UNIVERSITÄT BERN

Datenstrukturen und Algorithmen 26.4.18

Binäre Suchbäume

Aufgabe 1

1. Gegeben seien die Schlüssel $\{2, 4, 9, 13, 17, 21, 24\}$. Zeichnen Sie binäre Suchbäume der Höhe $2, 3, 4, 5$ und 6. **1 Punkt**

- Beinahe alle richtig gelöst

- **Wichtig: Auch wenn die Übung trivial ist, binäre Suchbaum-Eigenschaften müssen erhalten bleiben.**
- **Root gehört nicht zur Höhe!**

Aufgabe 2

2. Was ist der Unterschied zwischen der binären Suchbaum-Eigenschaft und der Min-Heap Eigenschaft? Kann die Min-Heap Eigenschaft benutzt werden, um Schlüssel in einem Baum mit n Knoten in sortierter Reihenfolge in $O(n)$ Zeit auszugeben? Gibt es einen vergleichenden Algorithmus, der für beliebige Schlüsselfolgen einen binären Suchbaum in $O(n)$ aufbaut? Erklären Sie Ihre Antwort. **1 Punkt**

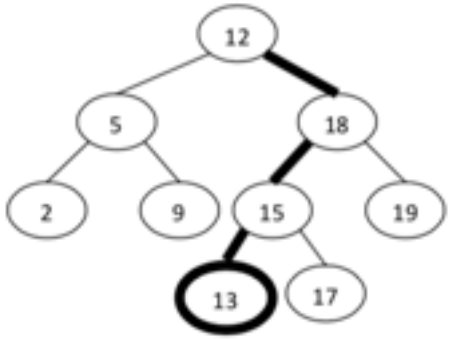
- Binärer Suchbaum: Für jeden Knoten x gilt, linker Teilbaum ist kleiner gleich x.key. Analog für rechter Teilbaum grösser gleich x.key.
- Min Heap: Parent < Child
- Min Heap sagt nichts über die Ordnung aus! Daher kann er nicht verwendet werden.
- Vergleichende Algorithmen optimal *O(n log n)*

Aufgabe 3

3. Angenommen, die Suche nach einem Schlüssel k in einem binären Suchbaum endet in einem Blatt. Wir unterscheiden drei Mengen: A, die Schlüssel links vom Suchpfad, B die Schlüssel auf dem Suchpfad, und C, die Schlüssel rechts vom Suchpfad. Die Vermutung ist, dass für jeweils drei Schlüssel $a \in A, b \in B$ und $c \in C$ gilt, dass $a \leq b \leq c$. Widerlegen Sie diese Vermutung mit einem Gegenbeispiel, das einen möglichst kleinen Baum verwendet. **1 Punkt**

$A = 5, 2, 9$
$B = 12, 18, 15, 13$
$C = 19, 17$

$a \leq b \leq c \rightarrow 18 \leq 17$ (!) Widerspruch



Aufgabe 4

4. Schreiben Sie Pseudocode für eine rekursive Version des Einfügens eines Knotens in einen nicht leeren binären Suchbaum. Beschreiben Sie ihren Algorithmus in 1-2 Sätzen. **1 Punkt**

```
binInsert(Node n, Object x)
    if(n.key < x.key) // Rechter Teilbaum
        if(n.rightChild = NIL)
            n.rightChild = x
            x.parent = n
        else
            binInsert(n.rightChild, x)
    else        // Linker Teilbaum
        if(n.leftChild = NIL)
            n.leftChild = x
            x.parent = n
        else
            binInsert(n.leftChild, x)
```

Aufgabe 5

5. Ein Knoten x in einem binären Suchbaum habe zwei Kinder. Zeigen Sie, dass der Nachfolger von x kein linkes Kind und der Vorgänger von x kein rechtes Kind hat.
1 Punkt

- Nachfolger von x := Minimum des rechten Teilbaumes
- Minimum des rechten Teilbaumes = Linkestes Kind des rechten Teilbaumes
- Widerspruch

