Rekursive Traversierungen

Anzahl der Elemente in eines Baum rekursiv:

```
anzahl(b: Binaerbaum):
  falls b == null:
    return 0
  sonst:
    t1 = anzahl(b.links)
    t2 = anzahl(b.rechts)
    return 1 + t1 + t2
```

Probleme, wenn der Baum jedoch sehr tief (> 20.000) ist:

```
java.lang.StackOverflowError
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
```

cc BY NC SA

Rekursive Traversierungen

Speicherplatz für den Aufruf-Stack ist in Java ca. 256 kB.

Jeder Methodenaufruf benötigt Platz auf dem Stack, wenn keine weiteren Aufrufe auf dem Stack abgelegt werden können → *Stack overflow*.

Abhilfe: Man schreibt die Methode in eine *iterative* Variante um und verwaltet manuell, welche Knoten bereits besucht wurden und welche als nächstes besucht werden müssen.

```
java.lang.StackOverflowError
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
        at Baumalgorithmen.anzahl(Baumalgorithmen.java:73)
```

cc BY NC SA

Rekursion → Iteration

```
anzahl(b: Binaerbaum):
  falls b == null:
    return 0
  sonst:
    t1 = anzahl(b.links)
    t2 = anzahl(b.rechts)
    return 1 + t1 + t2
```

```
anzahl(b: Binaerbaum):
  todo = new Stack
  todo.push(b)
  zaehler = 0
  solange todo nicht leer:
    tmp = todo.pop()
    zaehler++
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  return zaehler
```

Wir holen uns den nächsten Baumknoten und zählen ihn.

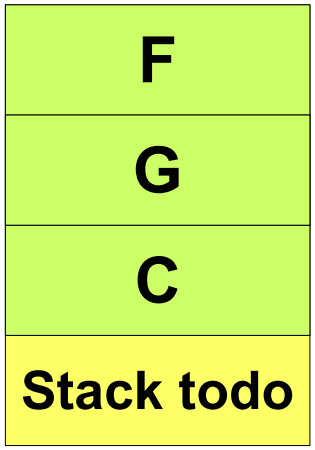
Dann merken wir uns die beiden Kinder, die als nächstes verarbeitet werden sollen, sofern sie existieren, in einem „eigenen“ Stack.

Das Programm geht davon aus, dass der Baum b nicht bereits leer ist.

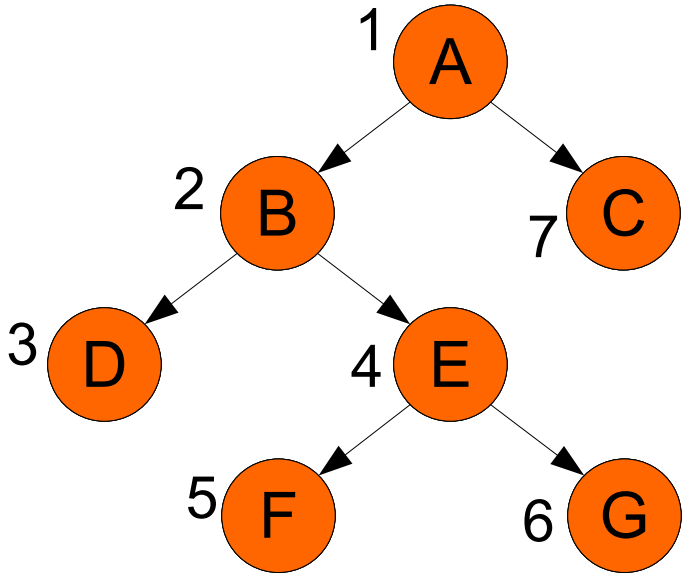
cc BY NC SA

Iterative Variante – Beispiel

```
anzahl(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  zaehler = 0
  solange todo nicht leer:
    tmp = todo.pop()
    zaehler++
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  return zaehler
```



zaehler = 6



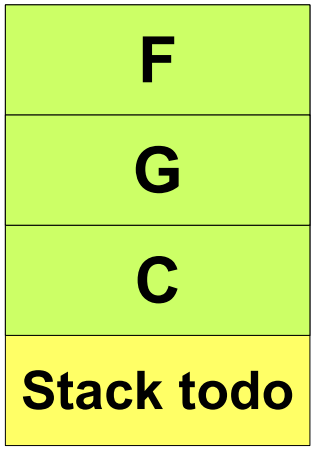
Aufgabe: Vollziehe schrittweise nach, wie der Baum durchlaufen wird.

