



Technische
Universität
Braunschweig



Algorithmen und Datenstrukturen – Übung #3

Dynamische Datenstrukturen

Arne Schmidt
12.12.2019

Heute

- Verkettete Listen

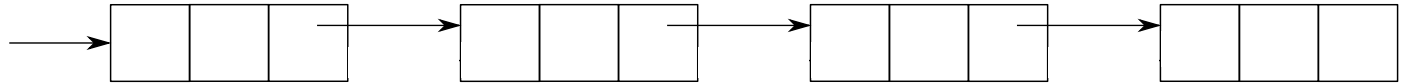
- Binäre Suchbäume
 - Operationen
 - Traversierung

- Verschmelzen von Suchbäumen

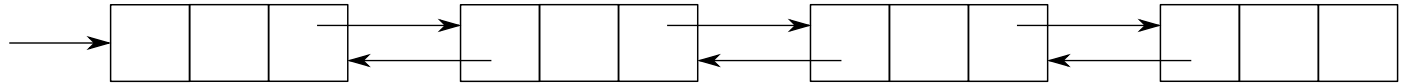
(Zyklisch) Verkettete Listen

Listen

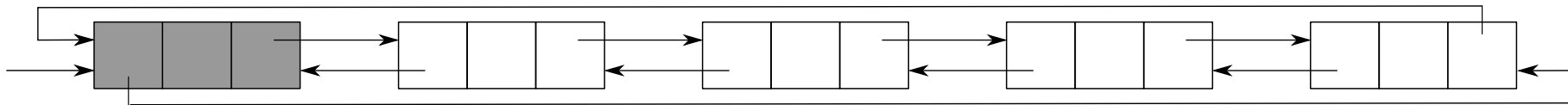
Einfach verkettet:



Doppelt verkettet:



Zyklisch doppelt verkettet (mit Wächter):



Laufzeiten in Listen

Operation	Einfach	Doppelt	Zyklisch
Suchen	$O(n)$	$O(n)$	$O(n)$
Einfügen	$O(1)$	$O(1)$	$O(1)$
Löschen	$O(n)$	$O(1)$	$O(1)$
Merge*	$O(1)**$	$O(1)$	$O(1)$






*: Verschmelze zwei Listen der Größe n und m .

** : Sofern Adresse des letzten Elements bekannt. Andernfalls $O(\max(n, m))$.


Laufzeit Hierholzer-Algorithmus


Eingabe: Zusammenhängender Graph G mit höchstens 2 ungeraden Knoten


Ausgabe: Ein Eulerweg, bzw. eine Eulertour in G


- 1:  starte in einem Knoten v
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
- 2:  verwende Algorithmus 2.7, um einen Weg W von v aus zu bestimmen
- 3: **while** es existieren unbenutzte Kanten **do**
- 4:  wähle einen Knoten w aus W mit positivem Grad im Restgraphen
- 5:  verwende Algorithmus 2.7, um einen Weg W' von w aus zu bestimmen
- 6:  verschmelze W und W'

Algorithmus 2.8: Hierholzers Algorithmus zum Finden eines Eulerweges oder einer Eulertour

