Grundzüge der Informatik 1

Vorlesung 18 - flipped classroom

Datenstrukturen

Rot-Schwarz-Bäume

- Balancierter Suchbaum
- Nach Einfügen/Löschen wird die Struktur des Suchbaums so modifiziert, dass eine Höhe von O(log n) garantiert wird
- Rebalancierung nach Einfügen/Löschen wird in O(log n) Zeit möglich sein
- Damit sind Operationen Suchen, Einfügen und Löschen in O(log n) Zeit möglich

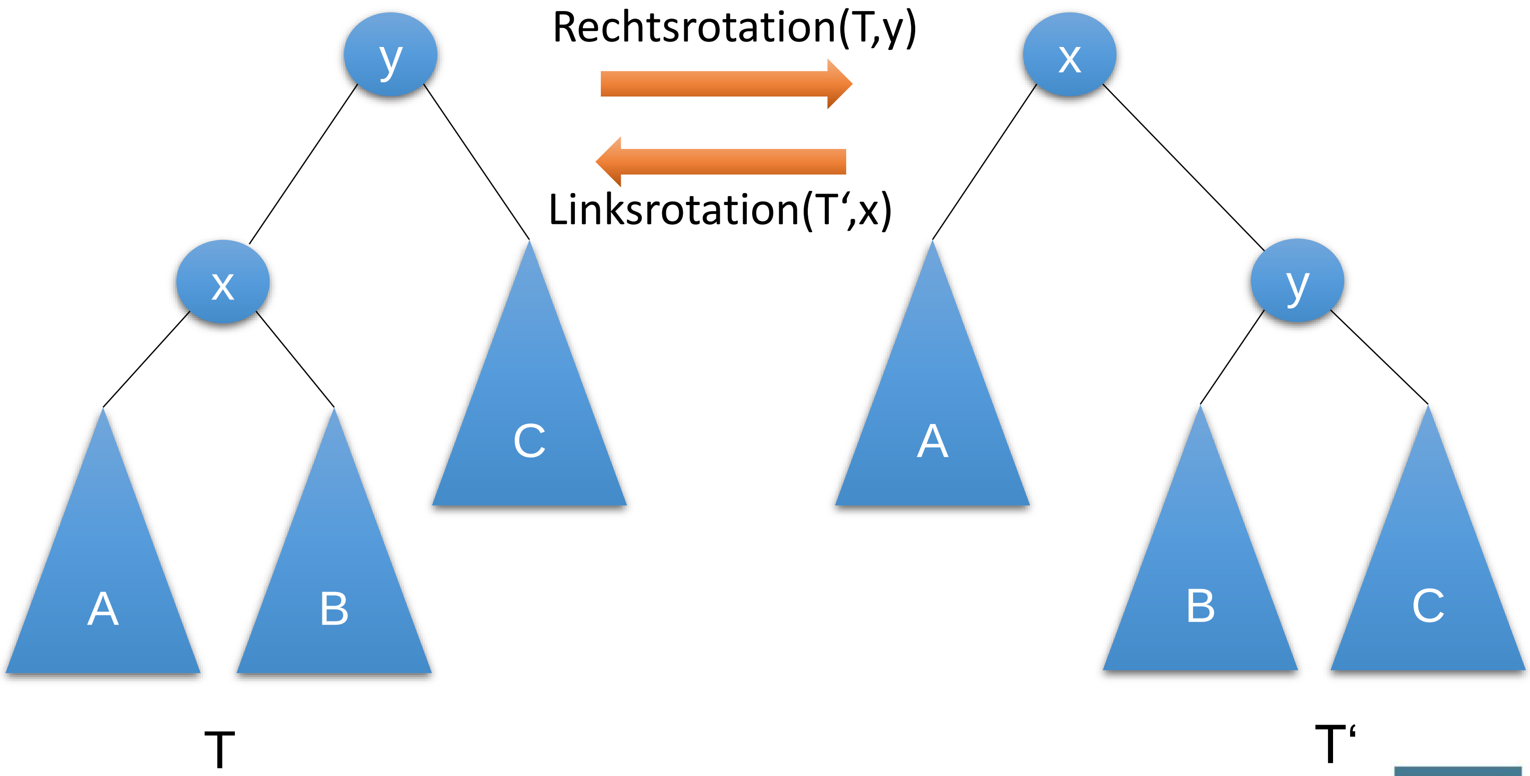
Datenstrukturen

Die Rot-Schwarz-Eigenschaften

- Jeder Knoten ist rot oder schwarz
- Die Wurzel ist schwarz
- Jedes Blatt ist schwarz
- Wenn ein Knoten rot ist, dann sind seine Kinder schwarz
- Für jeden Knoten v haben alle Pfade vom Knoten zu den Blättern im Unterbaum mit Wurzel v dieselbe Anzahl schwarzer Knoten

Datenstrukturen

Rotationen



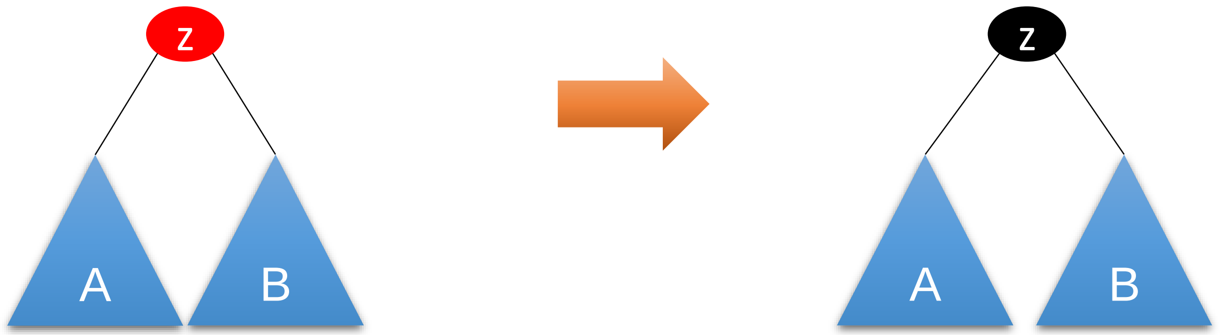
Datenstrukturen

Überblick: Wiederherstellen der Rot-Schwarz-Eigenschaften

- Starte mit eingefügtem Knoten z
- Stelle die Eigenschaft lokal wieder her, so dass sie nur von einem Knoten verletzt werden kann, der näher an der Wurzel ist
- Bei der Wurzel angekommen wird diese schwarz gefärbt

