




# 3

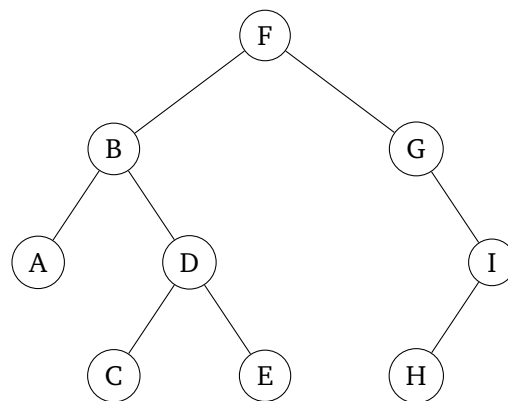
## Binaire zoekbomen (BST)

### Inleiding

Download het bestand `week03_BST_opgave.zip` van Toledo en importeer dit bestand in IntelliJ zoals aangegeven in de oefeningen van vorige week.

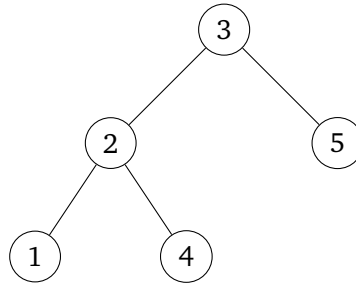
### Oefening 3.1

 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?

### 3 Binaire zoekbomen (BST)




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

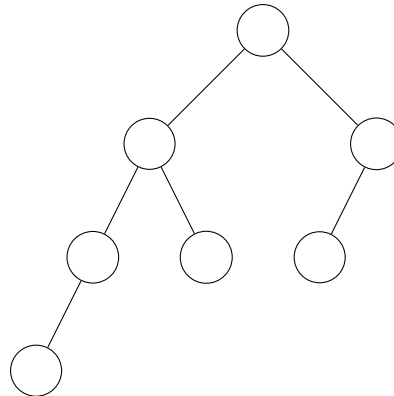
#### Oefening 3.2

 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.

- Teken op papier eerst de binaire zoekboom zoals die door de main methode in de klasse `BinarySearchTreeDriver` zal worden toegevoegd.
- Implementeer de `addNode` methode in de `BinarySearchTree` klasse. Maak optimaal gebruik van het feit dat de boom een binaire zoekboom is.
- 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 `BinaryTree` 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 `BinaryTree` 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 `BinaryTree` 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.
- 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 ...

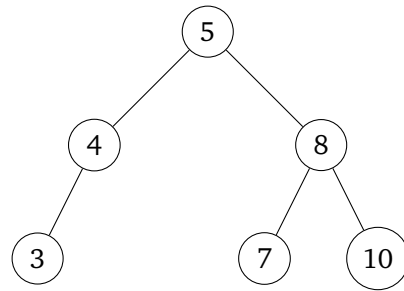
- Implementeer de `countNodes` methode in de `BinarySearchTree` klasse die het aantal knopen in een boom telt. Na de verwijdering van de knoop met data-veld 9 valt iets eigenaardigs op: het verwijderen van de knoop heeft geen effect op het aantal knopen. We onderzoeken dat in de volgende deelvragen.
- 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.


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



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

- 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 E.

