

# 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 lesweek.

### 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](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:  $abcd \rightarrow dcba$

**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

## 1 Recursie



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

## 1 Recursie

- 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

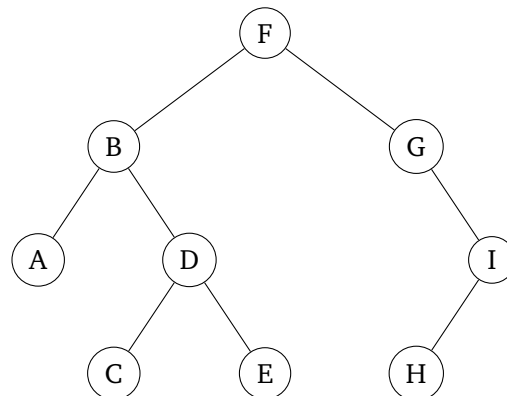
Je herkent de klassieke structuur met domain en ui. Je zal vooral methodes toevoegen in het java bestand `binaryTree` in domain en dan code uittesten in ui.

Als je de broncode hebt geopend moet je misschien nog aangeven dat de code in het mapje `src` de source is. Kies daarvoor in het menu `File > project structure` en duid bij het blad `Modules` de `src` map aan als `Sources`. De Java bestanden krijgen nu als icoontje een blauwe cirkel met een letter 'c' er in.

We maken eerst twee oefeningen op bomen op papier (📝). Vanaf oefening 2.3 schrijf je nieuwe methodes (💻).

### Oefening 2.1

📝 Bekijk de boom uit figuur 2.1 op pagina 5



**Figuur 2.1** Een (binaire?) boom

a) Is dit een binaire boom?

## 2 Binaire bomen

- 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 fig. 2.1) op in de volgorde waarin ze bezocht worden bij een pre-order wandeling.
- i) Schrijf de knopen van de gegeven boom (zie fig. 2.1) op in de volgorde waarin ze bezocht worden bij een in-order wandeling.
- j) Schrijf de knopen van de gegeven boom (zie fig. 2.1) 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 IntelliJ 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 verwachte output bij het doorlopen van de boom in pre-order.
- e) Test je verwachte output door de driver-klasse te runnen.
- f) Vervang de code in de driver-klasse door de constructie van de boom uit oefening 2.1.
- g) Run de code en controleer op die manier je antwoord op vraag h) van oefening 2.1.

### 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 2.1.

## 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 2.1.

## 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 2.1 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



Schrijf een recursieve implementatie 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.

- 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.
- Gebruik het vorige idee om een recursieve implementatie `countLeaves` te schrijven om het aantal bladeren van een binaire boom te bepalen.
- Controleer je implementatie van `countLeaves` met behulp van je testvoorbeeld in de `main` functie.

## Oefening 2.9



In deze oefening schrijf je een recursieve implementatie `getDataLeaves` 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 2.1. De volgorde zal natuurlijk afhangen van het soort wandeling dat je gebruikt.

## Oefening 2.10



Programmeer een recursieve implementatie 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.




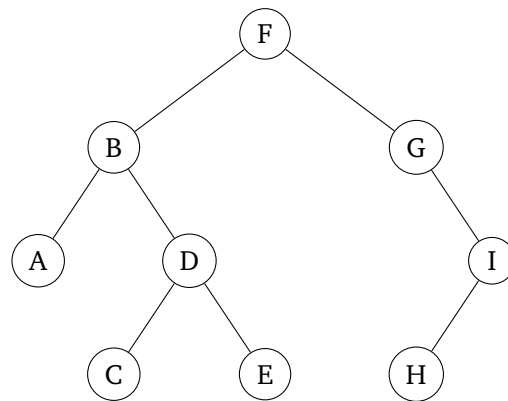
# 3

## Binaire zoekbomen (BST)

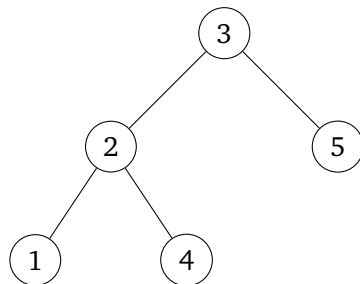
### Inleiding

### 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?




**Figuur 3.2** Een binare boom met getallen in de knopen

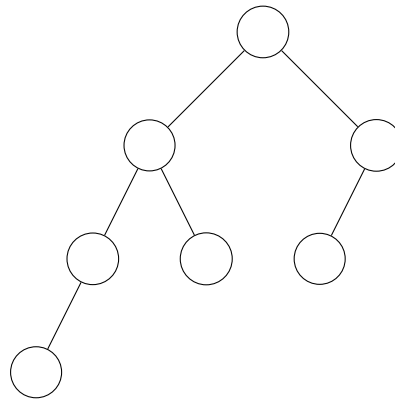
### Oefening 3.2

 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?

### Oefening 3.3

 In deze oefening krijg je een vaste structuur waar je de datavelden moet invullen.


- 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.



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

- 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 3.4

  Bestudeer de klasse `BinarySearchTree` en de `BinaryTree` in de package (domain) in de src folder. Een eerste 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. Maak optimaal gebruik van het feit dat de boom een binaire zoekboom is.

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

### Oefening 3.5



Implementeer nu ook de methode `lookUp` in de `BinarySearchTree` klasse en controleer je implementatie door in de `BinarySearchTreeDriver`-klasse een aantal knopen op te zoeken. Maak een efficiënte implementatie die de voordelen van een binaire zoekboom benut. Zoek ook een niet bestaande knoop op.

### Oefening 3.6



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

- Implementeer de `searchGreatest` methode in de `BinarySearchTree` klasse.
- Om je implementatie te controleren, run je de `BinarySearchTreeDriver` klasse uit de ui package. Verwachte uitvoer: De grootste waarde uit deze boom = 9

### Oefening 3.7



Programmeer een methode die de kleinste waarde uit de BST teruggeeft.

- Implementeer de `searchSmallest` methode in de `BinarySearchTree` klasse.
- Om je implementatie te controleren, run je de `BinarySearchTreeDriver` klasse uit de ui package. Verwachte uitvoer: De kleinste waarde uit deze boom = 3

### Oefening 3.8



*Deze en de volgende oefening beschouwen we niet als leerstof. Ze staan hier voor studenten die een extra uitdaging zoeken. Je moet het verwijderen van een node wel kennen voor het schriftelijk examen, maar hoeft het dus niet te kunnen programmeren.*

Een volgende methode is `removeNode` waarmee gegeven data uit de BST zal verwijderd worden indien mogelijk.

- Implementeer de `removeNode` methode in de `BinarySearchTree` klasse.

### 3 Binaire zoekbomen (BST)

- b) Om je implementatie te controleren, kan je de klasse `BinarySearchTreeDriver` aanpassen en uit de opgebouwde BST uit oefening 4 de knoop met dataveld 9 te verwijderen.

## Oefening 3.9



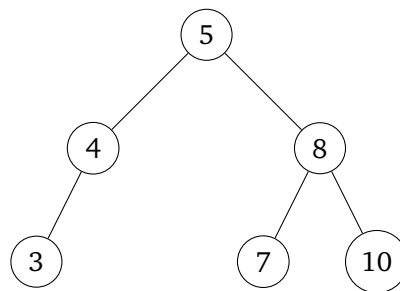
Een probleem dat weggesnoeid moet worden ...

- Implementeer de `countNodes` methode in de `BinarySearchTree` klasse die het aantal knopen in een boom telt. Na de verwijdering van de knoop met data-veld 9 valt iets eigenaardigs op: het verwijderen van de knoop heeft geen effect op het aantal knopen. We onderzoeken dat in de volgende deelvragen.
- 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.
- Blaadjes met als datawaarde `null` laten we niet aan de boom staan. Tijd om te snoeien! Schrijf een methode `cleanup()` 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.
- Voeg nog twee knopen toe aan de BST uit oefening 4 met data-velden 10 en 11. Schrijf de boom uit na een in-order wandeling (verwachte uitvoer: 3,4,5,6,7,8,9,10,11). Verwijder tenslotte uit deze boom data-veld 9, 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 3.10



In deze oefening schrijf je een methode 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.

### Oefening 3.11



Neem de boom van figuur 3.1. Teken achtereenvolgens de boom als je eerst knoop G verwijdert, dan knoop B en tenslotte knoop F.



# 4

## Binaire bomen en zoekbomen - Verdieping

In dit hoofdstuk vind je verdiepingsoefeningen bij de les 'Bomen' enerzijds en de les 'Binaire zoekbomen' anderzijds. Er zijn zowel theorie- als praktijkoefeningen. De praktijkoefeningen zijn vaak extra methodes die je aan de domain-klassen van de vorige hoofdstukken kan toevoegen.

### Oefening 4.1



(Examenvraag augustus 2019) Je weet wat een complete binaire boom is. Een *strikt* binaire boom wordt gedefinieerd als een binaire boom waar elke knoop, met uitzondering van de bladeren, *exact twee* kinderen heeft. Je krijgt nu 5 stellingen over beide bomen. Welke van de 5 stellingen is juist? Geef bij elke stelling die je af- of goedkeurt een duidelijke uitleg mbv een figuur.

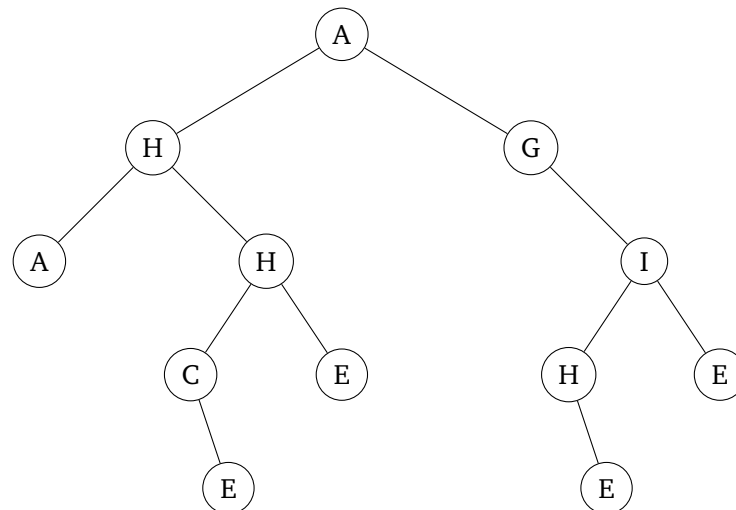
1. Elke binaire boom is ofwel compleet ofwel strikt.
2. Elke complete binaire boom is ook een strikt binaire boom.
3. Elke strikt binaire boom is ook een complete binaire boom.
4. Een binaire boom kan nooit tegelijk compleet en strikt zijn.
5. Geen enkele van bovenstaande zinnen is juist.

### Oefening 4.2



Gegeven de implementatie van een binaire boom zoals in les 2 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 4.1 maakt.




**Figuur 4.1** 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 4.3

 (Examen augustus 2019) Een preorder wandeling in een binaire zoekboom (BST) met in de knopen gehele getallen levert als resultaat volgende rij: 30, 20, 10, 15, 25, 23, 39, 35, 42. Geef het resultaat van een postorder wandeling in dezelfde BST.

### Oefening 4.4

 (Examen augustus 2019) Men zoekt in een BST van gehele getallen naar het getal 43. Men doorloopt zo een aantal knopen en het zesde getal is 43. Welke van volgende rijen van getallen zijn *niet* mogelijk als lijst van opeenvolgende knopen die men bekomt als resultaat van de zoektocht? Geef een duidelijke uitleg waarom je antwoord(en) niet kan (kunnen).

1. 61, 52, 14, 17, 40, 43



2. 2, 3, 50, 40, 60, 43
3. 10, 65, 31, 48, 37, 43
4. 81, 61, 52, 14, 41, 43
5. 17, 77, 27, 66, 18, 43

## Oefening 4.5



Zoek de knopen op een bepaalde afstand.

- 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 4.1 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 4.6



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 4.1:

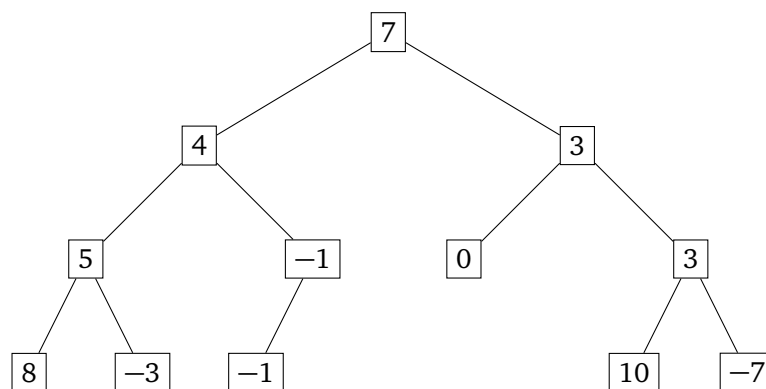
- a) boom.mystery(1)
- b) boom.mystery(2)
- c) boom.mystery(3)
- d) boom.mystery(4)
- e) boom.mystery(5)

### Oefening 4.7




(Examen juni 2017) 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 4.2 op pagina 18 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 4.2 te implementeren in een `main`methode en het resultaat van de toepassing van deze methode op deze boom naar de console te schrijven.



**Figuur 4.2** Binaire boom voldoet aan `kinderSom`

## Oefening 4.8

 (Examen juni 2019) Een binaire boom heeft een letter als waarde in elke knoop. Als je de boom *in-order* doorwandelt krijg je 'FHCADEGB'. Een *post-order* wandeling levert voor dezelfde boom 'FCHAGEBD'. Teken hieronder deze boom.


## Oefening 4.9

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


## Oefening 4.10

 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 4.11

 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 4.12


 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 4.13

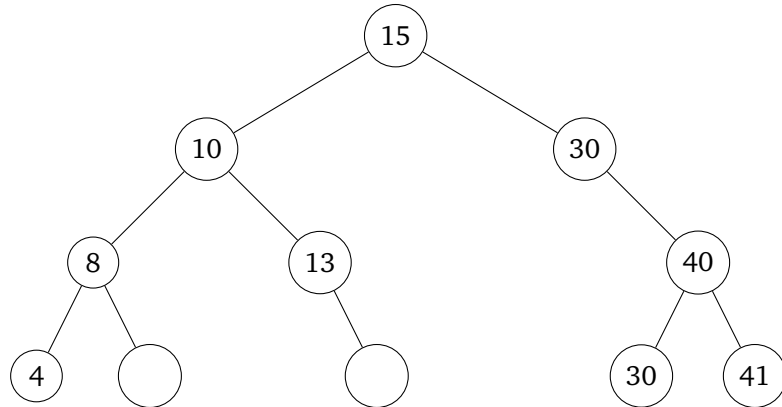
 Stel dat boom een BST is bestaande uit gehele getallen groter dan of gelijk aan 1 en kleiner dan of gelijk aan 100. Welke van de volgende paden kan (kunnen) niet?

- 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

### Oefening 4.14

 (Examenvraag juni 2017)


1. Gegeven de binaire zoekboom (BST) in figuur 4.3.
  - Vul een geheel getal in voor de ontbrekende knopen.
  - Verbeter eventuele fouten (door enkel aanpassingen te doen aan de bladeren!), zodat de BST eigenschap voor deze boom volledig klopt.



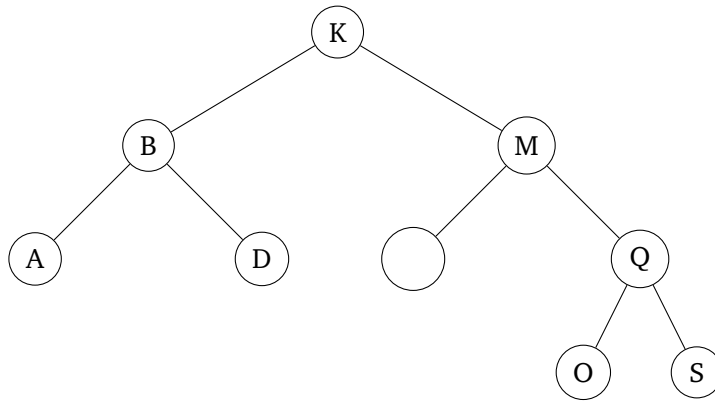
Figuur 4.3

2. Voeg aan de BST van figuur 4.3 een knoop met het getal 6 toe op de juiste plaats.
3. Vertrek nu van de boom die je in het vorige puntje bekwam. Teken hieronder de nieuwe toestand van de BST (teken de volledige boom) na verwijderen van de knoop met het getal 10.