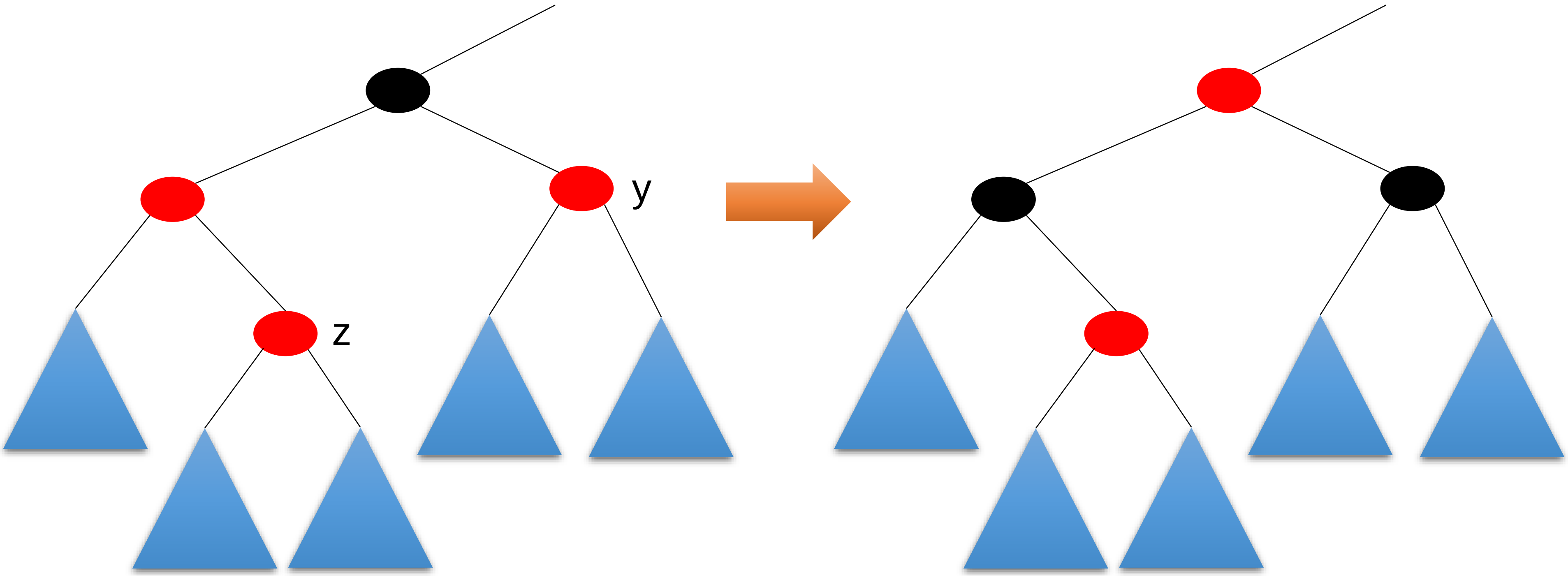
Datenstrukturen

Fall (1) (z ist die Wurzel)



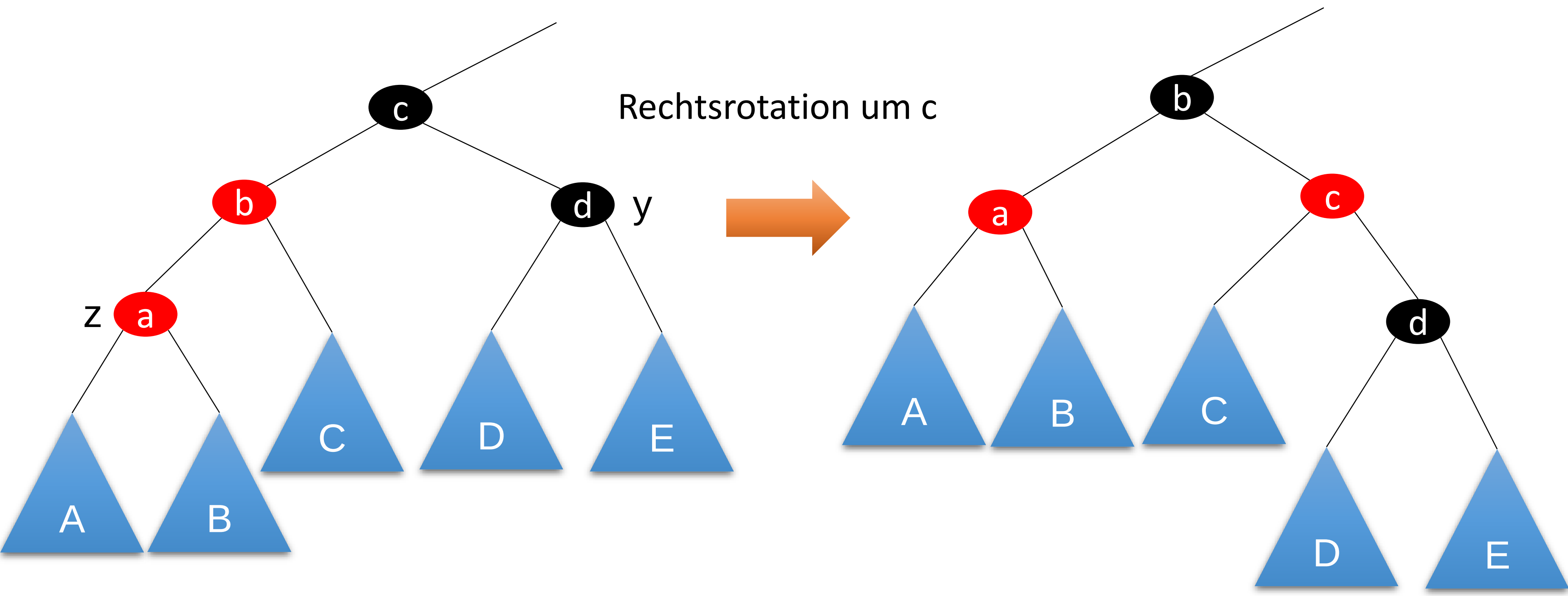
Datenstrukturen

Fall (2) (Onkel von z ist rot)



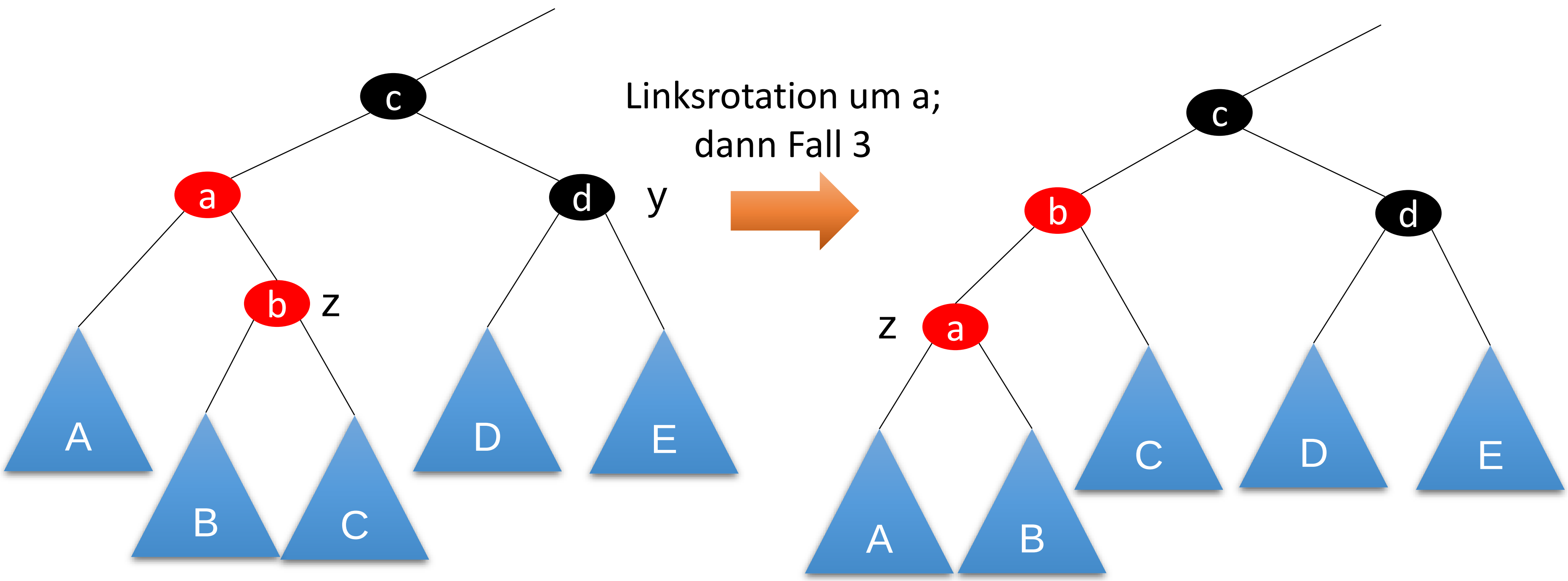
Datenstrukturen

Fall (3) (Onkel von z ist schwarz, z ist linkes Kind und parent(z) ist linkes Kind)



Datenstrukturen

Fall (4) (Onkel von z ist schwarz, z ist rechtes Kind und parent(z) ist linkes Kind)



Universität zu Köln

Datenstrukturen

Löschen in Rot-Schwarz-Bäumen

- Zunächst Löschen wie in binären Suchbäumen
- Dann Wiederherstellen der Rot-Schwarz-Eigenschaften

