ALGORITMES & DATASTUCTUREN

Joost Visser (jwjh.visser@avans.nl)

Academie voor Deeltijd

2 juni 2021

Opnemen

Deze lecture wordt opgenomen. Vinden jullie dit goed?

Les van vandaag

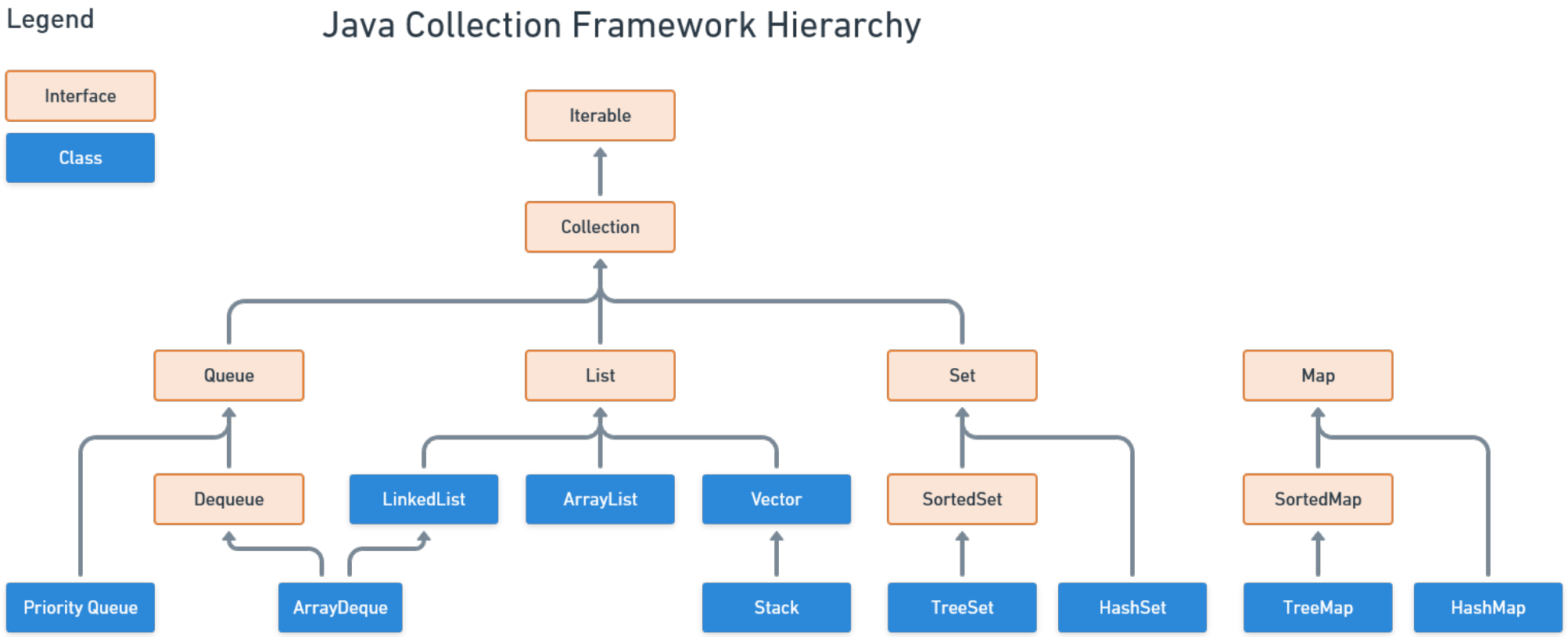
Lesplan

Les	Datum	Topic	Inhoud
1	4 mei 2021	Introductie	Inleiding, algoritme complexiteit, data-structuren, Linear zoeken, binair zoeken
2	11 mei 2021	Datastructuren 1	Herhaling algoritme complexiteit, Sorteren 1, Abstracte datatypen, Array, List, Set, Hashing
3	18 mei 2021	Datastructuren 2	Sets 2, Maps, Stack, Deque, Heap, Priority Queue
4	25 mei 2021	Algoritmes	**Generieke datatypen**, Sorteren 2, recursie, binaire bomen, proeftoets

Vragen?

Les van vandaag

Java datastructuren



Les van vandaag

Inhoudsopgave

1. Generics
2. Recursie [komt niet op de toets]
3. Binaire zoekbomen
4. Toets
    1. Wat komt er op de toets?
    2. Topics herhaling
    3. Proeftoets bespreking

Les van vandaag

Inhoudsopgave

1. Generics

Waarom generics?

Mijn eigen vogelbekdier printer

```
public static void main(String[] args) {
    ArrayList<Vogelbekdier> vogelbekdieren = …

    printer(vogelbekdieren);
}
```

```
public void printer(List<Vogelbekdier> vogelbekdieren) {
    for (Vogelbekdier vogelbekdier : vogelbekdieren) {
        System.out.println(vogelbekdier);
    }
}
```

```
public class Vogelbekdier {
    int lengte;
    String naam;

    public Vogelbekdier(int lengte, String naam) {
        this.lengte = lengte;
        this.naam = naam;
    }

    @Override
    public String toString() {
        …
    }
}
```

Wat nou als ik een printerfunctie wil maken voor **alle zoogdieren**?

