JOHANNES KEPLER
UNIVERSITY LINZ

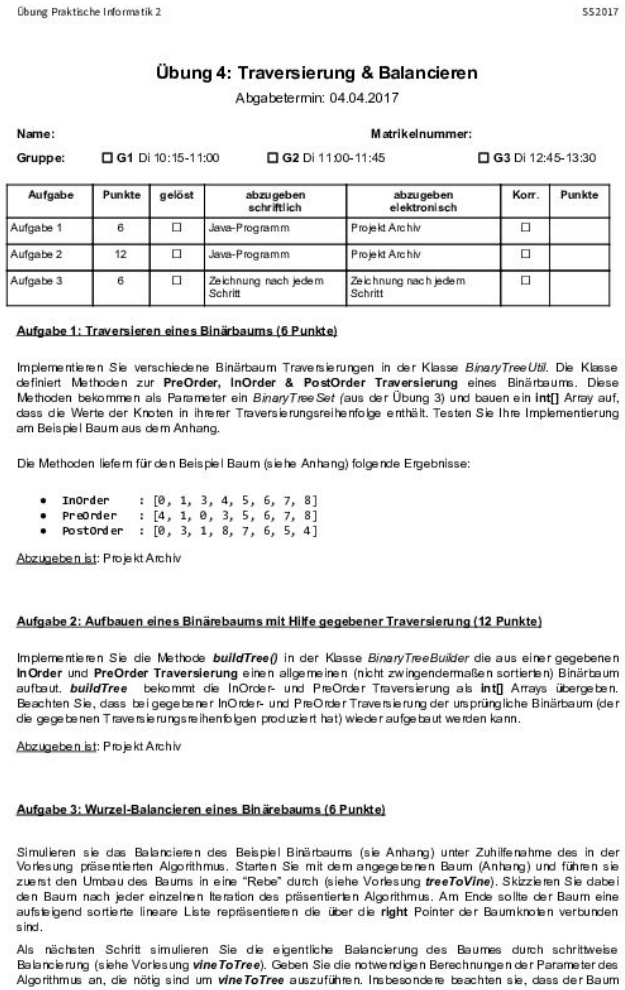
Praktische Informatik 2

Übung, 2017S
Institut für System Software (SSW)
DI Eisl & DI Leopoldseder

Ablauf

1. Fragen Übung 3
2. Fragen VO (Traversieren & Balancieren)
3. Übung 4
    a. Implementierung Traversierung
    b. Implementierung Binärbaum aus InOrder + PreOrder bauen

Übung 4 - Traversieren & Balancieren



Übung Praktische Informatik 2 SS2017

Übung 4: Traversierung & Balancieren

Abgabetermin: 04.04.2017

Name: **Matrikelnummer:**

Gruppe: ☐ **G1** Di 10:15-11:00 ☐ **G2** Di 11:00-11:45 ☐ **G3** Di 12:45-13:30

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Punkte
Aufgabe 1	6	☐	Java-Programm	Projekt Archiv	☐	
Aufgabe 2	12	☐	Java-Programm	Projekt Archiv	☐	
Aufgabe 3	6	☐	Zeichnung nach jedem Schritt	Zeichnung nach jedem Schritt	☐	

Aufgabe 1: Traversieren eines Binärbaums (6 Punkte)

Implementieren Sie verschiedene Binärbaum Traversierungen in der Klasse *BinaryTreeUtil*. Die Klasse definiert Methoden zur **PreOrder, InOrder & PostOrder Traversierung** eines Binärbaums. Diese Methoden bekommen als Parameter ein *BinaryTreeSet (*aus der Übung 3) und bauen ein **int[]** Array auf, dass die Werte der Knoten in ihrerer Traversierungsreihenfolge enthält. Testen Sie Ihre Implementierung am Beispiel Baum aus dem Anhang.

Die Methoden liefern für den Beispiel Baum (siehe Anhang) folgende Ergebnisse:

- `InOrder : [0, 1, 3, 4, 5, 6, 7, 8]`
- `PreOrder : [4, 1, 0, 3, 5, 6, 7, 8]`
- `PostOrder : [0, 3, 1, 8, 7, 6, 5, 4]`

Abzugeben ist: Projekt Archiv

Aufgabe 2: Aufbauen eines Binärebaums mit Hilfe gegebener Traversierung (12 Punkte)

Implementieren Sie die Methode ***buildTree()*** in der Klasse *BinaryTreeBuilder* die aus einer gegebenen **InOrder** und **PreOrder Traversierung** einen allgemeinen (nicht zwingendermaßen sortierten) Binärbaum aufbaut. ***buildTree*** bekommt die InOrder- und PreOrder Traversierung als **int[]** Arrays übergeben. Beachten Sie, dass bei gegebener InOrder- und PreOrder Traversierung der ursprüngliche Binärbaum (der die gegebenen Traversierungsreihenfolgen produziert hat) wieder aufgebaut werden kann.

Abzugeben ist: Projekt Archiv

Aufgabe 3: Wurzel-Balancieren eines Binärebaums (6 Punkte)

Simulieren sie das Balancieren des Beispiel Binärbaums (sie Anhang) unter Zuhilfenahme des in der Vorlesung präsentierten Algorithmus. Starten Sie mit dem angegebenen Baum (Anhang) und führen sie zuerst den Umbau des Baums in eine "Rebe" durch (siehe Vorlesung ***treeToVine***). Skizzieren Sie dabei den Baum nach jeder einzelnen Iteration des präsentierten Algorithmus. Am Ende sollte der Baum eine aufsteigend sortierte lineare Liste repräsentieren die über die **right** Pointer der Baumknoten verbunden sind.

Als nächsten Schritt simulieren Sie die eigentliche Balancierung des Baumes durch schrittweise Balancierung (siehe Vorlesung ***vineToTree***). Geben Sie die notwendigen Berechnungen der Parameter des Algorithmus an, die nötig sind um ***vineToTree*** auszuführen. Insbesondere beachten sie, dass der Baum

- Abgabetermin: 04.04.2017 08:30
- Vorgabe Projekt
- ANT zum Builden
- **TODO** markierte Methoden in Klasse ***BinaryTreeUtil.java*** implementieren

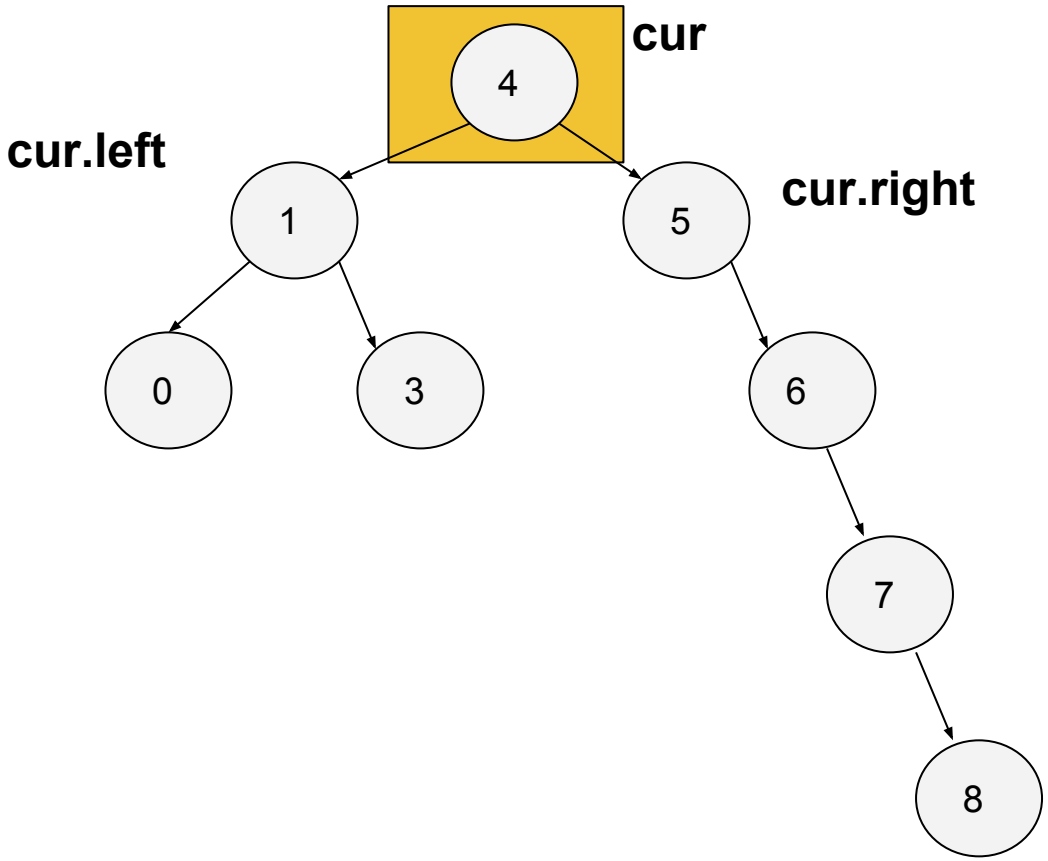
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[ ]