Datenstrukturen

RS-Löschen(T,z) * Zu löschender Knoten wird übergeben

1. **if** left[z]=NIL[T] or right[z]=NIL[T] **then** y=z
2. **else** y=NachfolgerSuche(z)
3. **if** left[y]≠NIL[T] **then** x=left[y] **else** x=right[y]
4. parent[x]=parent[y]
5. **if** parent[y]=NIL[T] **then** root[T]=x
6. **else if** y=left[parent[y]] **then** left[parent[y]]=x **else** right[parent[y]]=x
7. key[z]=key[y]
8. **if** color[y]=schwarz **then** RS-Löschen-Fix(T,x)
9. parent[NIL[T]]=NIL
10. **delete** y

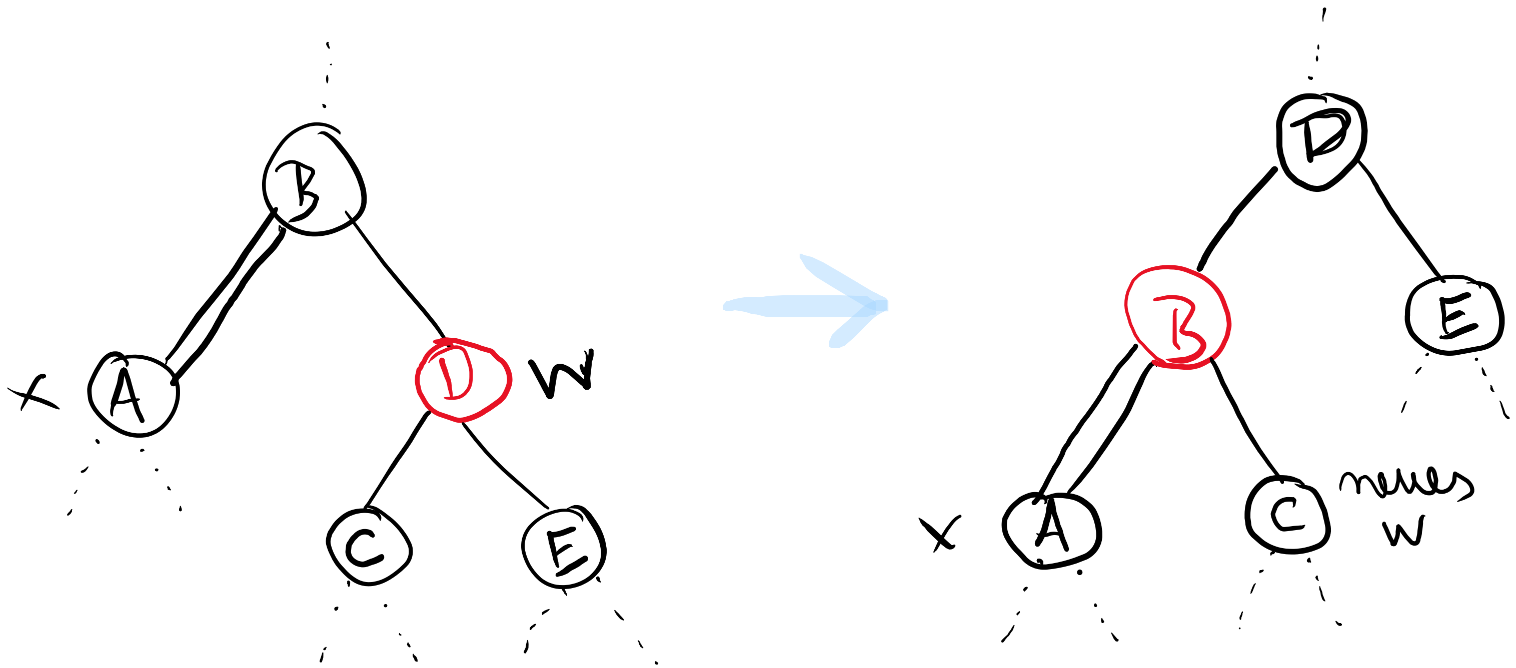
Datenstrukturen

Einfacher Fall

- Wenn der Knoten x rot ist, können wir diesen schwarz färben und der resultierende Baum ist ein Rot-Schwarz-Baum
- Im Folgenden gehen wir davon aus, dass x schwarz ist