### Oefening 4.15

 (Examenvraag augustus 2017)

1. Gegeven de binaire zoekboom (BST) in figuur 4.4. Vul een letter in voor de ontbrekende knoop, zodat de BST eigenschap voor deze boom nog klopt.
2. Voeg in figuur 4.4 aan de BST een knoop met dataveld C toe op de juiste plaats.



**Figuur 4.4**

3. Teken de nieuwe toestand van de BST (teken de volledige boom, ook met dataveld C) na verwijderen van de knoop met het dataveld M.

## Oefening 4.16



(Examenvraag juni 2018) Gegeven de volgende java-klasse:

```
package domain;
public class Persoon implements Comparable<Persoon>{
    private String naam, voornaam;
    private int lengte;
    private double gewicht;
    public Persoon(String voornaam, String naam, int lengte, double gewicht) {
        this.setVoornaam(voornaam);
        this.setNaam(naam);
        this.setLengte(lengte);
        this.setGewicht(gewicht);
    }

    private void setNaam(String naam) {
        if (naam == null || naam.trim().isEmpty()) throw new
            IllegalArgumentException();
        this.naam = naam;
    }

    private void setVoornaam(String voornaam) {
        if (voornaam == null || voornaam.trim().isEmpty()) throw new
            IllegalArgumentException();
        this.voornaam = voornaam;
    }

    private void setLengte(int lengte) {
        if (lengte <= 30) throw new IllegalArgumentException();
        this.lengte = lengte;
    }

    private void setGewicht(double gewicht) {
        if (gewicht <= 25) throw new IllegalArgumentException();
        this.gewicht = gewicht;
    }

    public int getBMI() {
        return (int)(Math.round(gewicht * 10000 / (lengte * lengte)));
    }

    @Override
    public String toString() {
        return voornaam + " " + naam + " BMI = " + this.getBMI();
    }

    @Override
    public int compareTo(Persoon o) {
        if (o == null) return 1;
        else {
            int i = this.getBMI() - o.getBMI();
            if (i != 0) return i;
            else {
```

```

        i = this.naam.compareTo(o.naam);
        if (i != 0) return i;
        else return this.voornaam.compareTo(o.voornaam);
    }
}
}
}

```

1. Volgende objecten worden aan een BST<Persoon> vervolgens toegevoegd:

```

els = new Persoon("Els", "Adams", 176, 86) //bmi = 28
an = new Persoon("Anne", "Janssen", 176, 68) //bmi = 22
tom = new Persoon("Tom", "Frederiks", 185, 105) // bmi = 31
tim = new Persoon("Tim", "Anders", 185, 85) //bmi = 25
joke = new Persoon("Joke", "Alders", 176, 86) //bmi = 28

```

Teken de resulterende boomstructuur

2. Hoeveel objecten zouden aan de boom uit het vorige puntje *minimaal* moeten toegevoegd worden om deze *compleet* te maken? Geef een concreet voorbeeld (zoals in bovenstaande code) voor elk van deze objecten. **Voeg daarna in een andere kleur je nieuwe objecten toe aan bovenstaande boom.**
3. Stel dat de compareTo-methode in de klasse Persoon er als volgt had uitgezien:


```

@Override
public int compareTo(Persoon o) {
    if ( o == null) return 1;
    else return this.getBMI() - o.getBMI();
}

```


Vertrek opnieuw van een lege boom. Teken de boomstructuur na het toevoegen van de objecten els, an, tom, tim en joke gegeven deze compareTo-methode

## Oefening 4.17

 (Examen augustus 2018)

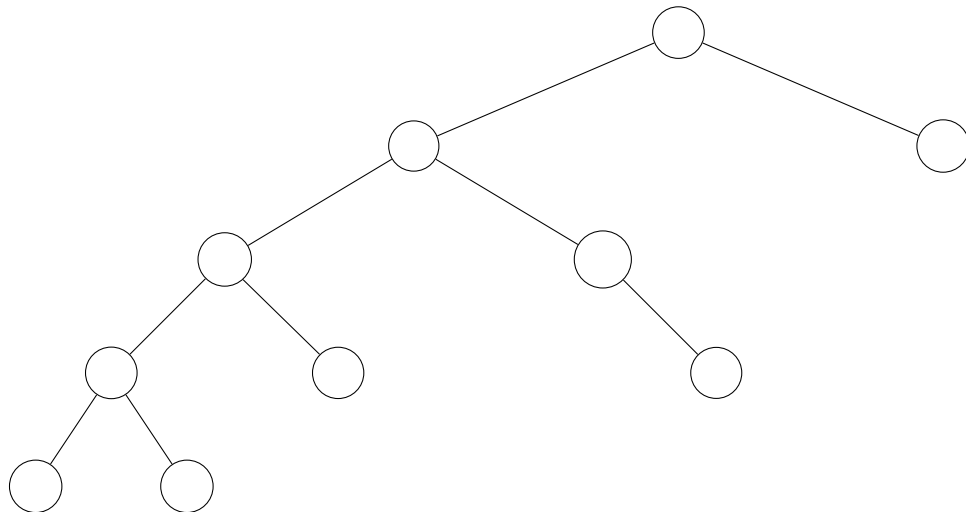
1. Gegeven een complete binaire boom met als datavelden gehele getallen. Een postorder-wandeling doorheen de boom levert als opeenvolgende getallen: -5, 4, 7, 9, 8, 2, 0, 11, 3, 5. Teken deze boom.
2. Gegeven de getallen 3, 9, -4, 6, 10, -3, -8, 5. In welke volgorde moet je deze 8 getallen toevoegen aan een lege boom zodat het resultaat een complete binaire zoekboom is? Geef de getallen in de juiste volgorde en teken de BST. Is deze volgorde uniek of zijn er meerdere mogelijke volgordes die dezelfde complete BST geven + leg uit?

## Oefening 4.18

 (Examen juni 2019)

In onderstaande binaire zoekboom met diepte 5 worden studentendossiers opgeslagen. De dossiers zijn sorteerbaar op familienaam. Er is een dossier voor volgende studenten: Aerts, Bériot, Colins, Decoster, Eerdeken, Lints, Mellaerts, Smolders, Tobback, Vansina. Stel deze namen verkort voor met hun eerste letter: A, B, ... , V.

1. Vul de eerste letter van de 10 namen op de juiste plaats in de BST van figuur 4.5.



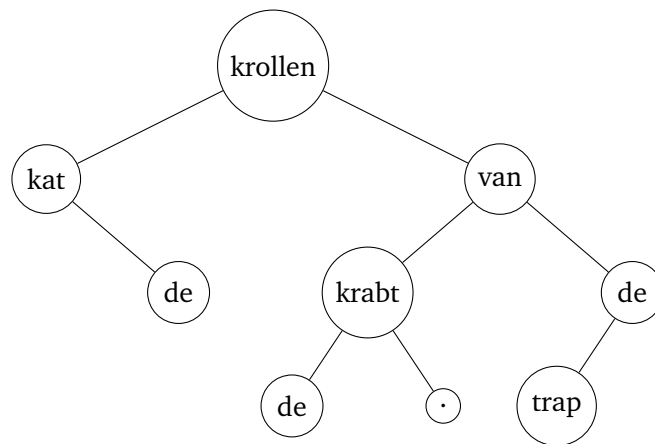
**Figuur 4.5** BST voor studentendossiers

2. Geef de/een (zie volgende vraag) volgorde waarin deze 10 dossiers moeten toegevoegd worden.
3. Is er een andere invoervolgorde mogelijk? Zo ja, geef er één. Zo nee, leg uit waarom er maar één volgorde kan zijn om deze BST te vullen.
4. (Deze en de volgende vragen kan je pas beantwoorden als de leerstof van heaps gezien is) Stel dat we deze 10 studentendossiers niet in een BST maar in een binaire max-heap bijhouden. Teken deze binaire max-heap.
5. Geef de arrayvoorstelling van deze max-heap.
6. Stel dat de binaire max-heap diepte 13 moet hebben. Reken uit hoeveel studentendossiers je dan *minimaal* zou moeten toevoegen bij de boom uit vraag (d). Je mag je berekeningen altijd aan de achterzijde van een blad maken. Verwijs er dan wel duidelijk naar ("zie achterzijde pg ...").



## Oefening 4.19


 (Examen augustus 2017) Gegeven de boom uit figuur 4.6.



Figuur 4.6 Boom 1

1. Geef de resulterende zin na een inorder-wandeling door de boom van figuur 4.6.
2. Gegeven de zin in de vorm van een array van Strings: `String[]{"Als", "de", "kat", "van", "huis", "is", "dansen", "de", "muizen", "altijd", "op", "tafel"}`. We kunnen deze zin omzetten naar een binaire boom bestaande uit knopen met als dataveld de individuele elementen van deze String-array waarbij een *postorder*-wandeling door deze boom de oorspronkelijke zin teruggeeft. Teken deze boom gegeven dat de boom *compleet* moet zijn.

## Oefening 4.20

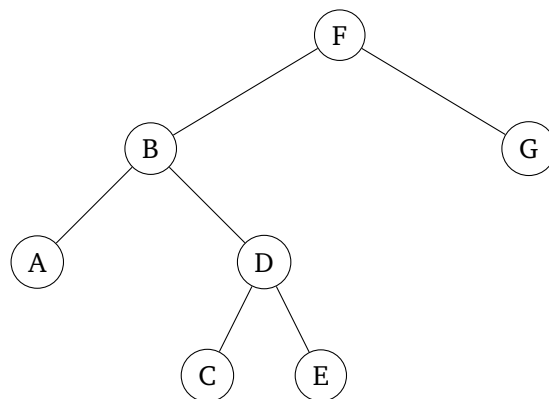
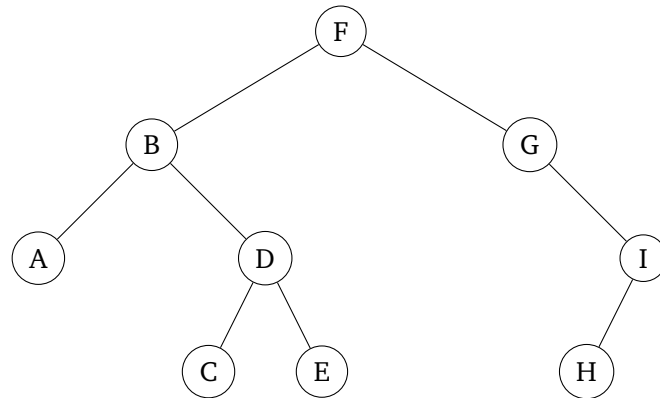
 (Examen augustus 2017) Herneem de domainklasse van Binary Tree. Schrijf in deze klasse een methode `deelZonder(E wortelInfo)` die een nieuwe `BinaryTree<E>` teruggeeft die gelijk is aan deze boom zonder de deelboom met als wortel de knoop met dataveld `wortelInfo`. Deze methode geeft `null` terug indien de parameter niet voorkomt in deze binary tree.

wordt na het oproepen van de methode `deelZonder(I)`:

UI-klasse:

```
public class BinaryTreeDriver {  
  
    public static void main(String[] args) {  
        BinaryTree<String> nodeA = new BinaryTree<>("A");  
        BinaryTree<String> nodeC = new BinaryTree<>("C");  
        BinaryTree<String> nodeE = new BinaryTree<>("E");  
        BinaryTree<String> nodeH = new BinaryTree<>("H");
```

#### 4 Binaire bomen en zoekbomen - Verdieping



```

        BinaryTree<String> nodeD = new BinaryTree<>("D", nodeC, nodeE);
        BinaryTree<String> nodeB = new BinaryTree<>("B", nodeA, nodeD);
        BinaryTree<String> nodeI = new BinaryTree<>("I", nodeH, null);
        BinaryTree<String> nodeG = new BinaryTree<>("G", null, nodeI);
        BinaryTree<String> boom = new BinaryTree<>("F", nodeB, nodeG);

        System.out.println("\nVolledige boom preorder:");
        boom.printPreorder();
        System.out.println("\nVolledige boom inorder:");
        boom.printInOrder();

        BinaryTree<String> boomZonderI = boom.deelZonder("I");
        System.out.println("\nBoom zonder I preorder: ");
        boomZonderI.printPreorder();
        System.out.println("\nBoom zonder I inorder: ");
        boomZonderI.printInOrder();

        BinaryTree<String> boomZonderB = boom.deelZonder("B");
        System.out.println("\nBoom zonder B preorder: ");
        boomZonderB.printPreorder();
        System.out.println("\nBoom zonder B inorder: ");
        boomZonderB.printInOrder();
    }
}

```

geeft volgende uitvoer:

```

Volledige boom preorder:
F B A D C E G I H
Volledige boom inorder:
A B C D E F G H I
Boom zonder I preorder:
F B A D C E G
Boom zonder I inorder:
A B C D E F G
Boom zonder B preorder:
F G I H
Boom zonder B inorder:
F G H I

```

## Oefening 4.21



(Examen juni 2018) We noemen een binaire boom *strikt* als alle knopen ofwel 0 ofwel 2 (dus niet 1!) kinderen hebben. Schrijf een methode `isStrikt()`: boolean die van een gegeven binaire boom nakijkt of hij strikt is.

Vertrek van de klasse `BinaryTree<E>` die we definieerden in de les:

#### 4 Binaire bomen en zoekbomen - Verdieping

```
package domain;

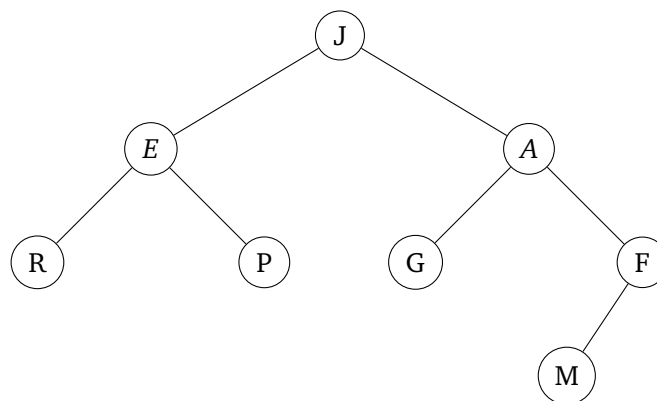
import java.util.ArrayList;

public class BinaryTree<E> {
    private E data;
    private BinaryTree<E> leftTree, rightTree;

    public BinaryTree(E data){
        this(data,null,null);
    }

    public BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree){
        this.data= data;
        this.leftTree= leftTree;
        this.rightTree= rightTree;
    }
}
```

Maak in het package ui een main methode in het bestand `BinaryTreeDriver.java` om de boom uit figuur 4.7 te maken.



**Figuur 4.7** Eenvoudige binaire boom van gehele getallen

Zorg dat je main methode volgend resultaat in de console toont:

```
volledige binaire boom strikt? -> false
binaire boom met enkel node R strikt? -> true
binaire boom met node E en kinderen R en P strikt? -> true
```

### Oefening 4.22



(Examen juni 2019) Gegeven een BST zoals we die implementeerden in de les. Schrijf

een methode `geefKnopenBinnenInterval(min:E, max:E): ArrayList<E>` die een arraylist geeft van alle knopen in stijgende volgorde en gelegen binnen het interval `[min,max]`.

UI klasse:

```
public static void main(String[] args) {
    BinarySearchTree<Integer> boom = new BinarySearchTree<>();
    boom.addNode(6);
    boom.addNode(4);
    boom.addNode(8);
    boom.addNode(3);
    boom.addNode(5);
    boom.addNode(7);
    boom.addNode(9);

    printBoomInfo(boom);

    System.out.println("\nKnopen tussen 5 en 8: "+boom.geefKnopenBinnenInterval(5,8));
    System.out.println("\nKnopen tussen 3 en 5: "+boom.geefKnopenBinnenInterval(3,5));
    System.out.println("\nKnopen tussen 8 en 9: "+boom.geefKnopenBinnenInterval(8,9));
    System.out.println("\nKnopen tussen -10 en 0: "+boom.geefKnopenBinnenInterval(-10,0));
    System.out.println("\nKnopen tussen 100 en 110: "+boom.geefKnopenBinnenInterval(100,110));
}
```

geeft volgende uitvoer:

Knopen tussen 5 en 8: [5, 6, 7, 8]

Knopen tussen 3 en 5: [3, 4, 5]

Knopen tussen 8 en 9: [8, 9]

Knopen tussen -10 en 0: []

Knopen tussen 100 en 110: []




# 5

## Binary min-heap


### Inleiding

Download het bestand `week05-06_Heap.zip` van Toledo. Ontzip het. Om de broncode van de oefeningen in IntelliJ te krijgen: map NIET openen, wel code IMPORTEREN (File > New > Project from Existing Sources of bij beginscherm: “import project”).