Repetition: Aufgabe 1

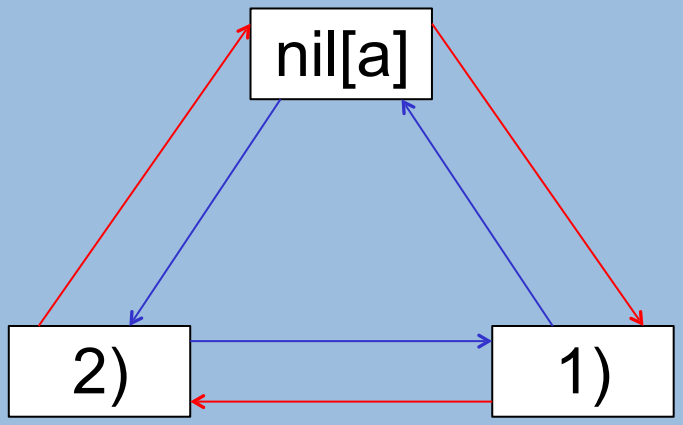
- Zwei zyklische, doppelt verkettete Listen a und b

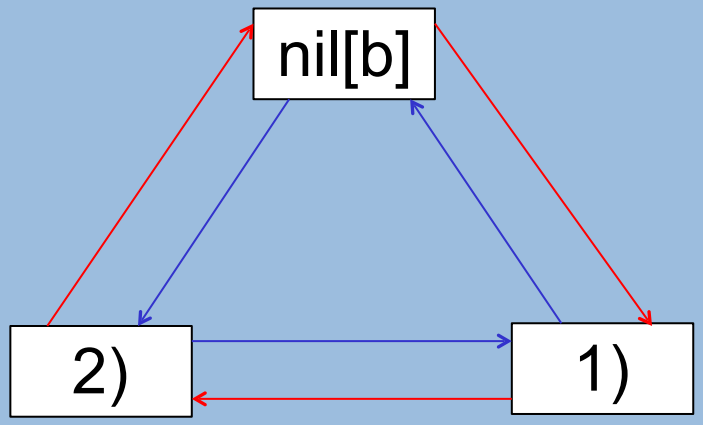
- CONCATENATE(nil[a], nil[b])

```
y ← nil[b].next
z ← nil[b].prev
if (y ≠ nil[b])
    nil[a].prev.next ← y
    y.prev ← nil[a].prev
    z.next ← nil[a]
    nil[a].prev ← z
```