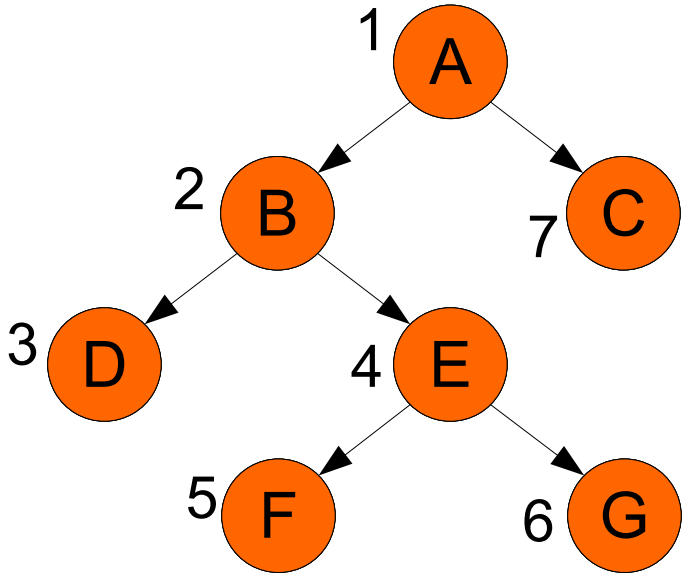
- Wie sieht der Inhalt des Stacks aus
  (vorne→unten, hinten→oben, der Stack im Beispiel wäre also `C,G,F`)
- Welchen Wert hat `zaehler` ?

R. Helfrich/F.Schiebel

Iterative Variante – Beispiel

```
anzahl(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  zaehler = 0
  solange todo nicht leer:
    tmp = todo.pop()
    zaehler++
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  return zaehler
```





zaehler = 6

Ablauf:

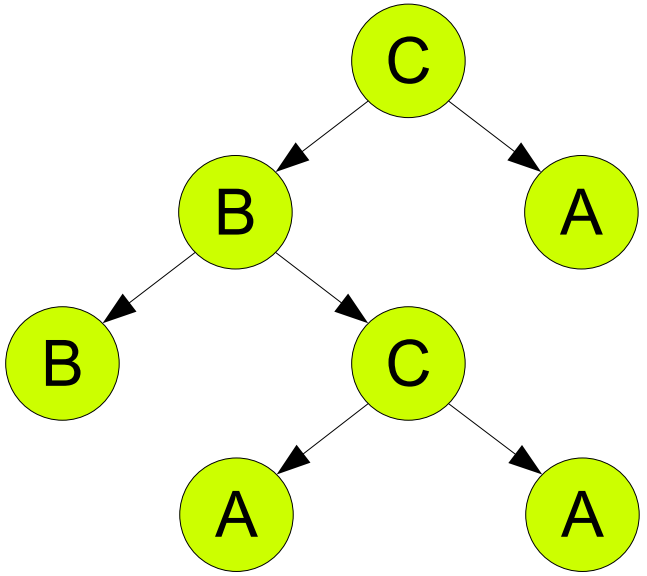
1. push(A)
   Stack: A
2. pop() → A
3. zaehler=1
4. push(C)
5. push(B)
   Stack: C,B
6. pop() → B
   Stack: C
7. zaehler=2
8. push(E)
   Stack: C,E
9. push(D)
   Stack C,E,D
10. pop() → D
    Stack C,E
11. zaehler=3
12. pop() → E
    Stack: C
13. zaehler=4
14. push(G)
15. push(F)
    Stack: C,G,F
16. pop() → F
    Stack: C,G
17. zaehler=5
18. pop() → G
    Stack: C
19. zaehler=6
20. pop() → C
    Stack: emtpy
21. zaehler=7

R. Helfrich/F.Schiebel

Suche im Baum

Mit einer kleinen Abwandlung kann man die Suche nach einem bestimmten Wert im Baum durchführen.