### Oefening 5.1


 Gegeven twee binaire bomen in figuren 5.1 en 5.2 op pagina 32. Stel dat je intern een arrayimplementatie voor bomen gebruikt. Welke array zou je dan bekomen voor deze beide bomen?

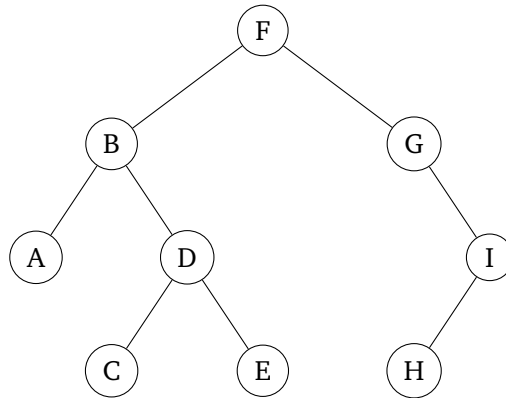
### Oefening 5.2

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

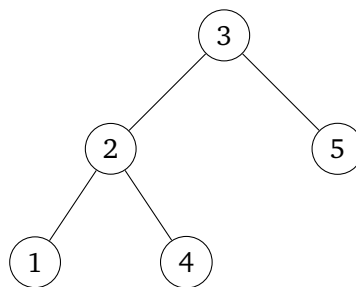
- a) Waar vind je de ouder van de knoop die zich in de array op index 256 bevindt?
- b) Waar vind je de rechterbuur (op hetzelfde niveau) van de knoop die zich in de array op index 72 bevindt? Op welk niveau staan die beiden? Hebben die beiden dezelfde ouder?
- c) Heeft de knoop die zich in de array op index 249 bevindt kleinkinderen?

### Oefening 5.3

 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 5.1** Welke arrayimplementatie voor deze binaire boom?



**Figuur 5.2** 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 5.4



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

- 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.
- Implementeer de `bubbleUp` functie in de `BinaryMinHeap` klasse.
- 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 5.5



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

- 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.
- Implementeer de `bubbleDown` functie in de `BinaryMinHeap` klasse.
- 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 5.6



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

# 6


## Binaire Heap - verdieping

### Oefening 6.1

 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 6.2


 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]

### Oefening 6.3

 (*Examen augustus 2019*) Een binaire min-heap wordt gemaakt door elk getal in het interval  $[1, 1023]$  juist één keer te gebruiken. Wat is de maximale diepte waarop het getal 9 kan staan? Gebruik in je antwoord zeker ook een figuur.

### Oefening 6.4


 (*Examen augustus 2019*) Een wandeling door een boom die we niet zagen is de zoge-

## 6 Binaire Heap - verdieping

naamde 'level order'. Hier overloopt men alle elementen van een boom niveau per niveau en binnen eenzelfde niveau van links naar rechts.


Een 'priority queue' is geïmplementeerd als een max-heap. Deze wachtrij heeft 5 elementen. Een 'level order' wandeling door deze max-heap levert als resultaat: 10, 8, 5, 3, 2. Twee nieuwe elementen worden toegevoegd aan deze max-heap: 1 en 7, in deze volgorde. Geef de 'level order' volgorde van een wandeling door deze uitgebreide boom.

### Oefening 6.5

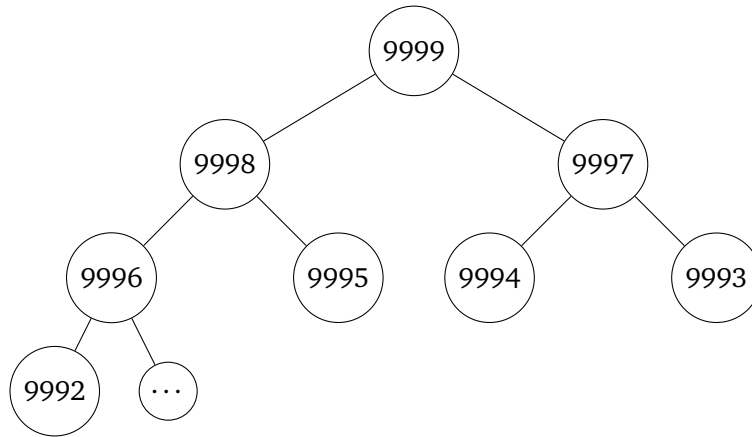
 (Examen augustus 2018) Op de luchthaven wachten vliegtuigen op de toestemming om te landen. De vliegtuigen mogen landen in een volgorde die gebaseerd is op volgende regel: *het vliegtuig met kleinste resterende hoeveelheid brandstof mag eerst landen*. De informatie met betrekking tot de brandstofhoeveelheden wordt bijgehouden in een binaire heap.

- Welke is de meest aangewezen structuur: binaire min-heap of binaire max-heap? Verklaar.
- Volgende brandstofinhouden van vliegtuigen worden vervolgens aan de heap toegevoegd: 1225, 340, 780, 840, 996, 125, 355. Welke is de resulterende array-voorstelling van de heap?
- Geef de resulterende array nadat 3 vliegtuigen geland zijn.

### Oefening 6.6

 (Examen juni 2018) Gegeven volgende max-heap uit figuur 6.1 op pagina 37. Op diepte 1 staat als root de knoop met waarde 9999. De volgende knoop heeft dan telkens als waarde één minder dan de vorige. Het laatste getal is 1.

1. Op welke diepte (= niveau) staat de knoop met waarde 1?
2. We stellen deze binaire max-heap voor met een array, dus  $A = [9999, 9998, \dots, 1]$ . Welke eigenschap van een binaire max-heap zorgt ervoor dat de keuze voor een array een efficiënt idee is?
3. Wat is de index van het getal 1000 in de array (+ uitleg)?
4. Wat is de waarde van het ouderelement van de knoop met waarde 1000 (+ uitleg)?
5. Geef de waarde van de twee kinderen van het element met waarde 1000 (+ uitleg).



**Figuur 6.1** max-heap

## Oefening 6.7



(Examen augustus 2017) Enkele vragen over heaps:

1. Een binaire min heap bestaat uit 355 getallen. Geef de indices van alle voorouders (ouder, grootouder, ...) van de knoop met index 250.
2. Een binaire min heap bestaat uit 360 getallen. Geef de indices van alle nakomelingen (kinderen, kleinkinderen, ...) van de knoop met index 44.
3. Gegeven de array-voorstelling van een binaire boom:  
 $\{4, 8, 10, 13, 12, 14, 12, 20, 18, 14, 17, 11\}$ .  
 Is deze boom een min heap? Verklaar.
4. Gegeven de array-voorstelling van een binaire boom:  
 $\{4, 8, 10, 13, 12, 14, 12, 20, 18, 14, 17, 11\}$ .  
 Geef de array-voorstelling van de boom met als root het linkerkind van de root van deze boom. Is deze boom een min heap?



# 7


## Breadth First Search Algoritme

### Inleiding

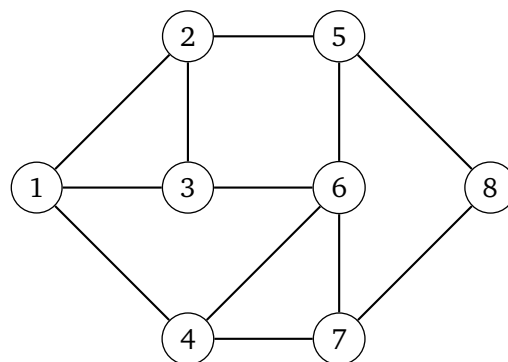
Download het bestand `week07-Grafen-BFS-opgave.zip` van Toledo. Importeer dit bestand in je IDE.

In deze oefenzitting leer je het Breadth First Search Algoritme (BFS) te implementeren in Java.

### Oefening 7.1

 In de cursustekst wordt BFS visueel uitgelegd. De knopen worden één voor één ingekleurd en de reeds bezochte knopen worden opgelijst in een boomstructuur. Gebruik deze methode om onderstaande oefening op te lossen. Zo krijg je inzicht hoe BFS werkt.

- Stel de verbindingsmatrix op van de getekende graaf.
- Zoek het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.



**Figuur 7.1** Niet-gerichte graaf met 8 knooppunten

## Oefening 7.2



Bestudeer de klasse `Graph` die je in IntelliJ importeerde. Een graaf heeft slechts één instantieveranderlijke, namelijk de verbindingsmatrix.

- De methode `isGeldigeVerbindingsmatrix(int[][] matrix)` controleert of `matrix` een geldige verbindingsmatrix is. Welke voorwaarden worden getest?
- De instantieveranderlijke is een matrix van booleans, niet van integers. Waarom?

## Oefening 7.3



We beginnen nu aan de implementatie van het BFS-algoritme. Houd de oplossing van oefening 7.2 bij de hand.

Bij BFS start je in een gegeven knoop. Je kijkt welke knopen allemaal verbonden zijn met deze startknoop. Dan neem je de eerste van die knopen en lijst je op wie met deze tweede knoop verbonden is enz. totdat je de eindknoop gevonden hebt. Dit resulteert in een array, de `ancestors`, waar je voor elke knoop kan aflezen wie zijn ‘voorganger’ is, of ‘ouder’ in de boomstructuur in afbeelding 1.8. In het schema blz. 11 is dit de array `A`.

Schrijf de methode `int[] findAncestors(int start, int destination)`. Invoer zijn start- en eindknoop. Uitvoer is de `ancestors`-array `A`, een array van integers.

De indices van `A` refereren naar (het nummer van) een knoop, maar zijn niet gelijk. Als je de voorganger van knoop met nummer `i` wil kennen, moet je in `A` de inhoud van het element met index `i - 1` opvragen. We behouden dit verschil omdat we in mensentaal niet spreken over het ‘nulde’ element, maar wel van het ‘eerste’ element.

We initialiseerden reeds alle elementen van `A` op `infty`. Zolang de waarde van een element op `infty` blijft staan, is de bijhorende knoop nog niet bezocht.

Om bij te houden welke knoop er onderzocht moet worden, gebruiken we de queue `Q`. In Java bestaat de Queue-datastructuur. Zoek op <https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html> hoe je elementen toevoegt en uitleest uit `Q`.

In het bestand `BreadthFirstSearchUI` vind je in de `main`-methode de verbindingsmatrix van het uitgewerkte voorbeeld in de cursustekst (figuur 1.5)

Verwachte uitvoer voor het voorbeeld uit de theorie:

Ancestors van 1 naar 7:

0 1 2 1 4 4 5

Ancestors van 7 naar 1:

infty infty infty infty infty infty 0



## Oefening 7.4



Schrijf nu de methode `List<Integer> findPath(int start, int destination)` die het gezochte pad berekent tussen `start` en `destination`.

Bepaal het pad van achter naar voren. Dit wil zeggen dat je start bij de eindknoop, zijn voorganger zoekt en zo opbouwt totdat je de startknoop vindt.

Verwachte uitvoer voor het voorbeeld uit de theorie:

Kortste pad van 1 naar 7 is 4 knopen lang en bestaat uit volgende knopen : [1, 4, 5, 7]

Er is geen pad van 7 naar 1

## Oefening 7.5



Voeg in de main methode de verbindingsmatrix van de graaf uit de eerste oefeningen (figuur 7.1 op pagina 39) toe en test je code met enkele verschillende start- en eindpunten in deze graaf.



# 8

## Floyd


### Inleiding

Download het bestand `week08-Grafen-Floyd-opgave.zip` van Toledo. Importeer dit bestand in je IDE.

In deze oefenzitting leer je het algoritme van Floyd te implementeren in Java.

De eerste oefening is op papier. Hiermee leer je het algoritme goed begrijpen. Verder kan je deze papieren versie gebruiken om je zelfgeschreven code te testen. Houd de oplossing dus goed bij!


### Oefening 8.1

 Een graaf bestaat uit zes knooppunten en wordt gegeven door zijn gewichtenmatrix

$$D^{(0)} = \begin{bmatrix} 0 & 7 & 3 & \infty & 5 & \infty \\ 2 & 0 & \infty & 12 & \infty & \infty \\ \infty & 3 & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix}$$

1. Teken het netwerk.
2. Gebruik de methode van Floyd om de kortste paden tussen de verschillende punten van het netwerk te berekenen.

### Oefening 8.2

 Schrijf de methode `int[][] getPointerMatrix()`. Deze methode berekent de pointermatrix  $P$ .

- De (lege) pointermatrix wordt gedeclareerd.

## 8 Floyd

- Er wordt een kloon van de gewichtenmatrix gemaakt. Dit is een kopie die verwijst naar een andere geheugenplaats. Deze kloon is de  $D$ -matrix van de cursustekst.
- Voeg de knopen één voor één toe als tussenstation en pas de  $D$ -matrix aan. Update de pointermatrix.

Verwachte uitvoer:

```
p_matrix:
0 0 4 0 4
5 0 0 0 4
5 5 0 0 4
5 5 0 0 0
0 1 4 1 0
```

### Oefening 8.3



Schrijf `ArrayList<Integer> getShortestPath(int i, int j, int[][] pointer)`. Deze methode berekent uit de pointermatrix `pointer` het kortste pad tussen knoop `i` en knoop `j`.

In deze oefening geldt dezelfde opmerking over de nummering van de knopen als we maakten in oefening 7.3. Het knooppunt dat laatst toegevoegd werd als tussenstation bij de berekening van het kortste pad tussen knooppunten `i` en `j`, is `pointer[i - 1][j - 1]`.

(Deel van) de verwachte uitvoer:

Kortste paden:

Er is geen pad van 1 naar 1

Kortste pad van 1 naar 2 lengte = 0 via : [1, 2]

Kortste pad van 1 naar 3 lengte = 0 via : [1, 4, 3]

Kortste pad van 1 naar 4 lengte = 0 via : [1, 4]

Kortste pad van 1 naar 5 lengte = 0 via : [1, 4, 5]

Kortste pad van 2 naar 1 lengte = 0 via : [2, 4, 5, 1]

Er is geen pad van 2 naar 2

Kortste pad van 2 naar 3 lengte = 0 via : [2, 3]

Kortste pad van 2 naar 4 lengte = 0 via : [2, 4]

Kortste pad van 2 naar 5 lengte = 0 via : [2, 4, 5]

Kortste pad van 3 naar 1 lengte = 0 via : [3, 4, 5, 1]

Kortste pad van 3 naar 2 lengte = 0 via : [3, 4, 5, 1, 2]

Er is geen pad van 3 naar 3

Kortste pad van 3 naar 4 lengte = 0 via : [3, 4]

Kortste pad van 3 naar 5 lengte = 0 via : [3, 4, 5]

## Oefening 8.4



Schrijf de methode `int berekenLengte(ArrayList<Integer> pad)`. Invoer is `pad`: een opeenvolging van knopen die resulteert in het kortste pad tussen eerste en laatste element van `pad`. Uitvoer is de lengte van dit pad. Deze wordt berekend met behulp van de instantie-variabele `gewichtenmatrix`. (Deel van) de verwachte uitvoer:

Kortste paden:

Er is geen pad van 1 naar 1

Kortste pad van 1 naar 2 lengte = 1 via : [1, 2]

Kortste pad van 1 naar 3 lengte = 3 via : [1, 4, 3]

Kortste pad van 1 naar 4 lengte = 1 via : [1, 4]

Kortste pad van 1 naar 5 lengte = 4 via : [1, 4, 5]

Kortste pad van 2 naar 1 lengte = 8 via : [2, 4, 5, 1]

Er is geen pad van 2 naar 2

Kortste pad van 2 naar 3 lengte = 3 via : [2, 3]

Kortste pad van 2 naar 4 lengte = 2 via : [2, 4]

Kortste pad van 2 naar 5 lengte = 5 via : [2, 4, 5]

Kortste pad van 3 naar 1 lengte = 10 via : [3, 4, 5, 1]

Kortste pad van 3 naar 2 lengte = 11 via : [3, 4, 5, 1, 2]

Er is geen pad van 3 naar 3

Kortste pad van 3 naar 4 lengte = 4 via : [3, 4]

Kortste pad van 3 naar 5 lengte = 7 via : [3, 4, 5]

Kortste pad van 4 naar 1 lengte = 6 via : [4, 5, 1]

Kortste pad van 4 naar 2 lengte = 7 via : [4, 5, 1, 2]

Kortste pad van 4 naar 3 lengte = 2 via : [4, 3]

Er is geen pad van 4 naar 4

Kortste pad van 4 naar 5 lengte = 3 via : [4, 5]

Kortste pad van 5 naar 1 lengte = 3 via : [5, 1]

Kortste pad van 5 naar 2 lengte = 4 via : [5, 1, 2]

Kortste pad van 5 naar 3 lengte = 6 via : [5, 1, 4, 3]

Kortste pad van 5 naar 4 lengte = 4 via : [5, 1, 4]

Er is geen pad van 5 naar 5



# 9

## Algoritme van Dijkstra


### Inleiding

Download het bestand `week9-Grafen-Dijkstra-opgave.zip` van Toledo. Importeer dit bestand in je IDE.