Repetition: Aufgabe 1

> Zwei zyklische, doppelt verkettete Listen a und b

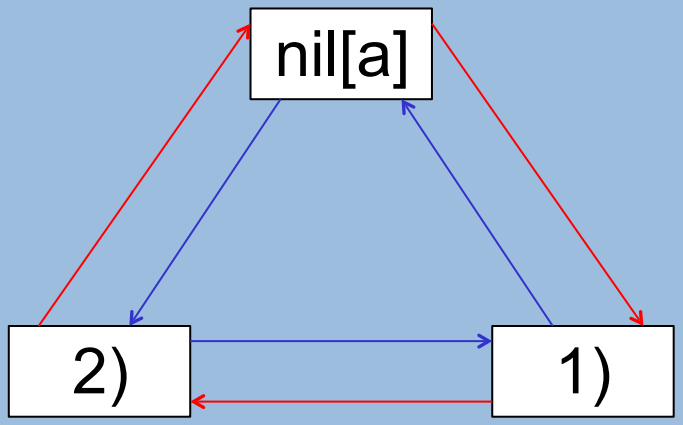


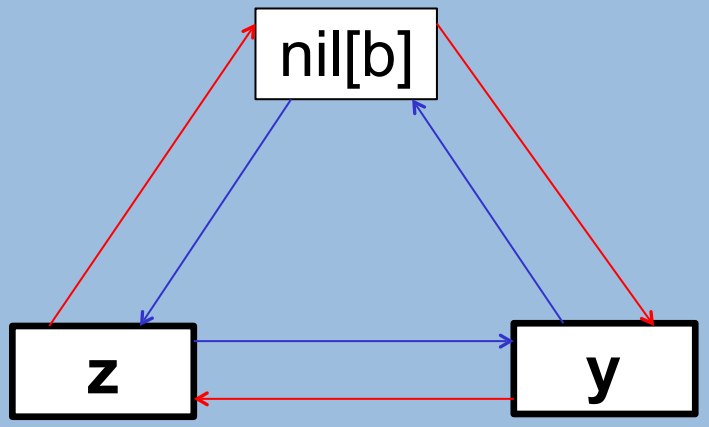



```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

> Zwei zyklische, doppelt verkettete Listen a und b

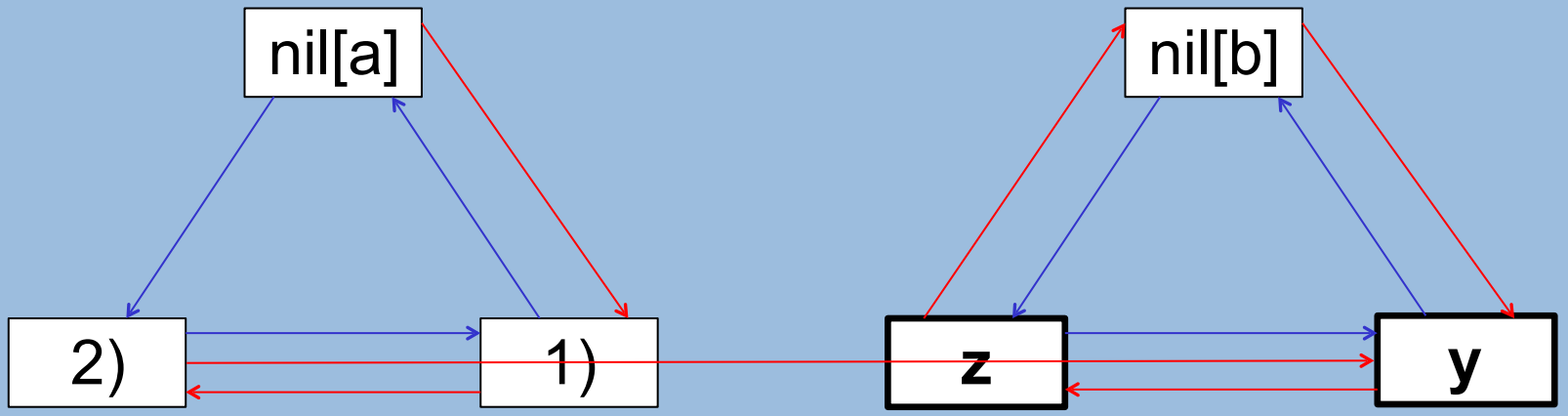





