Informatik II - Übung 2

Pascal Schärli

student.ethz@pascscha.ch

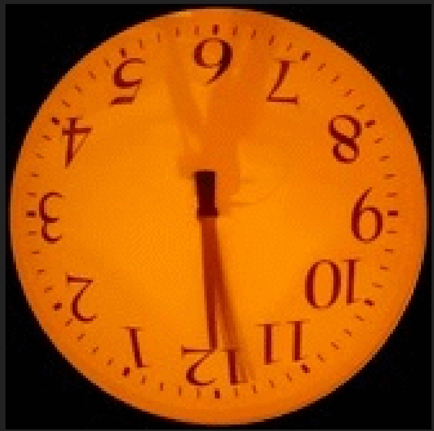
02.10.2020

ETH zürich

Administratives

- Die Übungen sind auf CodeExpert
- Die Files welche Ihr für die Übungen braucht sind auf der Vorlesungswebseite.
- Meine Slides findet ihr auf meiner Website pascscha.ch/info2
- Bei Fragen: info2@pascscha.ch

Nachbesprechung



Altägyptische Multiplikation

$$f(a,b) = \begin{cases} a & \text{, falls } b = 1 \\ f(2a, b/2) & \text{, falls b gerade} \\ f(2a, \frac{b-1}{2}) + a & \text{, sonst} \end{cases}$$

1. Induktionsbeweis über a möglich?
   - Nein! Der Induktionsanfang schlägt bereits für b>1 fehl!
   - Da a ständig wachsend ist, ist kein Rückschluss auf bereits bewiesene Fälle möglich.
2. Terminiert der Algorithmus?
   - Ja, da nach ⌊log 2 (b)⌋ Schritten b=1 sein wird
3. Wie ändert sich der Beweis wenn erst bei b=0 terminiert wird?
   - Der Algorithmus geht einen Schritt länger, da ⌊1/2⌋ = 0 ist. Der Rest ist analog zur Vorlesung.

Laufzeitkomplexität

```java
public static boolean gerade( int x ){
        if( x == 0 ) return true;
        return !gerade(x - 1);
}
```

x Aufrufe

```java
public static int verdopple( int x ){
        if( x == 0 ) return 0;
        return 2 + verdopple( x-1 );
}
```

x Aufrufe

```java
public static int halbiere( int x ){
        if( x == 0 || x == 1 ) return 0;
        return halbiere(x - 2);
}
```

$\lfloor \frac{x}{2} \rfloor$ Aufrufe

Laufzeitkomplexität

Ignorieren, dass f sich selbst aufruft

```
private static int f(int a, int b){
        if (b == 0) return 0;
        if (gerade(b)) return f(verdopple(a), halbiere(b));
        else return a + f(verdopple(a), halbiere(b));
}
```

- $gerade(b)$, $verdopple(a)$ und $halbiere(b)$ werden so oder so aufgerufen
- $\rightarrow$ Anzahl aufrufe: $b + 1 + a + 1 + \lfloor \frac{b}{2} \rfloor + 1 \approx a + \frac{3b}{2} + 3$

Laufzeitkomplexität

Ohne Ignorieren, dass f sich selbst aufruft

```
private static int f(int a, int b){
        if (b == 0) return 0;
        if (gerade(b)) return f(verdopple(a), halbiere(b));
        else return a + f(verdopple(a), halbiere(b));
}
```

Zusammen mit 2 b) ergibt sich:

$$N(a,b) = (a + \tfrac{3b}{2} + 3) + N(2a, \tfrac{b}{2})$$

$$= \sum_{i=0}^{k-1}(2^i a) + \sum_{i=1}^{k}(\tfrac{3b}{2^i}) + 3k$$

Fehler während der Übungsstunde

Überprüfung von Benutzereingaben

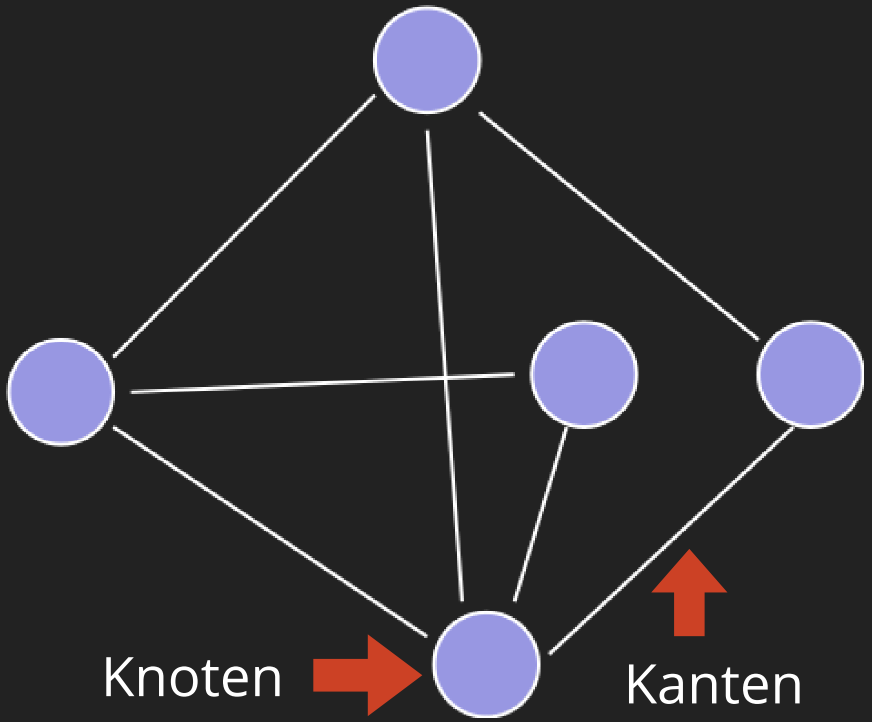
```java
/**
 * This function implements the ancient Egyptian multiplication.
 *
 * @param a must be a positive integer
 * @param b must be a positive integer
 * @return the product of a and b
 * @throws IllegalArgumentException if a or b is not positive
 */
    public static int mult(int a, int b) throws IllegalArgumentException {
        if (a < 1) throw new IllegalArgumentException("Parameter a must be a positive integer but is " + a);
        if (b < 1) throw new IllegalArgumentException("Parameter b must be a positive integer but is " + b);
        return f(a, b);
    }
}
```

Best of Vorlesung

Bäume

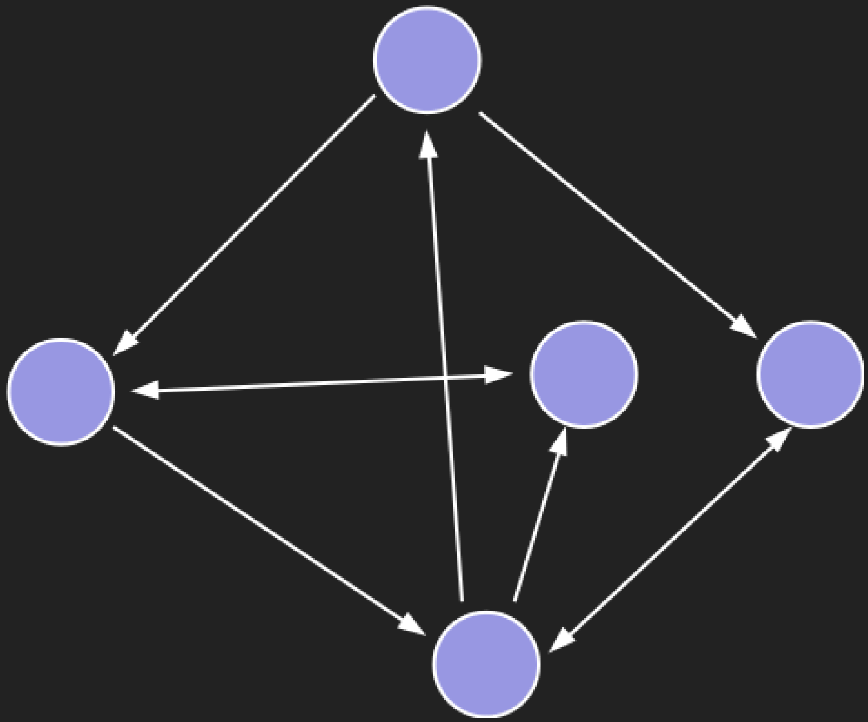
Bäume

1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. Binärbaum
6. Voller Baum
7. Vollständiger Baum
8. Entarteter Baum
9. Höhe
10. Pfad