In deze oefenzitting leer je het algoritme van Dijkstra te implementeren in Java.


De eerste oefening is op papier. Hiermee leer je het algoritme goed begrijpen. Verder kan je deze papieren versie gebruiken om je zelfgeschreven code te testen. Houd de oplossing dus goed bij!

### Oefening 9.1

 Figuur 9.1 op pagina 48 toont een netwerk. De aangegeven getallen bij elke verbinding zijn het aantal km tussen beide knooppunten. Stel een lijst op met de kortste afstanden vanuit *knooppunt 3* naar elk ander punt. Geef ook telkens de kortste route aan.

1. Pas het algoritme grafisch toe cfr. fig. 3.3 in de cursusnota's.
2. Gebruik de tabelmethode (tabel 3.3-3.11 in de cursusnota's).

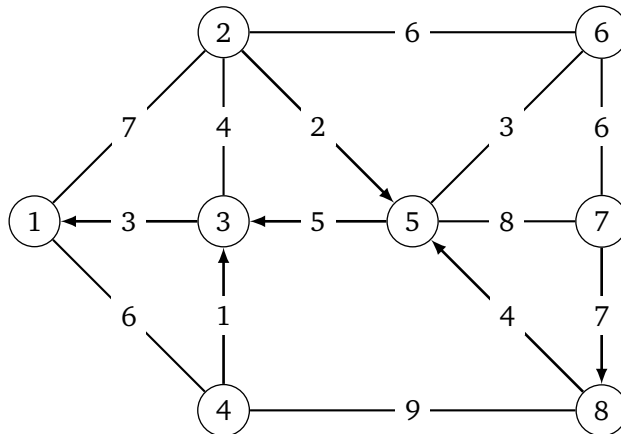
### Oefening 9.2

 Je bent eindelijk klaar met het papierwerk. Je kan nu beginnen met het opzetten van de klasse `Graph`.

Schrijf de methode `int[][] initMatrixDijkstra(int vanKnoop)`. Deze methode modelleert de gewichtenmatrix. Let op de indices die je gebruikt!

- Er wordt een `int[][]` aangemaakt die evenveel kolommen telt als de gewichtenmatrix, maar één extra rij.
- Om aan te geven dat de elementen van de laatste rij leeg zijn, stellen we ze gelijk aan `Integer.MAX_VALUE`.

## 9 Algoritme van Dijkstra



Figuur 9.1 Netwerk bij oefening 9.1

- De overige elementen krijgen de corresponderende waarde van de gewichtenmatrix waarbij infinity vervangen wordt door 0.
- Zet in de kolom die overeenkomt met de startknoop nullen.

Verwachte uitvoer bij het voorbeeld uit de cursustekst, waar een extra knoop 9 toegevoegd is cfr. main methode UIDijkstra:

Initiele matrix:

0	5	9	0	0	0	0	0	0
0	0	3	8	10	11	0	0	0
0	3	0	2	0	0	7	0	0
0	8	2	0	0	3	7	0	0
0	10	0	0	0	1	0	8	0
0	0	0	3	1	0	5	10	0
0	0	7	7	0	0	0	12	0
0	0	0	0	8	10	12	0	0
0	0	0	0	0	0	0	0	0
0	inf	inf	inf	inf	inf	inf	inf	inf

### Oefening 9.3



Schrijf de methode `int[][] Dijkstra(int vanKnoop)`. Deze methode implementeert het algoritme van Dijkstra. Ze geeft de aangepaste gewichtenmatrix terug, met extra rij onderaan met de kleinste afstanden.

Verwachte uitvoer:

Resulterende matrix:



```

0 5 0 0 0 0 0 0 0
0 0 3 0 0 0 0 0 0
0 0 0 2 0 0 7 0 0
0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 8 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 5 8 10 14 13 15 22 inf

```

## Oefening 9.4



Schrijf `ArrayList<Integer> vindPad(int vanKnoop, int naarKnoop, int[][] res)`, een methode die het kortste pad van `vanKnoop` naar `naarKnoop` berekent. De veranderlijke `int[][] res` is de aangepaste gewichtenmatrix zoals die teruggegeven wordt door de methode Dijkstra.

Verwachte uitvoer:

```

Kortste afstand van 1 naar 2 = 5
via [1, 2]
Kortste afstand van 1 naar 3 = 8
via [1, 2, 3]
Kortste afstand van 1 naar 4 = 10
via [1, 2, 3, 4]
Kortste afstand van 1 naar 5 = 14
via [1, 2, 3, 4, 6, 5]
Kortste afstand van 1 naar 6 = 13
via [1, 2, 3, 4, 6]
Kortste afstand van 1 naar 7 = 15
via [1, 2, 3, 7]
Kortste afstand van 1 naar 8 = 22
via [1, 2, 3, 4, 6, 5, 8]
Er is geen pad van 1 naar 9


```

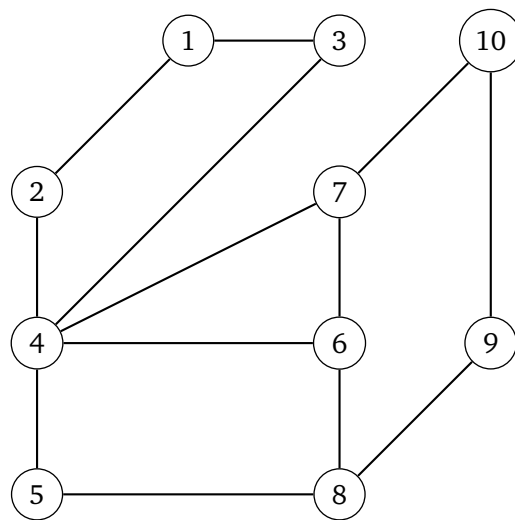


# 10

## Grafen - Verdieping


### Oefening 10.1

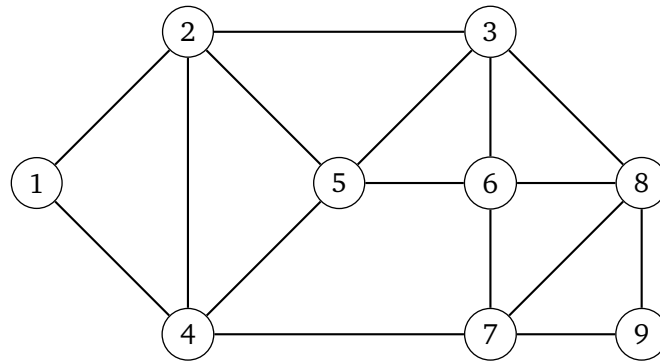
 Zoek in figuur 10.1 het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.



**Figuur 10.1** Zoek het pad van knoop 1 naar 10


### Oefening 10.2

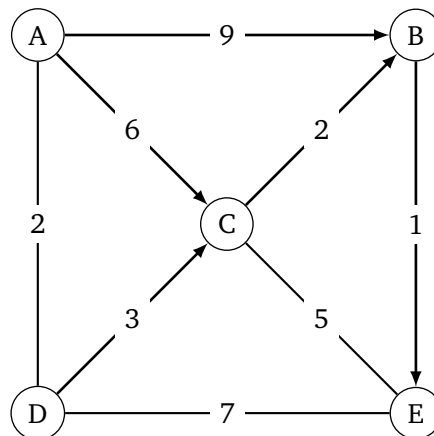
 Zoek in figuur 10.2 het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.



**Figuur 10.2** Zoek het pad van knoop 1 naar 9


### Oefening 10.3

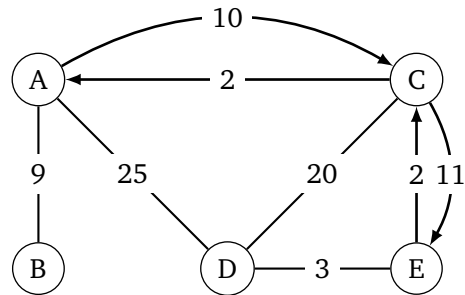
 Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 10.3 te berekenen.



**Figuur 10.3** Netwerk bij oefening 10.3

### Oefening 10.4

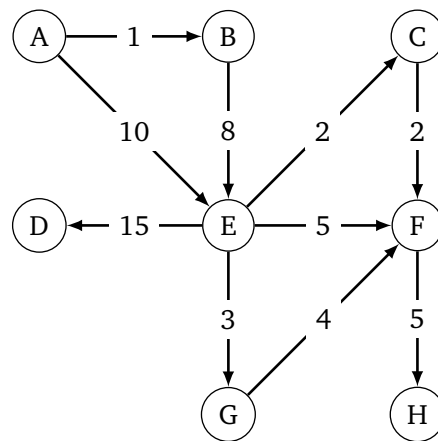
 Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 10.4 te berekenen.



**Figuur 10.4** Netwerk bij oefening 10.4

## Oefening 10.5

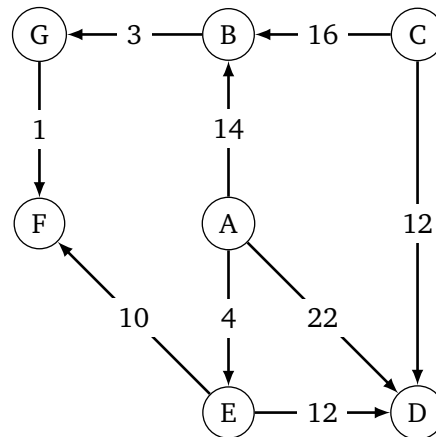
Gebruik de methode van Dijkstra om het kortste pad te bepalen van knooppunt A naar de andere knooppunten van het netwerk in figuur 10.5. Geef ook aan over welke knooppunten dat pad loopt.



**Figuur 10.5** Netwerk bij oefening 10.5

## Oefening 10.6

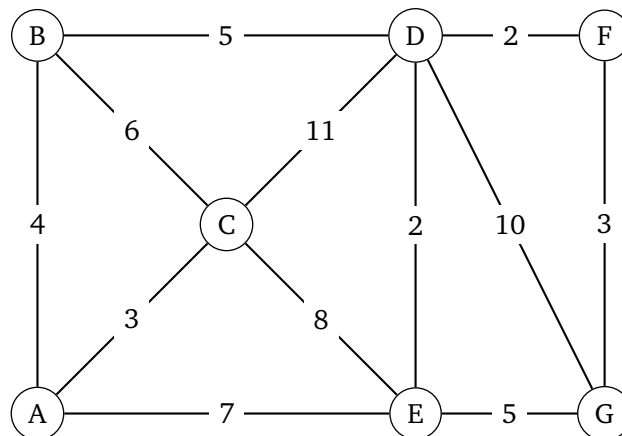
 Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 10.6 te berekenen.



**Figuur 10.6** Netwerk bij oefening 10.6

## Oefening 10.7

 Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 10.7 te berekenen.



**Figuur 10.7** Netwerk bij oefening 10.7

## Oefening 10.8



(Examen augustus 2018) Een *universele vergeetput* in een gerichte of gemengde graaf met  $n$  knooppunten is een knooppunt dat het doel is van  $n - 1$  pijlen en de bron van geen enkele pijl. Soms wordt dit ook een *beroemdheid* genoemd: iedereen in de ruimte herkent de beroemdheid, maar de beroemdheid herkent niemand.

Schrijf een methode die het nummer van het knooppunt dat een vergeetput is, teruggeeft. De graaf wordt voorgesteld door een verbindingsmatrix. Als de graaf geen vergeetput heeft, dan geeft de methode `null` terug.

Tip: teken een netwerk dat voldoet aan bovenstaande beschrijving. Stel de verbindingsmatrix hiervoor op. Ga op zoek naar de specifieke eigenschappen waaraan de verbindingsmatrix moet voldoen. Test je code met een driverklasse in `ui`.

## Oefening 10.9



(Examen juni 2019) Je beschikt over een *ongewogen, gerichte* graaf met  $n$  knooppunten. Elk knooppunt bevat een getal van 1 tot en met  $n$  dat met een persoon overeen komt. Een verbinding van knoop  $i$  naar knoop  $j$  betekent dat persoon  $i$  persoon  $j$  als één van zijn beste vrienden beschouwt (niet noodzakelijk omgekeerd).

Persoon  $i$  ( $i : 1 \dots n$ ) wint de lotto en verdient hiermee een bepaald bedrag. Hij houdt de helft zelf en verdeelt de andere helft gelijkmatig over zijn beste vrienden (behalve als die al iets van dit bedrag hebben verkregen van een andere persoon). Elke beste vriend zet dit proces verder totdat iedereen zijn deel gekregen heeft.

Schrijf een methode `verdeel(i: integer, bedrag: double)` die een array (met  $n$  elementen van het type `double`) teruggeeft. De elementen van deze array zijn het eindbedrag dat elke persoon uiteindelijk bezit als persoon  $i$  het gegeven bedrag heeft uitgedeeld en zijn vrienden het proces verder gezet hebben. Test je code via een `main` methode in `ui`.

Voor deze oefening zijn er vele juiste oplossingen mogelijk. Het hangt af van je implementatie en o.a. van de volgorde waarin je de mensen behandelt. Je mag altijd commentaar aan je code toevoegen als je wilt verduidelijken hoe je dit probleem aangepakt hebt.

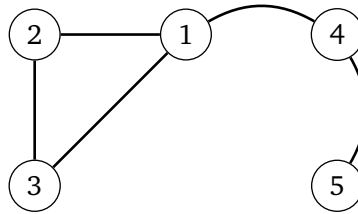
Tip: Maak voor jezelf eerst een kleine graaf die aan de beschrijving voldoet. Voer dit verdelen zelf eens uit op papier zodat je goed snapt hoe alles werkt. Pas dan probeer je een algoritme te bedenken.

## Oefening 10.10



(Examen augustus 2019) Een *brug* in een niet-gerichte graaf is een verbinding tussen twee punten die, als ze wegvalt als gevolg heeft dat de graaf niet meer verbonden is. Figuur 10.8 toont een eenvoudig voorbeeld van een niet-gerichte graaf met twee bruggen (de gebogen

lijnen). Verbinding 1–4 is een brug, want als je die weglaat zijn er twee subgrafen (1, 2, 3) en (4, 5) waar je onmogelijk van het ene stuk in het andere kan geraken. Analooog is de verbinding 4–5 ook een brug.



**Figuur 10.8** Een niet-gerichte graaf

Deze programmeervraag valt uiteen in twee delen die elk op punten staan. Wie weet kan je wel deel (b) programmeren, maar niet deel (a)? Schrijf dan die code (ook al kan je niet testen, want je hebt natuurlijk wel de code van deel (a) nodig). Nog een belangrijke tip: je kan veel code recyclen van BFS. Maak zeker ook gebruik van het feit dat de graaf niet-gericht is, en dus dat de verbindingsmatrix ... is). Werk eerst bovenstaand voorbeeld even op papier uit vooraleer je begint te programmeren!