```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

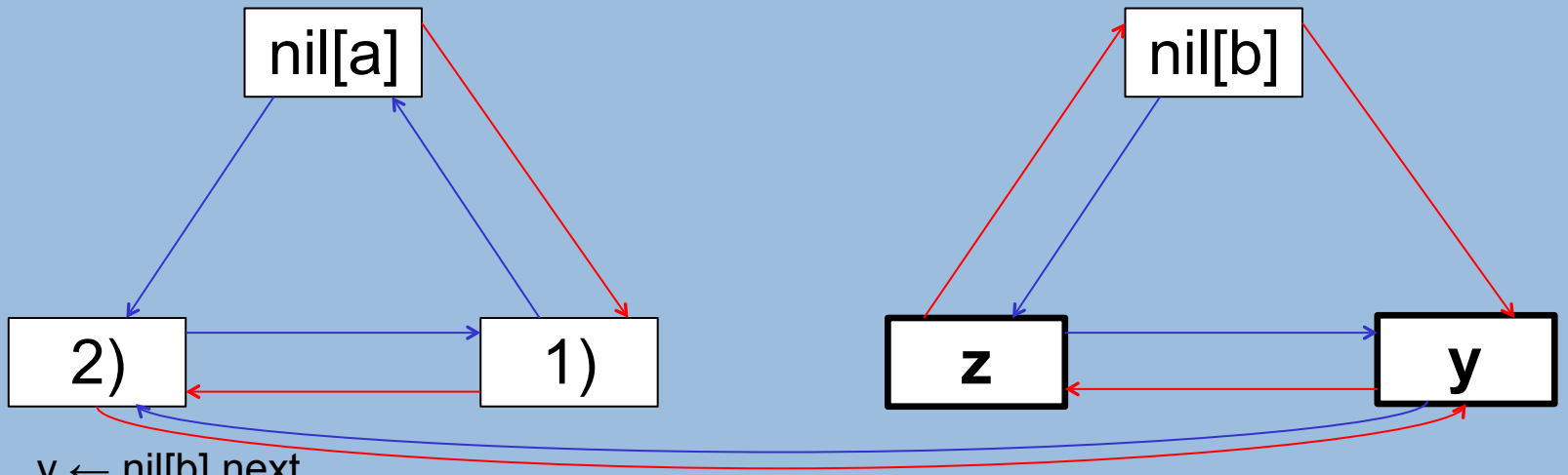
> Zwei zyklische, doppelt verkettete Listen a und b




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

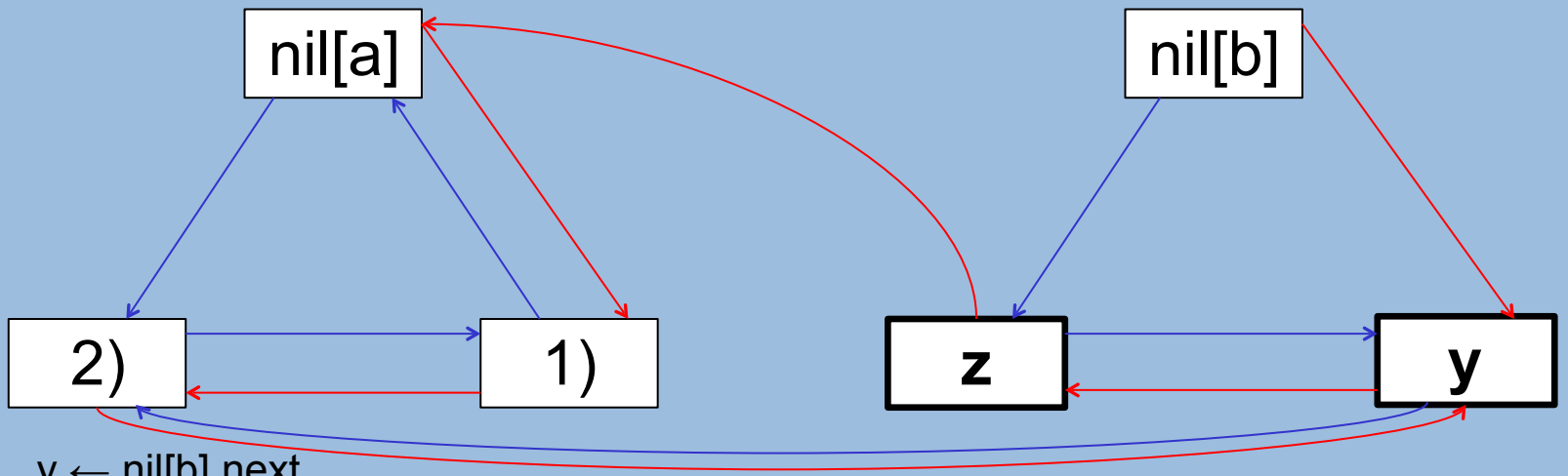
> Zwei zyklische, doppelt verkettete Listen a und b