→ Wir suchen *einen* Knoten, der ein **`A`** enthält



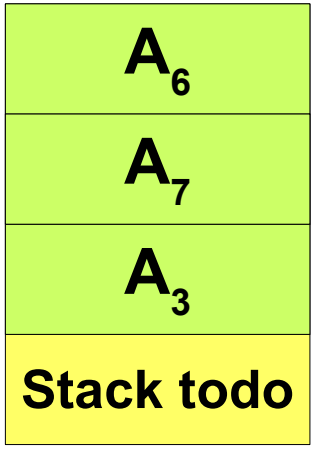

```
findeA(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  solange todo nicht leer:
    tmp = todo.pop()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  ausgabe → A wurde nicht gefunden
```

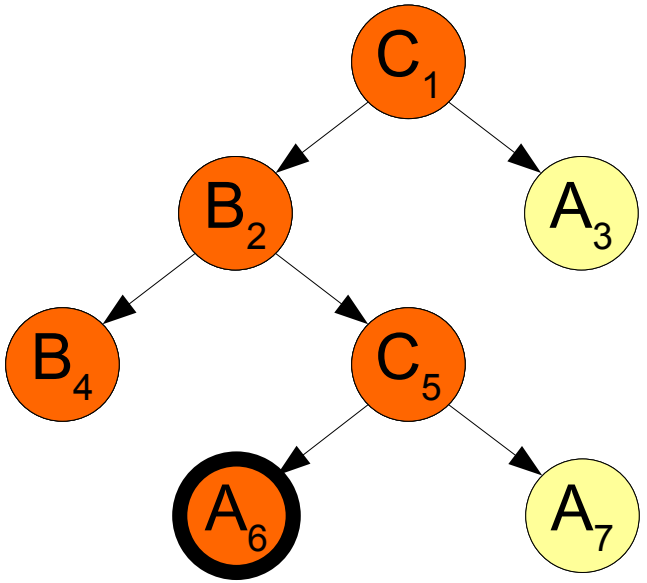
Aufgabe:
Ermittle, in welcher Reihenfolge die Knoten durchlaufen werden – welcher Knoten wird gefunden?

R. Helfrich/F.Schiebel

Suche im Baum

```
findeA(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  solange todo nicht leer:
    tmp = todo.pop()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  ausgabe „A nicht gefunden“
```





Ablauf:

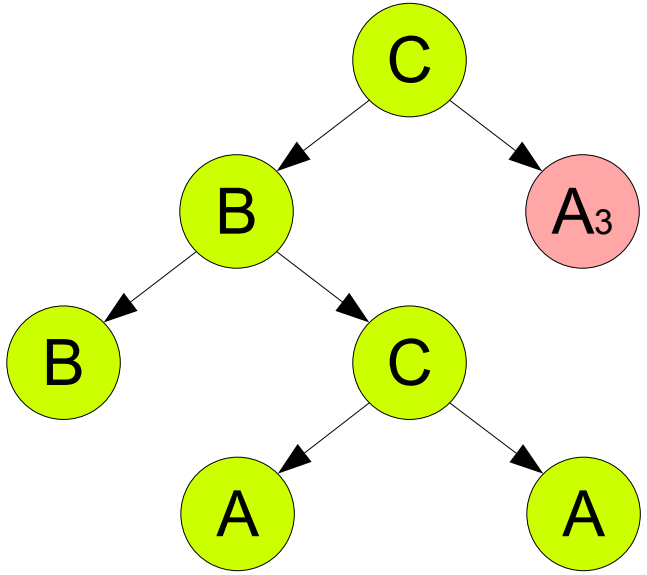
1. push(C_1)
2. pop() → C_1
3. push(A_3)
4. push(B_2)
5. pop() → B_2
6. push(C_5)
7. push(B_4)
8. pop() → B_4
9. pop() → C_5
10. push(A_7)
11. push(A_6)
12. pop() → A_6
13. Gefunden → Ende

R. Helfrich/F.Schiebel

Suche im Baum

Der Algorithmus findet den „am weitesten links stehenden“ Knoten, der ein A enthält („Tiefensuche“, links vor rechts).

Was, wenn man aber einen Knoten finden, möchte, der möglichst
nahe an der Wurzel steht (hier Knoten A_3)?