1. Schrijf een methode `isBrug(van: int, naar: int): boolean` die op een ongerichte graaf nakijkt of een verbinding tussen twee punten van en naar een brug vormt.
2. Schrijf een methode `aantalBruggen(): int` die het aantal bruggen telt in een gegeven ongerichte graaf, gebruik makend van de methode `isBrug`.



# 11

## Bomen: Overzichtsoefening

In deze oefening krijg je een klasse Vak die sorteerbaar is op studiepunten en vervolgens op naam. Het vak 'BOP' komt bijvoorbeeld voor het vak 'OOP' en na het vak 'Web1' omwille van verschil in studiepunten. Het vak 'Algo' komt voor het vak 'Web2' omdat ze beide evenveel studiepunten hebben maar 'Algo' alfabetisch voor 'Web2' komt.

### Doel van deze oefening

Je hebt verschillende boomstructuren leren opzetten. Elke structuur heeft zijn kenmerken, i.h.b. wat zoeken naar grootste/kleinste waarde betreft. In deze oefening word je gevraagd na te denken welke structuur het beste past in de gegeven situatie.

We maken in deze oefening een brug naar de opleidingsonderdelen BOP en Web 2, in het bijzonder naar de lijst- en db-klassen die daar in bod komen. We laten je inzien dat de klassen BinarySearchTree uit labo 4 en BinaryMinHeap uit labo 6 gelijkaardige db-klassen zijn.

## Studeren naar gelang het aantal studiepunten

De examens naderen, je hebt nog veel studeerwerk. Je plant om zoveel mogelijk vakken te studeren en studeert dus eerst de vakken met het minst aantal studiepunten.

1. Welke datastructuur gebruik je om de vakken op te slaan zodat je
  - snel het vak met het minst aantal studiepunten terugvindt
  - snel dat vak kan verwijderen
  - snel opnieuw het vak met het minst aantal studiepunten terugvindt
2. Maak een nieuw Java Project ‘VakkenStuderen’ aan. Organiseer de klassen nu zoals je leerde in de lessen BOP en Web2.
  - a) Maak een package `domain.model` waar je de gegeven klasse `Vak` in plaatst.
  - b) Maak een package `domain.db` en de klasse `VakkenLijst`. Deze klasse heeft als instantieveranderlijke de lijst met vakken, bewaard in de door jou gekozen datastructuur. Waarschijnlijk heb je gekozen voor een boomstructuur. Kopieer de overeenstemmende methodes uit één van de voorgaande labo’s naar de nieuwe klasse. De klasse die je nu aanmaakt is dus een lijst-klasse die je kent uit BOP of de db-klasse die je kent uit Web 2.
  - c) Maak een package `ui` aan die de user interface klasse bevat. Zet de gegeven klasse `VakkenMain` in deze package. In dit project is de user interface een main methode die gegevens wegschrijft naar het scherm met het commando `System.out.println()`.
3. Schrijf code in de main methode zodat de gegeven vakken opgeslagen/toegevoegd worden in een instantie van de db-klasse.
4. Schrijf naar het scherm welk vak je eerst gaat studeren (i.e. het vak met het minst aantal studiepunten).

Voorbeeld van uitvoer:  
Dit vak moet je eerst studeren: Algo met 3 studiepunten.
5. Als je klaar bent met Algo, voeg je in de main een lijn code toe die dit vak uit de lijst met te studeren vakken verwijdert.
6. Laat de db-klasse berekenen welk vak je vervolgens moet studeren. Schrijf dit naar het scherm zoals in oefening 4.
7. Je bent vergeten dat je het vak `Computersystemen` (3 studiepunten) ook nog moet studeren. Voeg dit vak toe aan de lijst via de main-methode.
8. Schrijf naar het scherm welk vak het meest aantal studiepunten heeft.
9. (*moeilijker*) Schrijf alle resterende vakken uit waarbij de vakken vermeld worden in stijgende orde van aantal studiepunten.

10. Maak een nieuwe db- en main-klasse aan. Kies nu voor een andere datastructuur waarbij je
  - snel een overzicht van vakken kan gegeven, geordend naargelang het aantal studiepunten
  - zowel het vak met het grootste en het vak met het minste aantal studiepunten relatief snel kan vinden.
11. Laat de nieuwe main klasse naar het scherm schrijven welk vak respectievelijk minst en meest aantal studiepunten heeft.
12. Maak oefening [9](#) opnieuw.



# Oplossingen

**Oplossing 1.1** De definitie 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));
}
```

## Oplossingen

**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

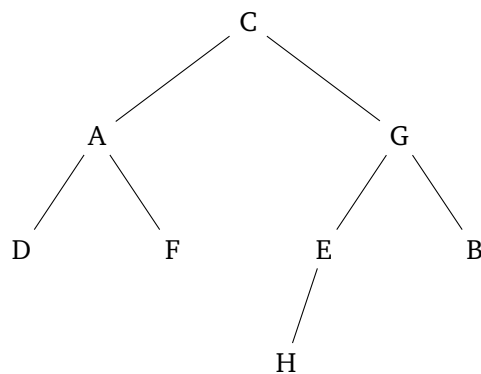
- a) Ja want elke knoop in deze boom heeft maximaal 2 kinderen.
- b) Knoop F
- c) Het maximaal aantal knopen van een pad van de wortel naar een blad is vier, dus de diepte van deze boom is vier.



- 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.

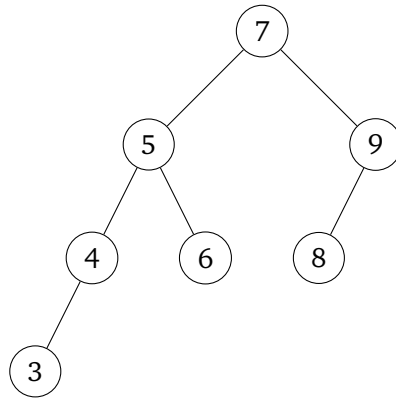
**Listing 21** Komt een bepaalde waarde voor in een boom?

```
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).

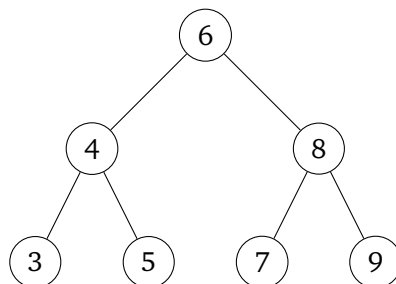
**Oplossing 3.2** We bekijken nog eens de definitie van binaire zoekboom: voor elke knoop in de boom geldt dat zijn waarde strikt groter is dan alle waarden in zijn linkersubboom en strikt kleiner dan alle waarden in zijn rechtersubboom. Deze volgorde is ook wat een in-order doorloop doet.

**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) {
        throw new IllegalArgumentException();
    }
    if (this.data.compareTo(data) == 0) {
        return false; //geen twee keer zelfde data in BST
    } else if (this.data.compareTo(data) > 0) { //ga naar linkersubboom
        if (this.leftTree == null) {
            this.leftTree = new BinarySearchTree<>(data);
            return true;
        }
    }
}
  
```

## Oplossingen

```
    } 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 lookUp(data) methode

```
public boolean lookup(E data) {  
  if (data == null) {  
    return false;  
  }  
  if (this.data.compareTo(data) == 0) {  
    return true;  
  } else {  
    if (this.data.compareTo(data) > 0) {  
      return (this.leftTree == null ? false : this.leftTree.lookup(data));  
    } else {  
      return (this.rightTree == null ? false : this.rightTree.lookup(data));  
    }  
  }  
}
```

### Oplossing 3.6

Listing 24 searchGreatest methode

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

### Oplossing 3.7

Listing 25 searchSmallest methode

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

## Oplossing 3.8

Listing 26 removeNode methode

```
public boolean removeNode(E data) {
    if (data == null) {
        throw new IllegalArgumentException();
    }
    if (this.data == null) {
        return false;
    }
    if (this.data.compareTo(data) == 0) {//data gevonden
        if (this.isLeaf()) {
            this.data = null;
            return true;
            // in dit geval blijft een leeg blaadje achter
            // clean kan dan enkel via gehele boom
        } else {
            if (this.leftTree != null) {//linkerboom is niet leeg
                E grootsteLinks = this.leftTree.searchGreatest();
                this.data = grootsteLinks;
                boolean verwijderenGelukt = this.leftTree.removeNode(grootsteLinks);
                if (verwijderenGelukt) {
                    this.leftTree.cleanUp();
                }
                return verwijderenGelukt;
            } else {//rechterboom is niet leeg
                E kleinsteRechts = this.rightTree.searchGreatest();
                this.data = kleinsteRechts;
                boolean verwijderenGelukt = this.rightTree.removeNode(kleinsteRechts);
                if (verwijderenGelukt) {
                    this.rightTree.cleanUp();
                }
                return verwijderenGelukt;
            }
        }
    }
} else {
    if (this.data.compareTo(data) > 0) {//zoek in linkerboom
        if (this.leftTree == null){
            return false;
        } else {
            boolean result = this.leftTree.removeNode(data);
            this.leftTree.cleanUp();
            return result;
        }
    } else {//zoek in rechterboom
        if (this.rightTree == null){
            return false;
        } else {
            boolean result = this.rightTree.removeNode(data);
            this.rightTree.cleanUp();
            return result;
        }
    }
}
```

```
}  
}
```

### Oplossing 3.9

Listing 27 ruimOp methode

```
private void cleanUp() {  
    if (this.data != null) {  
        if (this.leftTree != null) {  
            if (this.leftTree.data == null) {  
                this.leftTree = null;  
            } else {  
                this.leftTree.cleanUp();  
            }  
        }  
        if (this.rightTree != null) {  
            if (this.rightTree.data == null) {  
                this.rightTree = null;  
            } else {  
                this.rightTree.cleanUp();  
            }  
        }  
    }  
}
```

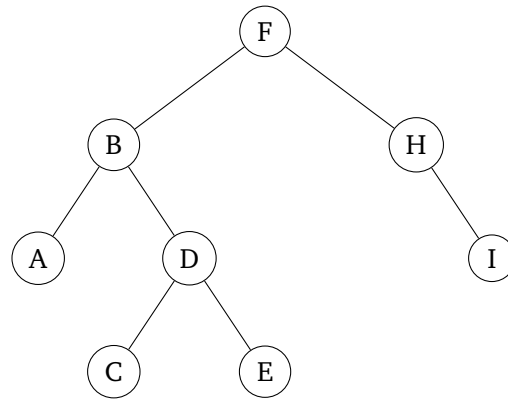
### Oplossing 3.10

Listing 28 getPath methode

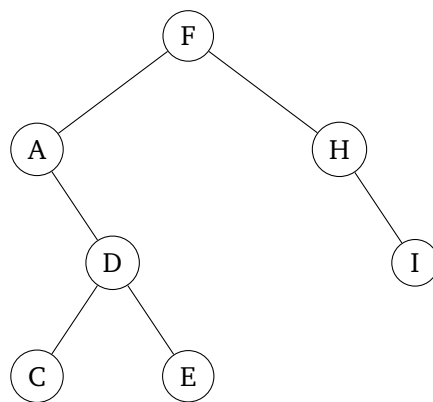
```
public ArrayList<E> getPath(E data) {  
    if (!lookup(data)) {//data komt niet voor in BST  
        return null;  
    }  
    ArrayList<E> pad = new ArrayList<>();  
    if (this.data.compareTo(data) == 0){  
        pad.add(data);  
        return pad;  
    } else {  
        pad.add(this.data);  
        if (this.data.compareTo(data) > 0) {//ga links, data komt zeker voor!  
            pad.addAll(this.leftTree.getPath(data));  
        } else {// ga rechts, data zit daar gegarandeerd  
            pad.addAll(this.rightTree.getPath(data));  
        }  
    }  
    return pad;  
}
```

Oplossing 3.11 Zie figuren [4](#), [5](#) en [6](#).

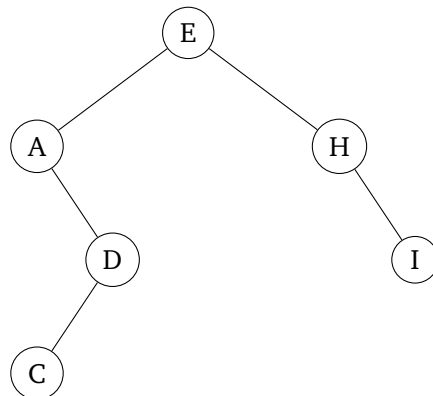




**Figuur 4** Boom 3.1 na verwijderen van knoop G



**Figuur 5** Boom 3.1 na verwijderen van knoop G en B

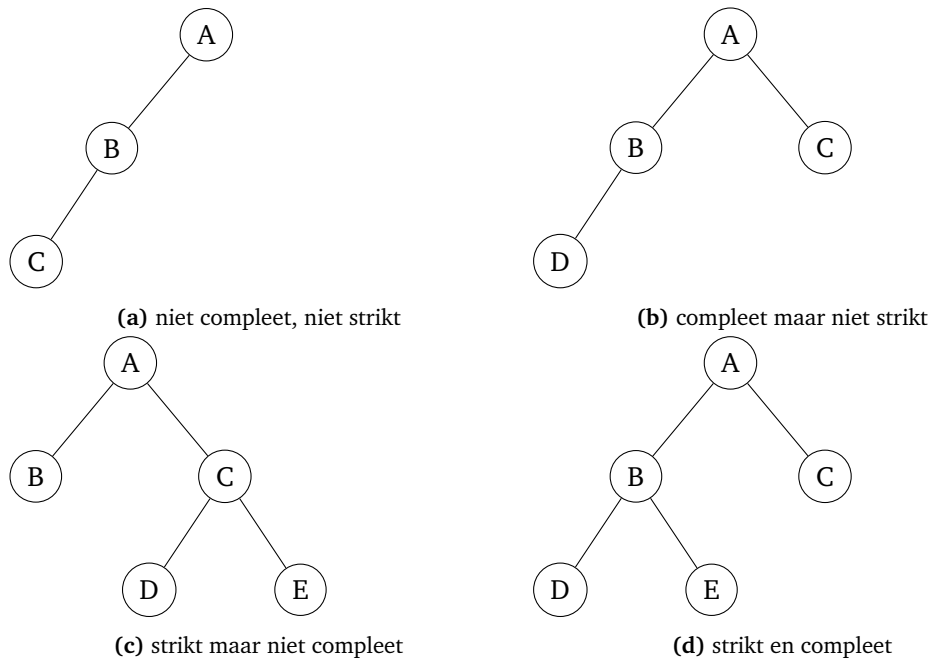


**Figuur 6** Boom 3.1 na verwijderen van knoop G, B en F

**Oplossing 4.1** De laatste stelling is juist, nl. geen van de vier vorige is correct. Figuur 7 toont

## Oplossingen

voor elke van deze vier stellingen een tegenvoorbeeld.