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

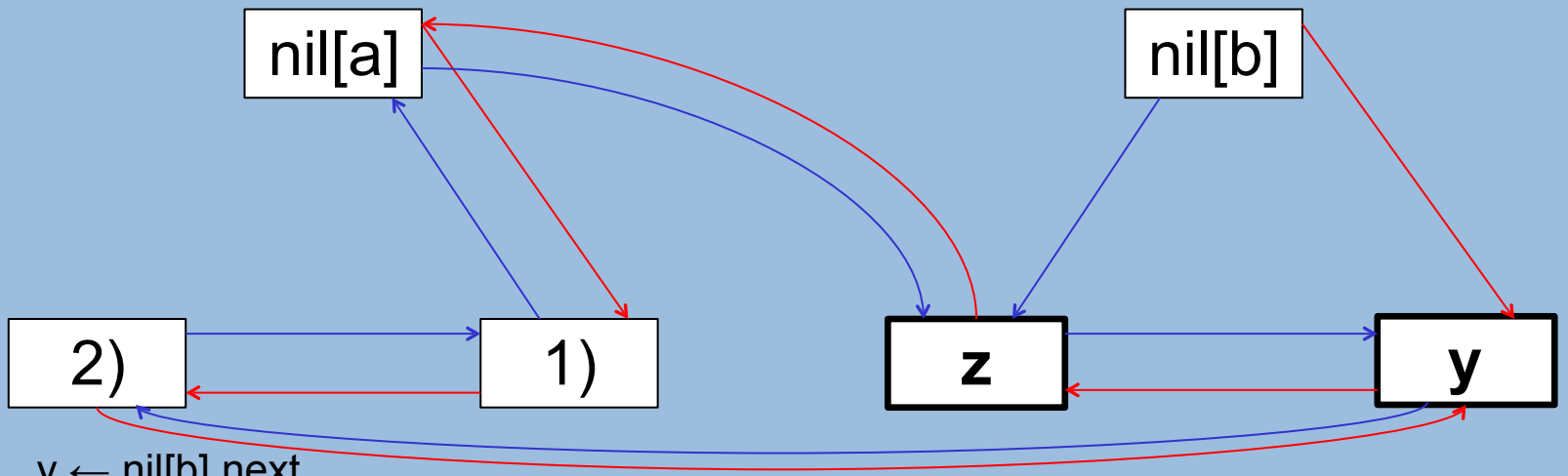
- Zwei zyklische, doppelt verkettete Listen a und b




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

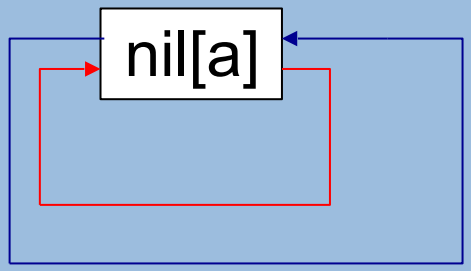
- Zwei zyklische, doppelt verkettete Listen a und b