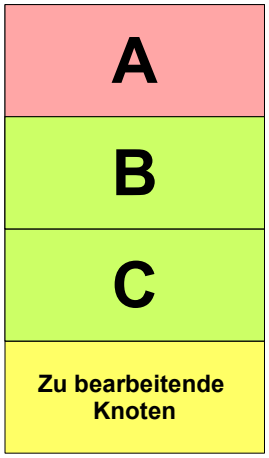


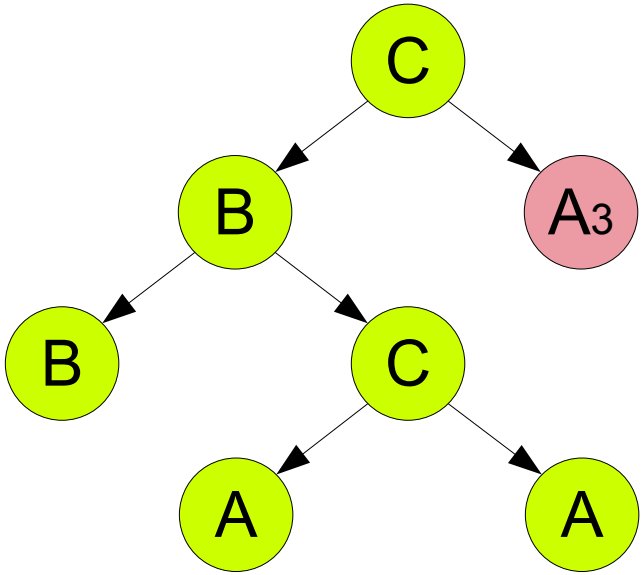
In diesem Fall muss man den Baum “schichtenweise“ abarbeiten, d.h. erst die Wurzel, dann die Kinder, dann deren Kinder, … → „Levelorder“

cc BY NC SA

Suche im Baum

```
findeA(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  solange todo nicht leer:
    tmp = todo.pop()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  ausgabe „A nicht gefunden“
```



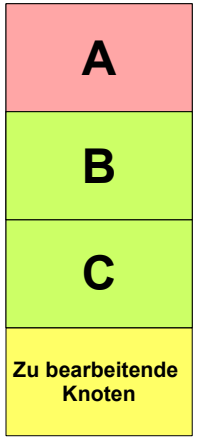


Frage:
Was könnte man am gegebenen Algorithmus verändern, damit der altrosa gefärbte knoten mit dem A_3 gefunden wird und die Knoten „Levelorder“ durchlaufen werden? (Also C→B→A→ B→ C → A → A)

cc BY NC SA

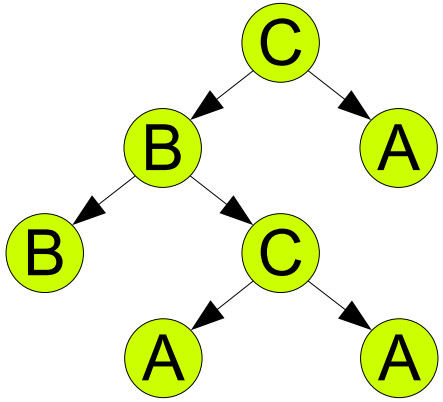
Suche im Baum

```
findeA(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  solange todo nicht leer:
    tmp = todo.pop()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  ausgabe „A nicht gefunden“
```




```
findeA(b: Binaerbaum):
  todo = neue Queue
  todo.enqueue(b)
  solange todo nicht leer:
    tmp = todo.dequeue()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.enqueue(tmp.rechts)
    falls tmp.links != null:
      todo.enqueue(tmp.links)
  ausgabe „A nicht gefunden“
```

Die Todo-Liste ist ein Stack

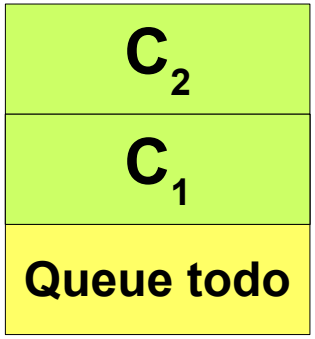


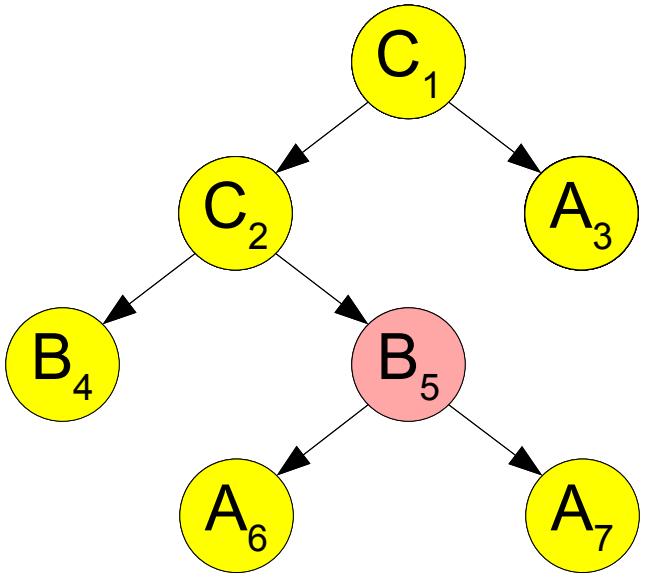
Die Todo-Liste ist eine Schlange (Queue)