 $O(n + m)$

 $O(m)?$

 $O(n + m)?$

 $O(|W| + |W'|)?$

Laufzeit Hierholzer-Algorithmus

Eingabe: Graph G

Ausgabe: Ein Weg in G

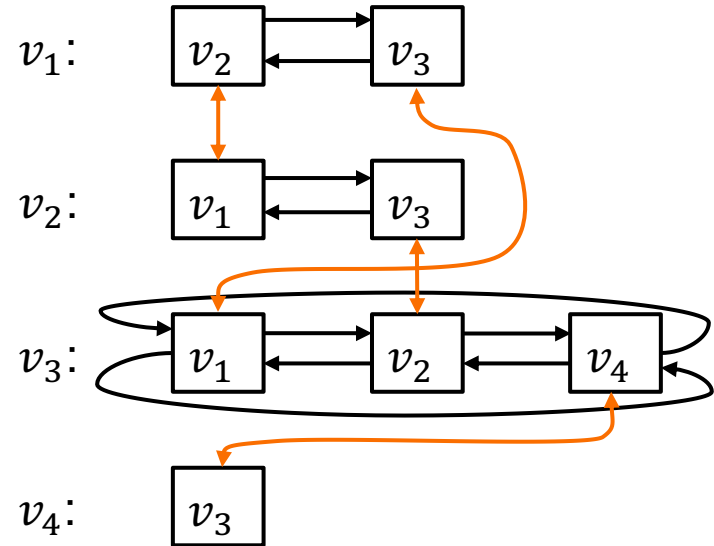
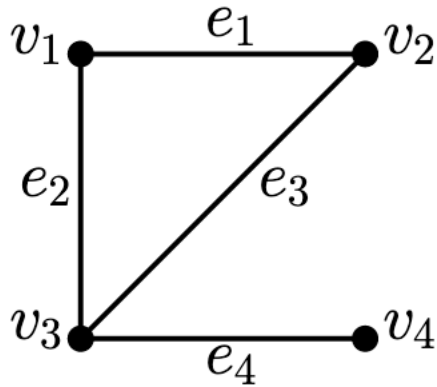
- 1: starte in einem Knoten v_0
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
- 2: $i \leftarrow 0$
- 3: **while** es gibt eine zu v_i inzidente, unbenutzte Kante **do**
- 4: wähle eine zu v_i inzidente, unbenutzte Kante $\{v_i, v_j\}$
- 5: laufe zum Nachbarknoten v_j
- 6: lösche $\{v_i, v_j\}$ aus der Menge der unbenutzten Kanten
- 7: $v_{i+1} \leftarrow v_j$
- 8: $i \leftarrow i + 1$

Algorithmus 2.7: Algorithmus zum Finden eines Weges in einem Graphen

Können wir einen Weg W in $O(|W|)$ Zeit bestimmen? Ideen?

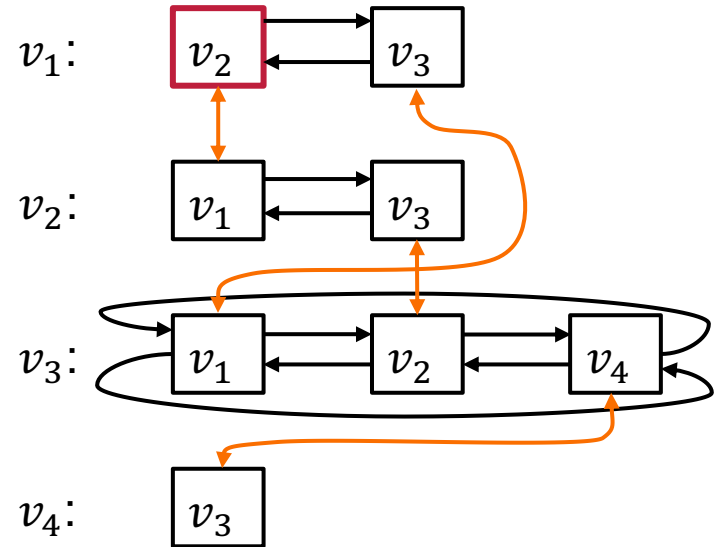
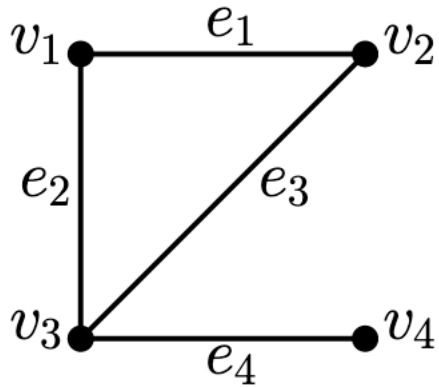
Laufzeit Hierholzer-Algorithmus

Benutze Adjazenzliste und Doppelt-Verkettete Listen.
Verwende zusätzliche **Pointer** für gleiche Kante.