1. Generics

Waarom generics?

Mijn eigen vogelbekdier printer

```
public static void main(String[] args) {
    ArrayList<Vogelbekdier> vogelbekdieren = …

    printer(vogelbekdieren);
}
```

```
public void printer(List<Zoogdier> zoogdierList) {
    for (Zoogdier zoogdier : zoogdierList) {
        System.out.println(zoogdier);
    }
}
```

```
public class Vogelbekdier extends Zoogdier {

    public Vogelbekdier(int lengte, String naam) {
        super(lengte, naam);
    }

    @Override
    public String toString() {
        …
    }
}
```

Werkt dit?

Nee!

1. Generics

Waarom generics?

We kunnen dit oplossen door gebruik te maken van generics.

Door gebruik te maken van generics kunnen we argumenten doorgeven van meerdere types.

Sterker nog, in de afgelopen lessen hebben we constant gebruik gemaakt van generics, bijvoorbeeld:

```
ArrayList<Vogelbekdier> vogelbekdieren = new ArrayList<>()
```

ArrayList accepteert elk generiek object om te gebruiken als lijst. Zo kunnen we een ArrayList van Integers, Strings of Vogelbekdieren maken.

1. Generics

Waarom generics?

```
public void printer(List<Zoogdier> zoogdierList) {
    for (Zoogdier zoogdier : zoogdierList) {
        System.out.println(zoogdier);
    }
}
```

```
public static void main(String[] args) {
    ArrayList<Vogelbekdier> vogelbekdieren = …

    printer(vogelbekdieren);
}
```

Hoe lossen we dit op?

We willen alle zoogdieren printen, waaronder vogelbekdieren.

```
public void printer(List<? extends Zoogdier> zoogdierList) {
    for (Zoogdier zoogdier : zoogdierList) {
        System.out.println(zoogdier);
    }
}
```

```
public <T extends Zoogdier> void printer(List<T> zoogdierList) {
    for (Zoogdier zoogdier : zoogdierList) {
        System.out.println(zoogdier);
    }
}
```

1. Generics

Waarom generics?

Je kunt generics op twee plekken gebruiken:

1. Generic methods en wildcards
2. Generic classes

1. Generics

Waarom generics?

Je kunt generics op twee plekken gebruiken:

1. **Generic methods en wildcards**
2. Generic classes

1. Generics

Generic methods en wildcards

Generic methods is handig om wat algemenere objecten mee te geven aan functies:

1. Algemene functies die voor een classe object werken
2. Functie die bijvoorbeeld een ArrayList van de inputobjecten teruggeeft.

Er zijn twee manieren om dit voor elkaar te krijgen:

1. Generic methods
2. Wildcards: ?

1. Generics

Generic methods en wildcards

Generic methods

Een generic method gebruikt een algemene <T> om een bepaalde Type aan te duiden, zoals Strings of Integers.

Deze declareer je vóór de return type:

```
public static <type> <return_type> functie(<parameters>)
```

Bijvoorbeeld:

```
public <T> void print(T param)
```

1. Generics

Generic methods en wildcards

Generic methods

Als je deze generieke type wilt bounden, bijvoorbeeld dat het een bepaalde classe moet extenden, dan kan je dat bij de type parameter aangeven:

```
public <T extends Iterable> void print(T param)
```

Daarnaast kan je deze T ook als return type geven:

```
public <T extends Iterable> T printAndReturn(T param)
```

En je kan zelfs helemaal funky gaan met meerdere types:

```
public <T, U> Set<T> getKeyset(Map<T, U> map)
```

1. Generics

Generic methods en wildcards

Wildcards

Soms is het makkelijker en overzichtelijker om Wildcards te gebruiken:

```
public <T, U> Set<T> getKeyset(Map<T, U> map)
```

Wordt dan:

```
public Set<?> getKeyset(Map<?, ?> map)
```

De wildcard kan je alleen gebruiken als het Collections of andere generic classes betreft. Het moet tussen <> zitten. Dit werkt dus niet:

```
public void print(? param)
```

Daar hebben we Object voor:

```
public void print(Object param)
```

1. Generics

Generic methods en wildcards

Generics vs Wildcards

Wat is het verschil tussen de volgende twee regels code?

```
public <T> List<T> appendLists(List<T> list1, List<T> list2)
  public List<?> appendLists(List<?> list1, List<?> list2)
```

Antwoord: bij de tweede functie hoeven de types niet hetzelfde te zijn, bijvoorbeeld `appendLists(List<String>, List<Integers>)`

Als laatste zijn er nog een paar andere, kleinere verschillen tussen de twee. Je kan ze zelfs combineren, zoals bij de officiële copy functie:

```
public static <T> void copy(List<T> dest, List<? extends T> src)
```

1. Generics

Waarom generics?