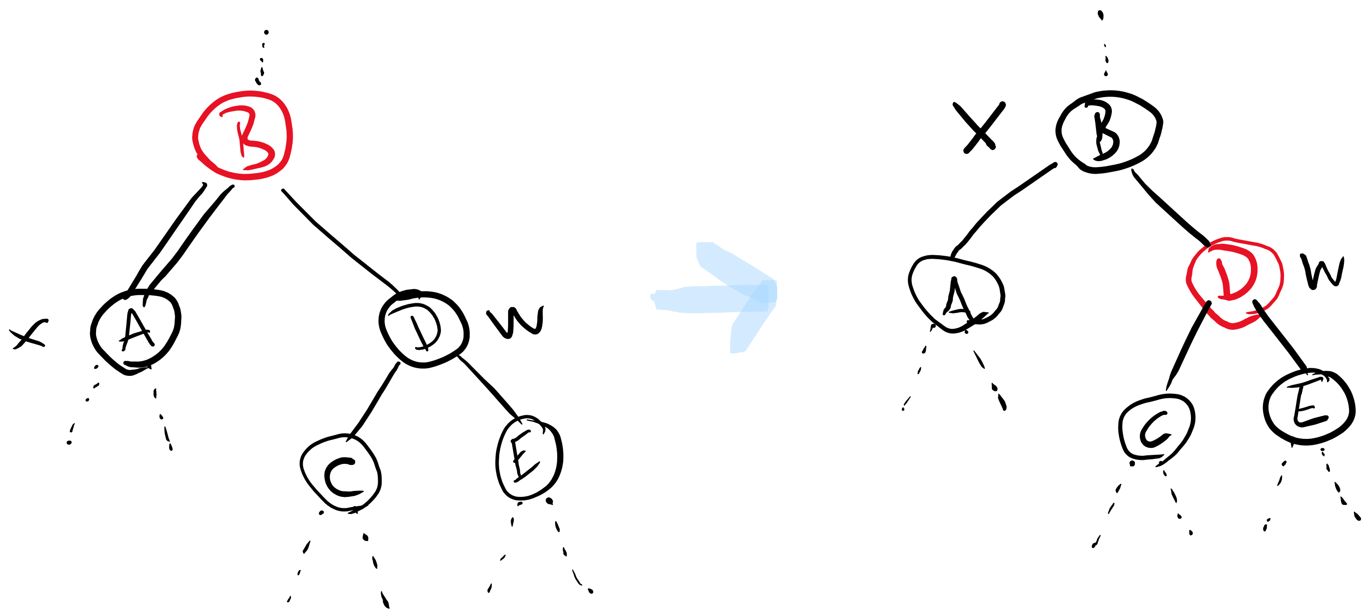
Datenstrukturen

Fall 1: Geschwisterknoten von x ist rot



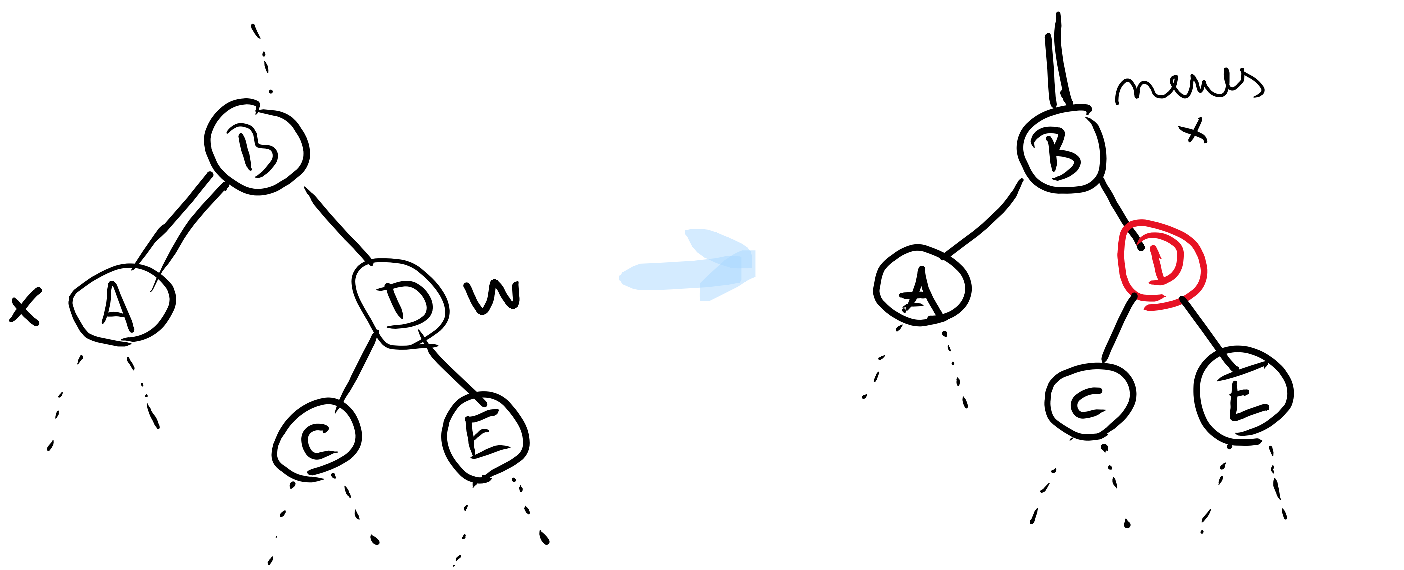
Datenstrukturen

Fall 2a: Geschwisterknoten w von x ist schwarz; parent[x] ist rot;
Kinder von w sind schwarz



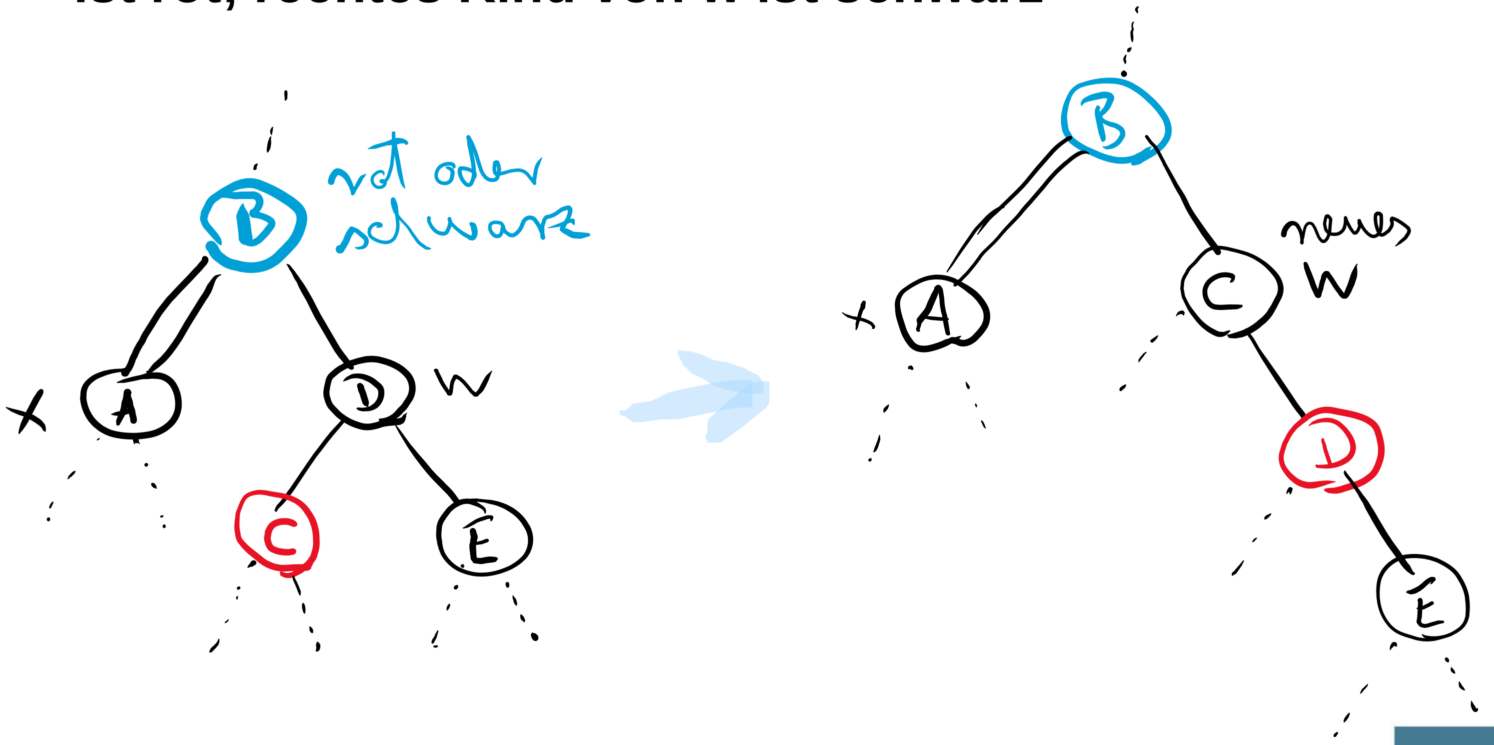
Datenstrukturen

Fall 2b: Geschwisterknoten w von x ist schwarz; parent[x] ist schwarz; Kinder von w sind schwarz



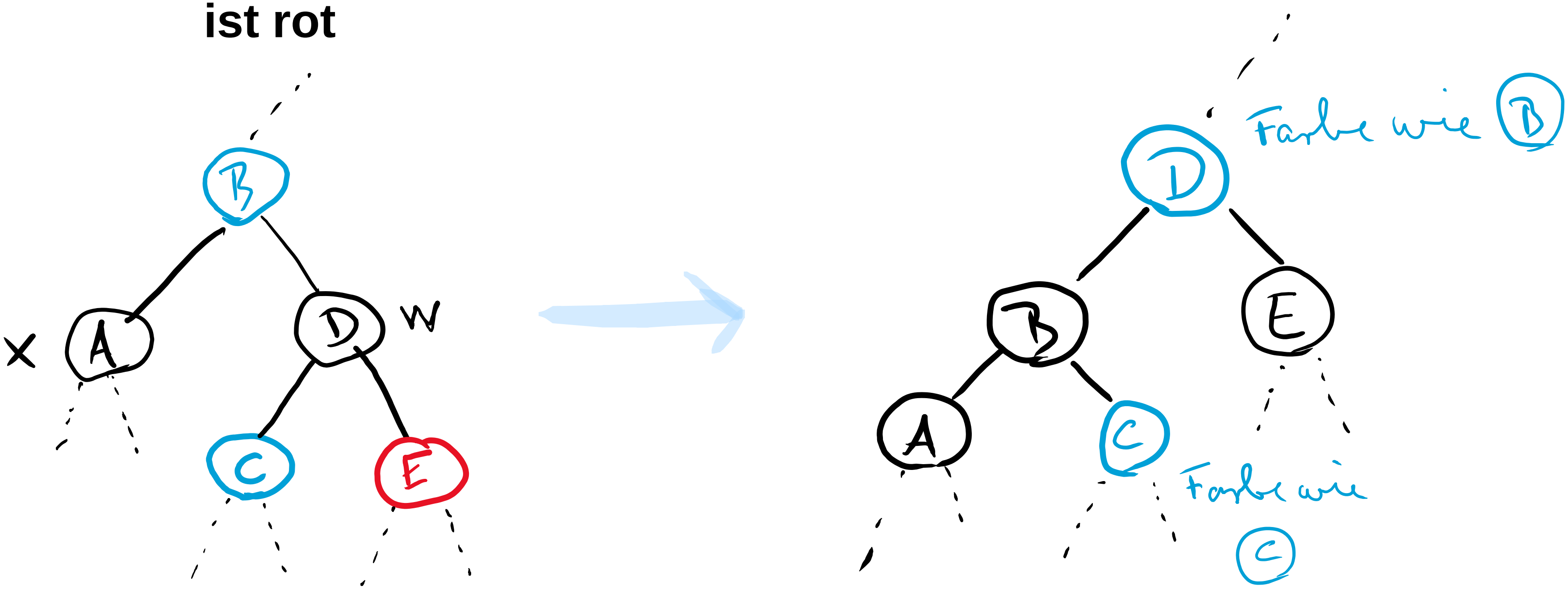
Datenstrukturen

Fall 3: Geschwisterknoten w von x ist schwarz; linkes Kind von w ist rot; rechtes Kind von w ist schwarz