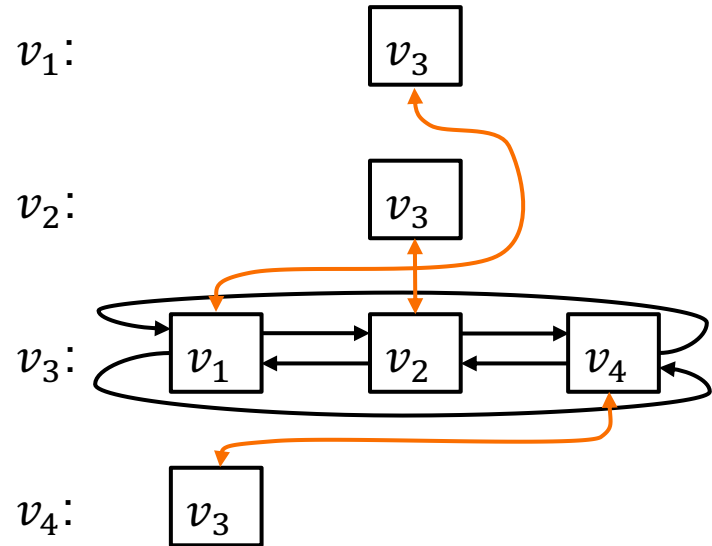
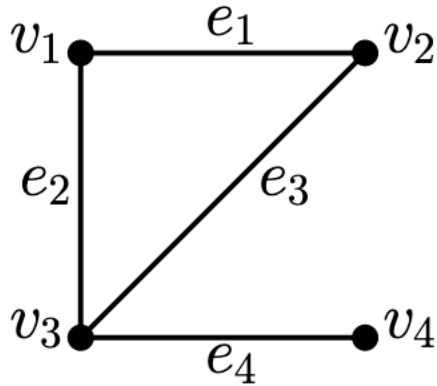
Laufzeit Hierholzer-Algorithmus

Starte bei v_1 und gehe zum nächsten Knoten.
Lösche die Kante aus der Adjazenzliste.
Alle Operationen kosten $O(1)$ Zeit.



Laufzeit Hierholzer-Algorithmus






Starte bei v_1 und gehe zum nächsten Knoten.
Lösche die Kante aus der Adjazenzliste.
Alle Operationen kosten $O(1)$ Zeit.




Laufzeit Hierholzer-Algorithmus

Eingabe: Zusammenhängender Graph G mit höchstens 2 ungeraden Knoten


Ausgabe: Ein Eulerweg, bzw. eine Eulertour in G

- 1:  starte in einem Knoten v
(wenn einer mit ungeradem Grad existiert, dort, sonst beliebig)
- 2:  verwende Algorithmus 2.7, um einen Weg W von v aus zu bestimmen
- 3: **while** es existieren unbenutzte Kanten **do**
- 4:  wähle einen Knoten w aus W mit positivem Grad im Restgraphen
- 5:  verwende Algorithmus 2.7, um einen Weg W' von w aus zu bestimmen
- 6:  verschmelze W und W'

Algorithmus 2.8: Hierholzers Algorithmus zum Finden eines Eulerweges oder einer Eulertour

 $O(n + m)$

 $O(|W|)$

 $O(n + m)?$

 $O(|W| + |W'|)?$

Laufzeit Hierholzer-Algorithmus

Nutze für die Eulertour/den Eulerweg auch eine zyklische doppelt-verkettete Liste.