Je kunt generics op twee plekken gebruiken:

1. **Generic methods en wildcards**
    1. Deze gebruik je in plaats van één specifiek object, een wat algemenere object als parameter door te geven (of als return type).
2. Generic classes

1. Generics

Waarom generics?

Je kunt generics op twee plekken gebruiken:

1. Generic methods en wildcards
2. **Generic classes**

1. Generics

Generic classes

Generic classes hebben wij al regelmatig gebruikt:

```
ArrayList<String> myList = new ArrayList<>();
TreeMap<String, Vogelbekdier> myMap = new TreeMap<>();
```

Je gebruikt ze dus wanneer je generieke typen in je classes nodig hebt.

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

1. Generics

Waarom generics?

Je kunt generics op twee plekken gebruiken:

1. Generic methods en wildcards
2. Generic classes

1. Generics

Samenvatting

Generic methods zijn methods die generieke objecten als parameters kunnen accepteren. Deze types definieer je voor de return type:

```
public <T> void print(T param)
public <T, U> Set<T> getKeyset(Map<T, U> map)
```

Soms is het ook handig om wildcards te gebruiken:

```
public void print(List<?> anyList)
```

We kunnen ook classes met generieke types maken:

```
public class Box<T> { … }
```

1. Generics

Generic methods en wildcards

Opgave 1: Implementeer een arrayToList() functie, die voor elke type list kan accepteren en een array kan teruggeven.

Opgave 2: Implementeer een generieke class `Pair<T, U>` voor objecten van type T en U zodat ik een ArrayList aan pairs bij kan houden:

1. Implementeer `Pair<T, U>`
2. Maak een ArrayList aan pairs
3. Print elke Pair in de ArrayList naar System.out.

Opgave 3 [extra]: Gegeven mijn IntLinkedList implementatie, zorg ervoor dat het generic types kan accepteren in plaats van alleen Integers.

Les van vandaag

Inhoudsopgave

1. Generics
2. **Recursie [komt niet op de toets]**
    1. Wat is recursie?
    2. Voorbeeld 1: Recursieve binary search
    3. Voorbeeld 2: MergeSort
3. Binaire zoekbomen
4. Toets

2. Recursie

Wat is recursie?

Wat gebeurt er als een functie zichzelf callt? Dit heet **recursie**.

Zijn jullie bekend met de Fibonacci reeks?
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Je telt de vorige getal en het getal daarvoor op.

2. Recursie

Wat is recursie?

Zijn jullie bekend met de Fibonacci reeks?

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

We zouden dit met een for-loop kunnen implementeren:

```
public static int fibonnaci(int n) {
    int int0 = 1;
    int int1 = 1;
    int curInt = 1;

    for (int i = 1; i < n; i++) {
        curInt = int0 + int1;
        int0 = int1;
        int1 = curInt;
    }
    return curInt;
}
```

2. Recursie

Wat is recursie?

Zijn jullie bekend met de Fibonacci reeks?

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Of we kunnen het veel eleganter implementeren:

```
public static int fibonnaciRecursive(int n) {
    if (n <= 1)
        return 1;
    return fibonnaciRecursive(n-1) + fibonnaciRecursive(n-2);
}
```

Dit werkt, maar hier zit een nadeel aan.
Wat is het nadeel? Weet je een mogelijke oplossing hiervoor?

2. Recursie

Wat is recursie?

In recursieve functies roep je dus jezelf aan.

Dit wordt erg veel in grafenalgoritmes gebruikt (out-of-scope), maar ook in andere algoritmes.

Tijd voor twee voorbeelden:

1. Recursieve binary search
2. MergeSort

2. Recursie

Voorbeeld 1: Recursieve Binary Search

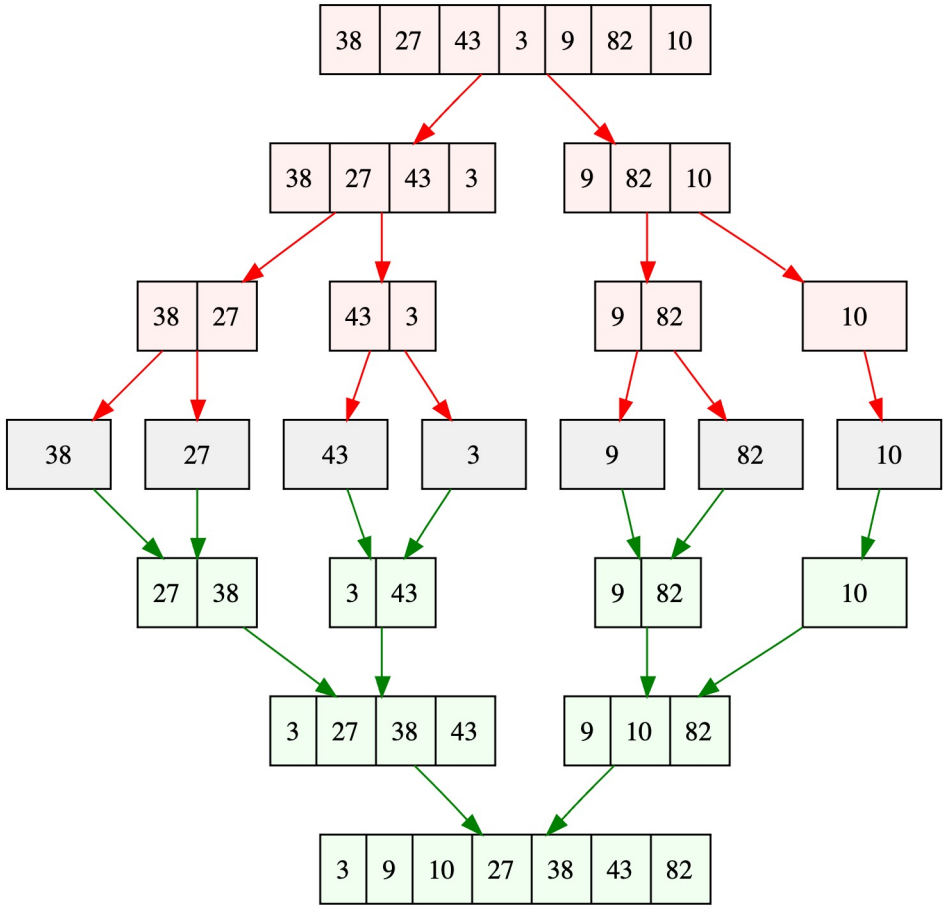
Weten jullie Binary Search nog? We kunnen die ook herschrijven als recursieve functie:

```
int binarySearch(int numberToFind, int[] numbers, int low, int high) {
    if (low >= high)
        return -1;

    int mid = (int) ((low + high) / 2);

    if (numberToFind == numbers[mid])
        return mid;
    else if (numberToFind < mid)
        return binarySearch(numberToFind, numbers, low, mid);
    else
        return binarySearch(numberToFind, numbers, mid, high);
}
```

2. Recursie

Voorbeeld 2: MergeSort

Nog een sorteeralgoritme!



2. Recursie

Voorbeeld 2: MergeSort

Nog een sorteeralgoritme!

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = l+ (r-l)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

https://visualgo.net/en/sorting

2. Recursie

Samenvatting

Recursie is wanneer een functie zichzelf aanroept.

Wordt erg veel gebruikt bij grafen en dynamic programming, maar af en toe ook in andere algoritmes.

Meestal kan je een recursieve functie omschrijven in een while-loop en andersom.

Voorbeeld recursiealgoritmes waren:

1. Fibonacci
2. Binary Search
3. MergeSort

Les van vandaag

Inhoudsopgave

1. Generics
2. Recursie [komt niet op de toets]
3. Binaire zoekbomen
4. Toets

Les van vandaag

Inhoudsopgave

1. Generics
2. Recursie [komt niet op de toets]
3. **Binaire zoekbomen**
   1. Wat is een binaire boom?
   2. Hoe werkt een binaire boom?
4. Toets

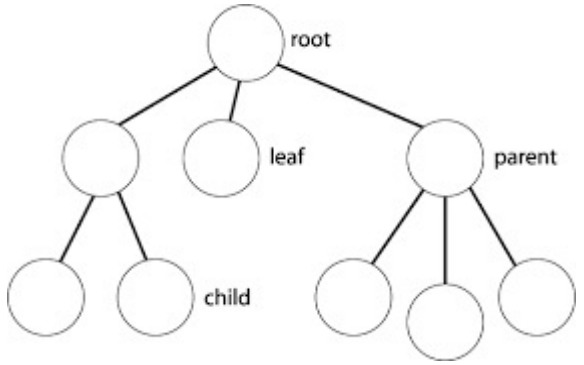
3. Binaire zoekbomen

Wat is een binaire boom?

Hoe zorgt ervoor dat een TreeSet zo snel dingen gesorteerd kan inserten?
→ Het gebruikt een (zelf-balancerende) binaire zoekboom

Een boom is een manier om data te structureren.
Elke *node* (knoop) heeft kinderen die eronder zitten.
Deze kinderen zijn zelf weer *nodes* met kinderen.

De boom begint bij de *root* en eindigt aan de *leaves*.