**Figuur 7** Tegenvoorbeelden voor de stellingen 1 t.e.m. 4

## Oplossing 4.2

**Listing 29** count methode

```
public int count(E geg) {  
    if (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 4.3** Postorder geeft: 15, 10, 23, 25, 20, 35, 42, 39, 30

Gebruik de preorder wandeling om de gevraagde boom te tekenen.

**Oplossing 4.4** Mogelijke oplossingen: nr 1, 3 en 4

- In nr 2 kan je knoop 60 niet tegenkomen in de zoektocht naar 43 want knoop 60 moet rechts van knoop 50 komen terwijl knoop 43 links moet staan.

- In nr 5 kan je knoop 18 niet tegenkomen omdat deze knoop links moet staan van knoop 27 terwijl je knoop 43 rechts moet zoeken.

## Oplossing 4.5

**Listing 30** 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 4.6** Je had al door dat dit een andere manier is om de vorige oefening in een opgave te gieten, niet?

## Oplossing 4.7

**Listing 31** kinderSom() methode

```
public class BinaryTreeInt {

    private int data;
    private BinaryTreeInt leftTree, rightTree;

    public BinaryTreeInt(int data) {
        this(data, null, null);
    }

    public BinaryTreeInt(int data, BinaryTreeInt leftTree, BinaryTreeInt rightTree) {
        this.data = data;
        this.leftTree = leftTree;
        this.rightTree = rightTree;
    }

    public boolean kinderSom() {
        // Controleer of wortel voldoet aan voorwaarden van kindersom
    }
}
```

## Oplossingen

```
// Als wortel een blaadje is, voldoet hij
if (this.isLeaf())
    // bij een blaadje is kinderSom altijd true
    return true;
// Als wortel geen blaadje is: controle uitvoeren
else {
    // Bereken som van de waardes van de kinderen
    // De somwaarde van een kind is gelijk aan zijn waarde,
    // tenzij het kind niet bestaat. Dan is de somwaarde gelijk aan 0
    int left_value = (this.leftTree != null ? this.leftTree.data : 0);
    int right_value = (this.rightTree != null ? this.rightTree.data : 0);
    // Als de som niet juist is, voldoet de boom niet aan de voorwaarden
    // Methode kan afgebroken worden
    if (this.data != left_value + right_value)
        return false;
}

// Als wortel wel voldoet aan voorwaarden, moet de rest van de boom gecontroleerd worden
return (this.leftTree != null ? this.leftTree.kinderSom() : true) &&
    (this.rightTree != null ? this.rightTree.kinderSom() : true);
}

...
}
```

### Oplossing 4.8

Deze boom kun je bekomen door volgend stappenplan:

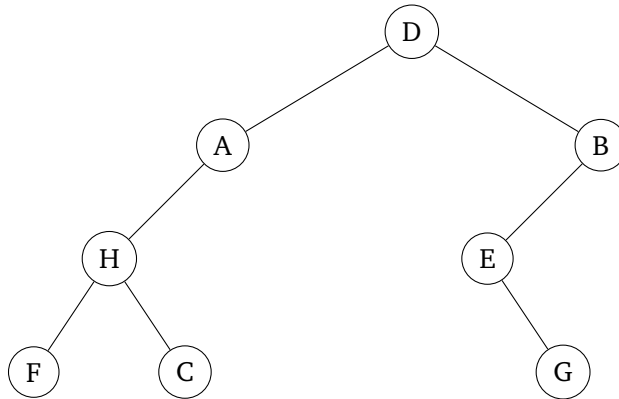
1. In *post-order* staat de root van de boom altijd achteraan.
2. Kijk in de *in-order* wandeling waar dat de root zich bevindt. Alles links van de root is de linker sub-boom en alles recht van de root is de rechter sub-boom.
3. Herhaal deze stappen opnieuw voor de linker en de rechter sub-boom.

Concreet voor deze opgave betekend dit:

1. Voor de gehele boom in *post-order* zien we dat 'D' achteraan staat en dus de root van onze boom gaat zijn.
2. De linker subboom bestaat uit 'FHCA' *in-order* en de rechter subboom uit 'EGB' *in-order*.
3. De linker subboom in *post-order* bestaat uit 'FCHA' en de rechter subboom in *post-order* bestaan uit 'GEB'.
4. Voor de linker subboom in *post-order* zien we dat 'A' achteraan staat en dus de root van onze subboom gaat zijn.
5. De linker sub-subboom bestaat uit 'FHC' *in-order* en de rechter sub-subboom is leeg.
6. De linker sub-subboom in *post-order* bestaat uit 'FCH' en de rechter sub-subboom is nog steeds leeg.

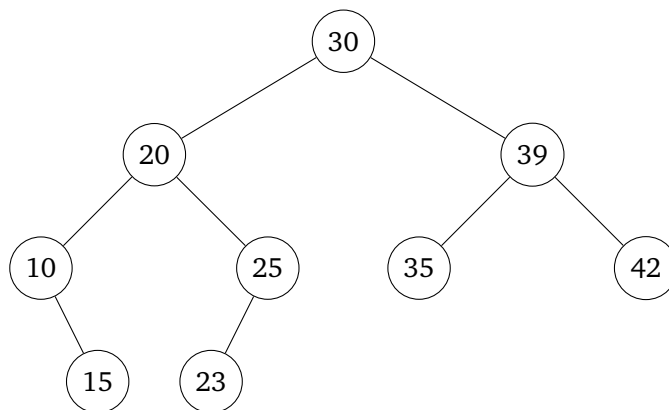
7. Voor de linker sub-subboom in *post-order* zien we dat 'H' achteraan staat en dus de root van onze volgende subboom gaat zijn.
8. Blijft dit herhalen tot links en rechts geen subbomen meer hebben.

Figuur 8 op pagina 77 toont de boom.



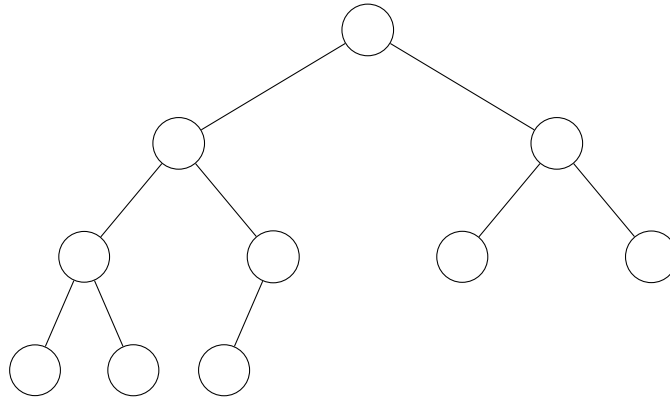
**Figuur 8** Binaire boom horend bij de gegeven in en post-order uitvoer

**Oplossing 4.9** 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 9. 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 9** BST horend bij de gegeven pre-order uitvoer

**Oplossing 4.10** Een goede manier om aan deze oefening te beginnen is een complete binaire boom tekenen met 10 knooppunten (figuur 10). Logisch redeneren met behulp van de

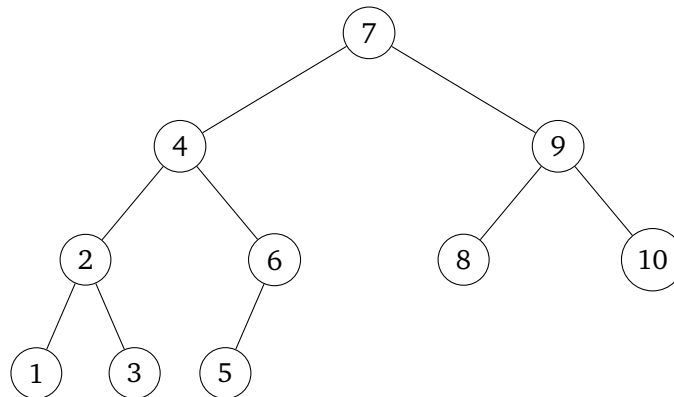


**Figuur 10** complete BST voor 10 getallen

basiskennmerken 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
- ...

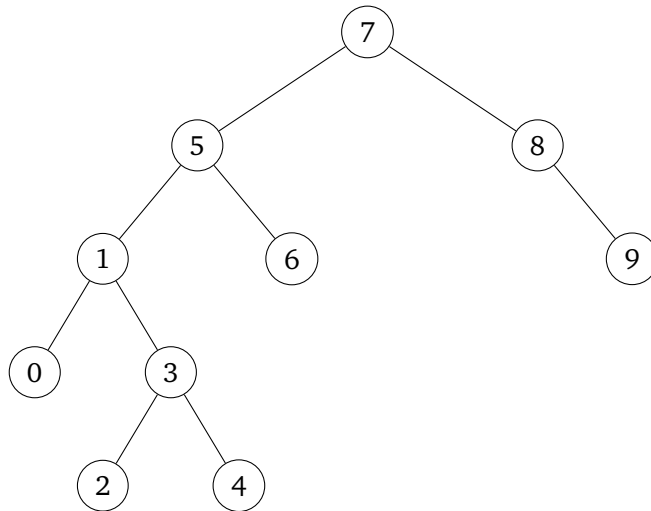
Je bekomt uiteindelijk figuur 11. Als je nu “laag per laag” aanvult komt alles waar het hoort



**Figuur 11** complete BST met getallen van 1 tot 10 ingevuld

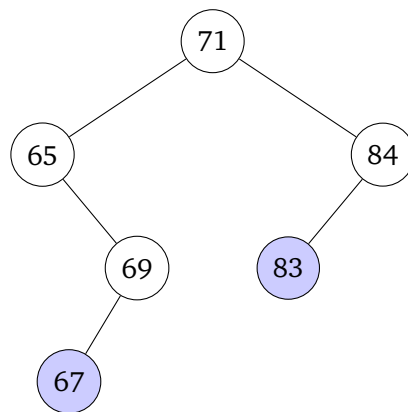
te staan. De volgorde wordt dus: 7, 4, 9, 2, 6, 8, 10, 1, 3, 5, of liever gezegd “één van de mogelijke volgordes”. Snap je dat bvb. 7, 9, 4, ... ook perfect mogelijk is?

**Oplossing 4.11** De resulterende boom heeft diepte 5 (figuur 12).



**Figuur 12** BST als resultaat van invullen in volgorde van 7, 5, 1, 8, 3, 6, 0, 9, 4 en 2

**Oplossing 4.12** De blaadjes van deze boom hebben als datavelden 67 en 83. De boom wordt getoond in figuur 13.



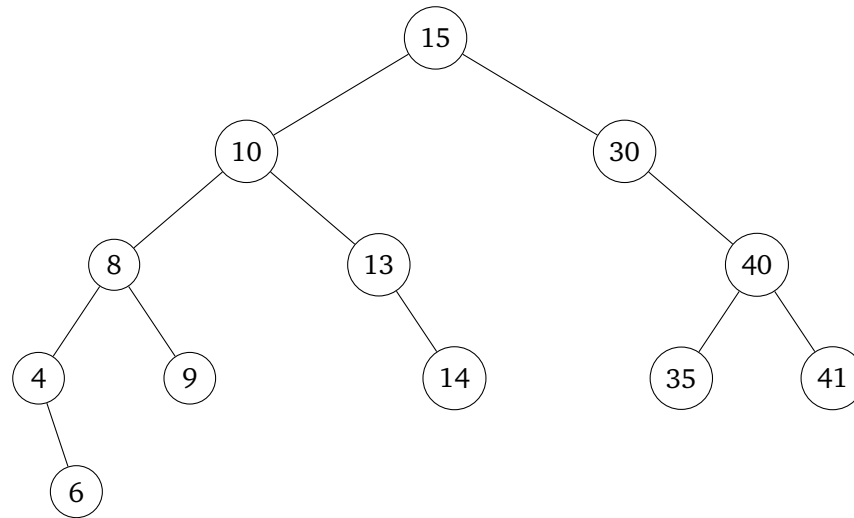
**Figuur 13** BST als resultaat van invullen in volgorde van 71, 65, 84, 69, 67 en 83

**Oplossing 4.13** 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 4.14** Figuur 14 op pagina 80 toont de oplossing. In de plaats van het getal 35 mag elk geheel getal tussen 30 en 40 komen (grenzen 30 en 40 niet inbegrepen). Wat het verwijderen van de knoop met waarde 10 betreft: ofwel verdwijnt het blaadje met waarde 9

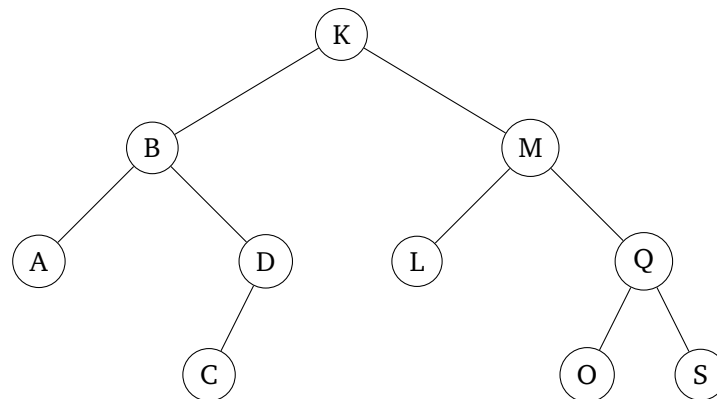
### Oplossingen

en komt de waarde in de knoop waar nu 10 staat. Ofwel schuift de knoop 13 eentje op en neemt de plaats in van de waarde 10.



**Figuur 14** Gewijzigde BST

**Oplossing 4.15** Figuur 15 op pagina 80 toont de oplossing van de eerste twee vragen. Voor het antwoord op de derde vraag kan ofwel de knoop met waarde L ofwel de knoop met waarde O de plaats innemen van M.



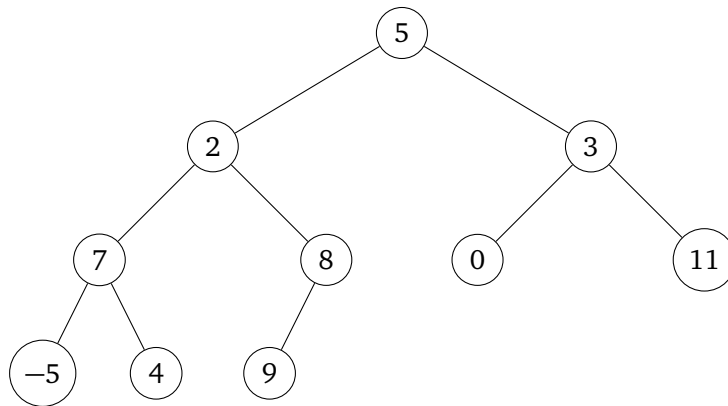
**Figuur 15** Gewijzigde BST

### Oplossing 4.16

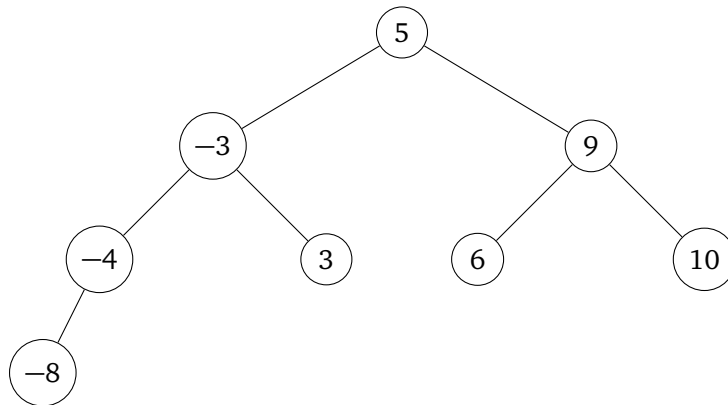
**Oplossing 4.17** De boom in figuur 16 op pagina 81 is de gevraagde complete binaire boom. Wat deel twee van de vraag betreft: er zijn vele mogelijke volgordes van getallen. Eén zo'n



volgorde is: 5, −3, 3, −4, −8, 9, 6, 10. Het volstaat om de laatste twee getallen in de andere volgorde te kiezen (dus eerst 10 en dan 6) en je bekomt dezelfde complete BST. Die wordt getoond in figuur 17 op pagina 81.



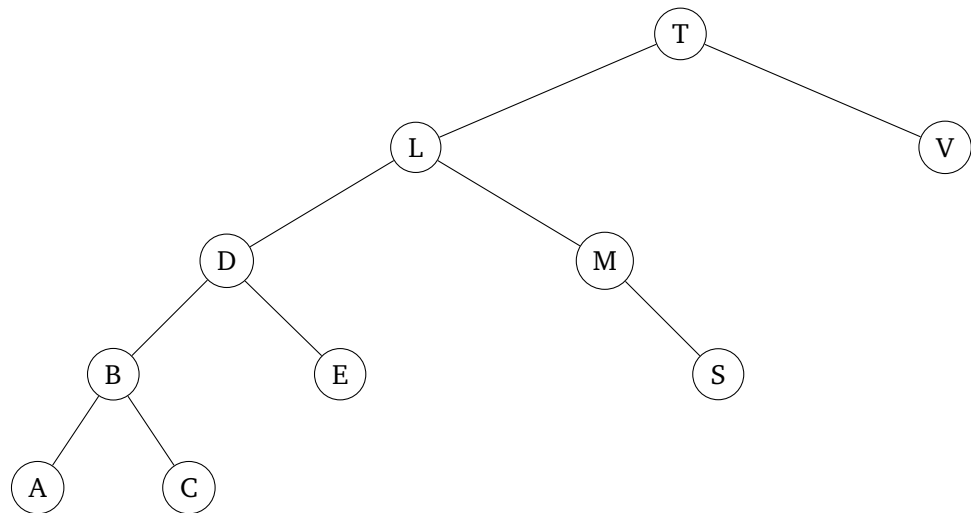
**Figuur 16** Oplossing voor de complete binaire boom



**Figuur 17** Deze complete BST bekom je

**Oplossing 4.18** Figuur 18 op pagina 82 toont de BST. Er zijn vele volgordes mogelijk om deze BST te construeren. Zo kan je laag per laag werken, of er voor kiezen om eerst de linkerboom af te werken en dan de rechter (of omgekeerd). Eén van vele mogelijk volgordes is: T, L, V, D, B, E, A, C, M, S. Het volstaat bvb. om de L en de V van plaats te verwisselen om een andere goede volgorde te bekomen. In arrayvoorstelling geeft dit het volgende (we stellen een lege cel voor door het symbool \*): [T, L, V, D, M, \*, \*, B, E, \*, S, \*, \*, \*, \*, A, C].

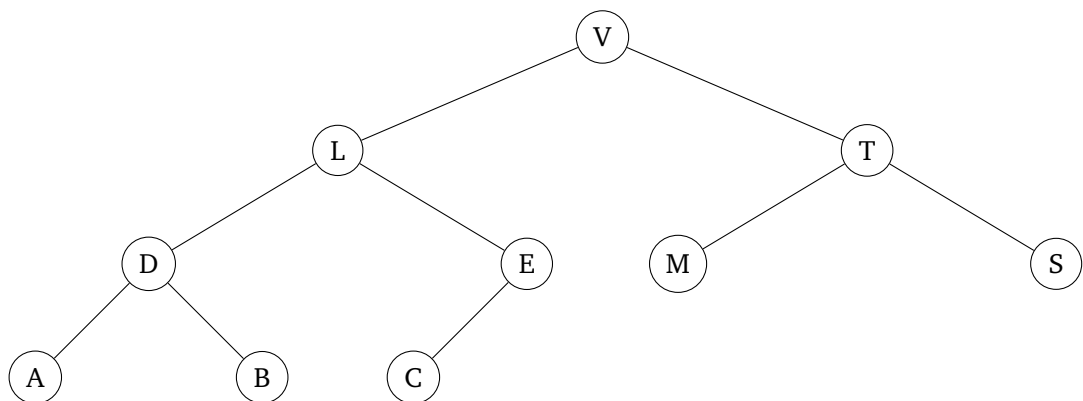
Om deze 10 dossiers voor te stellen in een heap zijn er vele mogelijkheden. Figuur 19 op pagina 82 toont er eentje. Om diepte 13 te bereiken moet je veel nodes toevoegen, want de boom moet compleet zijn. Op elk niveau staat een ‘macht van 2’ aantal getallen. Op diepte 4 moeten er  $2^3$  knopen zijn. Er staan er al drie, dus je voegt er nog  $2^3 - 3$  toe. Op de



**Figuur 18** BST voor studentendossiers

volgende dieptes voeg je  $2^4 + 2^5 + \dots + 2^{11}$  knopen toe. Tenslotte op diepte 13 nog 1 knoop. We berekenen dus volgende som:

$$2^3 - 3 + 2^4 + 2^5 + 2^6 + \dots + 2^{10} + 2^{11} + 1 = 4086$$

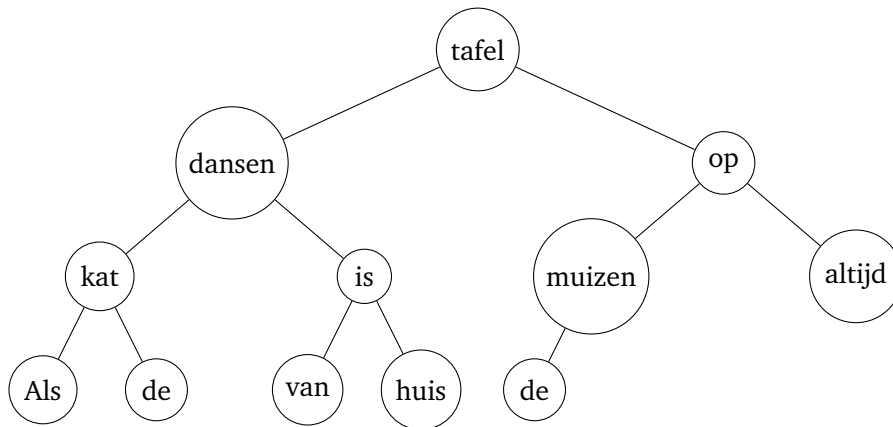


**Figuur 19** Max-heap voor studentendossiers

#### Oplossing 4.19

1. “kat de krollen de krabt . van trap de”
2. Figuur 20 op pagina 83 toont de complete binaire boom.

#### Oplossing 4.20



**Figuur 20** Als de kat van huis is dansen de muizen altijd op tafel