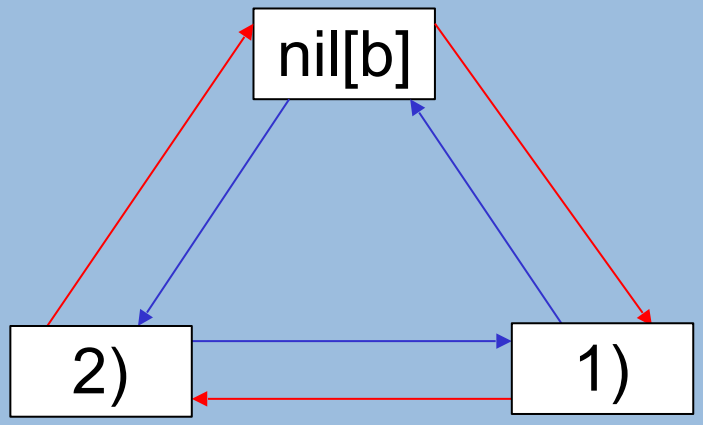



```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

> Was passiert wenn a leer ist?

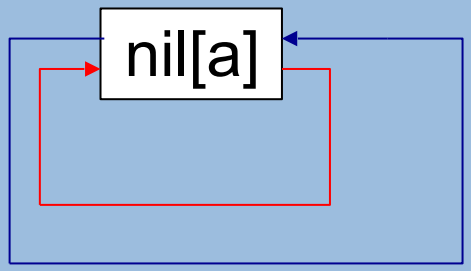


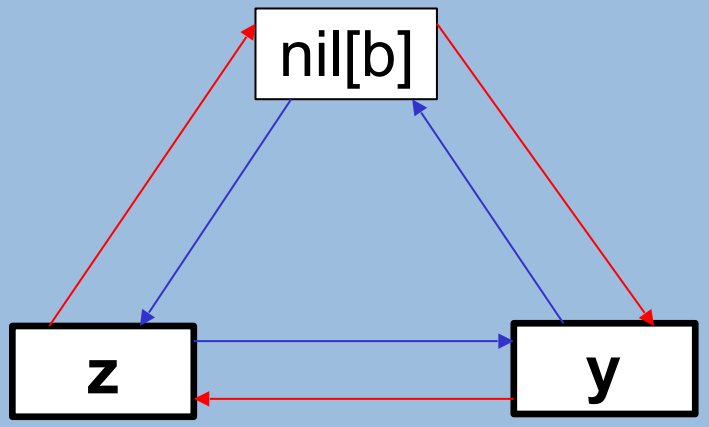



```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

> Was passiert wenn a leer ist?