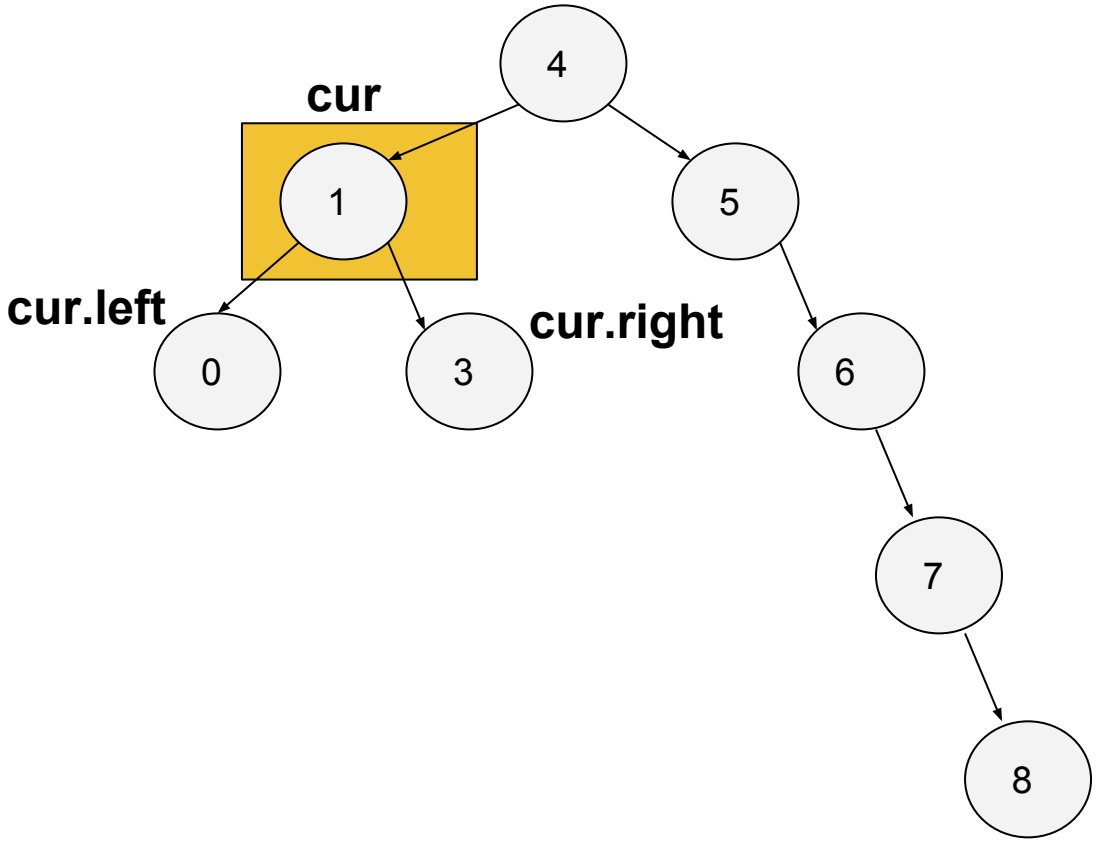
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[ ]



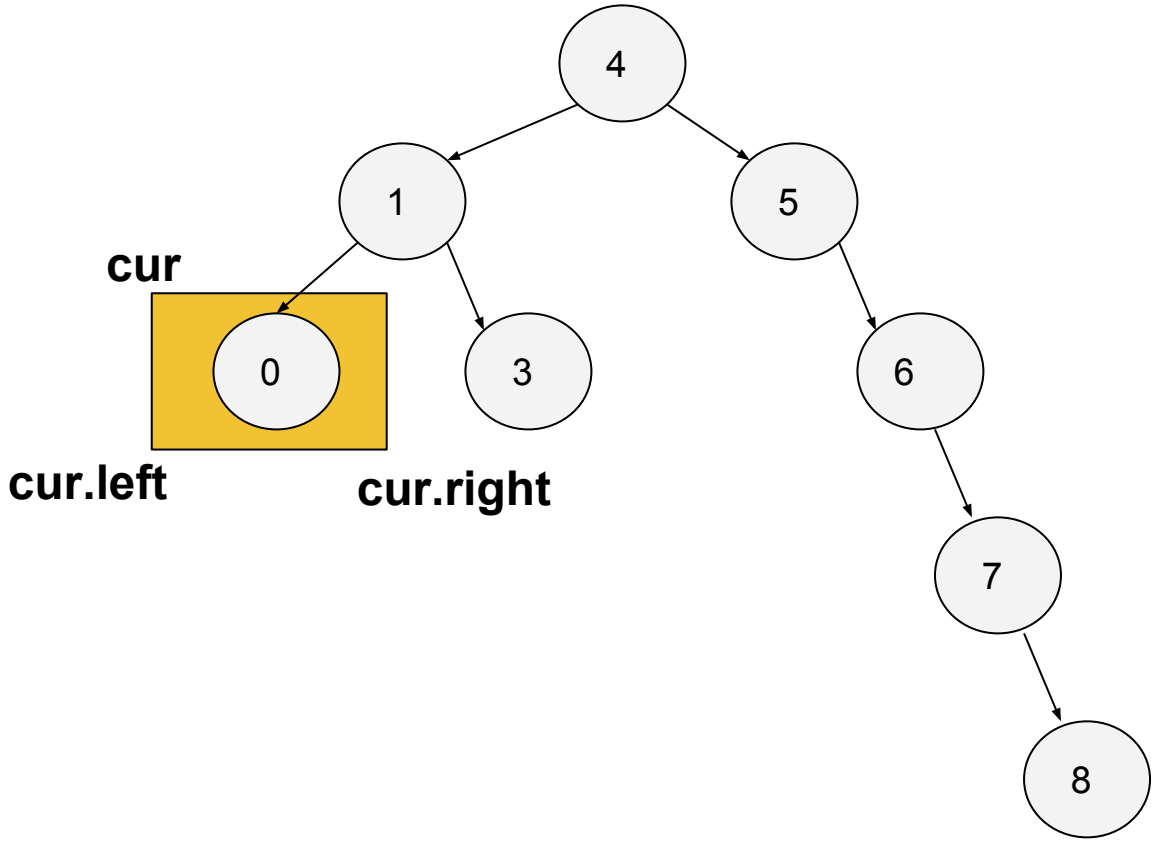
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[ ]



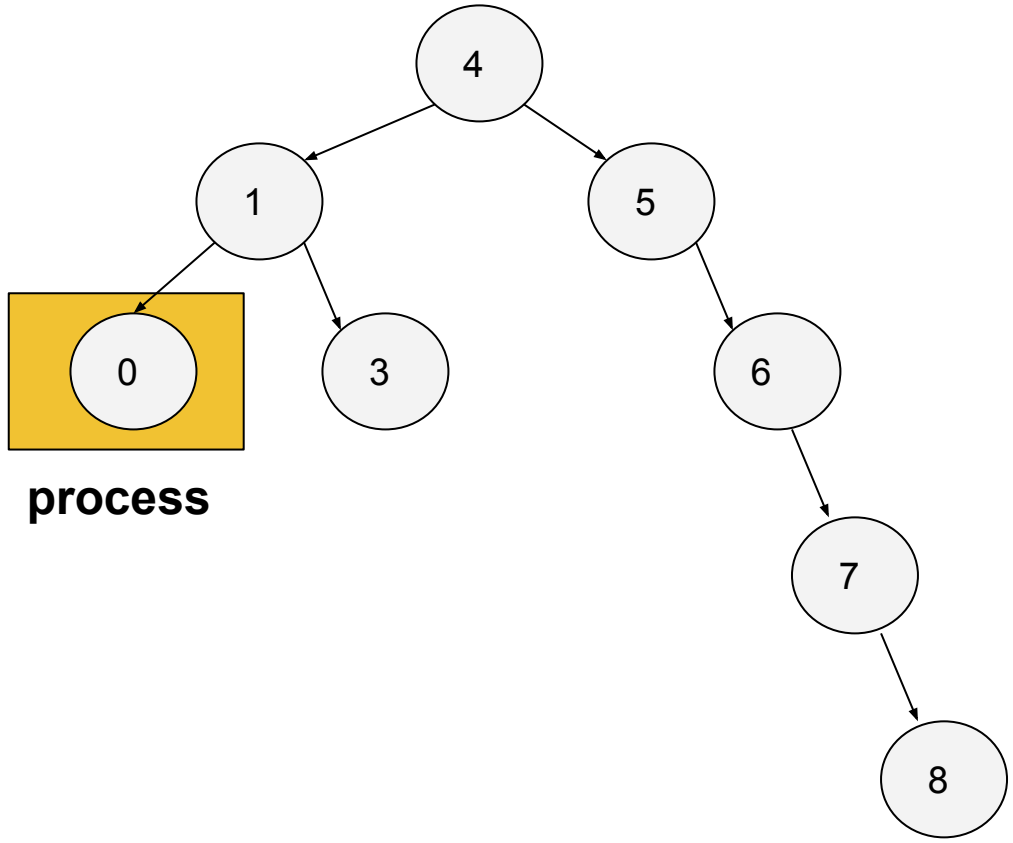
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0]



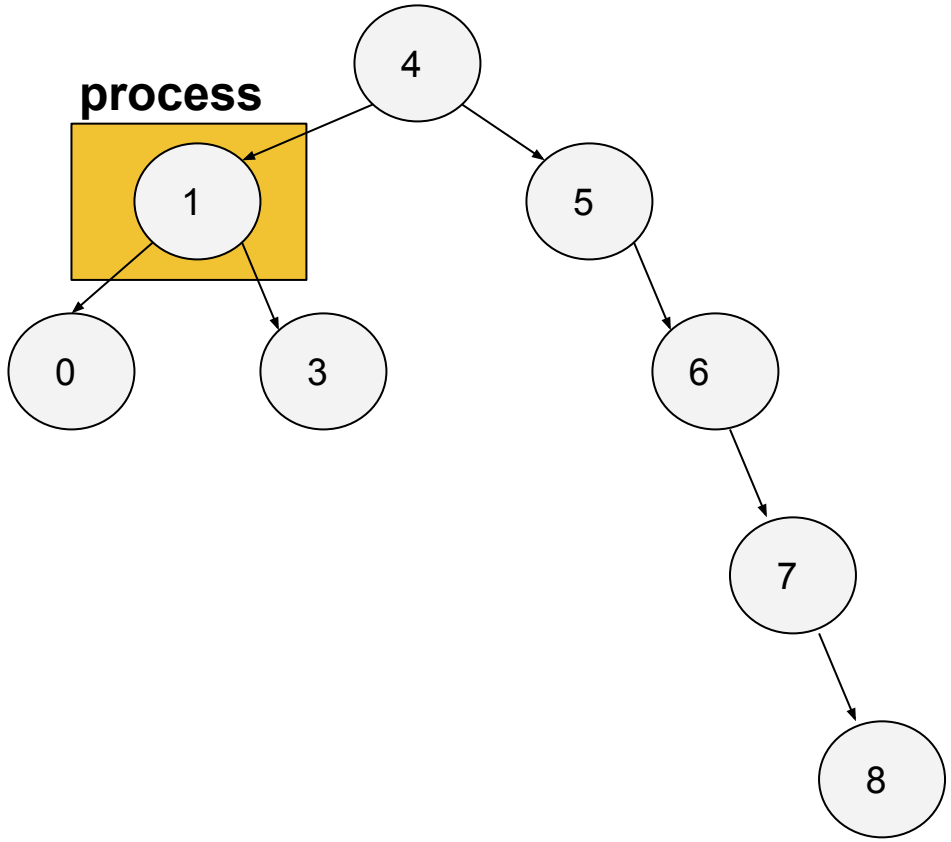
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1]