Bäume

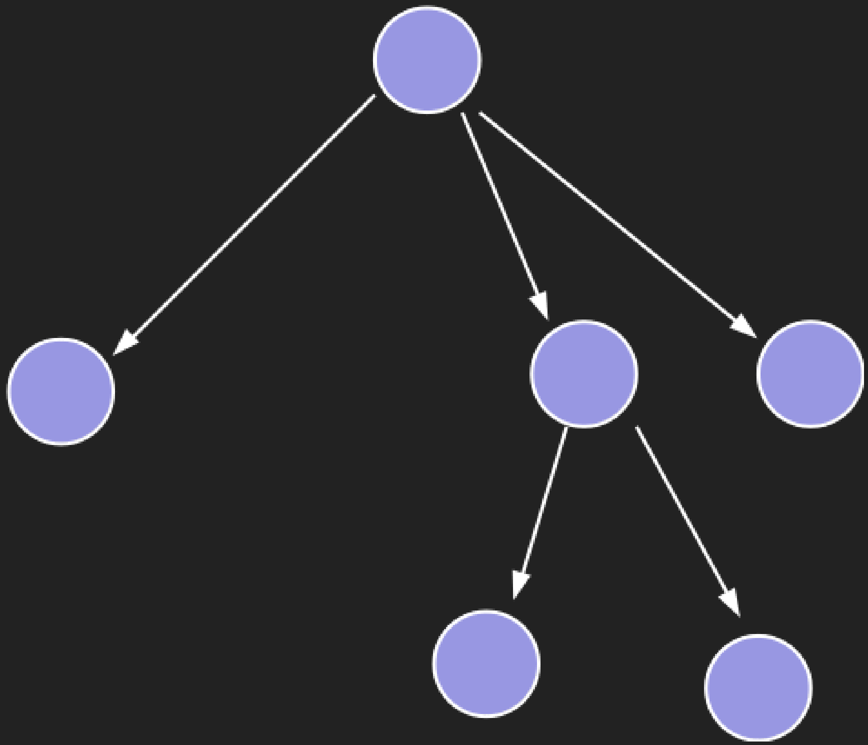
```
1  Graph
2  Gerichteter Graph
3  Baum
4  Blätter
5  Binärbaum
6  Voller Baum
7  Vollständiger Baum
8  Entarteter Baum
9  Höhe
10 Pfad
```



Kanten haben eine Richtung

Bäume

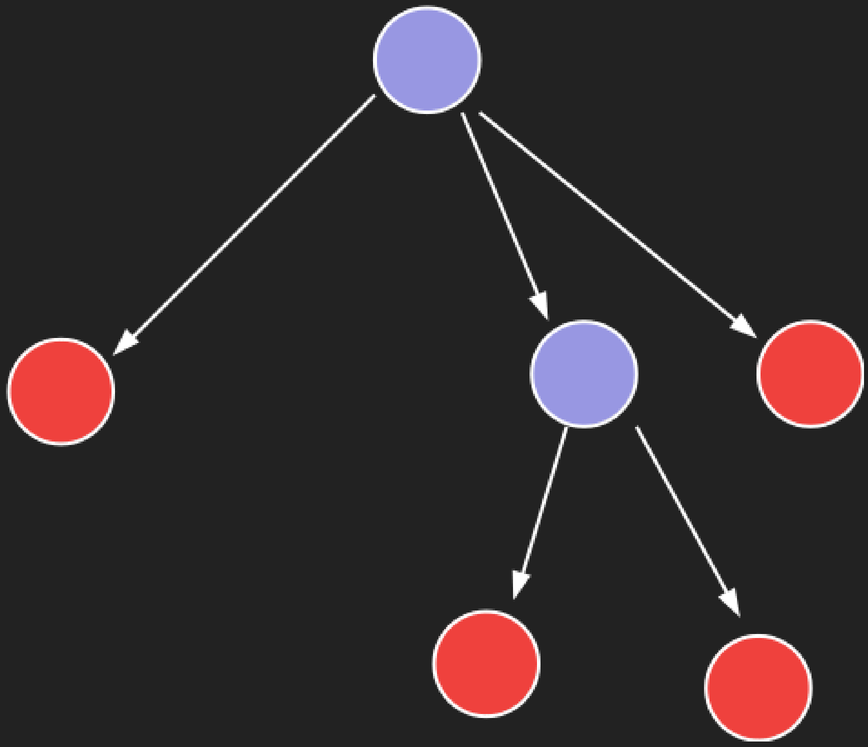
1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. Binärbaum
6. Voller Baum
7. Vollständiger Baum
8. Entarteter Baum
9. Höhe
10. Pfad



Zusammenhängender, Gerichteter Graph
ohne geschlossenen Pfade

Bäume

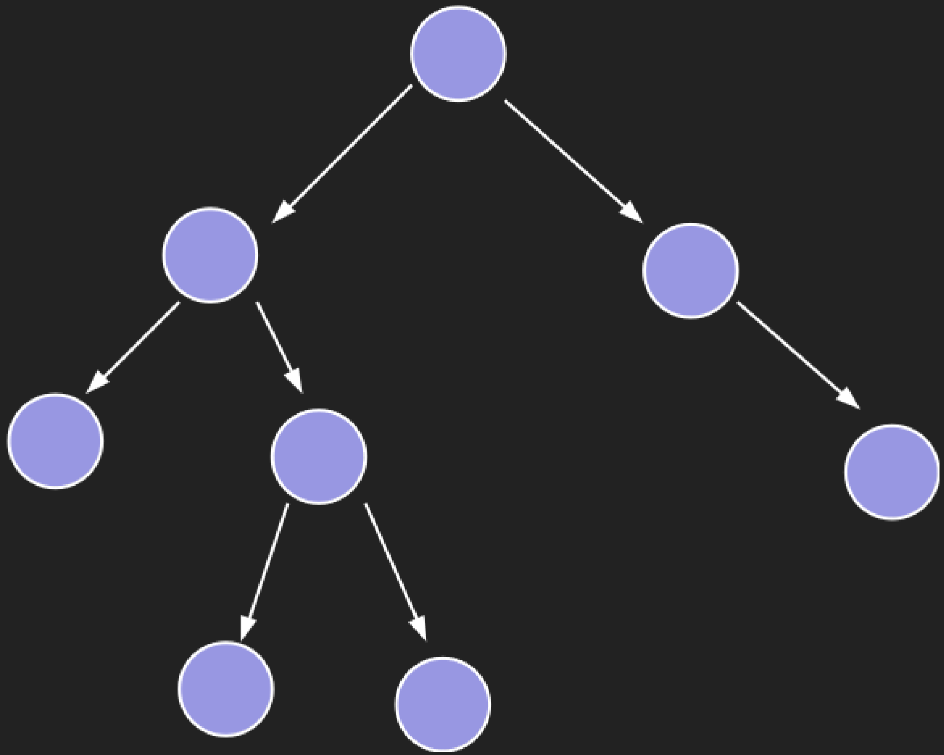
1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. Binärbaum
6. Voller Baum
7. Vollständiger Baum
8. Entarteter Baum
9. Höhe
10. Pfad