Damit:

- Kosten für das Verschmelzen: $O(1)$
- Nächsten Startknoten suchen: $O(m)$ über alle Iterationen

Damit ist die Laufzeit also:

$$O(n + m) + O(m) + \sum_{\text{Wege } w} O(|W|) = O(n + m) + O(m) + O(m) = O(m)$$

Ersten Startknoten suchen Weitere Startknoten suchen Finden und Verschmelzen aller Wege Da Graph zusammenhängend (also $n \leq m$)

Vergleich zu Fleury

Algorithmus	Hierholzer	Fleury	Fleury (mit Optimierungen)
Laufzeit	$O(m)$	$O(m^2)$	$O(m(\log m)^3 \log \log m)$

Binäre Suchbäume

Bin. Suchbäume

Anstatt einen Nachfolger (Liste), benutze zwei:

- Ein linkes Kind ($l(v)$)
- Ein rechtes Kind ($r(v)$)

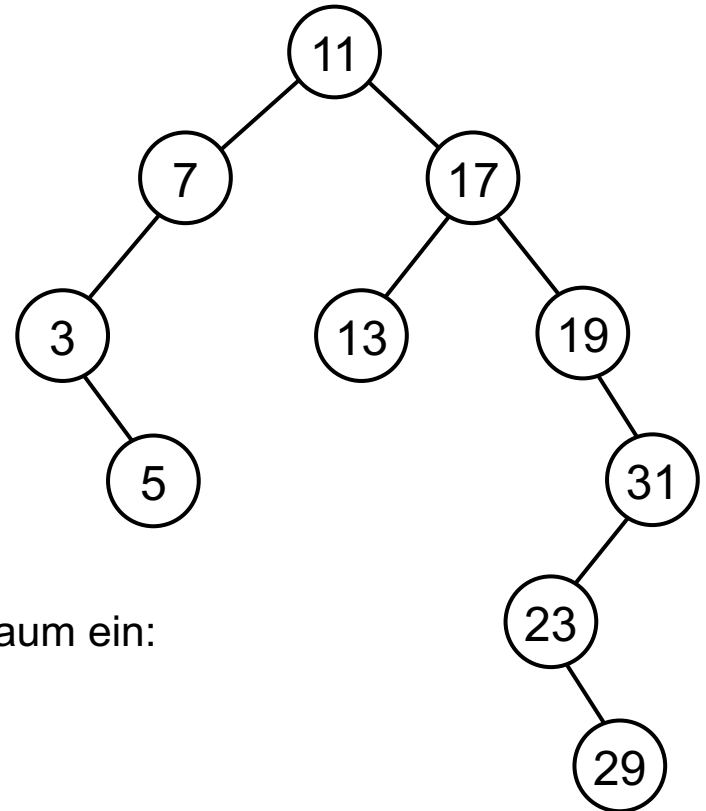
Verwalte zudem eine Totalordnung der Elemente:

- Schlüssel im linken Teilbaum: kleiner
- Schlüssel im rechten Teilbaum: größer

Beispiel:

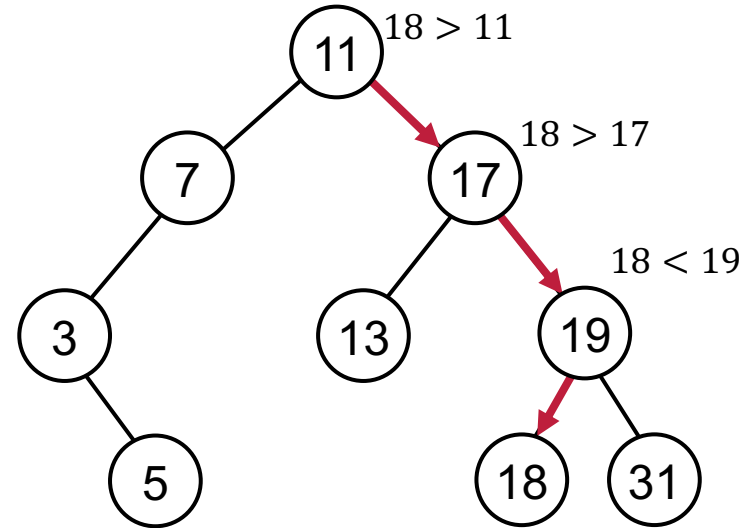
Füge folgende Sequenz von Zahlen in einen bin. Suchbaum ein:

11, 17, 13, 19, 7, 3, 31, 23, 5, 29



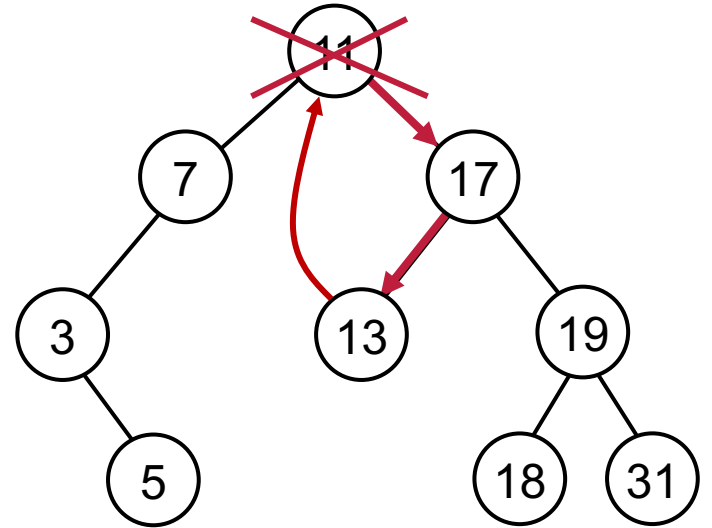
Operationen

- Insert (eben gesehen)
- Search
 - Search(18)



Operationen

- Insert (eben gesehen)
- Search
 - Search(18)
- Vorgänger (pred)/Nachfolger (succ)
 - $\text{pred}(5) = 3$
 - $\text{pred}(17) = 13$
 - $\text{succ}(11) = 13$
 - $\text{succ}(5) = 7$
- Delete
 - Delete(11)

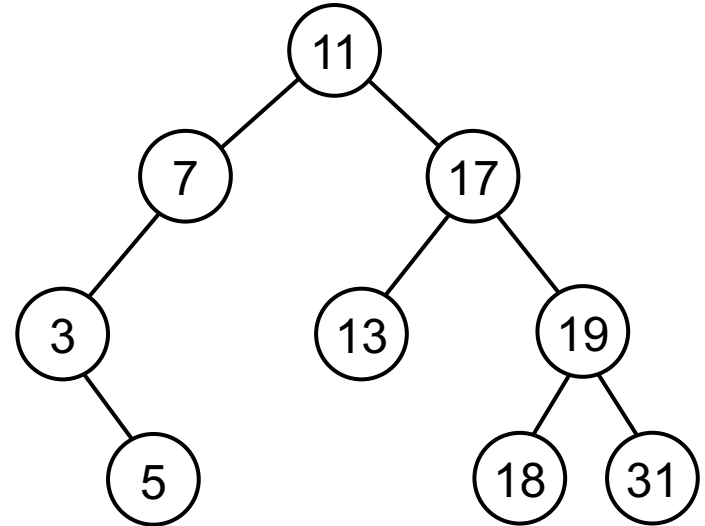


Traversierung

Wie durchläuft man einen bin. Suchbaum?

Man unterscheidet unter anderem drei Möglichkeiten:

- Inorder (Links, Wurzel, Rechts)
 - 3, 5, 7, 11, 13, 17, 18, 19, 31
- Preorder (Wurzel, Links, Rechts)
 - 11, 7, 3, 5, 17, 13, 19, 18, 31
- Postorder (Links, Rechts, Wurzel)
 - 5, 3, 7, 13, 18, 31, 19, 17, 11



Traversierung

Dank der Baumstruktur kann man das ganz einfach als Algorithmus aufschreiben.

<pre>Function inorder(v) if ($v \neq \text{NIL}$) inorder($l(v)$) print $S(v)$ inorder($r(v)$)</pre>	<pre>Function postorder(v) if ($v \neq \text{NIL}$) postorder($l(v)$) postorder($r(v)$) print $S(v)$</pre>	<pre>Function preorder(v) if ($v \neq \text{NIL}$) print $S(v)$ preorder($l(v)$) preorder($r(v)$)</pre>
Inorder	Postorder	Preorder

Laufzeiten der Operationen

Operation	Einfach
Suchen	$O(h)$
Einfügen	$O(h)$
Löschen	$O(h)$
Traversierung	$O(n)$
Merge	??