Datenstrukturen

Fall 4: Geschwisterknoten w von x ist schwarz; rechtes Kind von w ist rot



Datenstrukturen

Aufgabe 1

- Modifizieren Sie den Rot-Schwarz-Baum so, dass an den Knoten auch das Attribut Größe (Anzahl der Knoten im Unterbaum) aus der letzten Übung aufrecht erhalten werden kann
- Identifizieren Sie dazu zunächst die Schritte, in denen der Pseudocode geändert werden muss

Datenstrukturen

Aufgabe 1

- Modifizieren Sie den Rot-Schwarz-Baum so, dass an den Knoten auch das Attribut Größe (Anzahl der Knoten im Unterbaum) aus der letzten Übung aufrecht erhalten werden kann
- Identifizieren Sie dazu zunächst die Schritte, in denen der Pseudocode geändert werden muss

(a) Rotationen

(b) Einfügen

(c) Löschen

Datenstrukturen

Aufgabe 1

- Modifizieren Sie den Rot-Schwarz-Baum so, dass an den Knoten auch das Attribut Größe (Anzahl der Knoten im Unterbaum) aus der letzten Übung aufrecht erhalten werden kann
- Identifizieren Sie dazu zunächst die Schritte, in denen der Pseudocode geändert werden muss

(a) Rotationen

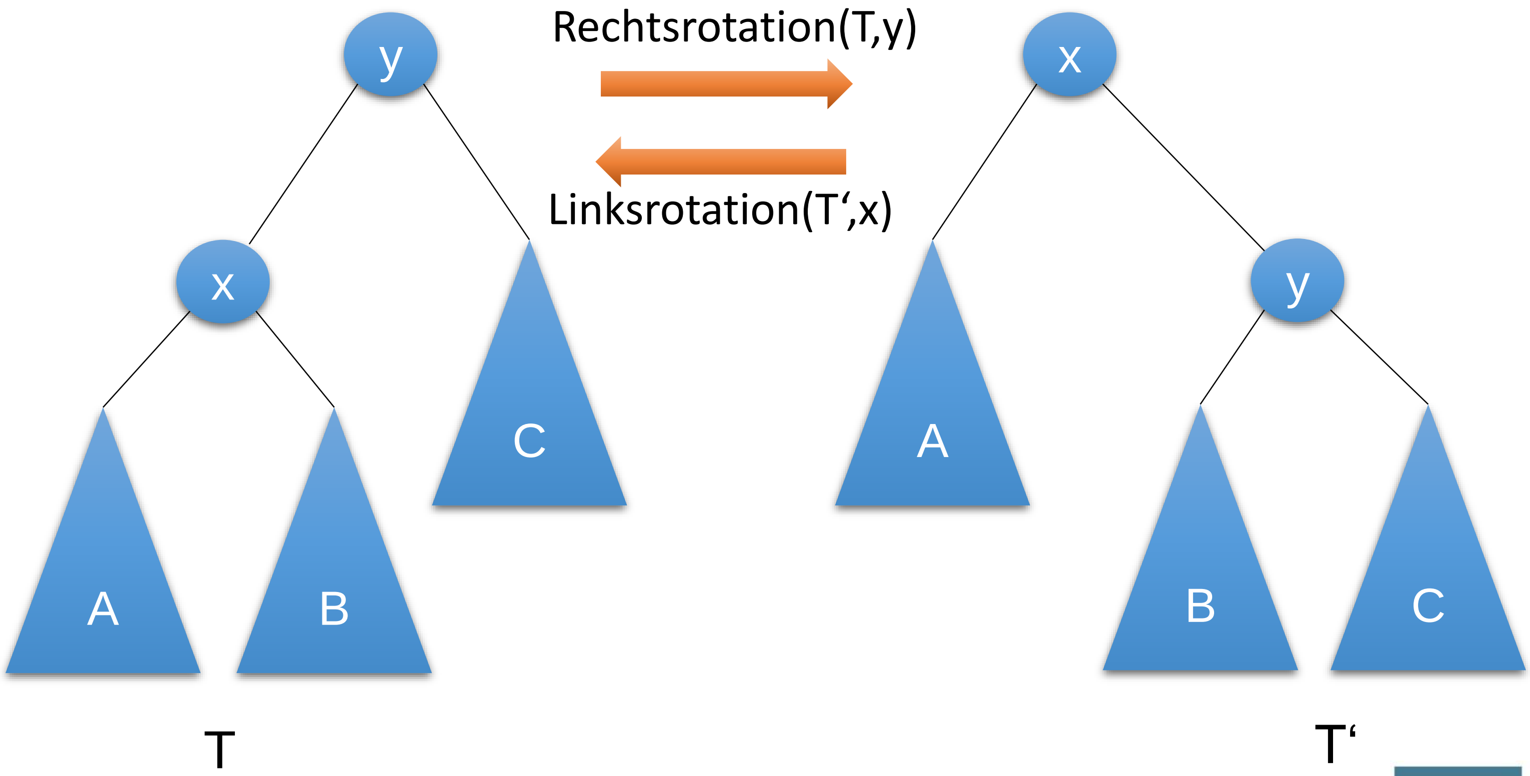
(b) Einfügen

(c) Löschen

- Geben Sie die Modifikationen der Prozeduren für Rotationen und Einfügen im Pseudocode an

Datenstrukturen

Rotationen



Datenstrukturen

Linksrotation(T,x)

1. y = right[x]
2. right[x] = left[y]
3. **if** left[y]$\neq$NIL **then** parent[left[y]] = x
4. parent[y] $\leftarrow$ parent[x]
5. **if** parent[x]=NIL **then** root[T] = y
6. **else if** x=left[parent[x]] **then** left[parent[x]] = y
7. **else** right[parent[x]] $\leftarrow$ y
8. left[y] $\leftarrow$ x
9. parent[x] $\leftarrow$ y
10. **Größe[x] = Größe[left[x]]+Größe[right[x]]+1**
11. **Größe[y] = Größe[x]+Größe[right[y]]+1**

Datenstrukturen

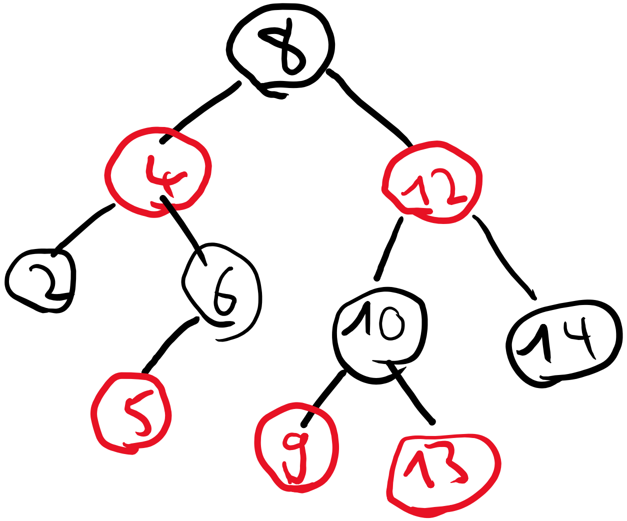
RS-Einfügen(T,k)

z = **new** Knoten
y=NIL[T]; x = Root[T]
while x$\neq$NIL[T] **do**
    y=x; **Größe[x] = Größe[x] +1**
    if k<key[x] **then** x=left[x] **else** x=right[x]
parent[z] = y
if y=NIL[T] **then** root[T] = z
else if k<key[y] **then** left[y] = z **else** right[y]=z
key[z] = k; left[z] = NIL[T]; right[z] = NIL[T]; color[z] = rot; **Größe[z] = 1**
RS-Einfügen-Fix(T,z)