Knoten ohne Kindknoten

Bäume

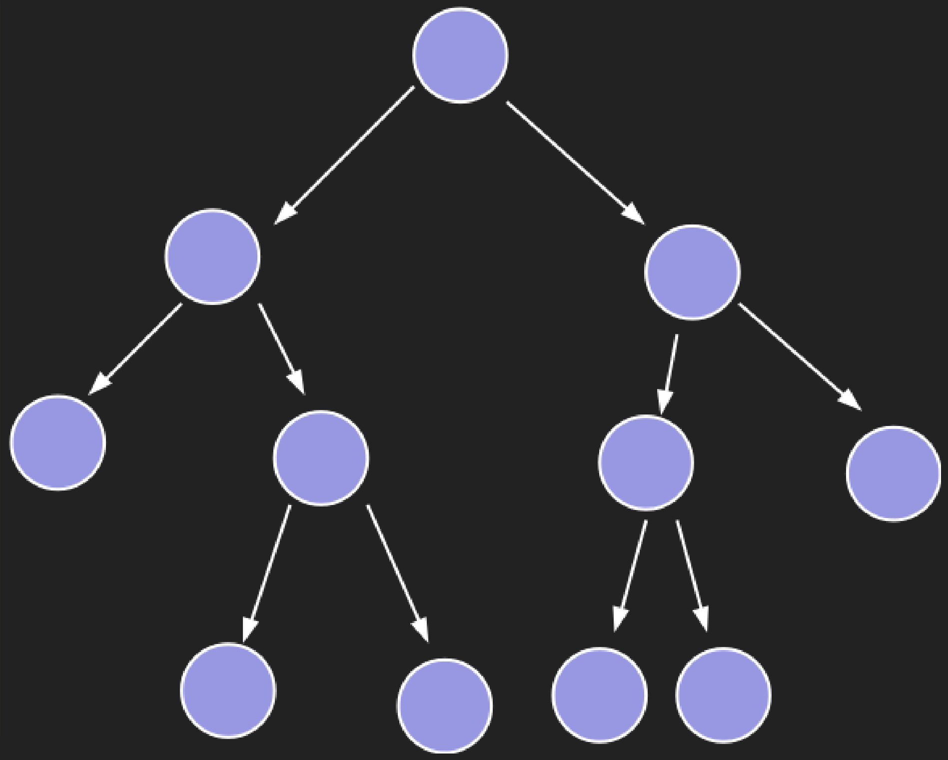
1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. **Binärbaum**
6. Voller Baum
7. Vollständiger Baum
8. Entarteter Baum
9. Höhe
10. Pfad



jeder Knoten besitzt *höchstens* zwei Kindknoten, die Reihenfolge der Kindknoten ist relevant.

Bäume

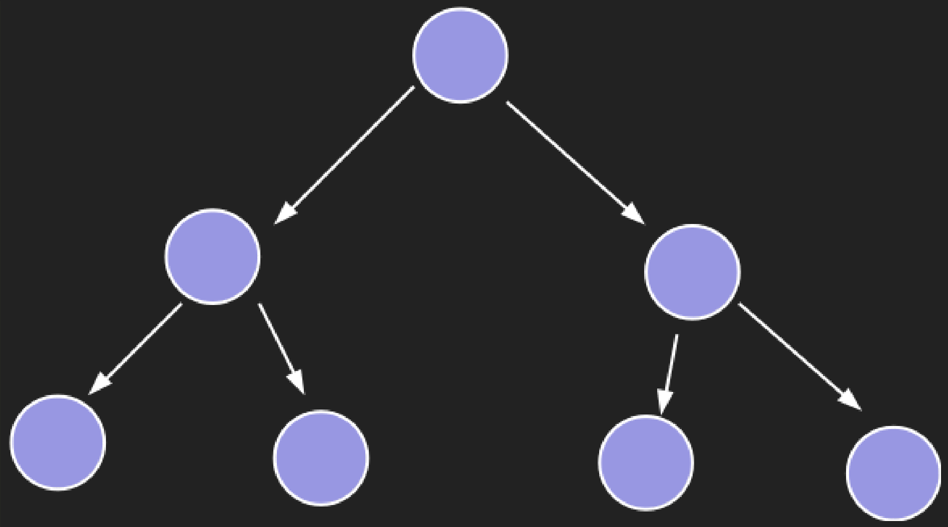
1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. Binärbaum
6. Voller Baum
7. Vollständiger Baum
8. Entarteter Baum
9. Höhe
10. Pfad



jeder Knoten besitzt entweder zwei oder keine Kinder

Bäume

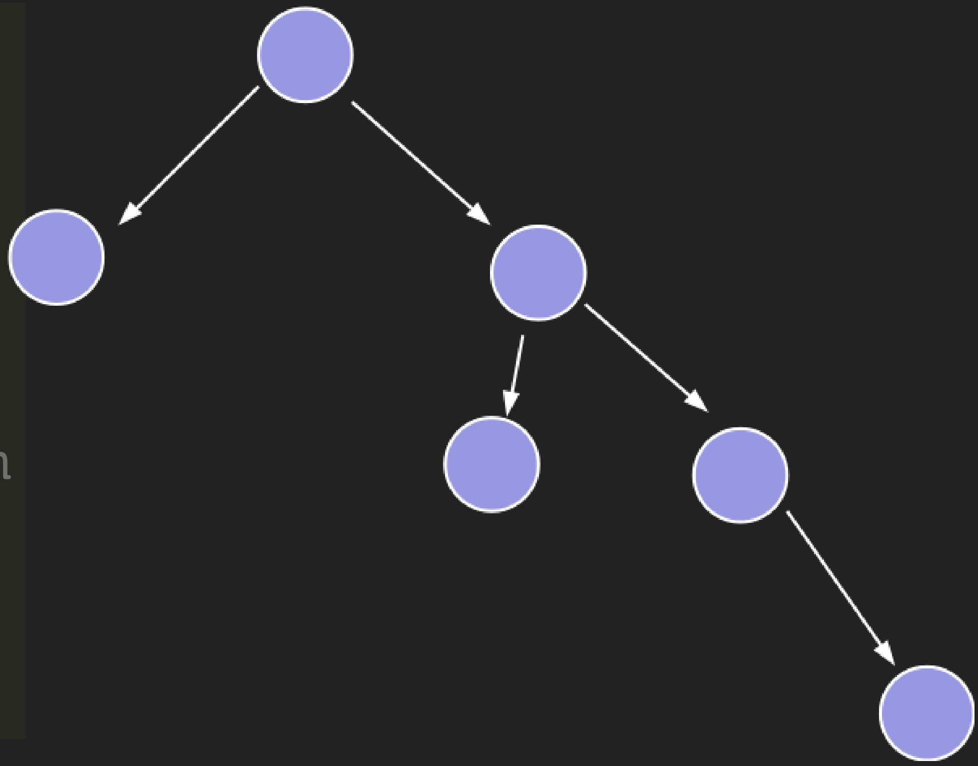
1 Graph
2 Gerichteter Graph
3 Baum
4 Blätter
5 Binärbaum
6 Voller Baum
7 **Vollständiger Baum**
8 Entarteter Baum
9 Höhe
10 Pfad



Alle Blätter haben die selbe Tiefe

Bäume

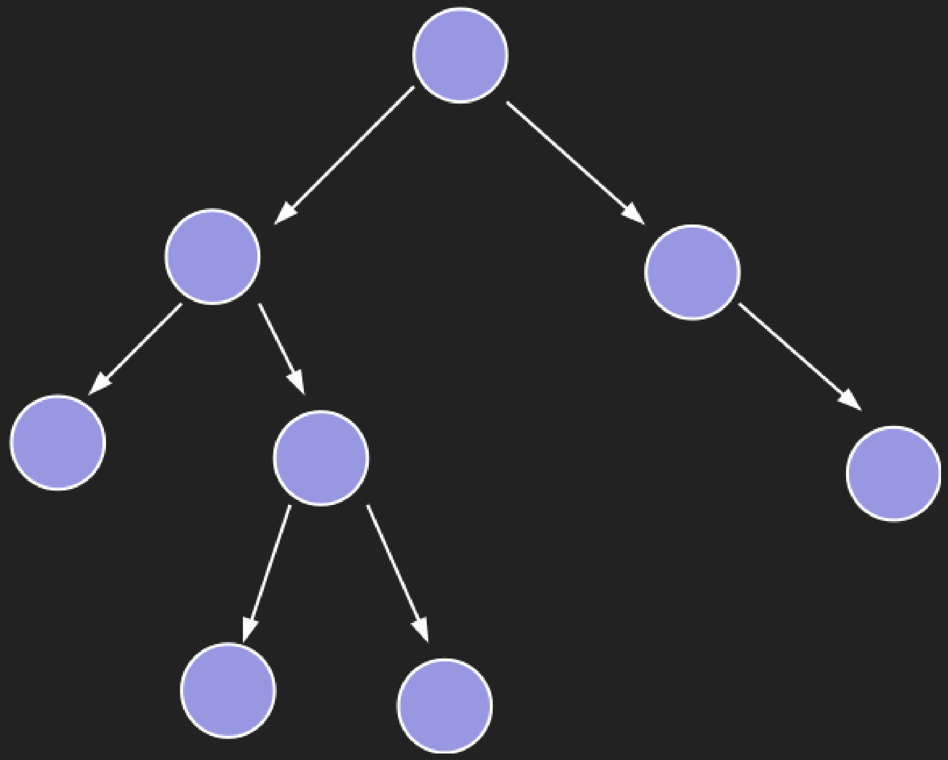
1. Graph
2. Gerichteter Graph
3. Baum
4. Blätter
5. Binärbaum
6. Voller Baum
7. Vollständiger Baum
8. **Entarteter Baum**
9. Höhe
10. Pfad