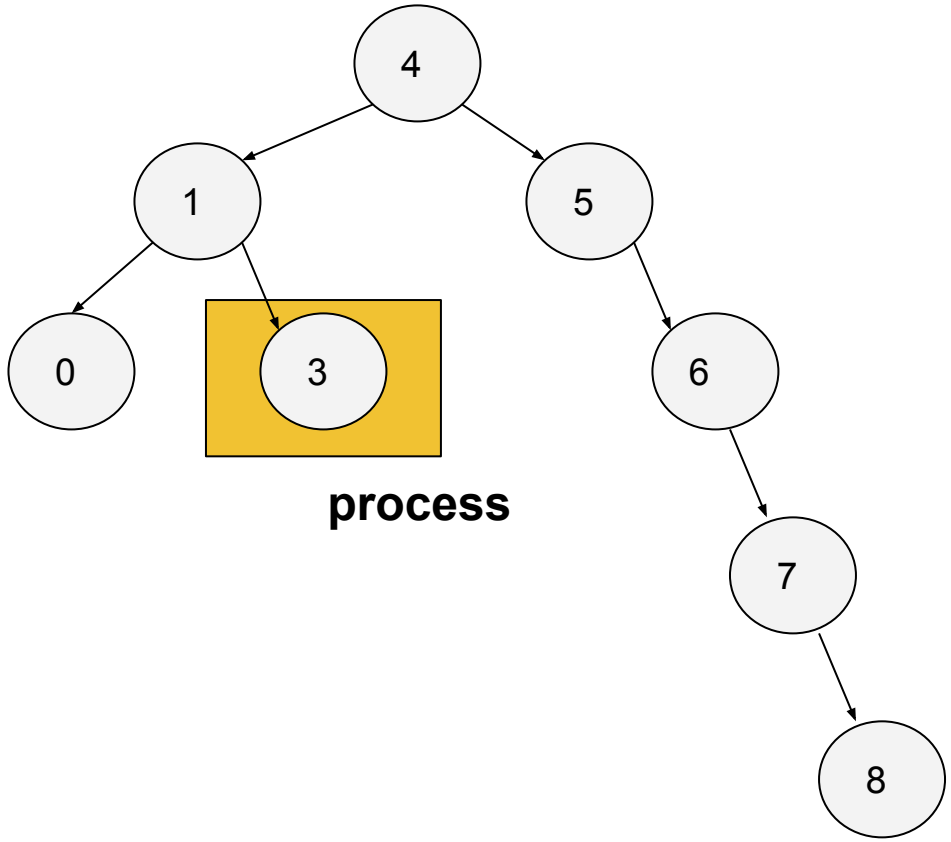
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3]



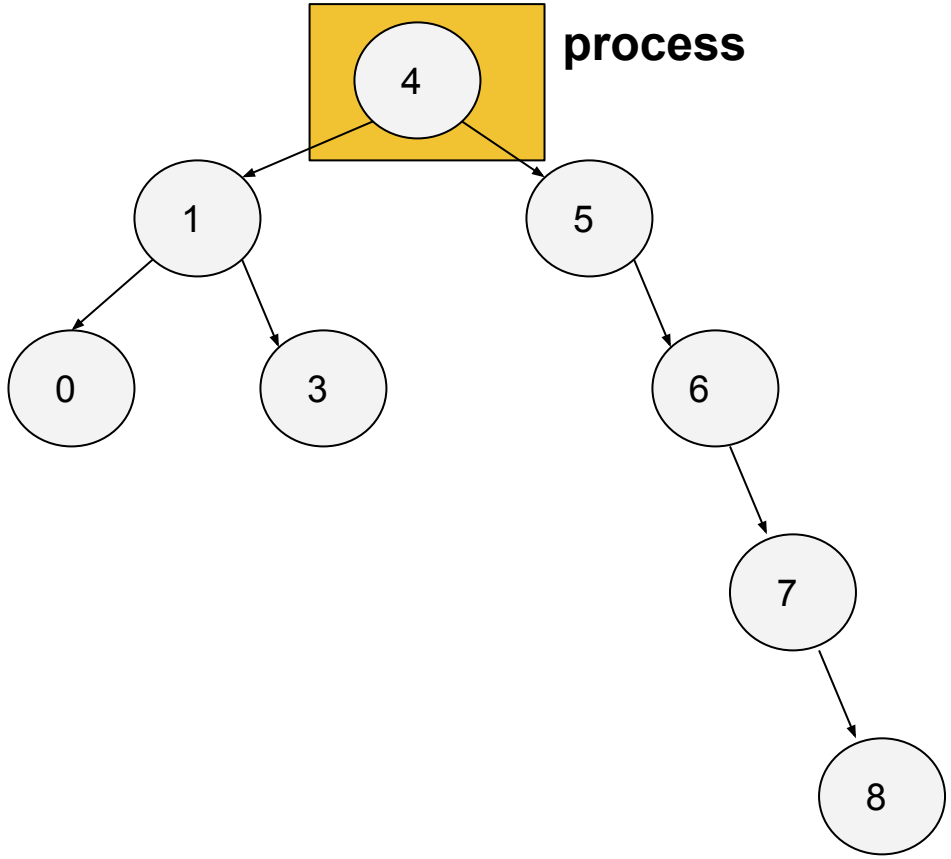
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4]



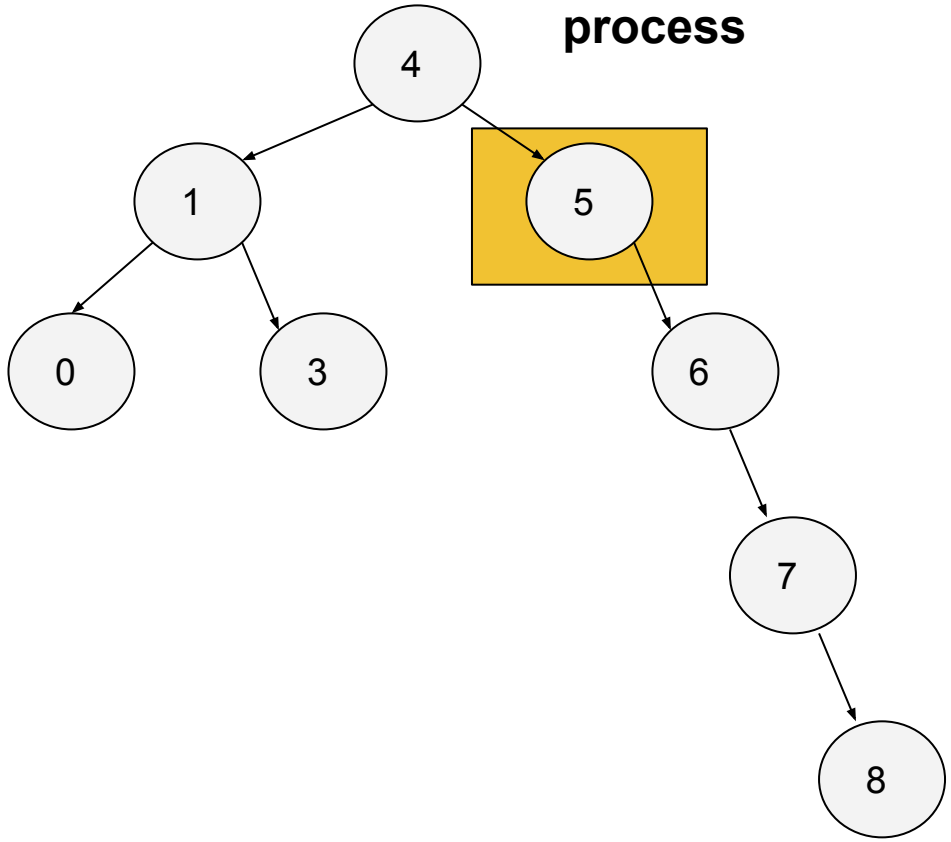
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4 - 5]



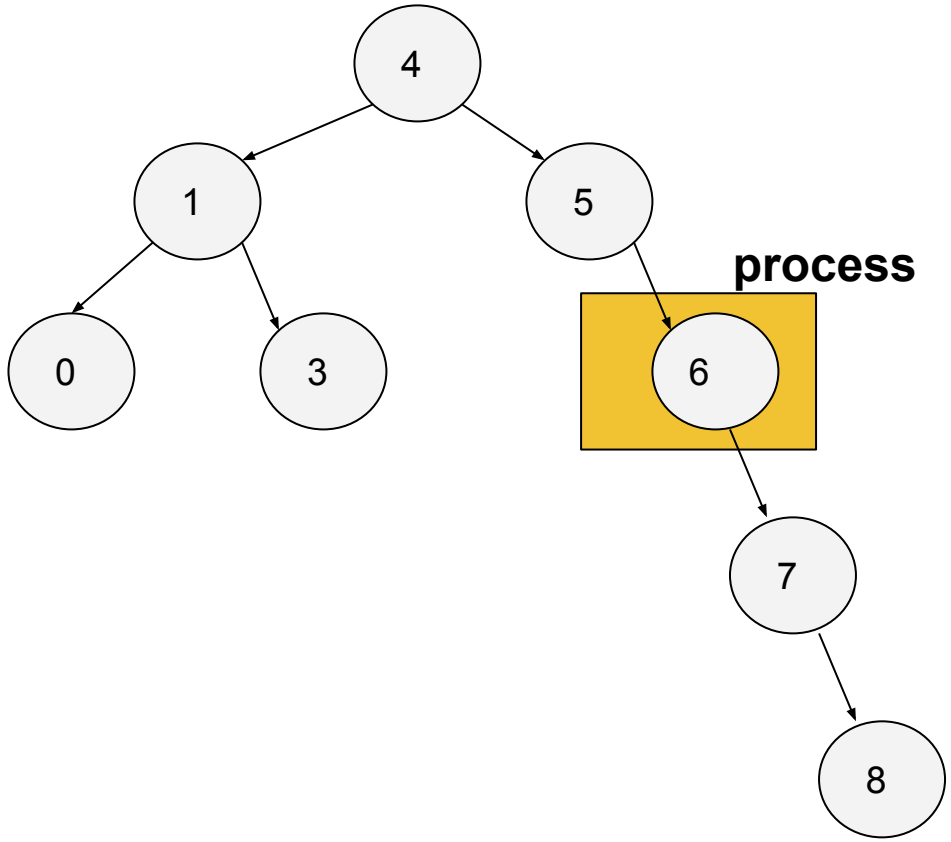
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4 - 5 - 6]