Datenstrukturen

Aufgabe 2

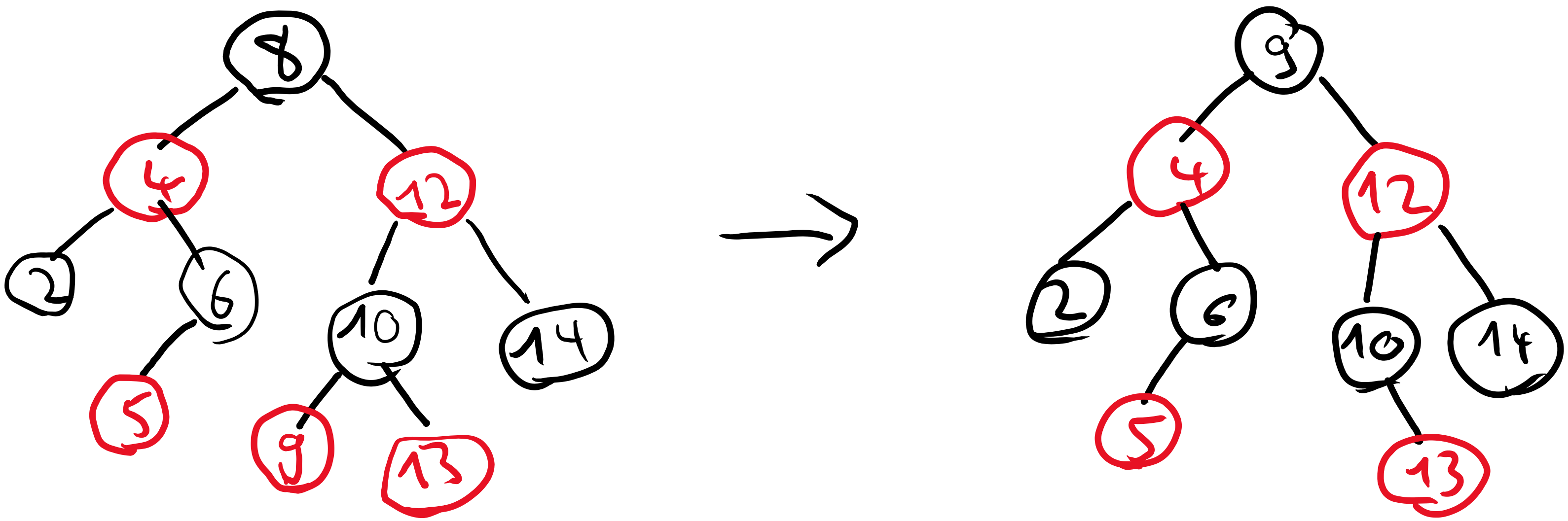
- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

Aufgabe 2

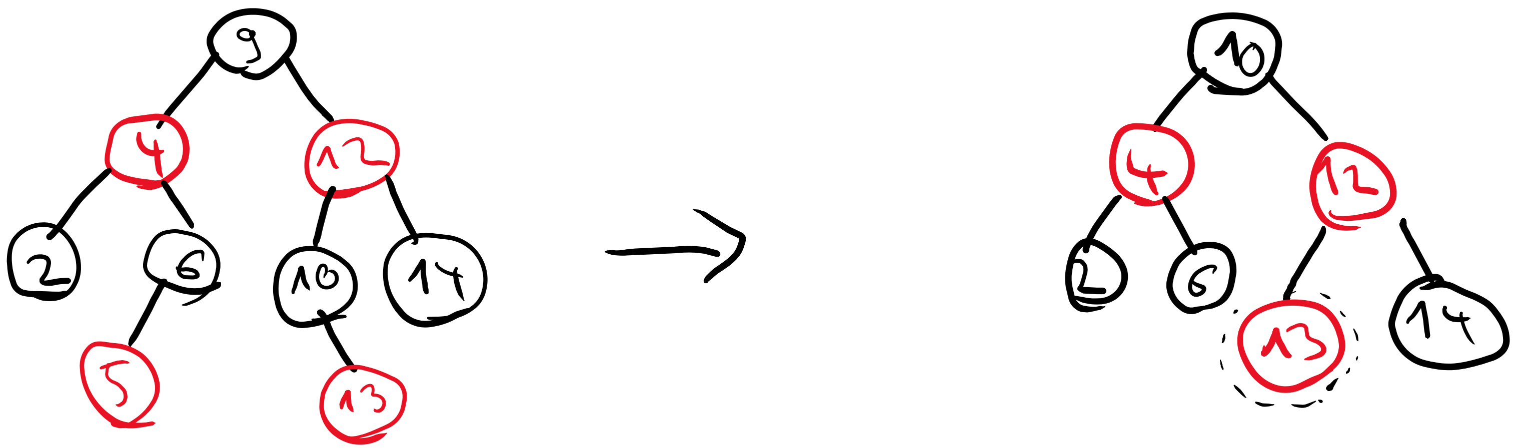
- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

Aufgabe 2

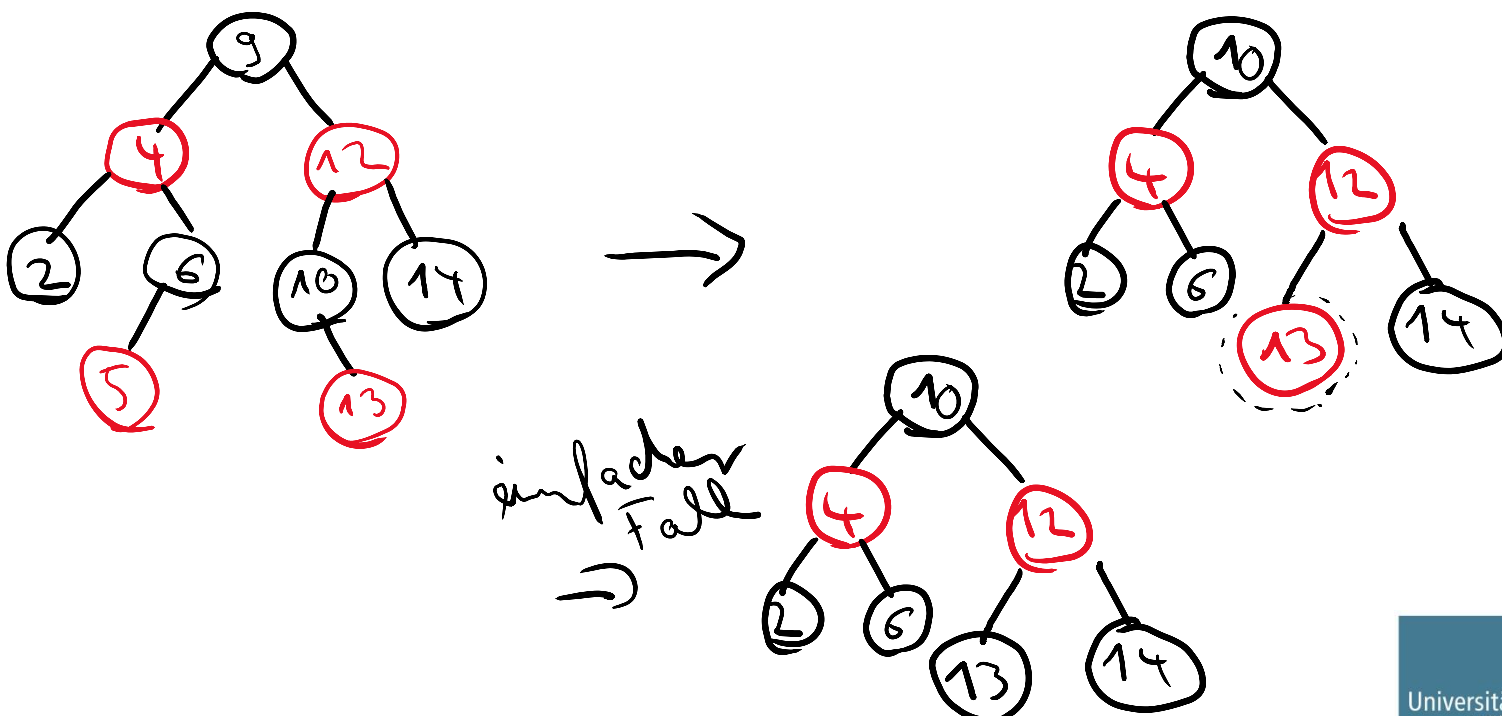
- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