Ungleich verteilter Baum

Bäume

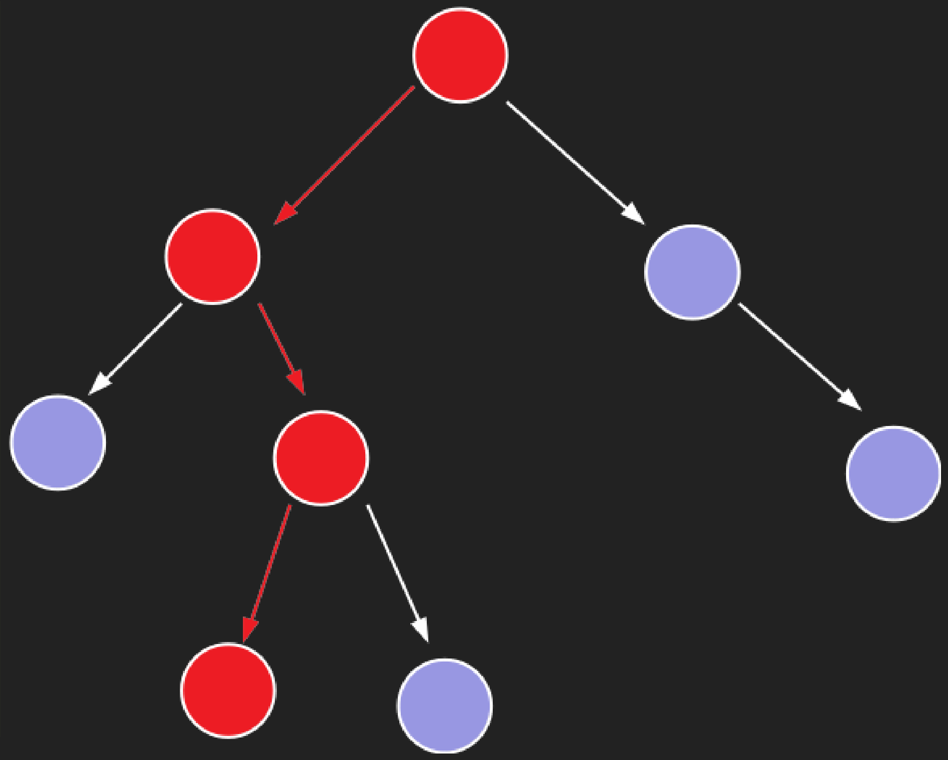
1 Graph
2 Gerichteter Graph
3 Baum
4 Blätter
5 Binärbaum
6 Voller Baum
7 Vollständiger Baum
8 Entarteter Baum
9 **Höhe**
10 Pfad



Anzahl “Levels” ohne Wurzel
→ hier: 3

Bäume

```
Graph
Gerichteter Graph
Baum
Blätter
Binärbaum
Voller Baum
Vollständiger Baum
Entarteter Baum
Höhe
Pfad
```



Weg durch Graph (Nur in Pfeilrichtung)