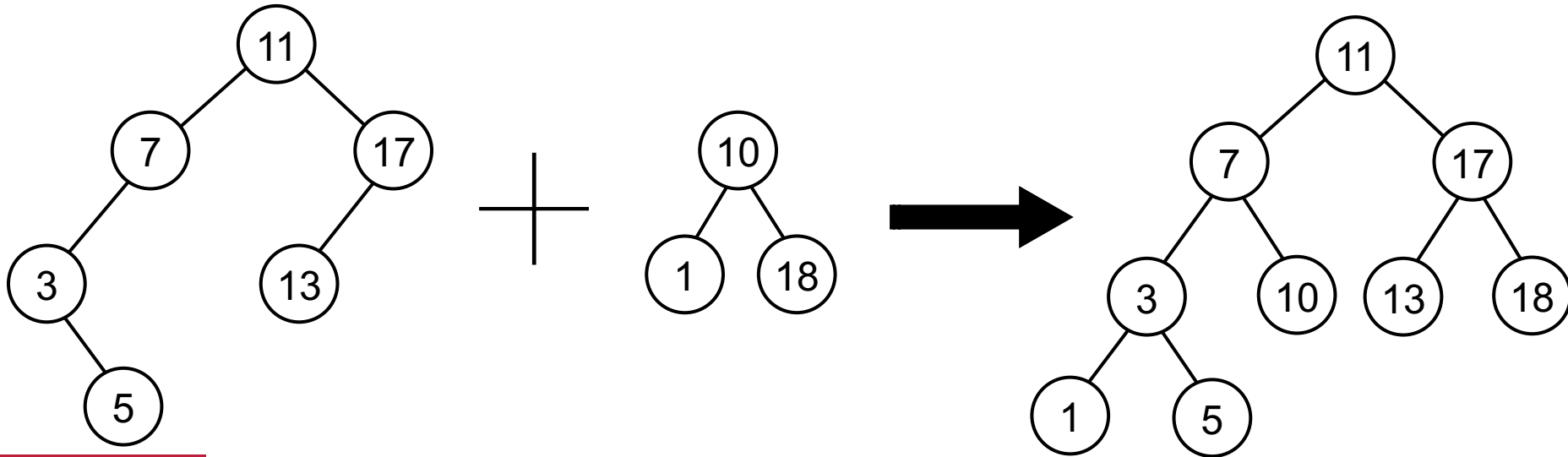
Wie fügt man zwei bin. Suchbäume zusammen?

Verschmelzen von Binären Suchbäumen

Merge – Das Problem

Gegeben: Zwei binäre Suchbäume mit n bzw. m Elementen.
Aufgabe: Konstruiere daraus einen Suchbaum mit $n + m$ Elementen.

Wie schnell geht das?



Merge – Ideen und Schranken

Erste Möglichkeit:

Für jeden Schlüssel S im ersten Suchbaum: Füge S in den zweiten Suchbaum ein.

Laufzeit:

$O(nh_m)$, wobei h_m die Höhe des zweiten Suchbaums ist.

Worst-Case: $O(nm)$

Wie lange braucht man mindestens?

Man muss jeden Schlüssel mindestens einmal betrachten.

Also: $\Omega(n + m)$

Können wir diesen Unterschied zwischen oberer und unterer Schranke schließen?