cc BY NC SA

Suche im Baum

```
findeB(b: Binaerbaum):
  todo = neue Queue
  todo.enqueue(b)
  solange todo nicht leer:
    tmp = todo.dequeue()
    falls tmp = B
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.enqueue(tmp.rechts)
    falls tmp.links != null:
      todo.enqueue(tmp.links)
  ausgabe „B nicht gefunden“
```





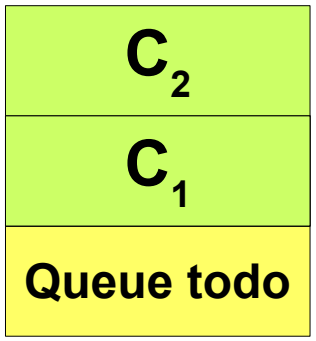
Aufgabe: Vollziehe schrittweise nach, wie der Baum durchlaufen wird, wenn mittels Breitensuche B_5 gefunden wird.

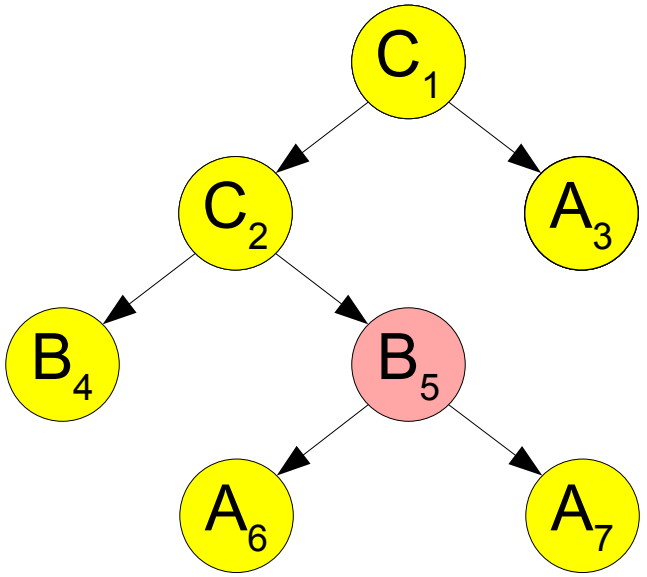
- Wie sieht jeweils der Inhalt der Queue aus

cc BY NC SA

Suche im Baum

```
findeA(b: Binaerbaum):
  todo = neue Queue
  todo.enqueue(b)
  solange todo nicht leer:
    tmp = todo.dequeue()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.enqueue(tmp.rechts)
    falls tmp.links != null:
      todo.enqueue(tmp.links)
  ausgabe „A nicht gefunden“
```





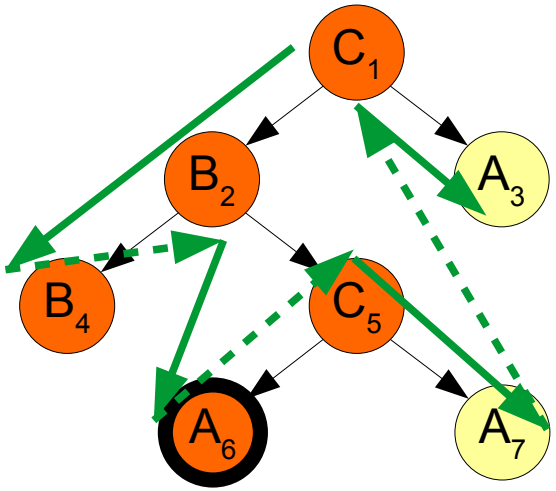
Ablauf:

1. enqueue(C_1)
2. dequeue() → C_1
3. enqueue(C_2)
4. enqueue(A_3)
   Queue: A3,C2
5. dequeue() → C_2
6. enqueue(B_5)
7. enqueue(B_4)
   Queue: B4,B5,A3
8. dequeue() → A_3
9. dequeue() → B5 → Ende