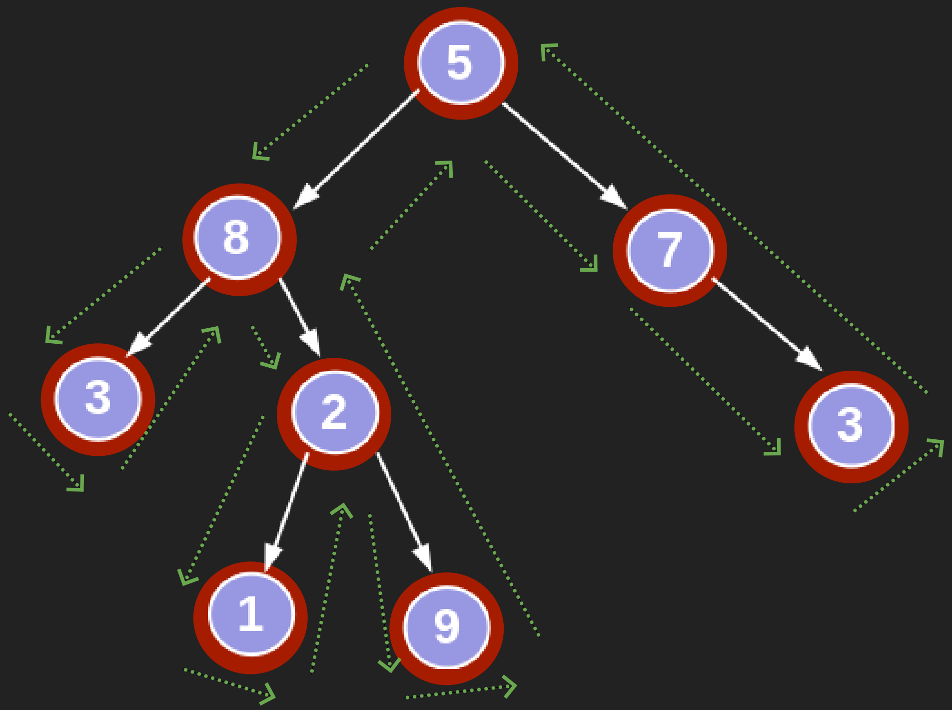
Eingerückte Darstellung

Indent = 0

Indent = 1

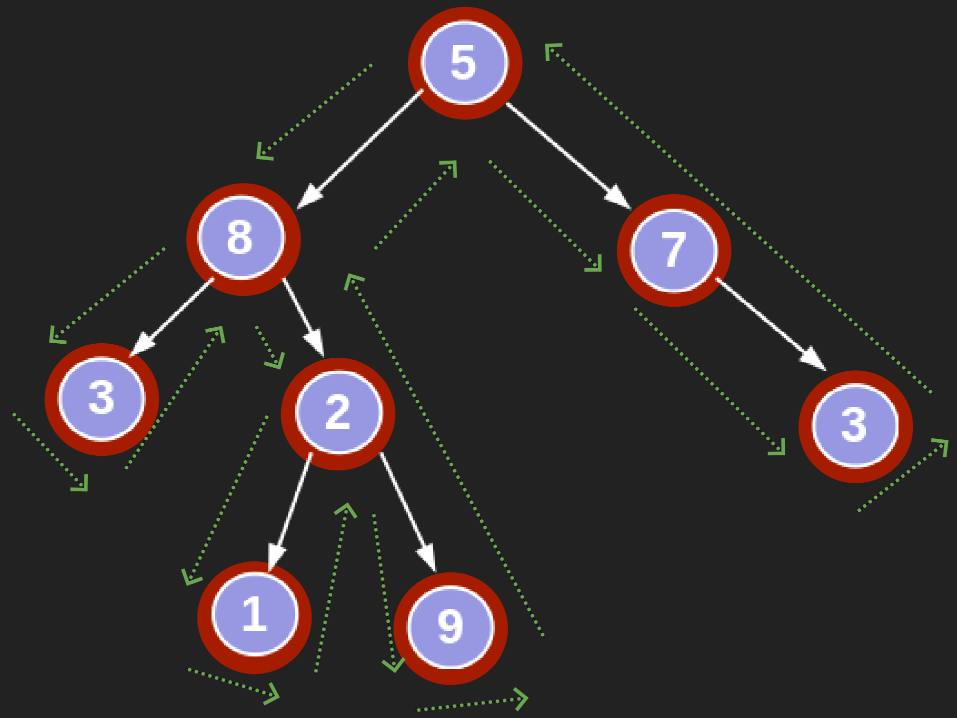
Indent = 2

Indent = 3



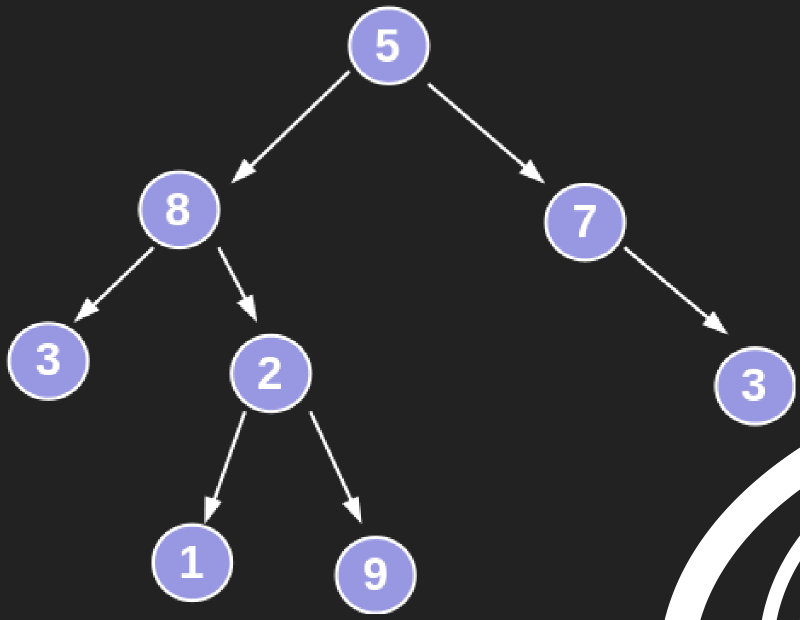

```
5
 8
  3
  2
   1
   9
 7
  3
```

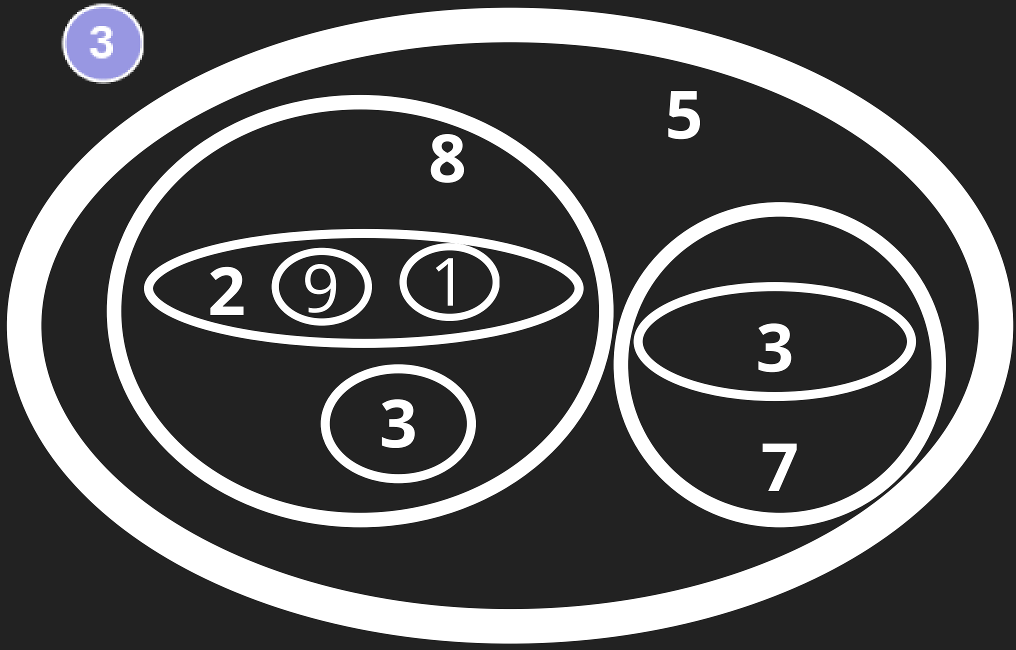
Klammer Darstellung




```
5(8(3,2(1,9)),7(3))
```

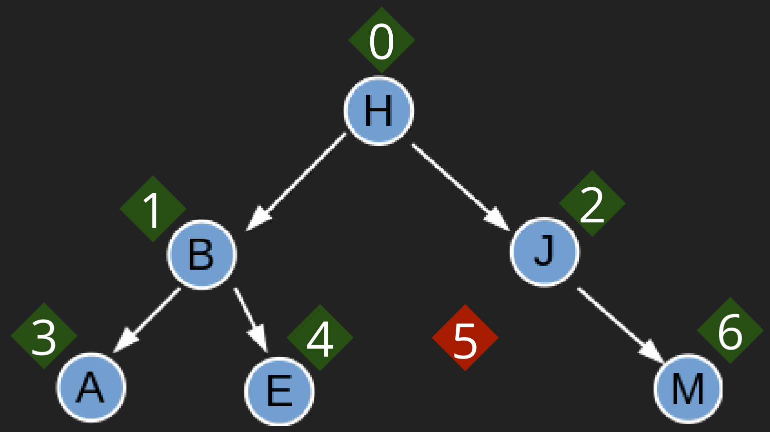
Mengendiagramm





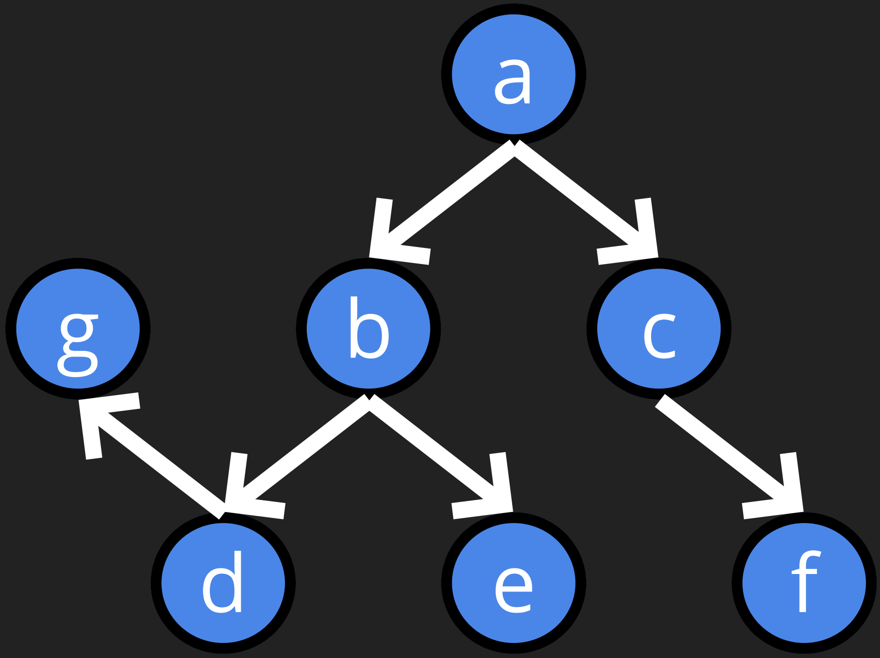
Binärbäume als Array

```
left child:  2*i+1
right child: 2*i+2
parent:      ⌊(i-1)/2⌋
```