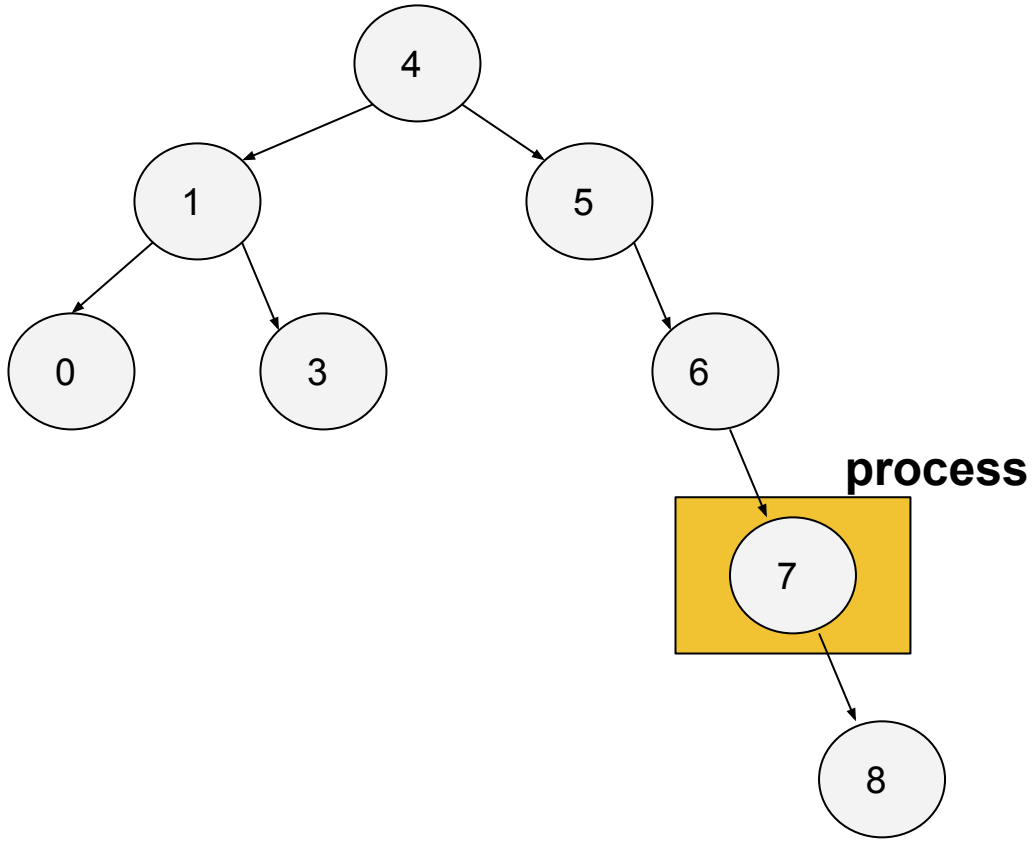
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4 - 5 - 6 - 7]



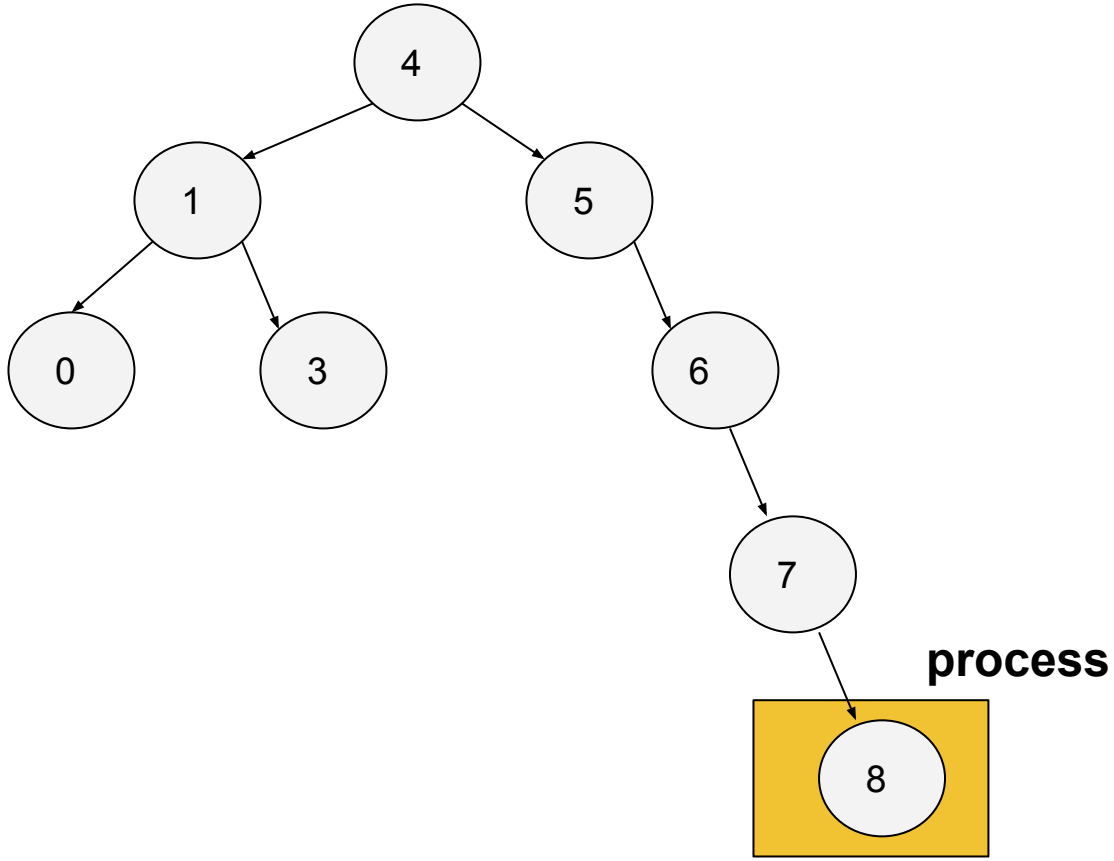
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4 - 5 - 6 - 7 - 8]



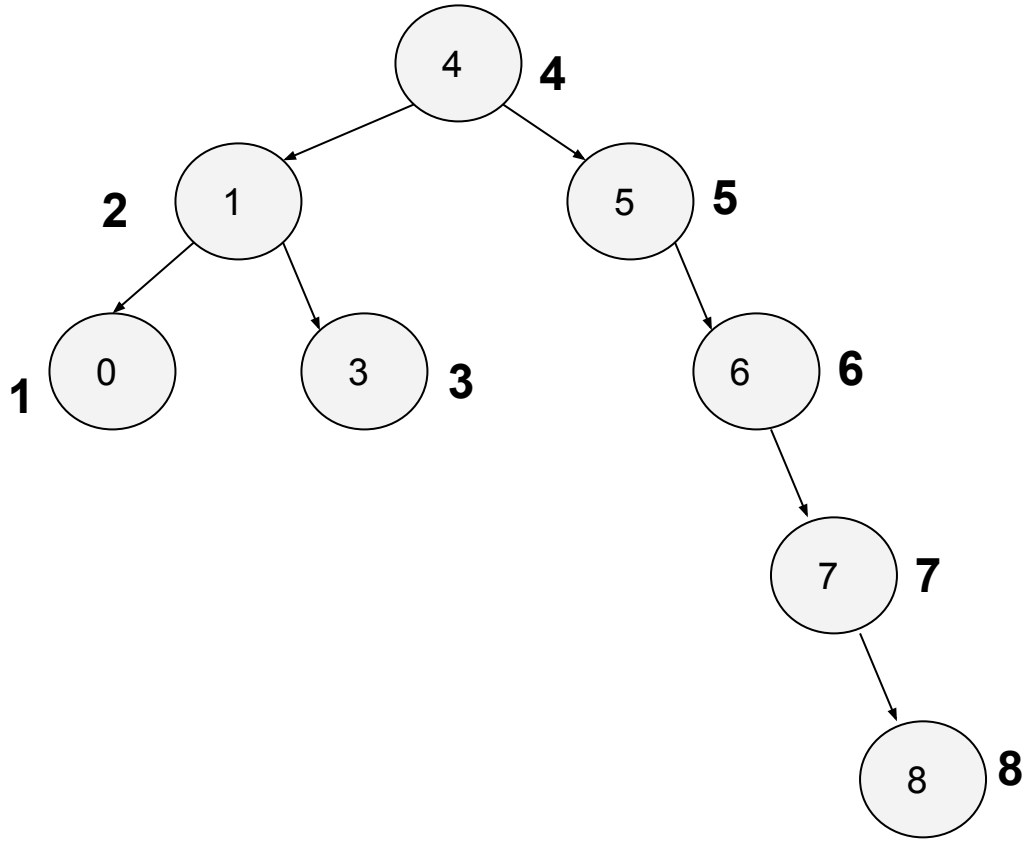
Übung 4 - Traversieren & Balancieren

InOrder

1. process cur.left
2. process cur
3. process cur.right

Ausgabe

[0 - 1 - 3 - 4 - 5 - 6 - 7 - 8]