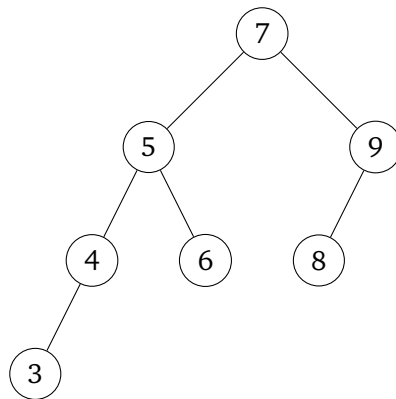


## Oplossingen

**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).

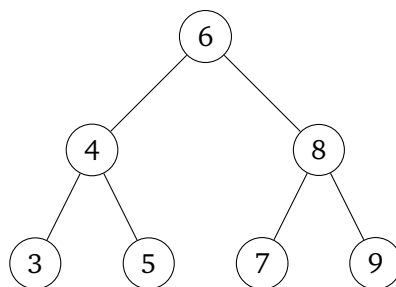
**Oplossing 3.2** We bekijken nog eens de definitie van binaire zoekboom: voor elke knoop in de boom geldt dat zijn waarde strikt groter is dan alle waarden in zijn linkersubboom en strikt kleiner dan alle waarden in zijn rechtersubboom. Deze volgorde is ook wat een in-order doorloop doet.

**Oplossing 3.3** Er is maar één mogelijke oplossing, nl. figuur 1.



**Figuur 1** Getallen van 3 t.e.m. 9 invullen in de knopen

**Oplossing 3.4** De main methode construeert de (gebalanceerde) BST uit figuur 2.



**Figuur 2** ResultaatBST van de main methode



**Listing 2** 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;
        } 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 3** 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 4** searchGreatest methode

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

### Oplossing 3.7

#### Listing 5 searchSmallest methode

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

### Oplossing 3.8

#### Listing 6 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}
            return (this.leftTree == null ? false : this.leftTree.removeNode(data));
        } else {//zoek in rechterboom}
            return (this.rightTree == null ? false : this.rightTree.removeNode(data));
        }
    }
}
```

```
}
```

### Oplossing 3.9

Listing 7 ruimOp methode

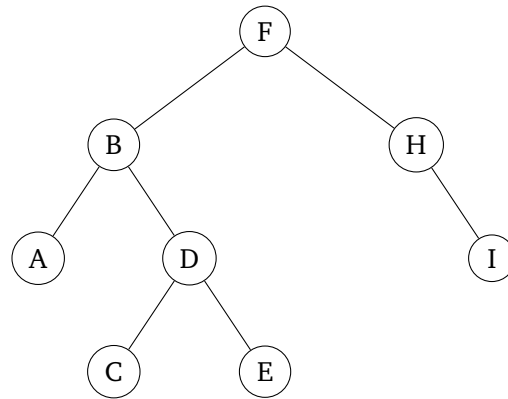
```
private void cleanUp() {
    if (this.data != null) {
        if (this.leftTree != null) {
            if (this.leftTree.data == null) {
                this.leftTree = null;
            } else {
                this.leftTree.cleanUp();
            }
        }
        if (this.rightTree != null)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

### Oplossing 3.10

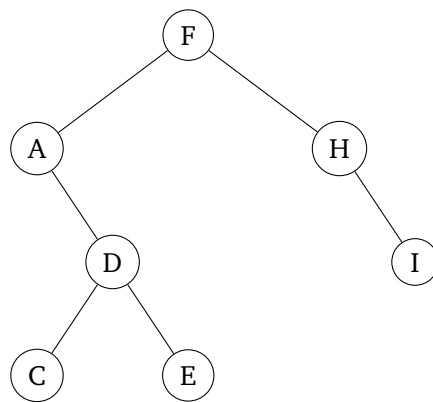
Listing 8 getPath methode

```
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;
}
```

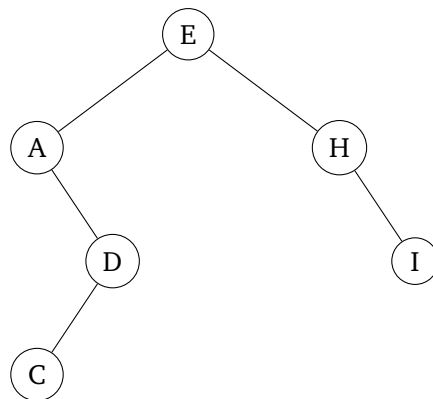
Oplossing 3.11 Zie figuren [3](#), [4](#) en [5](#).



**Figuur 3** Boom 3.1 na verwijderen van knoop G



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



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