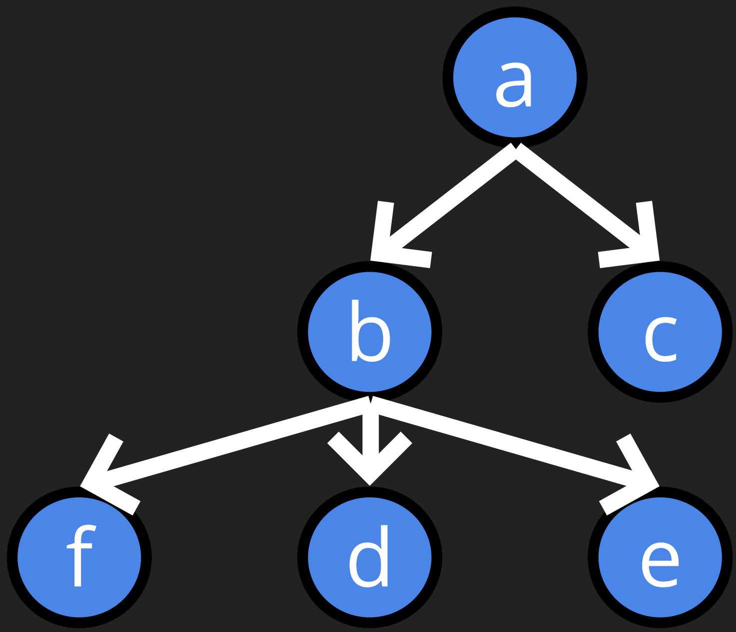

```
[ 'H', 'B' , 'J' , 'A' , 'E' , ' ' , 'M']
   0    1     2     3     4     5     6
```

Was ist die Höhe von diesem Baum?



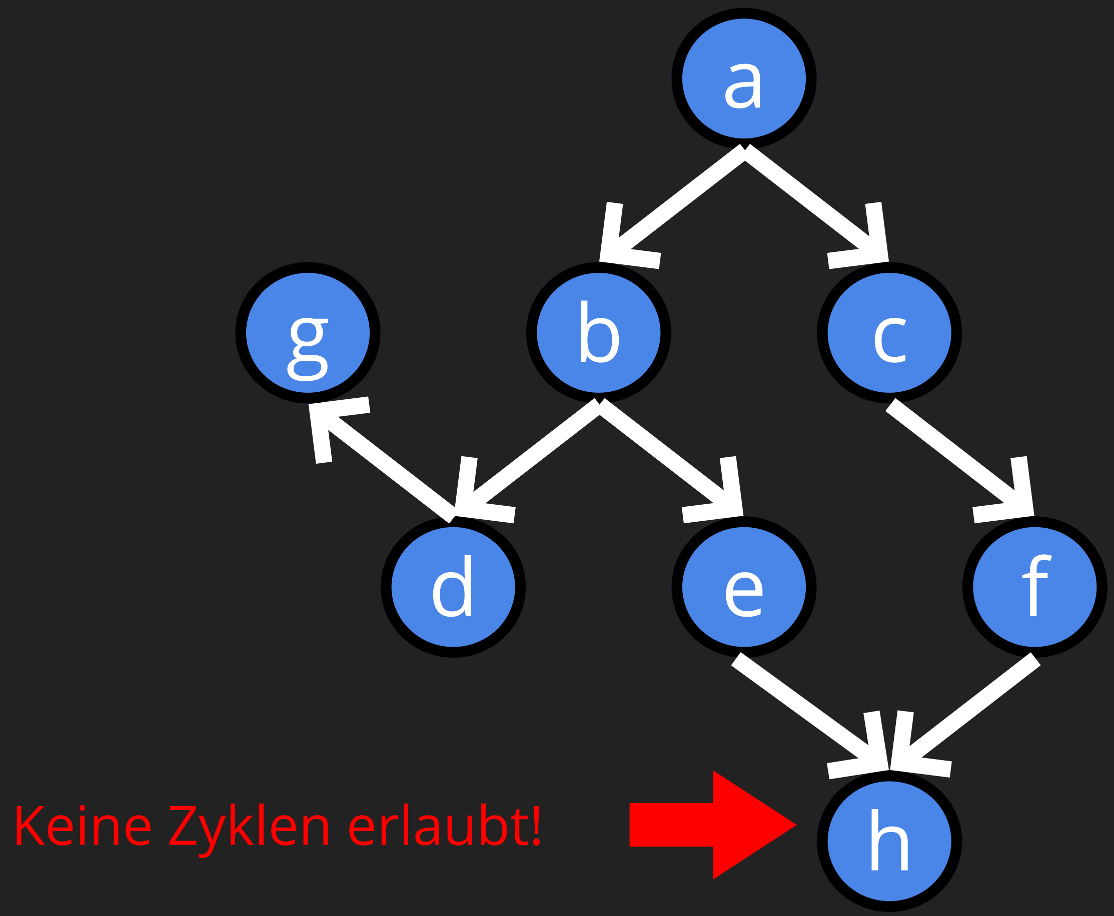
Lösung: **3**

Wie viele Blätter hat dieser Baum?



Lösung: **4**

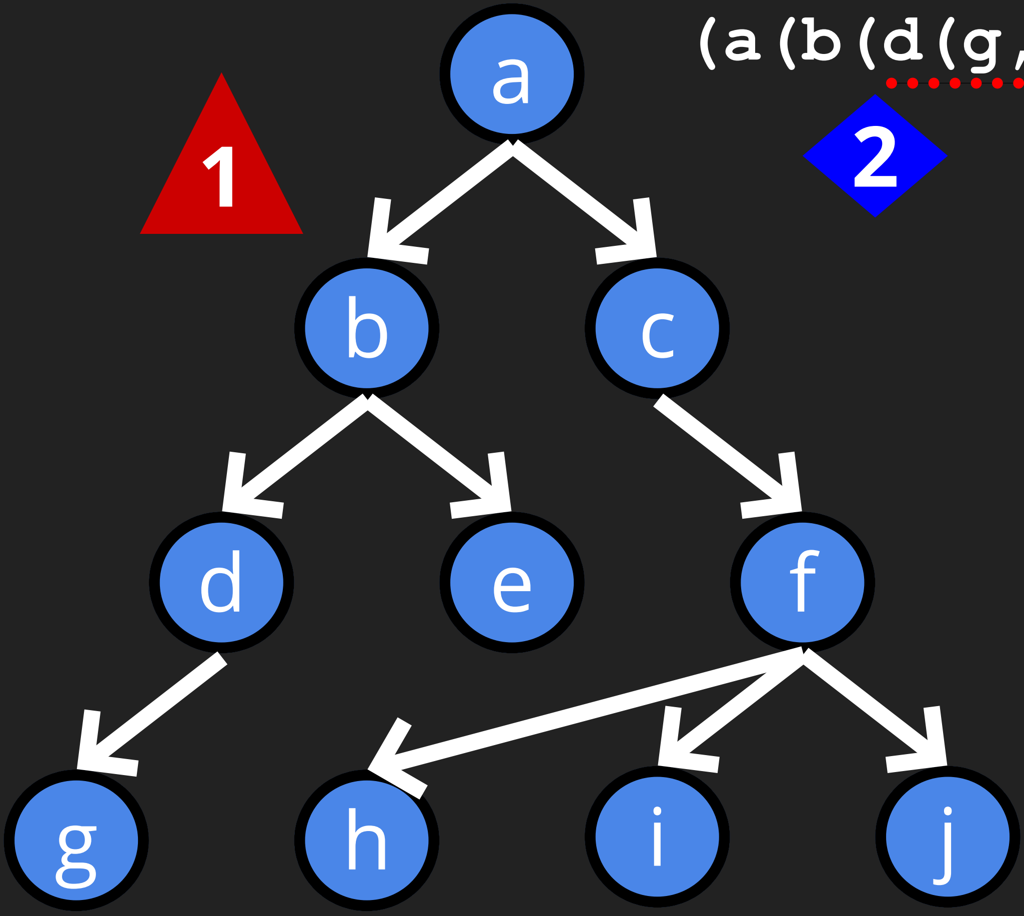
Wie viele Knoten hat der Längste Pfad?



Lösung: **kein korrekter Baum**

Welche Darstellung zeigt einen anderen Wurzelbaum?