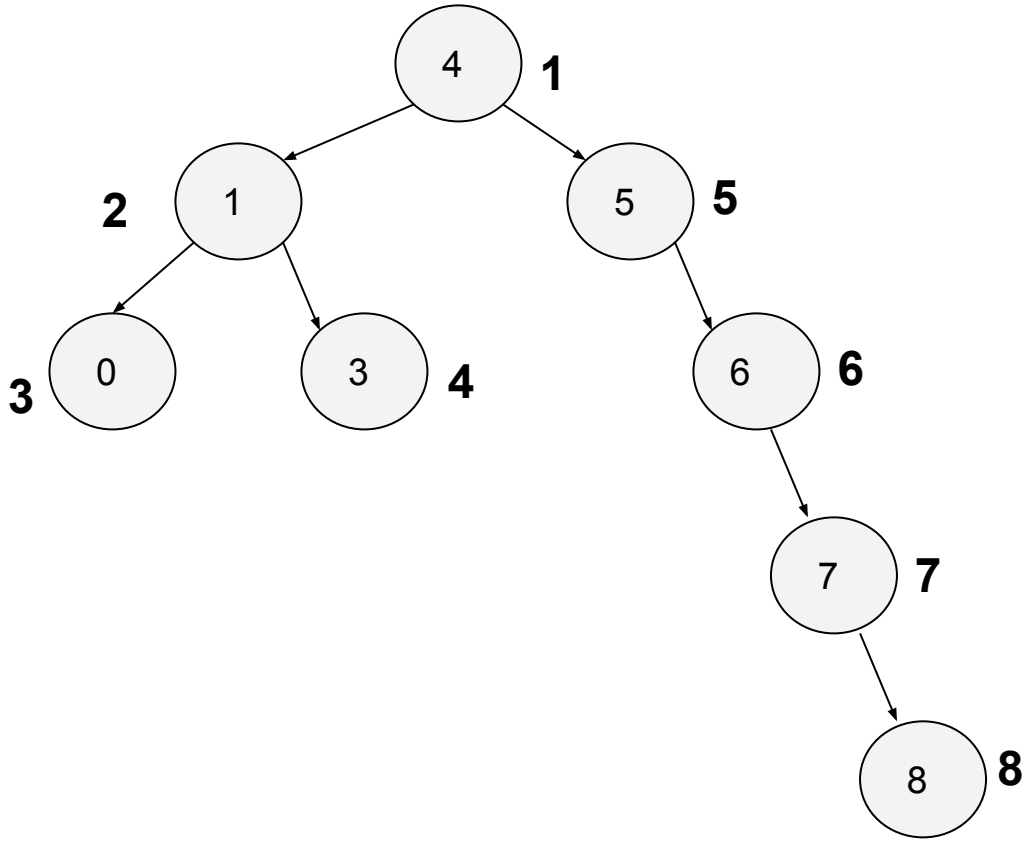
Übung 4 - Traversieren & Balancieren

PreOrder

1. process cur
2. process cur.left
3. process cur.right

PreOrder

[4 - 1 - 0 - 3 - 5 - 6 - 7 - 8]



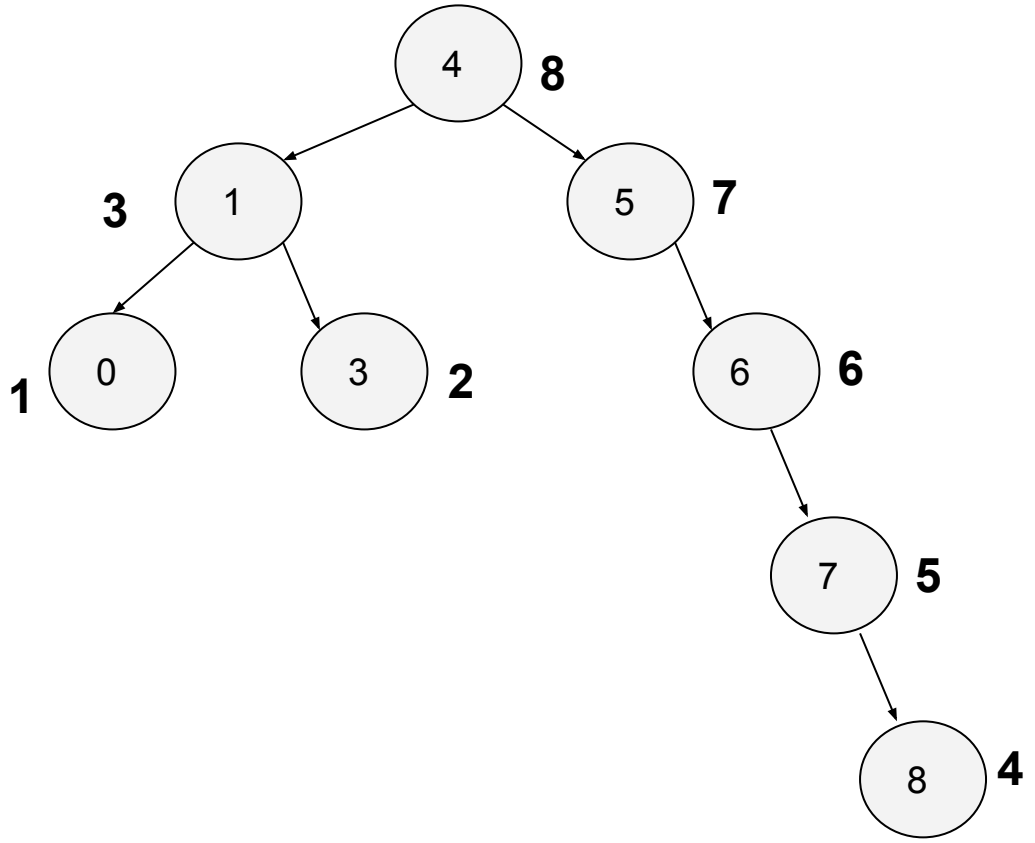
Übung 4 - Traversieren & Balancieren

PostOrder

1. process cur.left
2. process cur.right
3. process cur

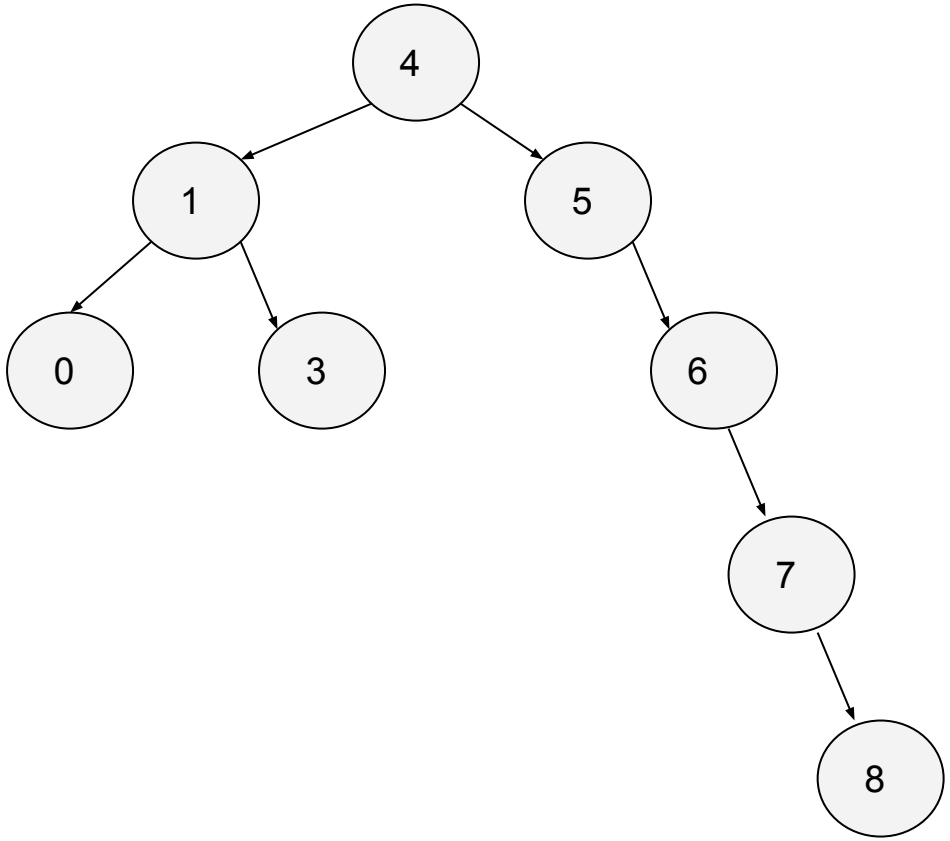
PostOrder

[0 - 3 - 1 - 8 - 7 - 6 - 5 - 4]



Übung 4 - Baum aus InOrder + PreOrder aufbauen

Zielbaum



<u>Gegeben</u>

PreOrder:

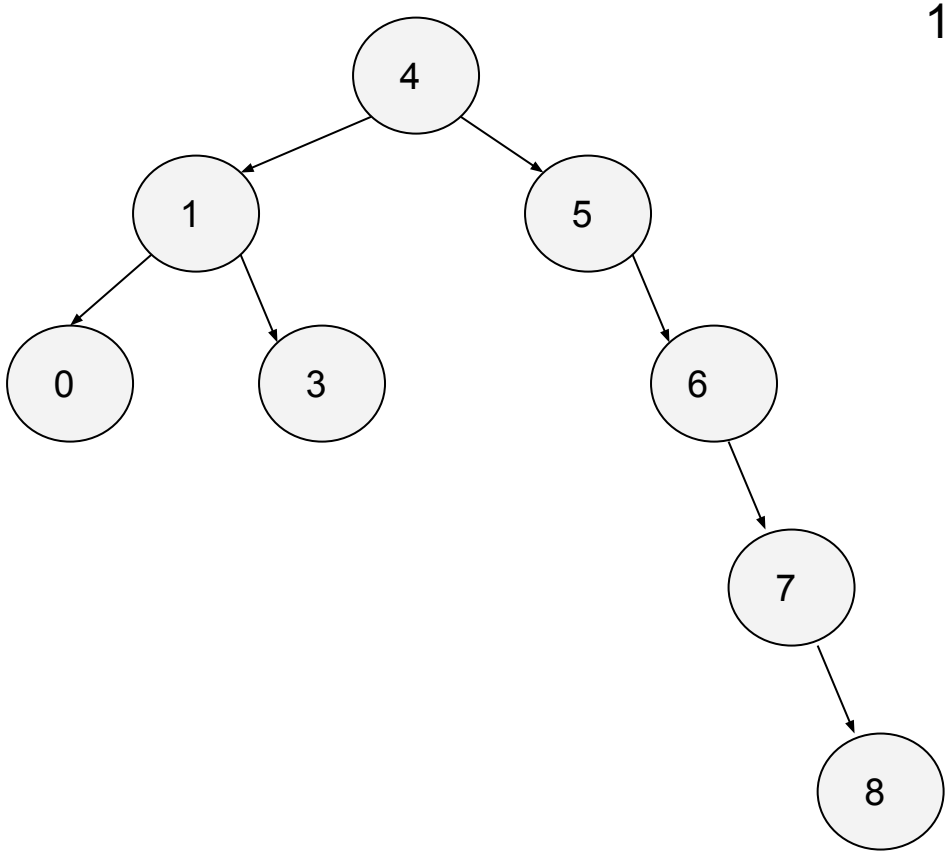
[4 - 1 - 0 - 3 - 5 - 6 - 7 - 8]

