


# 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 34. Stel dat je intern een arrayimplementatie voor bomen gebruikt. Welke array zou je dan bekomen voor deze beide bomen?

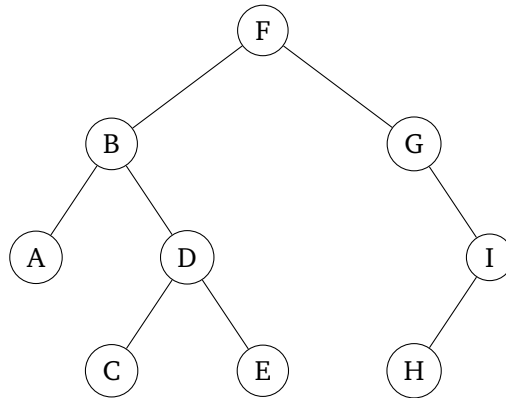
### Oefening 5.2

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

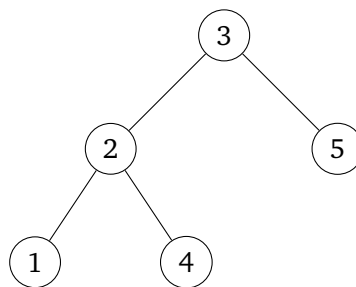
- a) Waar vind je de ouder van de knoop die zich in de array op index 256 bevindt?
- b) Waar vind je de rechterbuur (op hetzelfde niveau) van de knoop die zich in de array op index 72 bevindt? Op welk niveau staan die beiden? Hebben die beiden dezelfde ouder?
- c) Heeft de knoop die zich in de array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



# Oplossingen

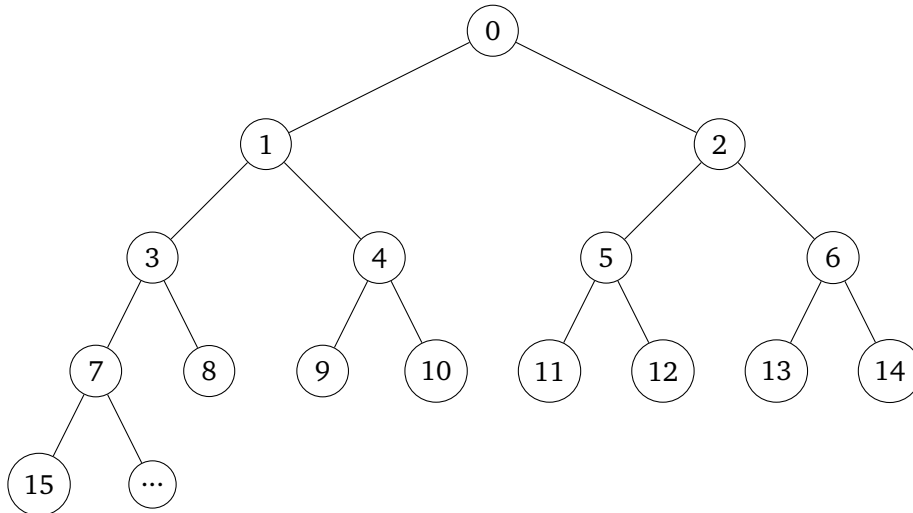
**Oplossing 5.1** Zoals we in de theorie zagen, moet je een bepaalde waarde afspreken om aan te geven dat een bepaald kind ontbreekt. Laten we dat voor de eenvoud hier gewoon voorstellen door het getal 0. Een mogelijke arrayimplementatie voor figuur 5.1 is:

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



**Figuur 1** Op weg naar 1000 knooppunten, genummerd van 0 tot 999

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

Met deze informatie kunnen we nu de vragen beantwoorden:

- 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.

**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

- b) Waar vind je de rechterbuur (op hetzelfde niveau) van de knoop die zich in de array op index 72 bevindt? Knoop 72 bevindt zich op niveau 7, meerbepaald de tiende knoop op dit niveau. De rechterbuur van 72 is natuurlijk knoop 73. Beide knopen hebben een verschillende ouder (welke?).
- 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 2** 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 3** 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 4 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;
    }
}

```



## Oplossingen

```
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 5 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;
    }
}
```