3. Binaire zoekbomen

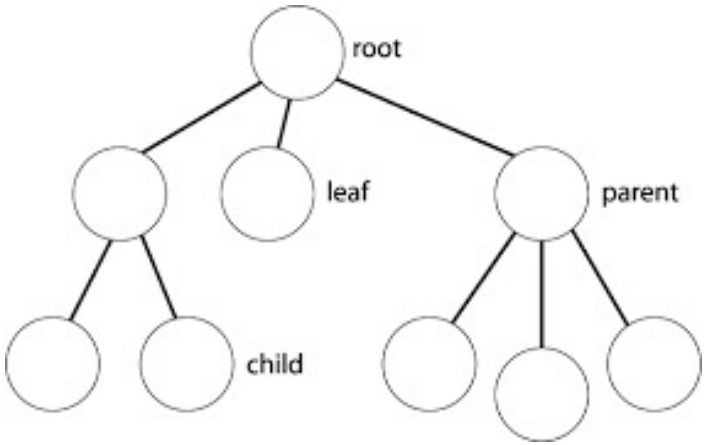
Wat is een binaire boom?

Hoe zorgt ervoor dat een TreeSet zo snel dingen gesorteerd kan inserten?

→ Het gebruikt een (zelf-balancerende) binaire zoekboom

Een implementatie van een boom in Java kan gewoon door classes. Voor twee kinderen:

```
class Tree {
    Node root;
}

class Node {
    Node left;
    Node right;
    int data; // Kan ook Strings etc zijn
}
```

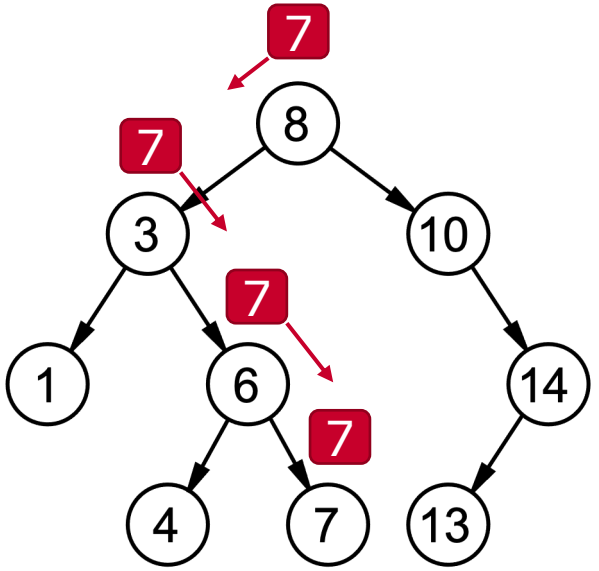


3. Binaire zoekbomen

Hoe werkt een binaire zoekboom?

Een binaire boom ziet er als volgt uit:

1. Alleen getallen in de nodes.
2. Links is kleiner, rechts is groter.



Functie `contains(7)`:

- Zit getal 7 in de set? Boom afgaan!
- Hoeveel operaties zijn dit?
  - Maximaal de diepte/hoogte van de boom.
    Dit is $O(\log(n))$ bij een gebalanceerde boom.
- Hoe weten we of een getal er niet inzit?

3. Binaire zoekbomen

Hoe werkt een binaire zoekboom?

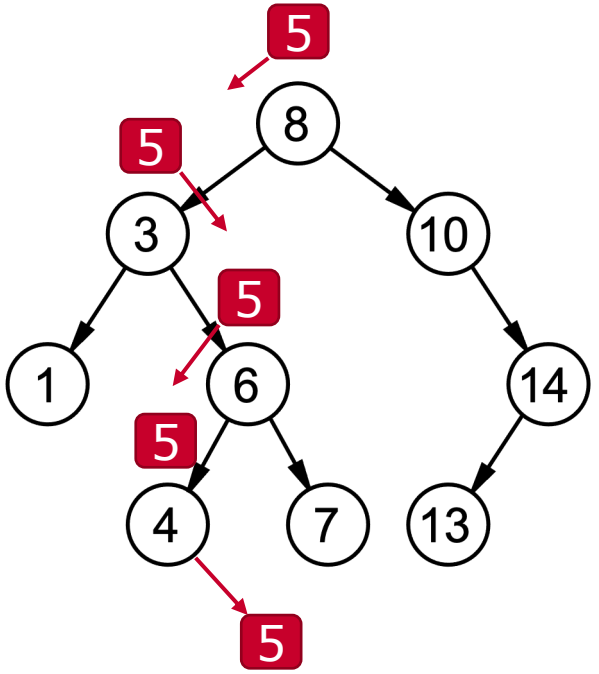
Een binaire boom ziet er als volgt uit:

1. Alleen getallen in de nodes.
2. Links is kleiner, rechts is groter.

Functie `add(5)`:

1. Boom afgaan naar beneden.
2. Getal toevoegen zodra je bij een leaf bent.

Wat kan er mis gaan?