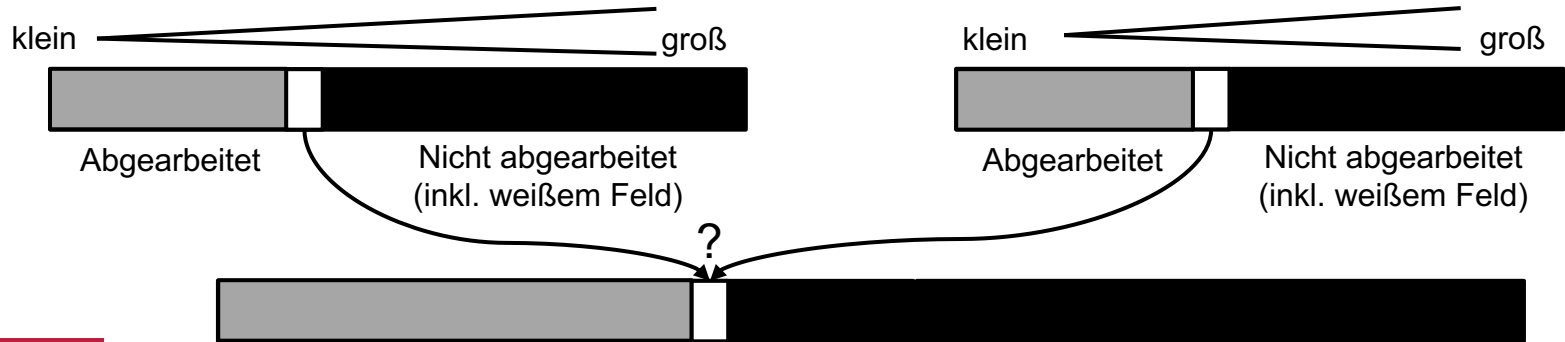
Merge – Alternative

Probieren wir folgende Strategie:

1. Transformiere beide Suchbäume in sortierte Arrays (durch inorder Traversierung).
2. Verschmelze beide Arrays in ein sortiertes Array.
3. Konstruiere aus dem sortierten Array einen Suchbaum.

Punkt 1 benötigt offensichtlich $O(n + m)$ Zeit.

Für Punkt 2 benötigen wir $O(n + m)$ Zeit:

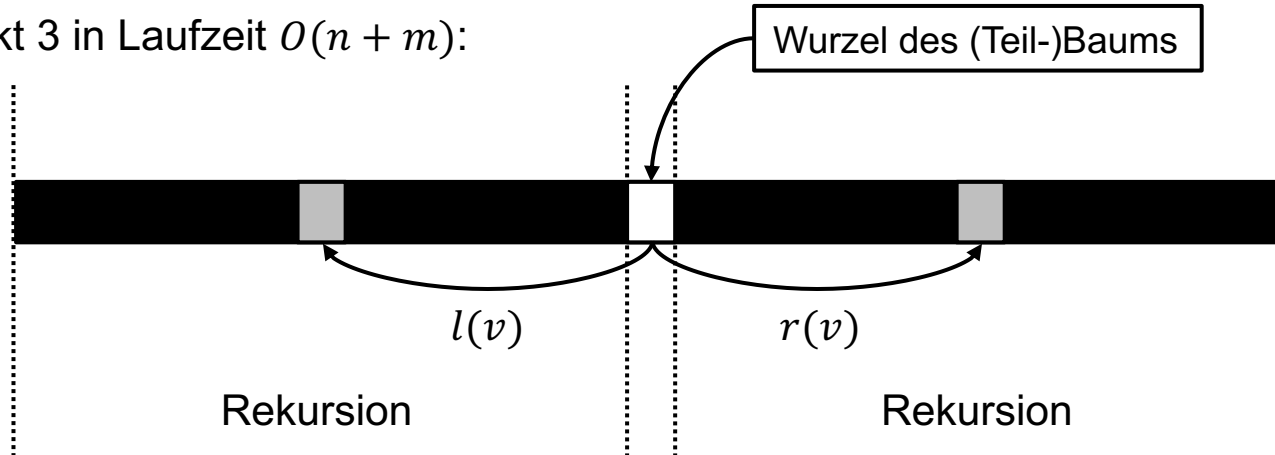


Merge – Alternative

Probieren wir folgende Strategie:

1. Transformiere beide Suchbäume in sortierte Arrays (durch inorder Traversierung).
2. Verschmelze beide Arrays in ein sortiertes Array.
3. Konstruiere aus dem sortierten Array einen Suchbaum.

Für Punkt 3 in Laufzeit $O(n + m)$:



Merge - Beispiel