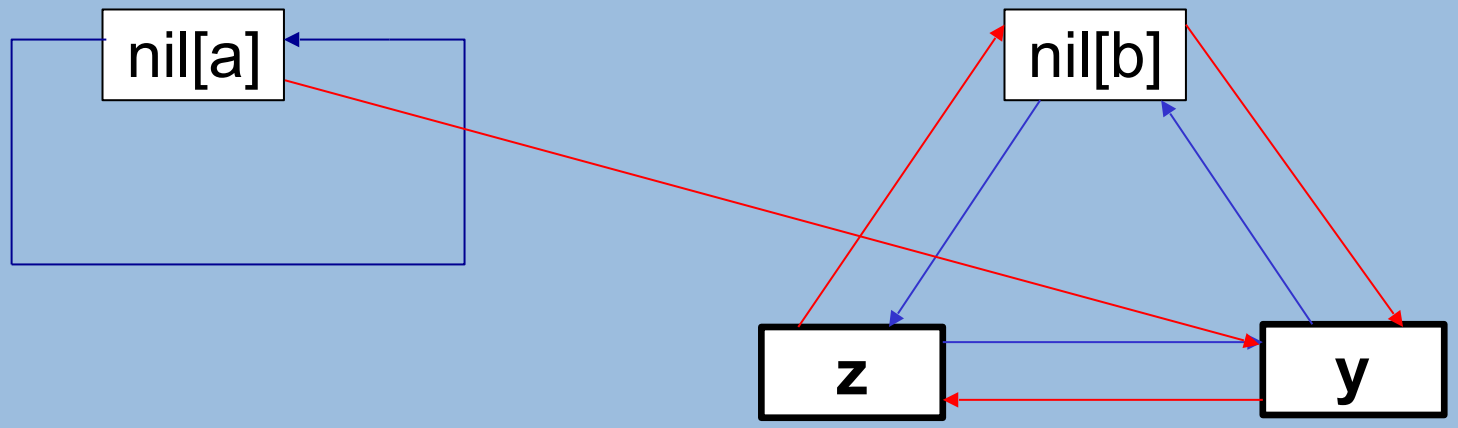





```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

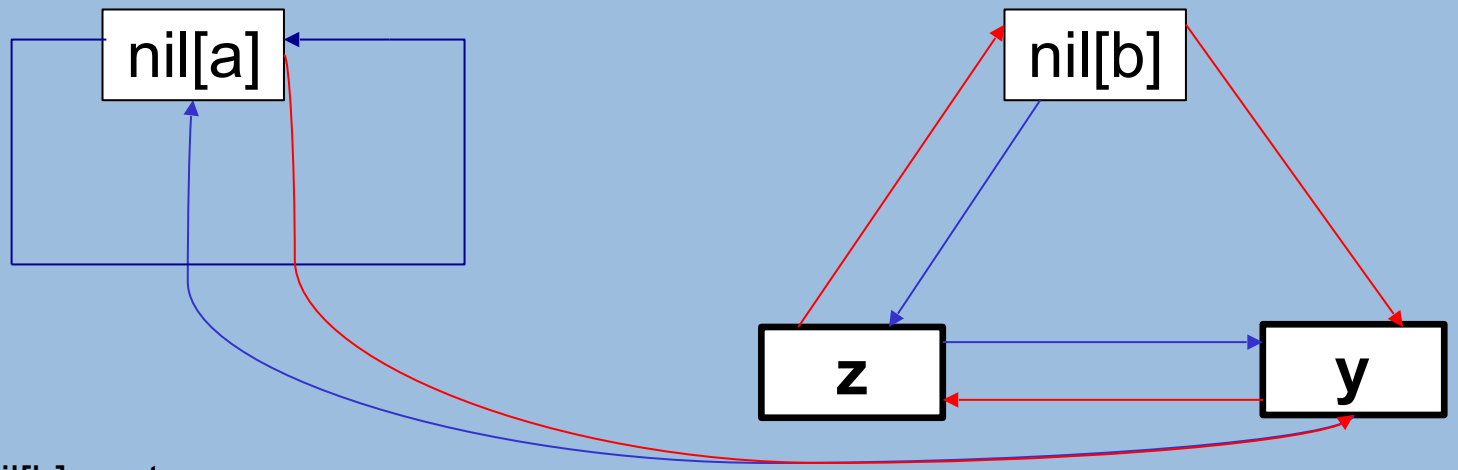
> Was passiert wenn a leer ist?




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

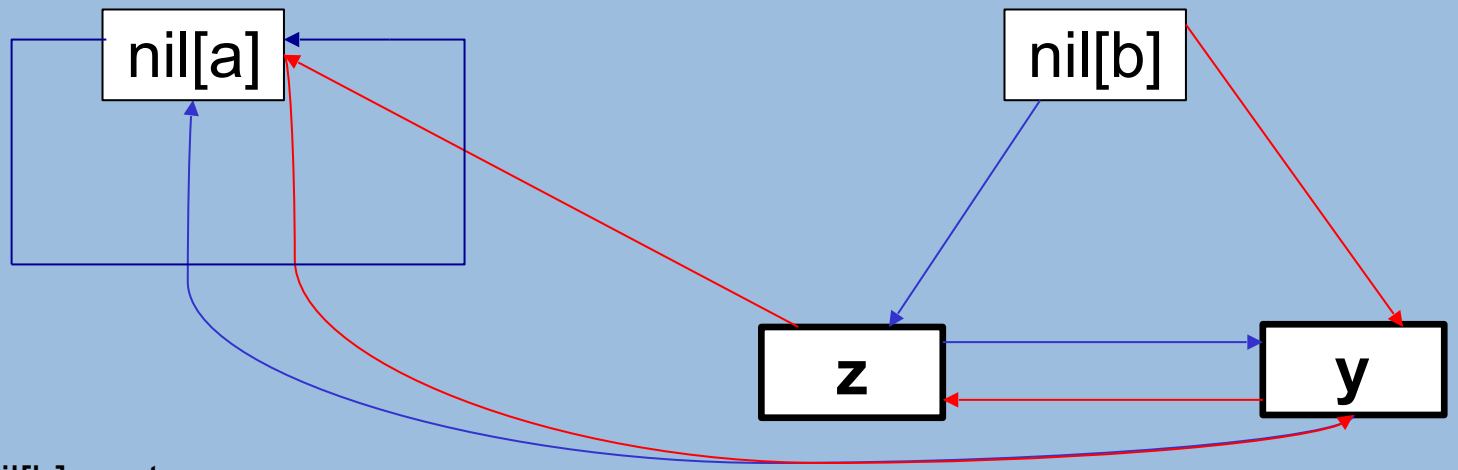
> Was passiert wenn a leer ist?




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

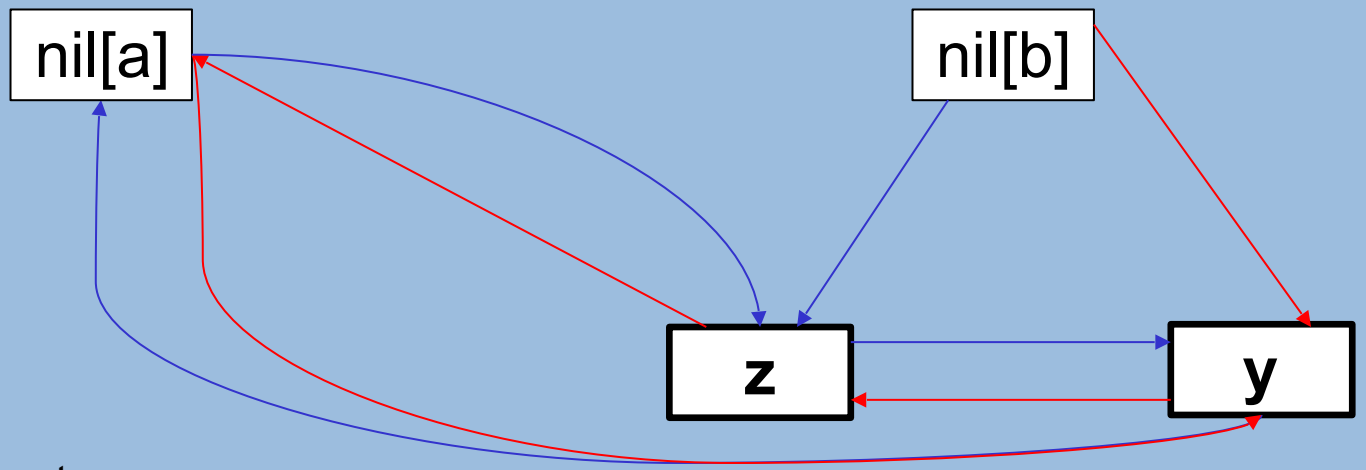
> Was passiert wenn a leer ist?




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 1

> Was passiert wenn a leer ist?