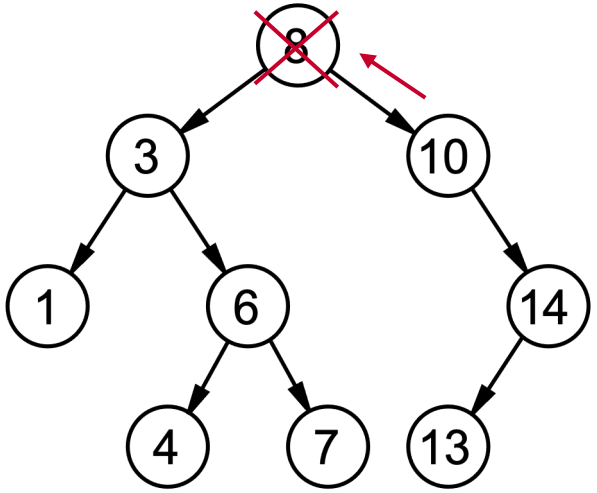


3. Binaire zoekbomen

Hoe werkt een binaire zoekboom?

Een binaire boom ziet er als volgt uit:

1. Alleen getallen in de nodes.
2. Links is kleiner, rechts is groter.



Functie `delete(8)`:

1. Vind het volgende grootste getal
   1. (Ga 1x naar rechts en zoveel naar links)
2. Swap het nieuwe getal naar de huidige plaats
   1. 0 kinderen? → Het is een blad, kan direct.
   2. 1 kind? → Geef het kind een nieuwe ouder mee
   3. 2 kinderen? → Onmogelijk

3. Binaire zoekbomen

Hoe werkt een binaire zoekboom?

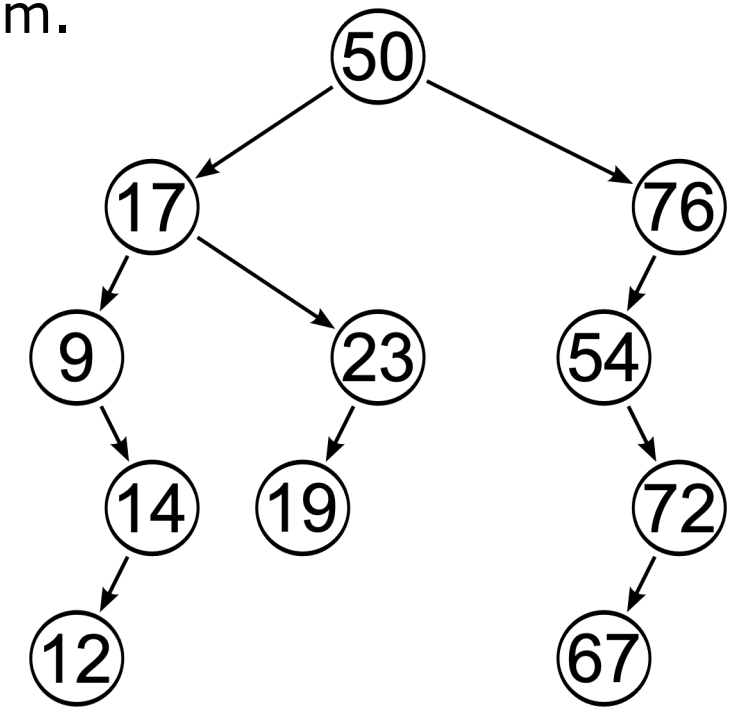
Al deze operaties duren maximal de diepte van de boom.

Bij een gebalanceerde boom is dit $O(\log n)$.

De binaire boom die we hebben laten zien heeft één nadeel: het wordt niet automatisch gebalanceerd.

Er zijn betere opties die zichzelf balanceren:

- B-tree
- Red-black tree
- Meer



By Me (Intgr) - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2809694

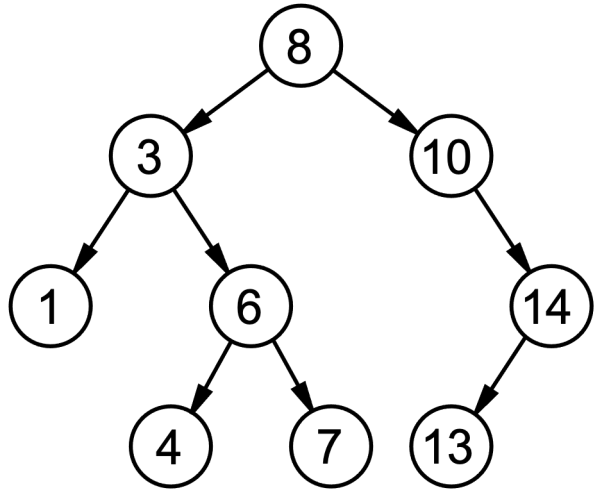
3. Binaire zoekbomen

Hoe werkt een binaire zoekboom?