InOrder:

[0 - 1 - 3 - 4 - 5 - 6 - 7 - 8]

Übung 4 - Idee Rekursive Lösung

Zielbaum



Divide and Conquer

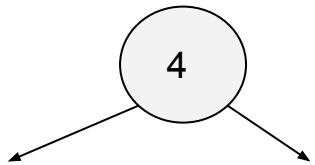
1. Baue current Node
   a. Baue current.left rekursiv
   b. Baue current.right rekursiv

PreOrder	4	1	0	3	5	6	7	8

InOrder	0	1	3	4	5	6	7	8

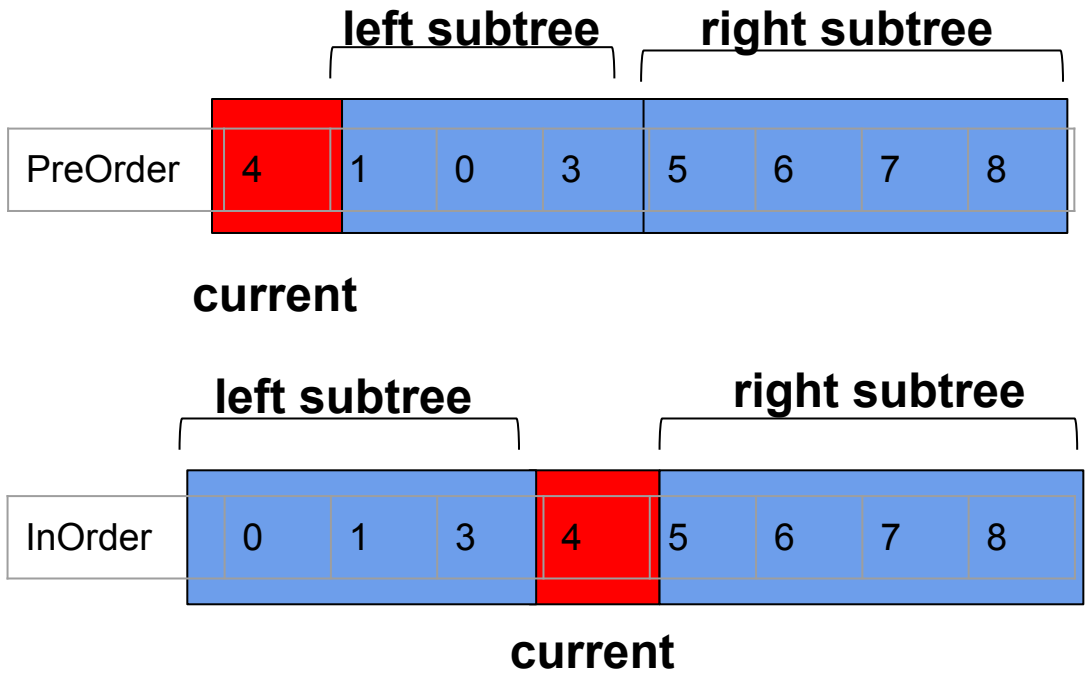
Übung 4 - Idee Rekursive Lösung

Baum



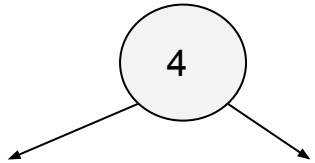

```
BinaryTreeNode buildTree0(int[] inorder,int[] preorder,int
minPreIdx,int maxPreIdx,int minInIdx,int maxInIdx)
```

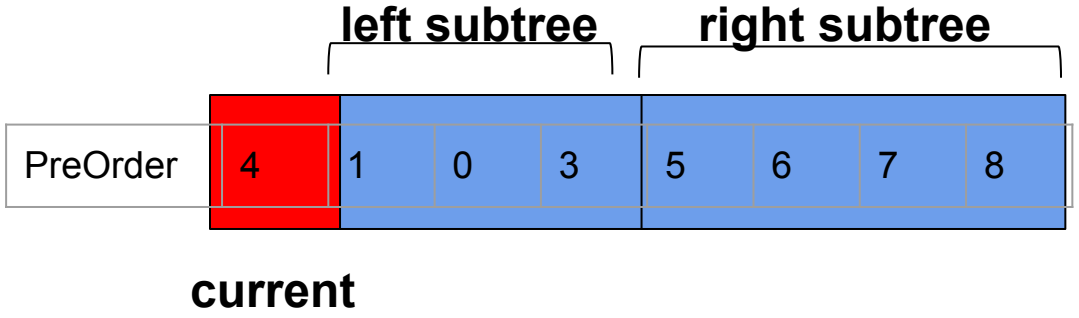
- Wir müssen in jedem Array den linken & rechten Subbaum finden.
- Rekursion über **preOrderArray** ist in preOrder.
- **InOrderArray** gibt uns die Größe des linken und rechten Subbaum
- `current.left = buildTree0(...)`
- `current.right = buildTree0(...)`



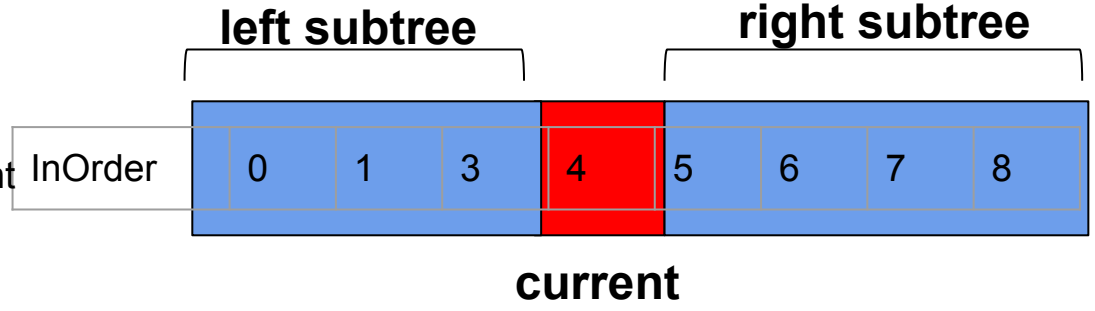
Übung 4 - PreOrder Indizes Berechnen

Baum



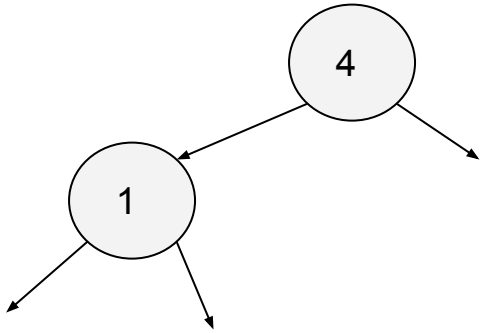


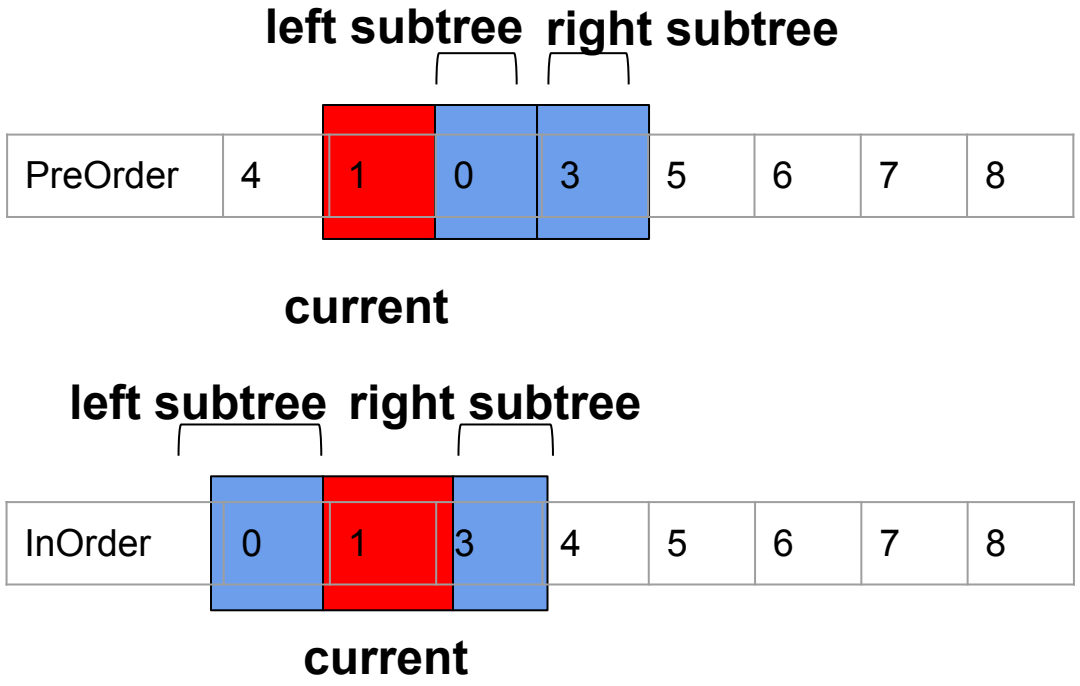
- Rekursion für `current.left`
  - `current = root = 4`
  - `currentPreOrderIndex = 0`
  - `current.left.preOrderIndex = 1`
- Rekursion für `current.right`
  - `preOrder[oldPreIndex + (currentInOrderIndex-oldInOrderIndex)+1]`
  - `oldPreIndex = 0 (at start)`
  - `currentInOrderIndex = 3`
  - `oldInOrderIndex = 0`
  - → `preOrder[0+(3-0)+1]=preOrder[4]=5`
- InOrder Traversierung wird benutzt (mit sub tree size) um left & right Index in PreOrder zu finden