```

public BinaryTree<E> deelZonder(E wortelInfo) {
    if (wortelInfo == null || !this.contains(wortelInfo))
        return null;
    // wortelInfo == data, dus hele boom verwijderen
    if (this.data == wortelInfo) {
        return null;
    }
    // wortelInfo = linkerkind van data, dus hele linkertak verwijderen
    if (this.leftTree != null && this.leftTree.data.equals(wortelInfo)){
        return new BinaryTree(this.data,null,this.rightTree);
    }
    // wortelInfo == rechterkind van data, dus hele rechtertak verwijderen
    if (this.rightTree != null && this.rightTree.data.equals(wortelInfo)){
        return new BinaryTree(this.data,this.rightTree,null);
    }
    BinaryTree newLeftTree, newRightTree;
    // als wortelInfo in linkertak zit: nieuwe linkertak maken zonder wortelInfo
    if (this.leftTree != null && this.leftTree.contains(wortelInfo))
        newLeftTree = this.leftTree.deelZonder(wortelInfo);
    // anders linkertak behouden
    else newLeftTree = this.leftTree;
    // idem voor rechtertak
    if (this.rightTree != null && this.rightTree.contains(wortelInfo))
        newRightTree = this.rightTree.deelZonder(wortelInfo);
    else
        newRightTree = this.rightTree;
    // resulterende boom terug samenstellen
    return new BinaryTree<>(this.data,newLeftTree,newRightTree);
}

```

#### Oplossing 4.21

```

public boolean isStrict() {

```

```
// wortel == blaadje
if (this.isLeaf())
    return true;
// wortel heeft linker- en rechtertak, dus rest van boom controleren
if ((this.leftTree != null && this.rightTree != null)) {
    return this.leftTree.isStrict() && this.rightTree.isStrict();
}
// bij wortel ontbreekt linker- of rechtertak
return false;
}
```

**Oplossing 4.22** Werkwijze: Bekijk de code van de methode `printInOrder()`. In die methode schrijf je eerst recursief de waarden uit van de linkerboom, dan de waarde van de wortel, dan behandel je de rechterboom. Hier kan je ook zo te werk gaan. De waarden zullen dan van klein naar groot geordend zijn.

In tegenstelling tot `printInOrder()` schrijven we de waarden nu niet uit naar de console, maar verzamelen we ze in een lijst. Dat is gelijkend op hetgeen je deed in de methode `getDataLeaves()`.

Tenslotte hoef je niet steeds te zoeken over het volledige interval. Immers, in een BST zijn de elementen links van wortel kleiner dan wortel en rechts ervan groter. Je kan het interval waarin je zoekt dus steeds kleiner maken.

```
public class BinarySearchTree<E extends Comparable<E>> {
    private BinaryTree<E> root;

    ...

    public List<E> geefKnopenBinnenInterval(E min, E max) {
        if (this.isEmpty())
            throw new IllegalStateException("Empty tree");
        return this.root.geefKnopenBinnenInterval(min, max);
    }

    private class BinaryTree<E extends Comparable<E>> {
        private E data;
        private BinaryTree<E> leftTree, rightTree;

        ...

        public List<E> geefKnopenBinnenInterval(E min, E max) {
            if (min == null || max == null)
                throw new IllegalArgumentException("Geen effectief interval");
            List<E> result = new ArrayList<>();
            if (min.compareTo(max) > 0)
                return result;

            // zoek eerst waarden in linkerboom
            // je hoeft niet over volledig interval te zoeken
        }
    }
}
```

```

        // want alle knopen in linkerboom zijn kleiner dan wortel
        if (this.leftTree != null)
            result.addAll(this.leftTree.geefKnopenBinnenInterval(min, getMinimum(this.data, max)));
        // controleer of data in gevraagde interval zit
        if (this.data.compareTo(min) >= 0 && this.data.compareTo(max) <= 0)
            result.add(this.data);
        // behandel rechterboom
        if (this.rightTree != null)
            result.addAll(this.rightTree.geefKnopenBinnenInterval(getMaximum(min, this.data), max));
        return result;
    }

    private E getMinimum(E object1, E object2) {
        if (object1.compareTo(object2) <= 0)
            return object1;
        else
            return object2;
    }

    private E getMaximum(E object1, E object2) {
        if (object1.compareTo(object2) >= 0)
            return object1;
        else
            return object2;
    }
}

```

**Oplossing 5.1** 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 5.1 is:

[F, B, G, A, D, 0, I, 0, 0, C, E, 0, 0, H]

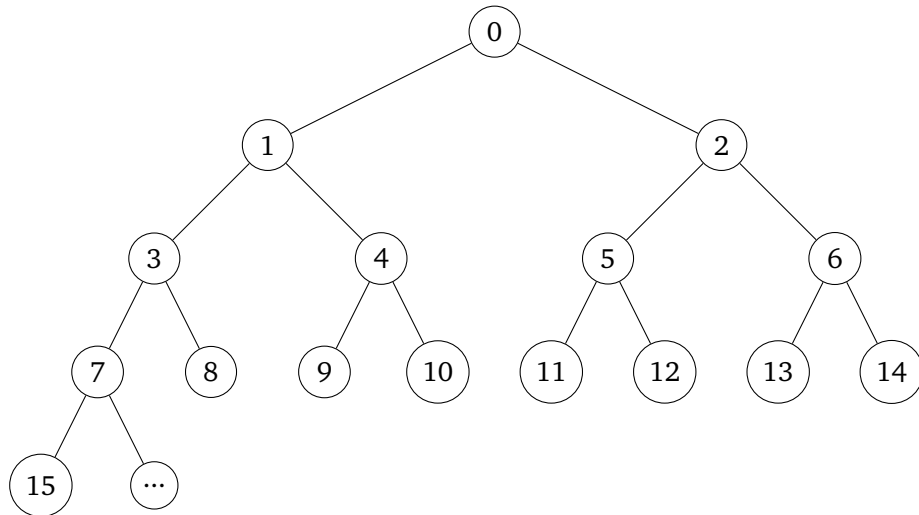
Voor figuur 5.2: [3, 2, 5, 1, 4]. Aangezien dit een complete binaire boom is, bevat de array geen “nullen”.

**Oplossing 5.2** 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 21. 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.

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:



**Figuur 21** Op weg naar 1000 knooppunten, genummerd van 0 tot 999

**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

- 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, meer bepaald de tiende knoop op dit niveau. De rechterbuur van 72 is natuurlijk knoop 73. Beide knopen hebben een verschillende ouder (welke?).
- 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 5.3

**Listing 32** kleinste waarde van min-heap

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

### Oplossing 5.4

**Listing 33** 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 5.5

Listing 34 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 5.6

Listing 35 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 {

```

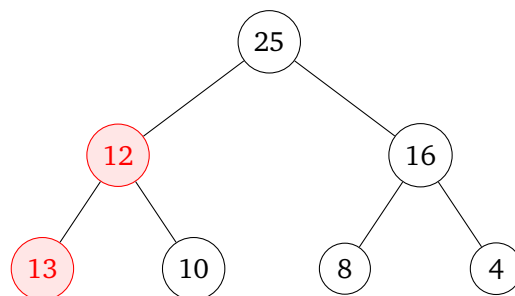


```

//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;
}
}

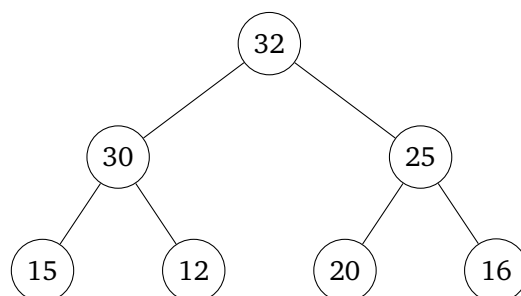
```

**Oplossing 6.1** 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 krijgt figuur 22. 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 22** Foute max-heap uit a)

**Oplossing 6.2** Voeg alle elementen element per element toe, gevolgd door een eventuele “bubble up”. Je krijgt figuur 23 en dus antwoord a).

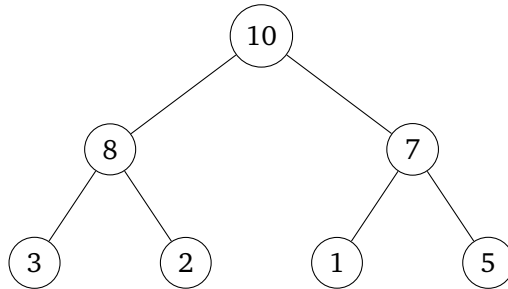


**Figuur 23** Max-heap met de getallen 32, 15, 20, 30, 12, 25 en 16 in die volgorde toegevoegd

**Oplossing 6.3** Diepte 9.

In een binaire min-heap van 1023 elementen zijn 10 lagen. Daarom kan je de getallen 2 tot en met 9 telkens als linkerkind van het vorige toevoegen.

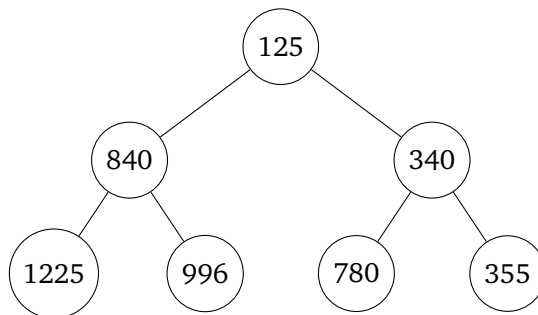
**Oplossing 6.4** Figuur 24 toont de originele boom plus de toevoeging van de getallen 1 en 7. Het getal 1 wordt gewoon het linkerkind van 5, 7 wordt het rechterkind met als gevolg dat de 7 'naar boven opborrelt' en dus gewisseld wordt met 5. De level-order output wordt dan: 10, 8, 7, 3, 2, 1, 5.



**Figuur 24** Max heap die bewandelt wordt in 'level order'

**Oplossing 6.5**

- een binaire min-heap: vliegtuig met minste brandstof moet eerst van de hoop gehaald kunnen worden
- [125, 840, 340, 1225, 996, 780, 355]



**Figuur 25** Min-heap van vliegtuigen

- [780, 840, 996, 1225]

**Oplossing 6.6**

1. Op diepte 1 staat er 1 element, op diepte 2 zijn er 2, op diepte 3 zijn er 4. Op diepte 4 staan er 8. Dat zijn dus allemaal machten van 2. Reken na dat er op diepte  $i$  juist  $2^{i-1}$  getallen kunnen staan. Laten we nu even het *cumulatieve aantal* getallen bekijken, d.w.z. het aantal getallen op een diepte gelijk aan het gegeven getal of een hoger niveau (met lagere diepte). Als voorbeeld: op diepte 4 en alle dieptes erboven (3, 2, 1) staan er in totaal 15 elementen. Kijk zelf na dat het hier telkens gaat over 1 minder dan een macht van 2. Het aantal mogelijke getallen op een diepte  $i$  of hoger is dus gelijk aan  $2^i - 1$ . Als  $i = 9$  als voorbeeld kan je berekenen dat op dit niveau en alle niveau's erboven samen  $2^9 - 1 = 511$  getallen kunnen staan. Van 9999 naar 1 zijn in totaal 10000 getallen. Als  $i = 13$  komen we aan een cumulatief aantal getallen van  $2^{13} - 1 = 8191$ , wat kleiner is dan 10000. Het getal 1 staat dus op diepte 14.
2. Het feit dat een binaire max-heap compleet is: er zijn dus geen “lege plaatsen” in de array
3. Op index 0 staat het getal 9999, op index 1 het getal 9998 enz. Je merkt dat de som van de index en het getal zelf altijd 9999 is. Dat wil dus zeggen dat het getal 1000 in de array op index  $9999 - 1000 = 8999$  staat.
4. Het ouderelement van het element 1000 dat op index 8999 staat is  $\frac{8999-1}{2}$  afgerond naar beneden, dus index 4499. De waarde van het element op deze index van de array is dus  $9999 - 4499 = 5500$ .
5. De twee kinderen van het element met waarde 1000 (op index 8999) zijn de getallen op indices  $8999 \cdot 2 + 1$  en  $8999 \cdot 2 + 2$ . Vermits de array maar tot index 9999 gaat, heeft de knoop met waarde 1000 geen kinderen in dit schema.