1



2

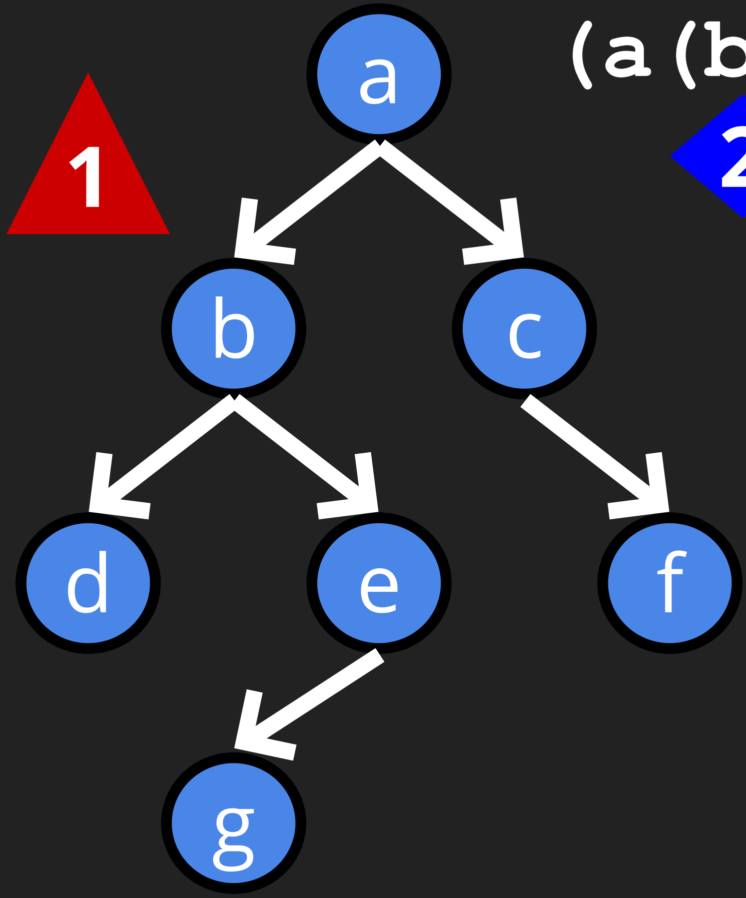
`(a(b(d(g,e)),c(f(h,i,j))))`

3

```
a
 b
  d
   g
  e
 c
  f
   h
   i
   j
```

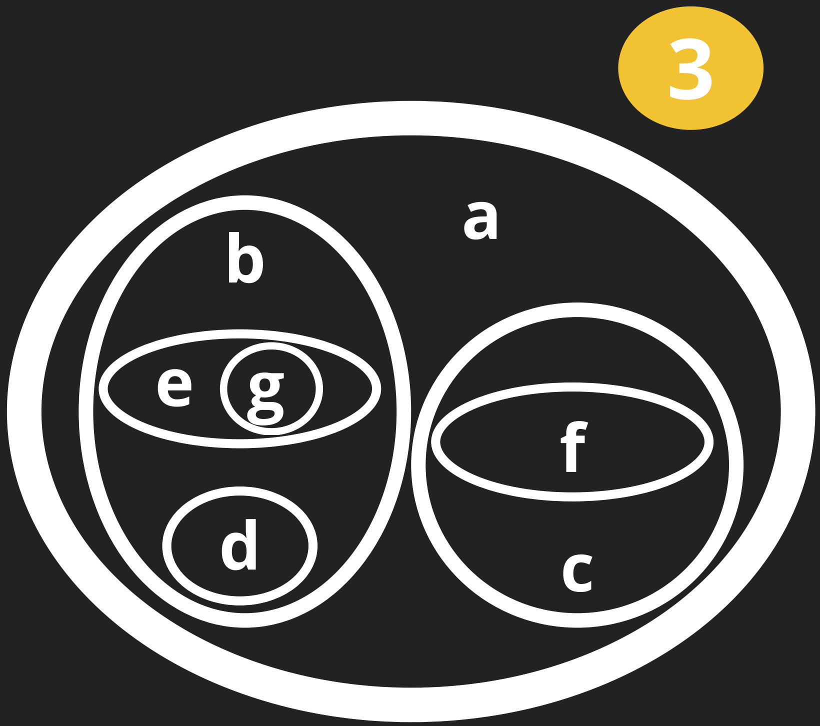
Lösung: **2**

Welche Darstellung zeigt einen anderen Wurzelbaum?



`(a(b(d,e(g)),c(f)))`

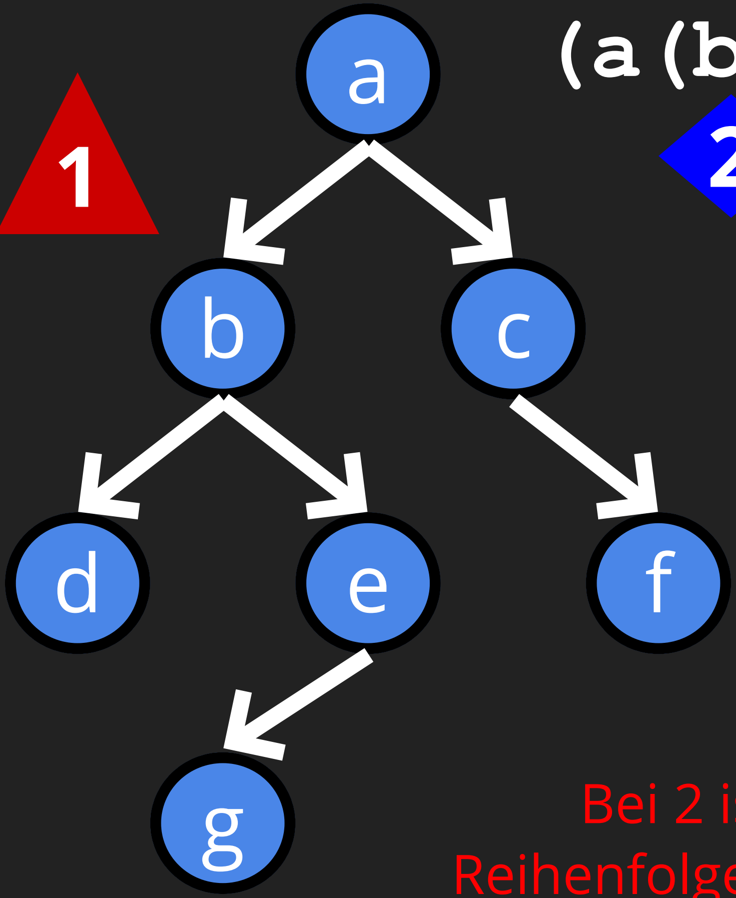
2



Lösung: **alles der gleiche Baum**

Welche Darstellung ist kein Binärbaum?

1




```
(a(b(d,e(g)),c(f)))
```

2

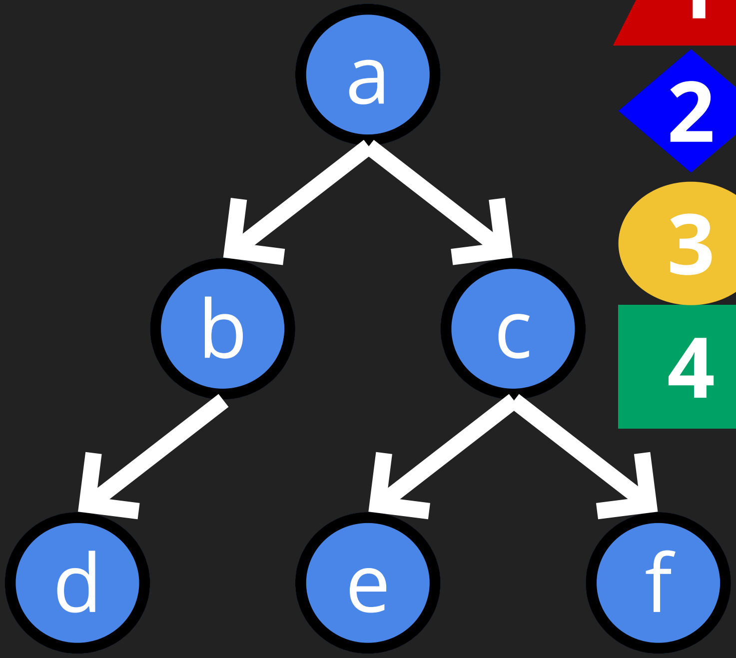
3

```
a
 b
  d
  e
   g
   -
 c
  -
  f
```

Bei 2 ist die Reihenfolge der Kind-knoten nicht gegeben

Lösung: **2**

Welches ist die richtige Arraydarstellung des Binärbaums?