Aufgabe 2

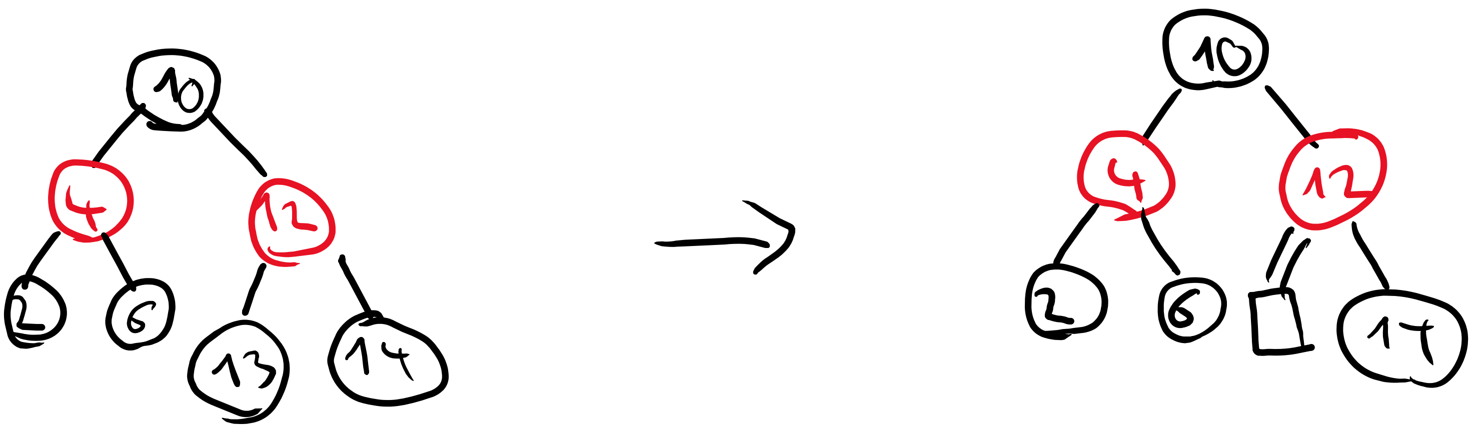
- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

Aufgabe 2

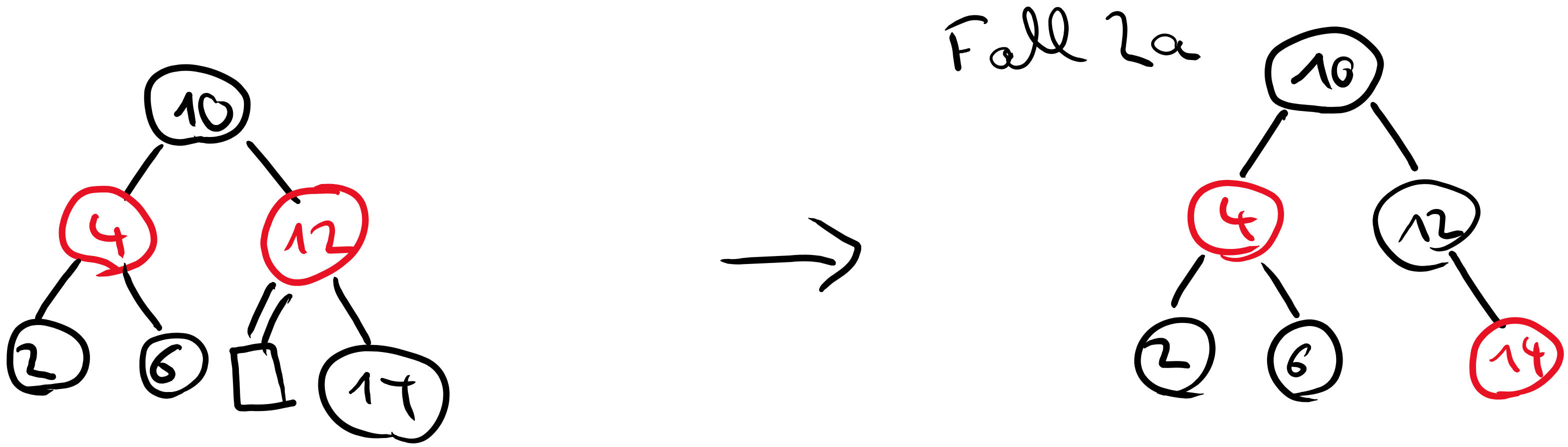
- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

Aufgabe 2

- Löschen Sie die Schlüssel 8, 9 und 13 aus dem unten stehenden Rot-Schwarz-Baum



Datenstrukturen

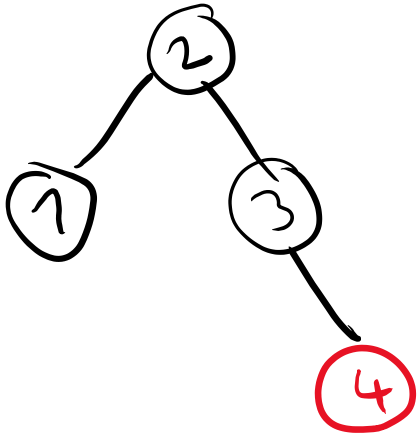
Aufgabe 3

- Wenn man eine Zahl x aus einem Rot-Schwarz-Baum löscht und danach wieder einfügt, ist die Struktur des Baums identisch zur Struktur vor dem Einfügen? Begründen Sie Ihre Antwort!

Datenstrukturen

Aufgabe 3

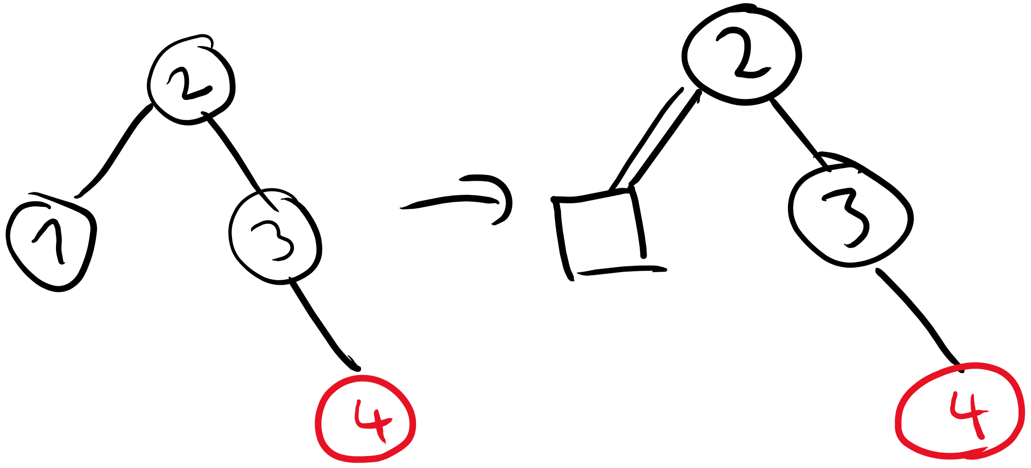
- Wenn man eine Zahl x aus einem Rot-Schwarz-Baum löscht und danach wieder einfügt, ist die Struktur des Baums identisch zur Struktur vor dem Einfügen? Begründen Sie Ihre Antwort!