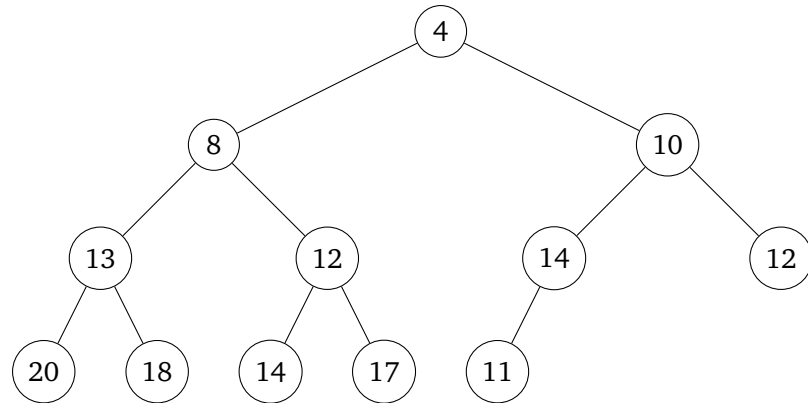
### Oplossing 6.7

1. 0, 2, 6, 14, 30, 61, 124
2. 89, 90, 179, 180, 181, 182, 359 (de 360ste knoop heeft index 359)
3. neen, want de laatste knoop is kleiner dan zijn ouder (zie figuur 26 op pg 92)
4. [8, 13, 12, 20, 18, 14, 17]: is wel een min-heap (zie figuur 27)

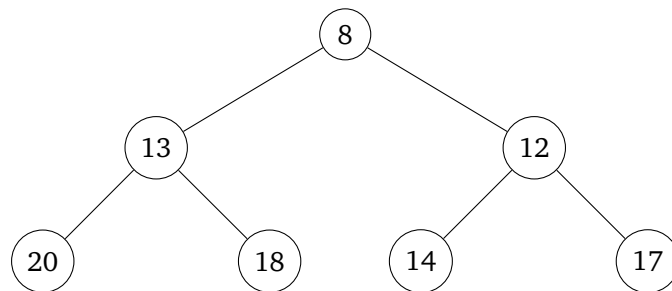
### Oplossing 7.1 $1 \rightarrow 2 \rightarrow 5 \rightarrow 8$ of $1 \rightarrow 4 \rightarrow 7 \rightarrow 8$

### Oplossing 7.2

1. Een verbindingsmatrix is correct indien
  - het aantal rijen gelijk is aan het aantal kolommen
  - de diagonaal van linksboven naar rechtsonder overal nul is
  - alle elementen gelijk zijn aan 0 of 1



**Figuur 26** Oplossing bij oefening 2 van 6.7 - geen min-heap



**Figuur 27** Oplossing bij oefening 3 van 6.7 - wel min-heap

2. De waarden van de verbindingsmatrix kunnen slechts gelijk zijn aan 0 en 1. Het vraagt minder geheugenruimte en het werkt efficiënter als de elementen als een boolean voorgesteld worden.

### Oplossing 7.3

Listing 36 findAncestors(int,int)

```
private boolean rechtstreekseVerbinding(int van, int tot) {
    //System.out.println("verbinding van "+van+" tot "+tot+"?");
    return this.getVerbindingsMatrix()[van - 1][tot - 1];
}

private int[] findAncestors(int start, int destination) {
    int aantalKnopen = this.getAantalKnopen();
    int[] ancestors = new int[aantalKnopen];
    initArray(ancestors, infty);

    Queue<Integer> queue = new LinkedList<>();
    queue.add(start);
    ancestors[start - 1] = 0;

    int huidig = queue.remove();
    while (huidig != destination) {
        //System.out.println("huidig = "+huidig);
        //zoek alle nog niet bezochte knooppunten vanuit huidig
        for (int i = 1; i <= aantalKnopen; i++) {
            if (rechtstreekseVerbinding(huidig, i) && (ancestors[i - 1] == infty)) {
                //System.out.println("ja");
                //voeg knoop i toe aan queue
                queue.add(i);

                //duid aan dat huidig de ouder is van i in ancestorematrix
                ancestors[i - 1] = huidig;
            }
        }
        //voerste element van queue wordt nieuwe huidige knoop
        if (!queue.isEmpty()) {
            huidig = queue.remove(); //of .poll() wat geen exception gooit
        } else {
            //queue is leeg, stop maar
            break;
        }
    }
    return ancestors;
}

public boolean[][] getVerbindingsMatrix() {
    return verbindingsMatrix;
}
```

## Oplossingen

```
private void initArray(int[] array, int value) {
    for (int i = 0; i < array.length; i++)
        array[i] = value;
}
```

### Oplossing 7.4

Listing 37 findPath

```
public List<Integer> findPath(int start, int destination) {
    if (start <= 0 || start > this.getAantalKnopen() || destination <= 0 ||
        destination > this.getAantalKnopen())
        throw new IllegalArgumentException();

    int[] ancestors = this.findAncestors(start, destination);
    List<Integer> path = new LinkedList<>();

    int ouder = ancestors[destination - 1];
    while (ouder != 0 && ouder != infy) {
        path.add(0, destination);
        destination = ouder;
        ouder = ancestors[destination - 1];
    }
    if (ouder == 0) {
        path.add(0, destination);
    }
    return path;
}
```

**Oplossing 8.1** Gewichtenmatrix  $D^{(6)}$  en bijhorende pointermatrix  $P$ :

$$D^{(6)} = \begin{bmatrix} 0 & 6 & 3 & 7 & 4 & 6 \\ 2 & 0 & 5 & 9 & 6 & 8 \\ 5 & 3 & 0 & 4 & 1 & 3 \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & 11 & 8 & 3 & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 3 & 0 & 6 & 3 & 5 \\ 0 & 0 & 1 & 6 & 3 & 5 \\ 2 & 0 & 0 & 6 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden:  $2 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6$  met lengte 8;  $5 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 4$  met lengte 3

### Oplossing 8.2

Listing 38 getPointerMatrix

```
public int[][] getPointerMatrix() {
    int aantal = this.gewichtenMatrix.length;
    int[][] P = new int[aantal][aantal];
    //double[][] D = this.gewichtenMatrix.clone(); fout = shallow clone
    //http://stackoverflow.com/questions/9106131/how-to-clone-a-multidimensional-array-in-java
}
```

```

//of manuele versie in de nieuwe opgave op toledo
//argument voor deze clone: is gezien in OOP
double[][] D = this.gewichtenMatrix.clone();
for (int i = 0; i < D.length; i++) {
    D[i] = D[i].clone();
}

for (int k = 0; k < aantal; k++) {
    for (int i = 0; i < aantal; i++) {
        for (int j = 0; j < aantal; j++) {
            if (D[i][k] + D[k][j] < D[i][j]) {
                D[i][j] = D[i][k] + D[k][j];
                P[i][j] = k + 1;
            }
        }
    }
}
return P;
}

```

## Oplossing 8.3

Listing 39 getShortestPath

```

public List<Integer> getShortestPath(int van, int tot, int[][] P) {
    List<Integer> pad = new ArrayList<>();
    if (van == tot) {
        return pad;
    } else {
        int via = P[van - 1][tot - 1];
        if (via == 0){
            pad.add(van);
            pad.add(tot);
        } else {
            pad = getShortestPath(van, via, P);
            pad.remove(pad.size() - 1); //anders dubbel
            pad.addAll(getShortestPath(via, tot, P));
        }
    }
    return pad;
}

```

## Oplossing 8.4

Listing 40 berekenLengte

```

public int berekenLengte(List<Integer> pad) {
    int som = 0;
    int aantalKnopen = pad.size();
    int huidigeKnoop, volgendeKnoop;
}

```

## Oplossingen

```
for (int i = 0; i < aantalKnopen - 1; i++) {
    huidigeKnoop = pad.get(i);
    volgendeKnoop = pad.get(i + 1);
    som += this.gewichtenMatrix[huidigeKnoop - 1][volgendeKnoop - 1];
}

return som;
}
```

### Oplossing 9.1

Stad	Kortste afstand vanuit 3	Route
1	3	3 → 1
2	4	3 → 2
3	0	
4	9	3 → 1 → 4
5	6	3 → 2 → 5
6	9	3 → 2 → 5 → 6
7	14	3 → 2 → 5 → 7
8	18	3 → 1 → 4 → 8

### Oplossing 9.3

#### Listing 41 Dijkstra

```
private int getAantalKnopen() {
    return gewichtenMatrix.length;
}

private int[][] initMatrixDijkstra(int vanKnoop) {
    int[][] res = new int[this.gewichtenMatrix.length + 1][this.gewichtenMatrix.length];
    // laatste rij is rij met kortste lengtes vanuit vanKnoop

    // oefening 3.3
    for (int i = 0; i < getAantalKnopen(); i++) {
        for (int j = 0; j < getAantalKnopen(); j++)
            res[i][j] = gewichtenMatrix[i][j] != inf ? gewichtenMatrix[i][j] : 0;
        res[getAantalKnopen()][i] = inf;
    }
    for (int i = 0; i <= getAantalKnopen(); i++) {
        res[i][vanKnoop - 1] = 0;
    }
    return res;
}
```



```

public int[][] Dijkstra(int vanKnoop) {
    int[][] res = initMatrixDijkstra(vanKnoop);

    System.out.println("Initiele matrix: \n");
    printIntMatrix(res);

    // oefening 3.4
    // herhaal voor alle knopen
    for (int i = 0; i < getAantalKnopen() - 1; i++) {
        // zoek nieuwe minimale afstand
        int min = inf;
        int[] knopenpaar = {inf, inf}; // index die het nieuwe minimum is
        for (int j = 0; j < getAantalKnopen(); j++) {
            // herhaal voor alle knopen die al bezocht zijn
            if (res[getAantalKnopen()][j] != inf) {
                for (int k = 0; k < getAantalKnopen(); k++) {
                    // als knoop k+1 nog niet gevonden is,
                    // als er een verbinding is tussen knoop j+1 en knoop k+1
                    // en als de verbinding tussen deze knopen korter is
                    // dan het minimum tot nog toe
                    if (res[getAantalKnopen()][k] == inf && res[j][k] != 0 &&
                        res[getAantalKnopen()][j] + res[j][k] < min) {
                        // onthoud (index van) dit knopenpaar en hun minimum
                        knopenpaar[0] = j;
                        knopenpaar[1] = k;
                        min = res[getAantalKnopen()][j] + res[j][k];
                    }
                }
            }
        }
        // tussenresultaat wegschrijven indien er verbetering is
        if (knopenpaar[0] != inf && knopenpaar[1] != inf) {
            // nieuwe minimum
            res[getAantalKnopen()][knopenpaar[1]] = min;
            for (int j = 0; j < getAantalKnopen() - 1; j++) {
                // kolom op nul zetten, maar niet op de plaats die het minimum aanlevert
                if (j != knopenpaar[0])
                    res[j][knopenpaar[1]] = 0;
            }
        }
    }
    return res;
}

```

## Oplossing 9.4

### Listing 42 vindPad

```

private ArrayList<Integer> vindPad(int vanKnoop, int naarKnoop, int[][] res) {
    ArrayList<Integer> pad = new ArrayList<>();
    // oefening 3.5
    // naarKnoop, vanKnoop en k zijn namen van knopen

```

## Oplossingen

```
// hun index in de matrix is altijd eentje minder want de rijen/kolommen tellen vanaf 0
pad.add(naarKnoop);

while (naarKnoop != vanKnoop) {
    int k = 1;
    while (k - 1 < getAantalKnopen() && res[k - 1][naarKnoop - 1] == 0)
        k++;
    pad.add(0, k);
    naarKnoop = k;
}
return pad;
}
```

**Oplossing 10.1**  $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 10$  of  $1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 10$

**Oplossing 10.2**  $1 \rightarrow 4 \rightarrow 7 \rightarrow 9$

**Oplossing 10.3** Enkele paden:  $A \rightarrow D \rightarrow C \rightarrow B \rightarrow E$ ;  $C \rightarrow B \rightarrow E \rightarrow D \rightarrow A$ ;  $D \rightarrow C \rightarrow B \rightarrow E$

**Oplossing 10.4**

$$D^{(5)} = \begin{bmatrix} 0 & 9 & 10 & 24 & 21 \\ 9 & 0 & 19 & 33 & 30 \\ 2 & 11 & 0 & 14 & 11 \\ 7 & 16 & 5 & 0 & 3 \\ 4 & 13 & 2 & 3 & 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 1 & 5 & 3 \\ 0 & 1 & 0 & 5 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden:  $4 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 2$  met lengte 16;  $3 \rightarrow 5 \rightarrow 4$  met lengte 14

**Oplossing 10.8** Om deze oefening op te lossen bekijken we de verbindingsMatrix. Als er in de verbindingsmatrix een rij bestaat met allen maar nullen (false) en als overeenkomstige kolom alleen maar eentjes (true) bevat uitgezonderd waar op de plaats waar de rij-index gelijk is aan de kolom-index, dan hebben we te maken met een vergeetput.

**Listing 43** isVergeetput(verbindingsMatrix) methode

```
public Integer isVergeetput(boolean[][] verbindingsMatrix){
    for (int i = 0; i < verbindingsMatrix.length; i++){
        boolean[] fromRow = verbindingsMatrix[i];
        boolean vergeetputFound = true;
        for(int j = 0; j < verbindingsMatrix.length; j++){
            if(fromRow[j] || (!verbindingsMatrix[j][i] && i!=j)){
                vergeetputFound = false;
                break;
            }
        }
    }
}
```

```

        if(vergeetputFound){
            return i;
        }
    }
    return null;
}

```

**Oplossing 10.9** Deze oefening is opgelost in 2 delen.

Het eerste deel maakt een vertaling van ons probleem waarbij 1 persoon en 1 bedrag gegeven zijn naar het zelfde probleem waarbij er n personen zijn en waar dat iedere persoon een ander bedrag kan hebben.

Vervolgens roepen we een helper functie op die dat de verdeling zal maken. Deze verdeling gebeurt van links naar rechts en per diepte. Deze methode zal zichzelf recursief oproepen waarbij we steeds verder kijken naar vrienden die verder weg zijn van de oorspronkelijke winnaar.

**Listing 44** verdeel(i, bedrag) methode

```

public ArrayList<Double> verdeel(int i, double bedrag){
    ArrayList<Double> verdeelt = new ArrayList<>();
    ArrayList<Integer> delers = new ArrayList<>();
    for(int j =0; j<this.getAantalKnopen(); j++) {
        if(i == j){
            verdeelt.add(bedrag);
            delers.add(i);
        }
        else{
            verdeelt.add(0.0);
        }
    }
    return verdeelHelper(verdeelt,delers);
}
private ArrayList<Double> verdeelHelper(ArrayList<Double> verdeelt,
ArrayList<Integer> delers){
    if(delers.size() == 0){
        return verdeelt;
    }
    else{
        ArrayList<Integer> nieuweDelers = new ArrayList<>();
        for(int i = 0; i < delers.size(); i++){
            int deler = delers.get(i);
            boolean[] kinderen = this.verbindingsMatrix[deler];
            ArrayList<Integer> deelMetKinderen = new ArrayList<>();
            for(int j = 0; j<kinderen.length; j++){
                if(kinderen[j] && verdeelt.get(j) == 0.0){
                    deelMetKinderen.add(j);
                }
            }
            if(deelMetKinderen.size() != 0){
                double verdeelValue = verdeelt.get(deler)/2;
                verdeelt.set(deler,verdeelValue);
            }
        }
    }
}

```

## Oplossingen

```
        verdeelValue /= deelMetKinderen.size();
        for(int j = 0; j < deelMetKinderen.size(); j++){
            int kind = deelMetKinderen.get(j);
            verdeelt.set(kind, verdeelValue);
            if (!nieuweDelers.contains(kind)) nieuweDelers.add(kind);
        }
    }
    return verdeelHelper(verdeelt, nieuweDelers);
}
```

## Oplossing 10.10