1 ['a', 'b', 'd', ' ', 'c', 'e', 'f']

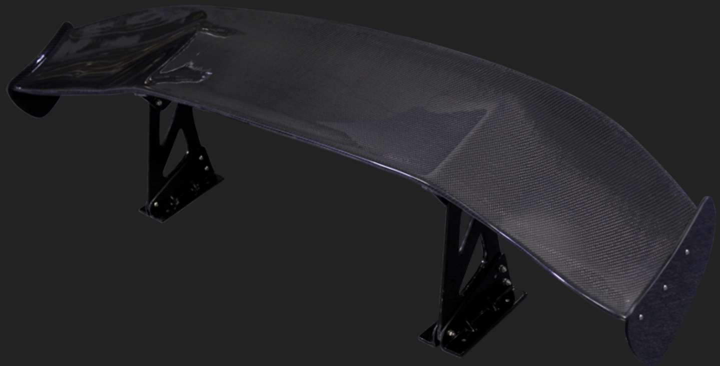
2 ['a', 'b', 'c', 'd', 'e', 'f']

3 ['a', 'b', 'c', 'd', ' ', 'e', 'f']

4 ['a', 'b', 'd', 'c', 'e', 'f']

Lösung: **3**

Vorbesprechung



Wurzelbäume

- Umwandeln zwischen Gerichteten Graphen, Klammer Darstellung und eingerückter Darstellung
- Höhe, längste Pfade & Blätter im Baum finden
- Alles ausführlich im BestOf diskutiert

Sortieren

Aufgabe 1 - Konstruktor

Implementiert den Konstruktor, der ein Array der angegebenen Grösse erstellt und mit Zufallszahlen füllt:

1. Der Konstruktor ist die Funktion RandomArray im File RandomArray.java
2. Verwendet die Klasse Random:
   ```
   import java.util.Random;
   ```
3. Initialisiert den Array numbers mit der gegebenen Grösse:
   ```
   numbers = new int[length];
   ```
4. Neues Objekt von der Klasse Random erstellen:
   ```
   Random r = new Random();
   ```
5. Zufallszahlen generieren:
   ```
   int random_integer = r.nextInt(1000)    //random Integer between 0 and 999
   ```

Sortieren

Aufgabe 2 - toString

Implementiert ie Funktion toString, die eine Stringrepräsentation des Arrays erzeugt.

1. In Java können Strings "addiert" werden:
   ```
   "Hello " + "World!" => "Hello World!"
   ```

2. Man kann Strings auch mit Zahlen addieren:
   ```
   "M" + 8 => "M8"
   ```

3. Gebt acht, dass Ihr die Leerschläge am richtigen Ort habt:

- `{1,2,3} =>'[1, 2, 3]'`

Sortieren

Aufgabe 3 - sortieren

- Implementiert die Funktion recursiveSort (von sort() aufgerufen mit der länge des arrays)

- Gegeben eine Liste mit n Elementen:
  Die ersten i Elemente der Liste können absteigend sortiert werden indem man:
  1. Die grössten i-1 Elemente absteigend sortiert
  2. Das grösste Element im Rest der Liste suchen...
  3. ... und dieses an der ersten Stelle vom Rest der Liste einsetzen

Sortieren

Aufgabe 3 - sortieren

```
[5, 1, 9, 2]
[5, 1, 9, 2]
[5, 1, 9, 2]
[5, 1, 9, 2]
[5, 1, 9, 2]
[5, 1, 9, 2]
[9, 1, 5, 2]
[9, 1, 5, 2]
[9, 5, 1, 2]
[9, 5, 1, 2]
[9, 5, 2, 1]
[9, 5, 2, 1]
```

```
Die grössten i-1 Elemente absteigend sortiert
Das grösste Element im Rest der Liste suchen…
… und dieses an der ersten Stelle vom Rest der Liste einsetzen
```

```
recursiveSort(4)
  recursiveSort(3)
    recursiveSort(2)
      recursiveSort(1)
        recursiveSort(0)
        -> Leere Liste ist Sortiert
      grösstes Element -> 9
      Swap 5 <-> 9
    grösstes Element -> 5
    Swap 5 <-> 1
  grösstes Element -> 2
  Swap 2 <-> 1
Fertig
```

Binärbäume als Array

Aufgabe 1

Implementiert die Funktionen leftChild, rightChild und father, die zu einem Index eines Knotens den Index des linken Kindes, des rechten Kindes bzw. des Vaters berechnen.

Binärbäume als Array

Aufgabe 2 - toString

- Implementiert die Funktion toString, die den Binärbaum in eingerückter Form zurückgibt.
- toString() ruft Hilfsfunktion toString(int node, String indentation) auf (zB. toString(0,“ “)

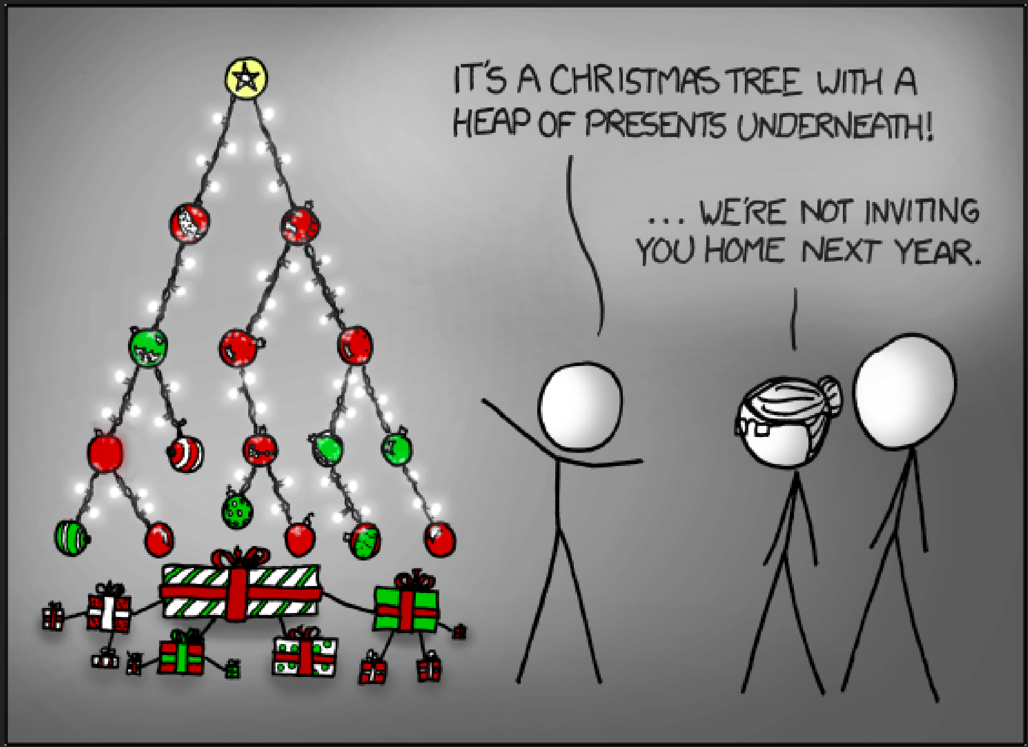
```
toString(int node, String indentation){
  //1. Print Node
  //2. Increase indentation
  //3. Recursive call for each child
}
```

Binärbäume als Array

Aufgabe 3 - checkTree

- Implementiert die Funktion checkTree, die das übergebene Array daraufhin überprüft, ob es eine gültige Repräsentation eines Binärbaumes ist.
- Das Array sollte mindestens ein Element haben (leerer Baum → [' '])
- Jedes nicht leere Element braucht eine nicht leere Parent Node
- „the root is its own parent“
  parent = (i-1)/2
  → (0-1)/2 = 0

Viel Spass!


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.