Datenstrukturen

Aufgabe 3

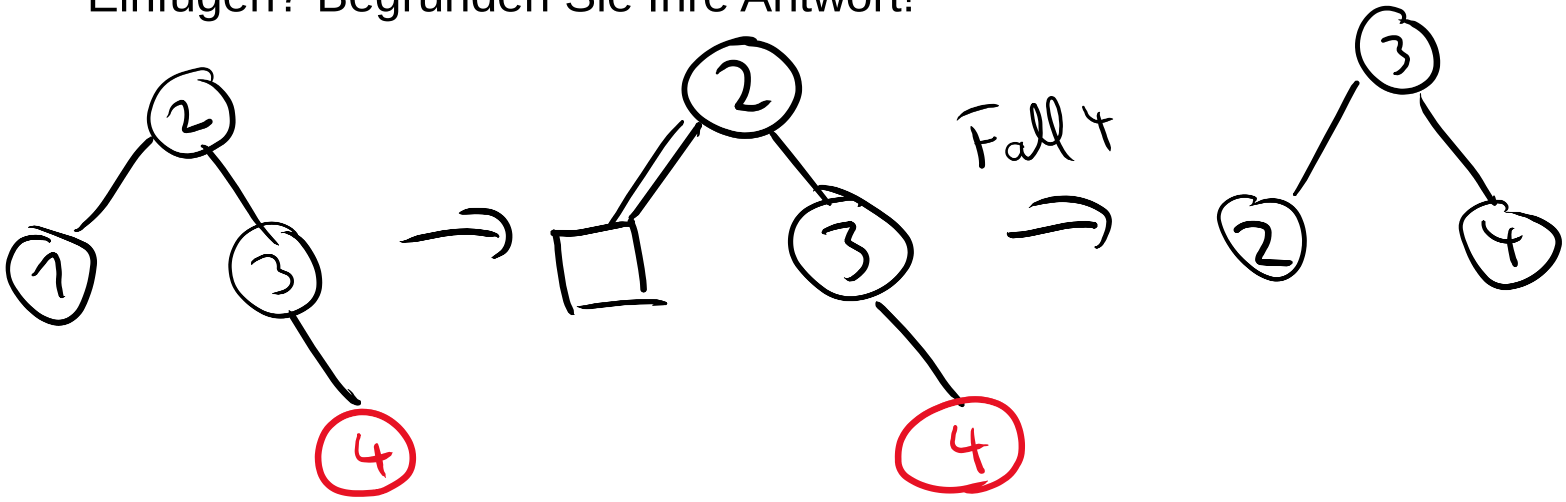
- Wenn man eine Zahl x aus einem Rot-Schwarz-Baum löscht und danach wieder einfügt, ist die Struktur des Baums identisch zur Struktur vor dem Einfügen? Begründen Sie Ihre Antwort!



Datenstrukturen

Aufgabe 3

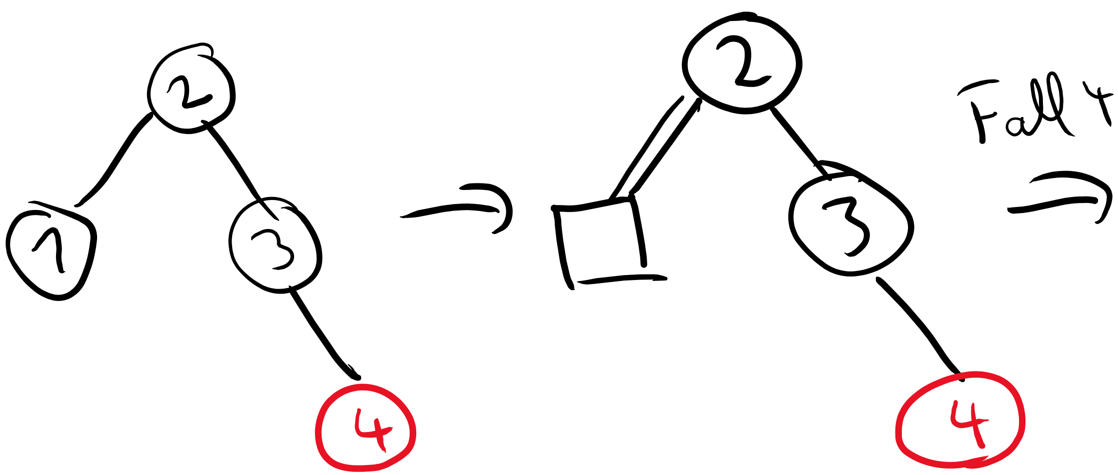
- Wenn man eine Zahl x aus einem Rot-Schwarz-Baum löscht und danach wieder einfügt, ist die Struktur des Baums identisch zur Struktur vor dem Einfügen? Begründen Sie Ihre Antwort!

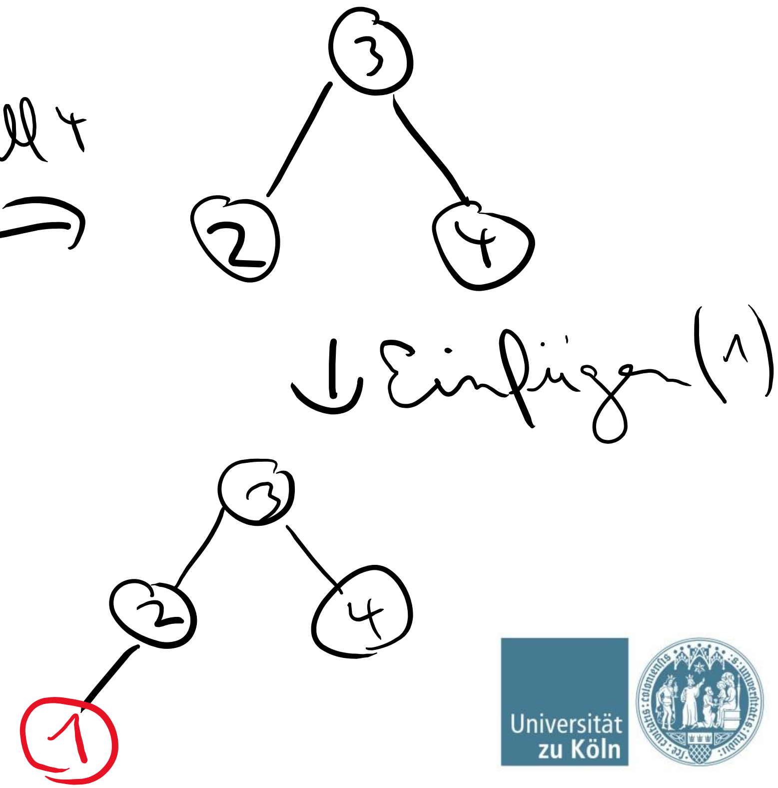


Datenstrukturen

Aufgabe 3

- Wenn man eine Zahl x aus einem Rot-Schwarz-Baum löscht und danach wieder einfügt, ist die Struktur des Baums identisch zur Struktur vor dem Einfügen? Begründen Sie Ihre Antwort!





Datenstrukturen

Aufgabe 4

- Seien T und T‘ binäre Suchbäume für eine Menge von n Zahlen S. Zeigen Sie, dass man T mit Hilfe von O(n) Rotationen in T‘ überführen kann.

Datenstrukturen

Aufgabe 4

- Seien T und T‘ binäre Suchbäume für eine Menge von n Zahlen S. Zeigen Sie, dass man T mit Hilfe von O(n) Rotationen in T‘ überführen kann.

Idee:

- Wir argumentieren zunächst, wie man einen beliebigen Suchbaum in einen Suchbaum umformen kann, der nur rechte Kinder besitzt (also eine Listenstruktur hat)

Datenstrukturen

Aufgabe 4

- Seien T und T‘ binäre Suchbäume für eine Menge von n Zahlen S. Zeigen Sie, dass man T mit Hilfe von O(n) Rotationen in T‘ überführen kann.

Beweis:

- Wir zeigen, dass man einen beliebigen Suchbaum T in einen Suchbaum umformen kann, der nur rechte Kinder besitzt (also eine Listenstruktur hat)
- Damit folgt das Ergebnis, weil man sowohl T als in solche Listenstruktur umwandeln kann und es zu jeder Rotation eine inverse Rotation gibt
- Für einen Suchbaum T sei L die Länge des rechten Asts
(also die Anzahl Knoten, die man besucht, wenn man so lange wie möglich immer nach rechts geht)

Datenstrukturen

Aufgabe 4

- Seien T und T‘ binäre Suchbäume für eine Menge von n Zahlen S. Zeigen Sie, dass man T mit Hilfe von O(n) Rotationen in T‘ überführen kann.

Beweis:

- Wir zeigen: So lange L<n gilt, gibt es eine Rotation, mit deren Hilfe man einen Suchbaum mit einem rechten Ast der Länge L+1 erhalten kann
- Da L<n ist, gibt es einen Knoten, der ein linkes Kind eines Knotens v im rechten Ast ist
- Wir machen eine Rechtsrotation um v und erhalten einen Baum, dessen rechter Ast eine Länge von L+1 hat
- Induktives Anwenden dieses Arguments liefert unsere Aussage