R. Helfrich/F.Schiebel

Suche im Baum

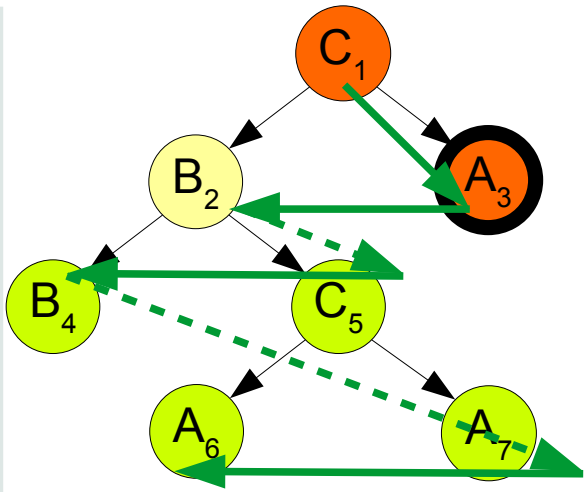
```
findeA(b: Binaerbaum):
  todo = neuer Stack
  todo.push(b)
  solange todo nicht leer:
    tmp = todo.pop()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.push(tmp.rechts)
    falls tmp.links != null:
      todo.push(tmp.links)
  ausgabe „A nicht gefunden“
```



Tiefensuche:
Man sucht erst „tief unten“ im Baum, bevor man sich zur Seite bewegt.

```
findeA(b: Binaerbaum):
  todo = neue Queue
  todo.enqueue(b)
  solange todo nicht leer:
    tmp = todo.dequeue()
    falls tmp = A
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.enqueue(tmp.rechts)
    falls tmp.links != null:
      todo.enqueue(tmp.links)
  ausgabe „A nicht gefunden“
```

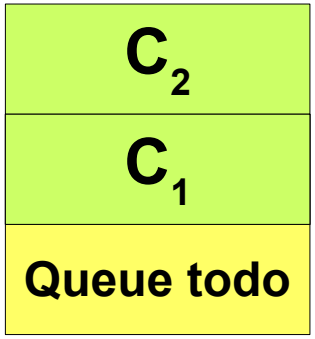
Breitensuche:
Man sucht erst die erste Schicht „auf ganzer Breite“, bevor man eine Ebene tiefer geht.

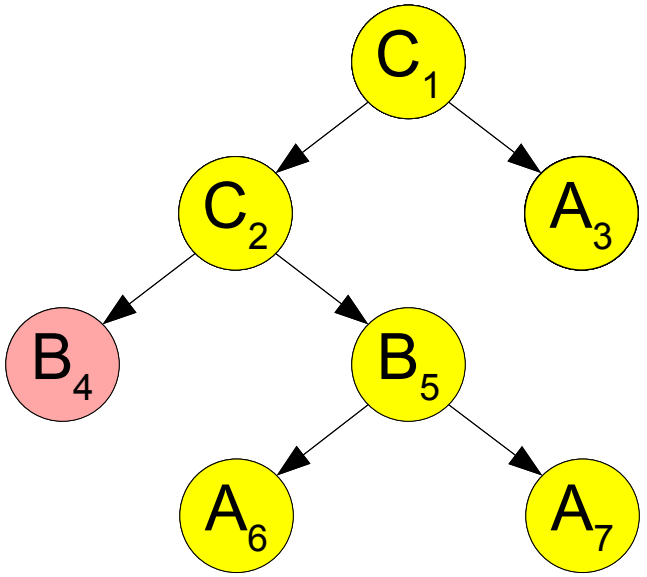


R. Helfrich/F.Schiebel

Suche im Baum

```
findeB(b: Binaerbaum):
  todo = neue Queue
  todo.enqueue(b)
  solange todo nicht leer:
    tmp = todo.dequeue()
    falls tmp = B
      markiere tmp
      beende Methode
    falls tmp.rechts != null:
      todo.enqueue(tmp.rechts)
    falls tmp.links != null:
      todo.enqueue(tmp.links)
  ausgabe „B nicht gefunden“
```





Aufgabe:
Was müsste man am Algorithmus anpassen, damit bei einer **Breitensuche** B_4 gefunden werden würde?

cc BY NC SA