Übung 4 - Idee Rekursive Lösung

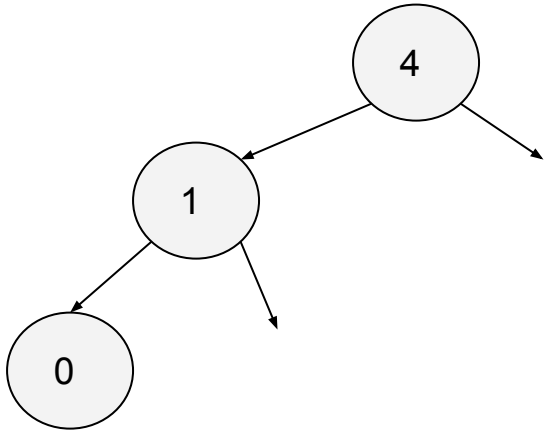
Baum



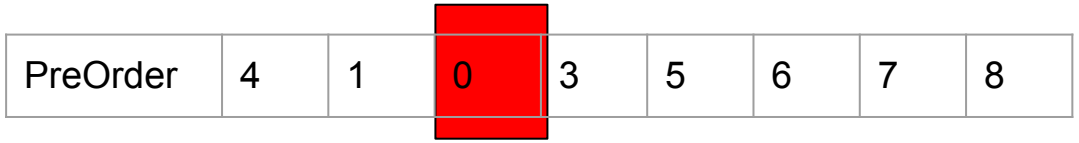


Übung 4 - Idee Rekursive Lösung

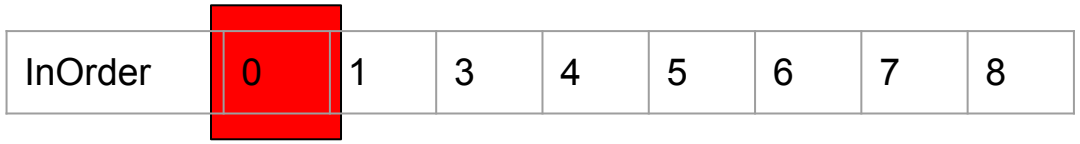
Baum



No children



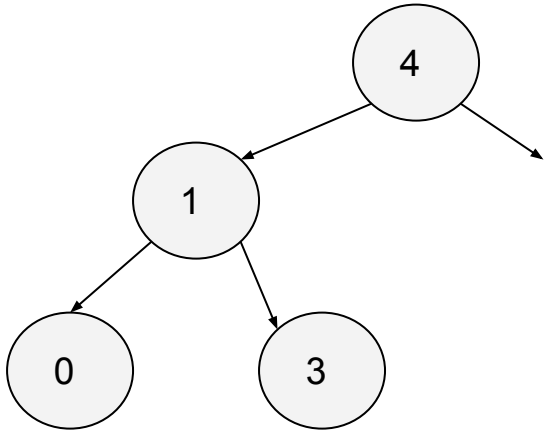
current



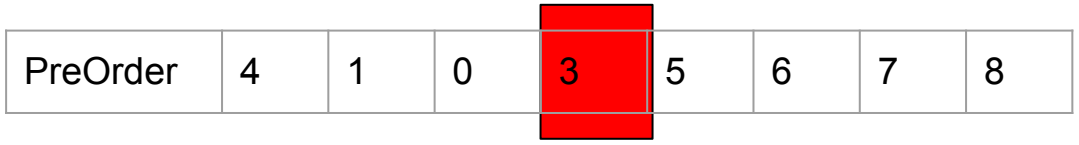
current

Übung 4 - Idee Rekursive Lösung

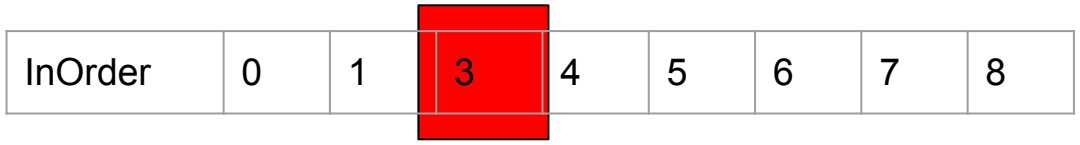
Baum



No children



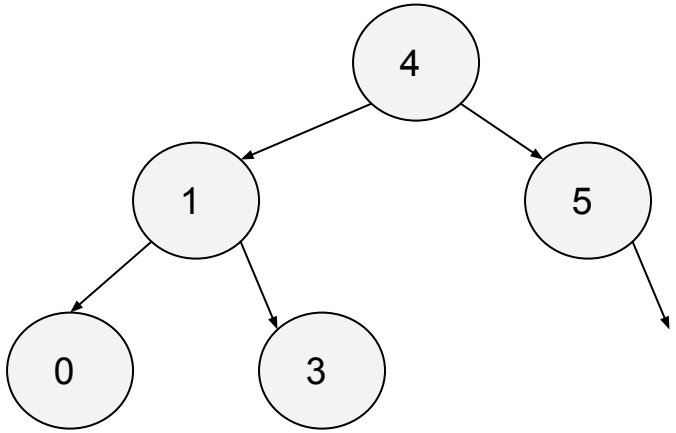
current

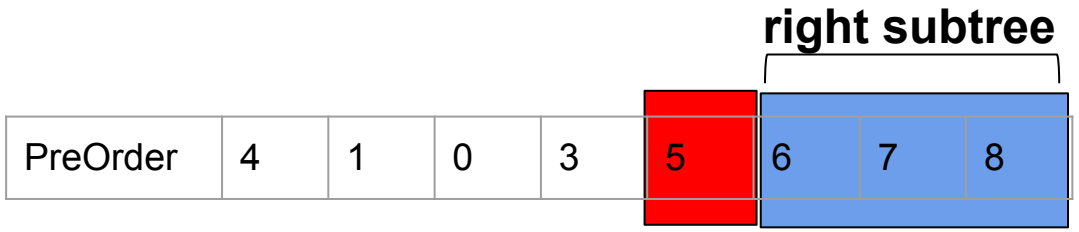


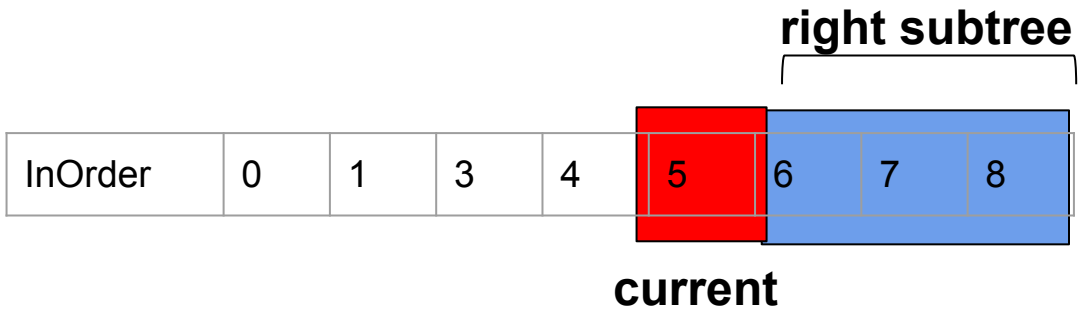
current

Übung 4 - Idee Rekursive Lösung

Baum

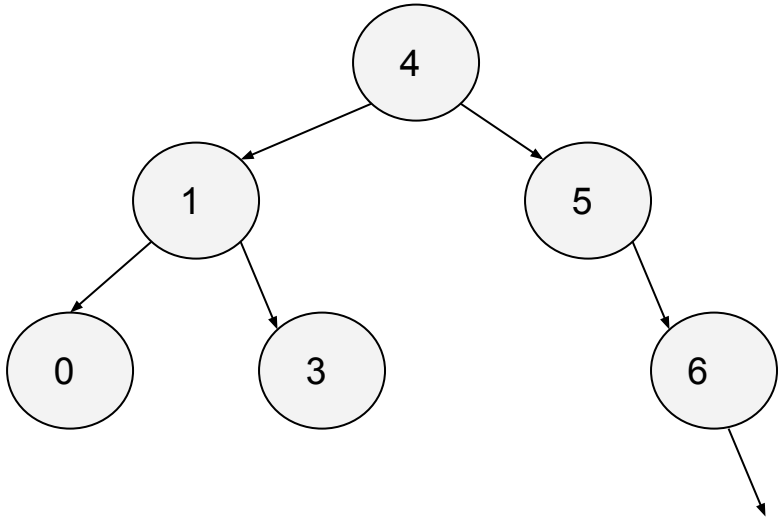


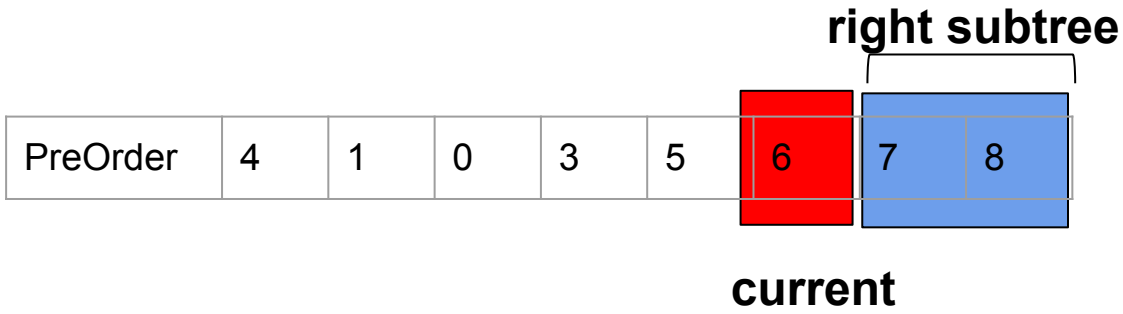