```
y ← nil[b].next
z ← nil[b].prev
      if (y ≠ nil[b])
            nil[a].prev.next ← y
            y.prev ← nil[a].prev
            z.next ← nil[a]
            nil[a].prev ← z
```

Repetition: Aufgabe 2

1. $T(\frac{n!}{n^n}) = \Theta(\frac{1}{n})$
2. $T(2^{128}) = \Theta(1)$
3. $T(\log(\log(4n))) = \Theta(\log(\log(n)))$
4. $T(5n-2) = \Theta(n) = T(2^{\log_8(n)})$
5. $T(3n^2 + 2n + 1) = \Theta(n^2)$
6. $T(n^3 \log(n)) = \Theta(n^3 \log(n))$
7. $T(\sqrt{n^7}) = \Theta(n^{3.5})$
8. $T(2^n) = \Theta(2^n)$

Repetition: Aufgabe 3

$O(n)$

3. Geben Sie die asymptotische Laufzeit in Abhängigkeit von n für folgenden Algorithmus in Theta-Notation an:

```
1  i ← 1
2  j ← 1
3  while i <= n
4        do
5           while j <= i + 5
6                 do
7                    j ← j + 1
8           i ← i + 1
```

- j keine Wertzuweisung mehr nach erster Iteration
- daher Laufzeit $O(n)$

Fragen?