Een TreeSet wordt geïmplementeerd door een zelf-balancerende binaire zoekboom. Dit is een boom met de volgende eigenschappen:

1. Elk kind links is kleiner dan de node.
2. Elk kind rechts is groter dan de node.

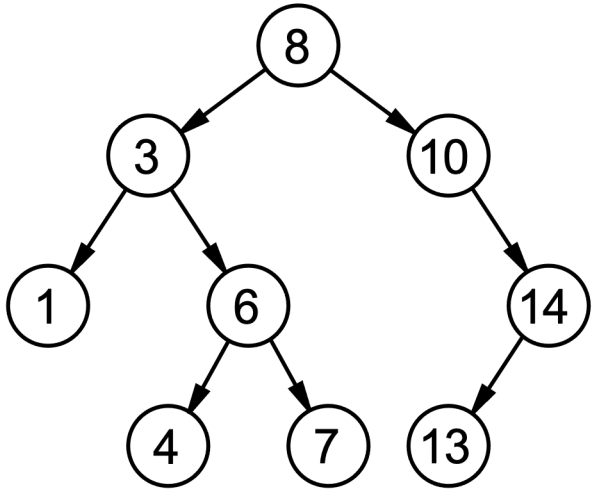
De functies add, remove en contains zijn allemaal afhankelijk van de hoogte van de boom.



3. Binaire zoekbomen

Opdracht

1. [Extra] Implementeer je eigen binaire zoekboom met integers.
   1. Begin met de functie add().
   2. Voeg daarna de functie contains() toe.
   3. Voeg als laatste de functie remove() toe.
2. [Extra] Implementeer extra functies `min()`, `max()` en een functie `printSorted()` die de boom gesorteerd print.
3. Implementeer een binaire zoekboom met generieke types.




```
class Tree {
    Node root;
}
```

```
class Node {
    Node left;
    Node right;
    int data;
}
```

Les van vandaag

Inhoudsopgave

1. Generics
2. Recursie [komt niet op de toets]
3. Binaire zoekbomen
4. **Toets**
    1. Wat komt er op de toets?
    2. Kleine herhaling
    3. Bespreking proeftoets

4. Toets

Wat komt er op de toets?

De toets bestaat uit twee delen:

1. 50% theorievragen
2. 50% programmeeropdracht

Tijdens de toets mag je bijna alles bijhouden. Je mag:

1. De slides en het boek erbijhouden
2. Zelf uitgewerkte opdrachten bijhouden
3. De proeftoets bijhouden
4. Een offline versie van de Java documentatie.

Je mag alleen **niet op het internet**, behalve voor het submitten van de code.

4. Toets

Wat komt er op de toets?

Topics voor de theorievragen:

1. Tijds- en ruimtecomplexiteit
2. Het snappen van de zoek- en sorteeralgoritmes
3. Arrays, ArrayLists, Linked Lists
4. HashSet, TreeSet, HashMap, TreeMap
5. ArrayDeque, Priority Queue
6. Generieke datatypen snappen
7. Binaire zoekbomen snappen

Vragen?

De programeeropdracht gaat met datastructuren te maken hebben en is in dezelfde vorm als vorig jaar, maar dit keer met de tijdscomplexiteit zoals $O(\log n)$ erbij gezet.

4. Toets

Kleine herhaling: tijds- en geheugencomplexiteit

```java
/**
 * Finds a number in an unordered list.
 * Returns index or -1 if not found.
 */
public int findNumber(int list[], int number) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] == number) {
            return i;
        }
    }
    return -1;
}
```

1. Gegeven een stuk simpele code:
   - *Wat is de worst-case verwerkingstijd?*
   - *Wat is het extra geheugengebruik?*

2. Je snapt wat de big-O notatie betekent
   - Als inputverzameling *n* 10x zo groot wordt, duurt het running van een $O(n^2)$ algoritme 100x zo lang.

3. Je kan verschillende snelheden vergelijken:
   - $O(n^2)$ is langzamer dan $O(n \log n)$ (voor grote n, in de worst-case).
   - $O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n^2) < O(n^6) < O(2^n)$
   - Stel ik heb een input van 1 000 000 getallen, hoelang duurt $O(n^2)$ ongeveer?

4. Toets

Kleine herhaling: tijds- en geheugencomplexiteit

Linear zoeken: Door een lijst heen gaan. $O(n)$

Binair zoeken: Door een gesorteerde lijst snel heen gaan. $O(\log n)$

Een sorteeralgoritme sorteert een lijst op de juiste volgorde, standaard van klein naar groot. Voorbeeldalgoritmes:

1. Selection Sort: Pak steeds het minimum van de lijst $\rightarrow O(n^2)$
2. (Counting Sort: Tel hoe vaak een getal in de lijst zit $\rightarrow O(kn)$)
3. Tree Sort: Voeg elementen toe aan een TreeSet $\rightarrow O(n \log n)$
4. (Merge Sort: Recursieve splits en merge ze terug inelkaar. $\rightarrow O(n \log n)$)

