk figuur 8. Als je nu "laag per laag" aanvult komt alles waar het hoort



Figuur 8 complete BST met getallen van 1 tot 10 ingevuld

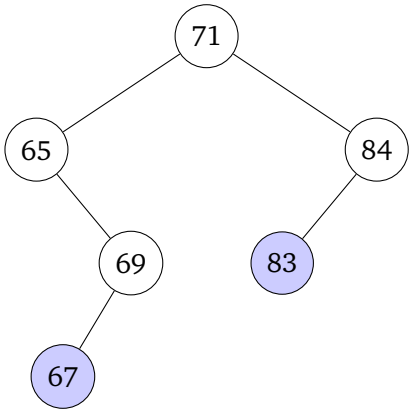
te staan. De volgorde wordt dus: 7, 4, 9, 2, 6, 8, 10, 1, 3, 5, of liever gezegd "één van de mogelijke volgordes". Snap je dat bvb. 7, 9, 4, ... ook perfect mogelijk is?

Oplossing 6.3 De resulterende boom heeft diepte 5 (figuur 9).



Figuur 9 BST als resultaat van invullen in volgorde van 7, 5, 1, 8, 3, 6, 0, 9, 4 en 2

Oplossing 6.4 De blaadjes van deze boom hebben als datavelden 67 en 83. De boom wordt getoond in figuur 10.

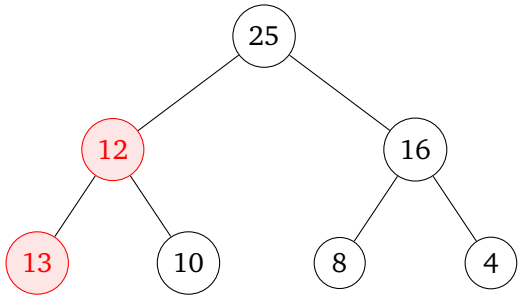


Figuur 10 BST als resultaat van invullen in volgorde van 71, 65, 84, 69, 67 en 83

Oplossing 6.5 Pad 9, 85, 47, 68, 43, 57, 55 kan niet omdat $43 < 47$. Maak zelf de figuur, waarbij je telkens links of rechts gaat afhankelijk van of het nieuwe getal kleiner of groter is dan het vorige. Als je ergens naar rechts gaat, moeten alle getallen die daarna komen groter zijn dan dit vorige getal en die voorwaarde wordt hier geschonden.

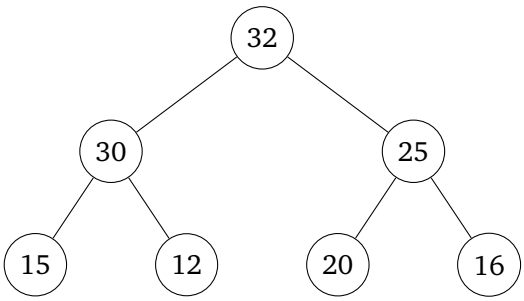
Oplossing 7.1 Het volstaat om de getallen in een boom te zetten en te kijken of de max-heap eigenschap gerespecteerd is. Het juiste antwoord is b), maar wel zullen bij wijze van uitleg laten zien waarom antwoord a) niet correct is. Zet de zeven getallen in volgorde in een

complete binaire boom. Je bekomt figuur 11. Voor een max-heap moet elke ouder groter zijn dan zijn eventuele kinderen. Aan deze eis is niet voldaan door de knopen met de getallen 12 en 13.



Figuur 11 Foute max-heap uit a)

Oplossing 7.2 Voeg alle elementen element per element toe, gevolgd door een eventuele "bubble up". Je bekomt figuur 12 en dus antwoord a).



Figuur 12 Max-heap met de getallen 32, 15, 20, 30, 12, 25 en 16 in die volgorde toegevoegd