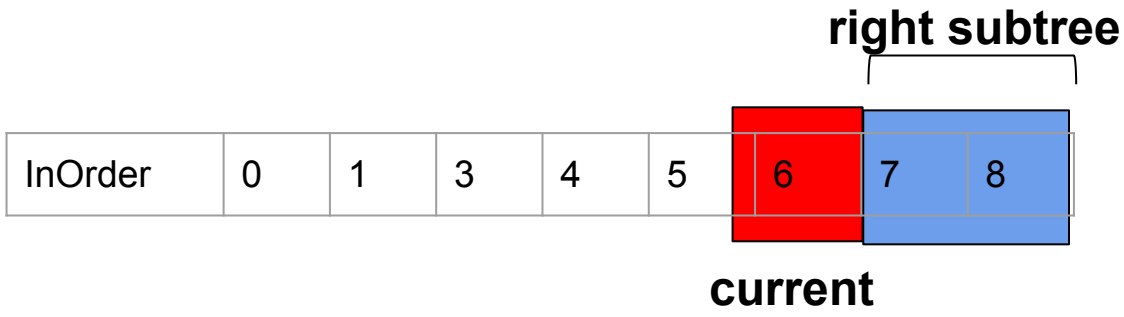


Übung 4 - Idee Rekursive Lösung

Baum

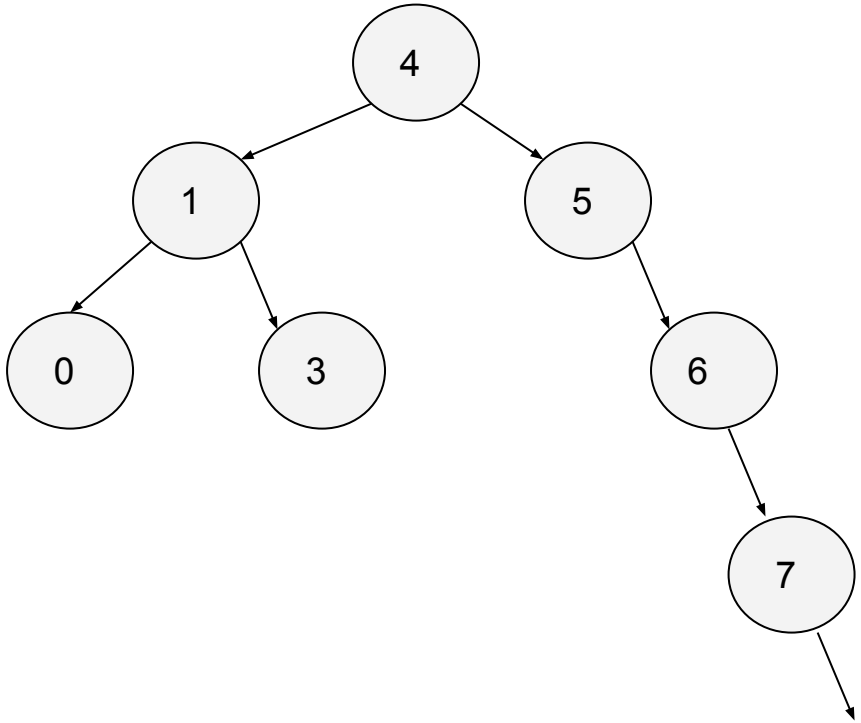


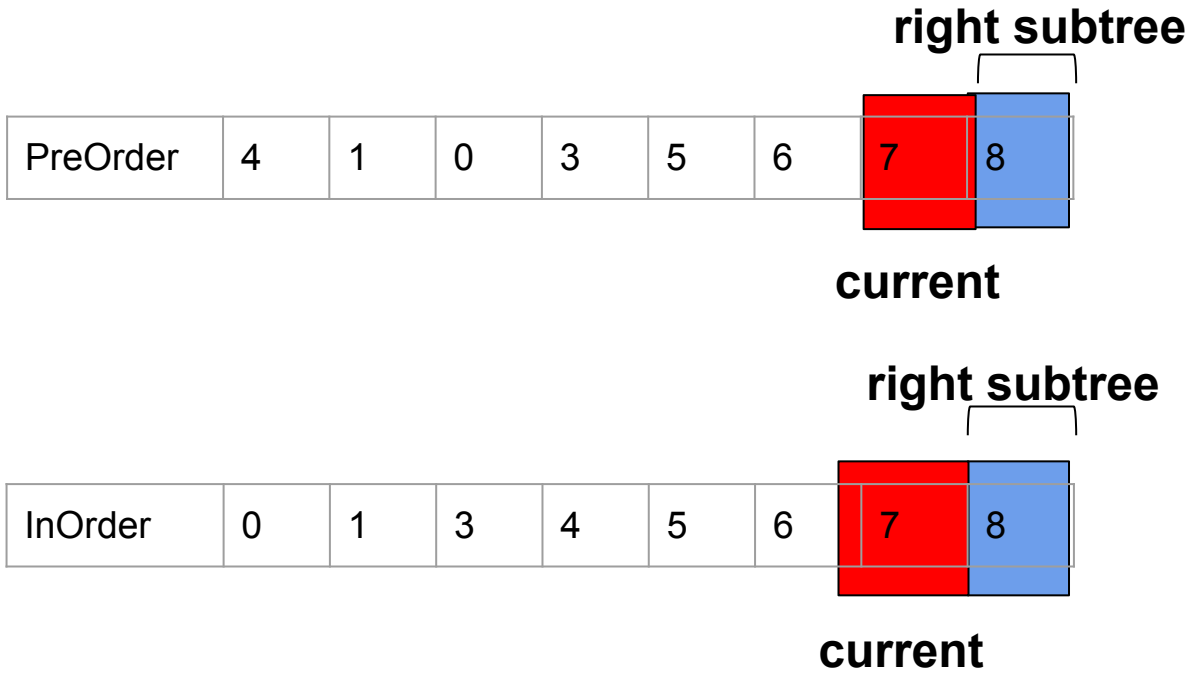




Übung 4 - Idee Rekursive Lösung

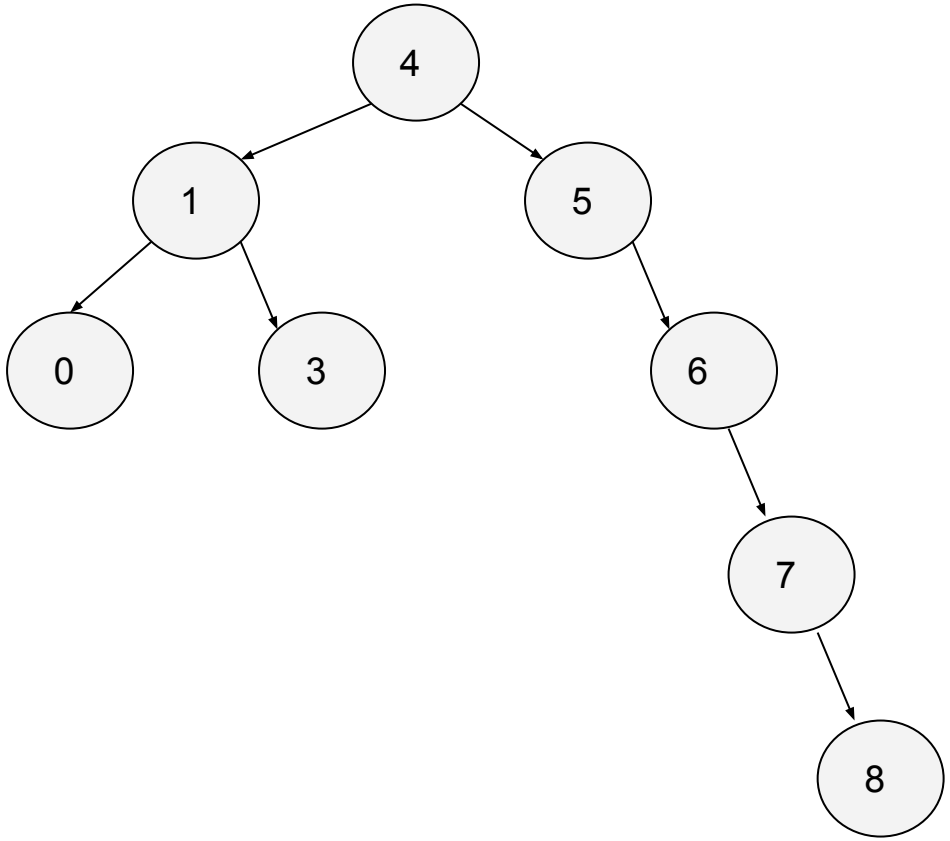
Baum



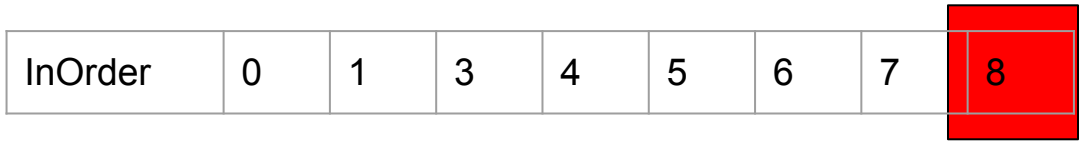


Übung 4 - Idee Rekursive Lösung

Baum



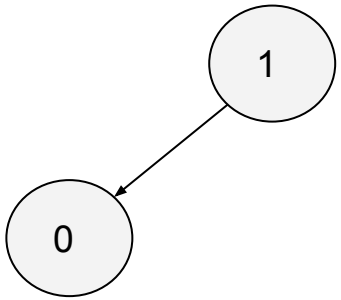
current

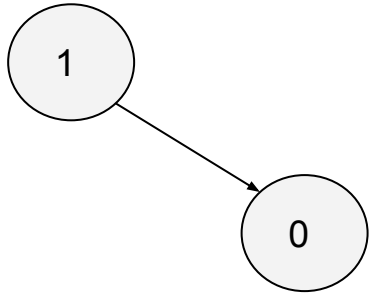


current

Übung 4 - Selbe Lösung PreOrder + PostOrder ?

Nein





PreOrder : 1 - 0

PostOrder: 0 - 1

PreOrder : 1 - 0

PostOrder: 0 - 1

JOHANNES KEPLER
UNIVERSITY LINZ

Danke

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at