4. Toets

Kleine herhaling: datastructuren

- Array<E>: een lijst aan getallen, kan niet worden veranderd.
- ArrayLists<E>: kan eigenlijk alles ermee, houdt arrays bij in de achtergrond.
- LinkedLists<E>: linkt naar volgende datapunt. In principe inferieur aan ArrayList en ArrayDeque.
- Sets<E>: Handig om te checken of iets geweest is (add, contains, remove)
  - HashSet<E>: Gebruikt hashing en hashCode(). Soms zijn er collisions.
  - TreeSet<E>: Gebruikt een sorteerorde en een gebalanceerde binaire zoekboom.
- Map<K, V>: Handig om dingen mee op te zoeken.
  - Net als bij Sets zijn er HashMaps en TreeMaps.
- ArrayDeque<E>: Goed om de volgorde van data bij te houden.
- PriorityQueue<E>: Wordt gebruikt om continu een minimum (of maximum) bij te houden.

4. Toets

Kleine herhaling: datastructuren

Operatie	Array	ArrayList	LinkedList	TreeSet	HashSet	ArrayDeque	PrioQueue(min)
Lezen/Updaten	$O(1)$	$O(1)$	$O(n)$	N/A	N/A	N/A	N/A
Zoeken	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$**	$O(n)$	$O(n)$
Toevoegen	N/A	$O(1)$*	$O(1)$	$O(\log n)$	$O(1)$**	$O(1)$	$O(\log n)$
Toevoegen orde	N/A	$O(n)$	$O(1)$	N/A	N/A	$O(1)$	N/A
Insertion	N/A	$O(n)$	$O(n)$	N/A	N/A	N/A	N/A
Verwijderen	N/A	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$**	N/A	N/A
Verwijderen orde	N/A	$O(n)$	$O(1)$	N/A	N/A	$O(1)$	N/A
Min bekijken	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(1)$
Min verwijderen	N/A	$O(n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(\log n)$
Max verwijderen	N/A	$O(n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	N/A

4. Toets

Kleine herhaling: datastructuren

Om custom objecten te gebruiken met datastructuren moet je soms extra functies implementeren:

1. Functie .equals() → Bekijk of twee verschillende instanties hetzelfde zijn.
   - Nodig voor elke .contains()
2. Functie .hashCode() → Een unieke code van een object.
   - Nodig voor een hashSet en hashMap
3. Functie .compareTo en implements Comparable → Een natural ordering
   - Nodig voor datastructuren die de data voor je sorteren op een bepaalde manier: TreeSet, TreeMap en PriorityQueue.
   - Als alternatief kan vaak ook een Comparator meegegeven worden aan de structuur.

4. Toets

Kleine herhaling: generieke typen

Er zijn twee manieren om generieke typen te gebruiken in Java:

1. Generic classes
2. Generic methods

Generic classes kun je generieke object accepteren, zoals in een ArrayList<E>.

```
public class Box<T> { … }
```

Met generic methods generieke objecten als parameters accepteren.

```
public <T, U> Set<T> getKeyset(Map<T, U> map)
```

Soms is het ook handig om wildcards te gebruiken:

```
public void print(List<?> anyList)
```

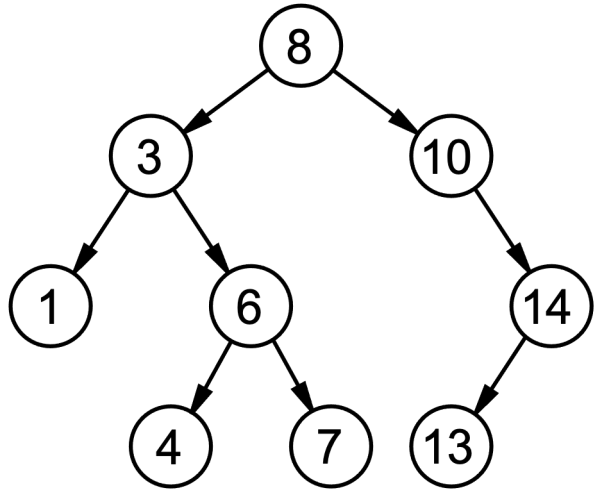
4. Toets

Kleine herhaling: binaire bomen

Een TreeSet wordt geïmplementeerd door een zelf-balancerende binaire zoekboom. Dit is een boom met de volgende eigenschappen:

1. Elk kind links is kleiner dan de node.
2. Elk kind rechts is groter dan de node.

De functies add, remove en contains zijn allemaal afhankelijk van de hoogte van de boom.



By No machine-readable author provided. Dcoetzee assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims)., Public Domain, https://commons.wikimedia.org/w/index.php?curid=488330

4. Toets

Proeftoets

Tijd om de proeftoets te bespreken.

Les van vandaag

Inhoudsopgave

1. Generics
2. Recursie [komt niet op de toets]
3. Binaire zoekbomen
4. Toets

Les van vandaag

Einde les

Dat was alles van ADS.

Ik wens jullie veel success op de toets!