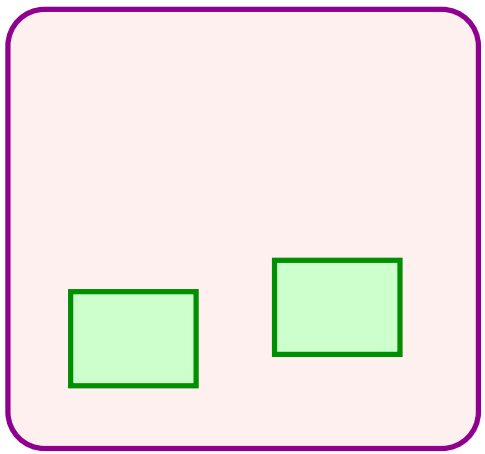
Informatik 1

Wintersemester 2007/2008

Helmut Seidl

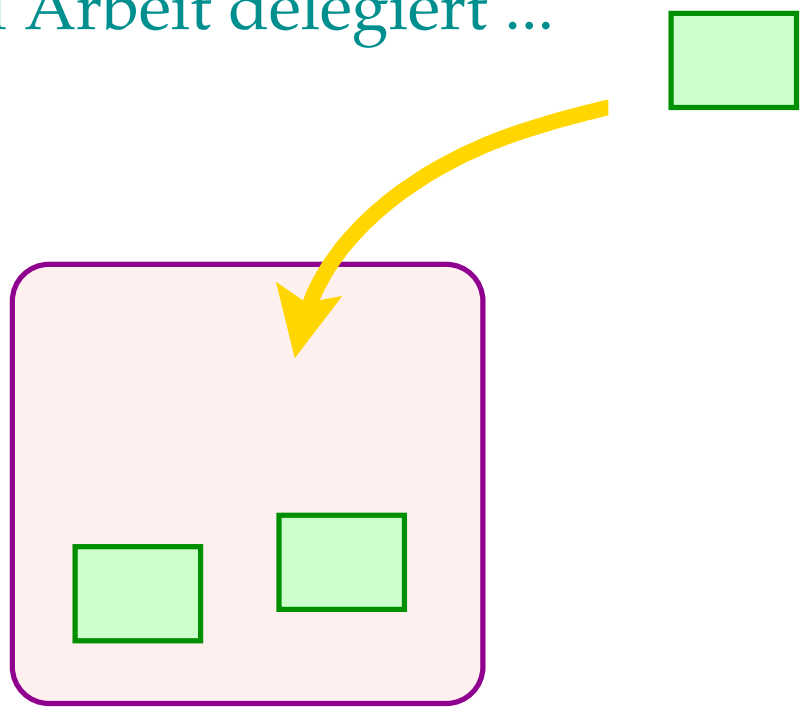
Institut für Informatik
TU München

Anwendung: Schreibtisch



Anwendung: Schreibtisch

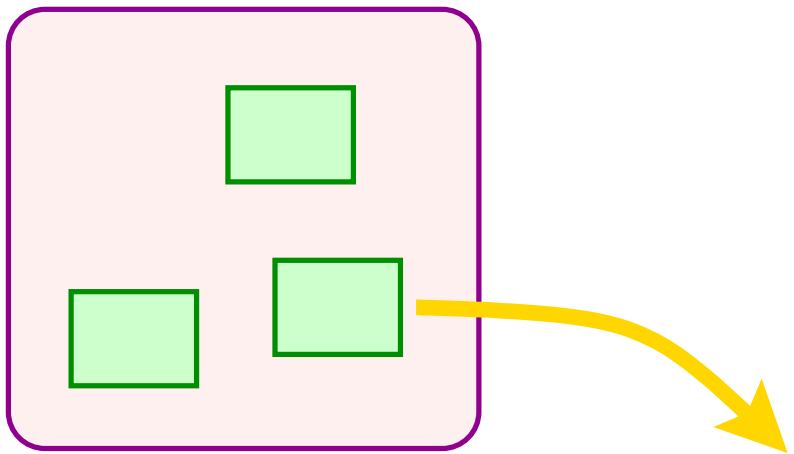
An uns wird Arbeit delegiert ...



Operation: insert(task)

Anwendung: Schreibtisch

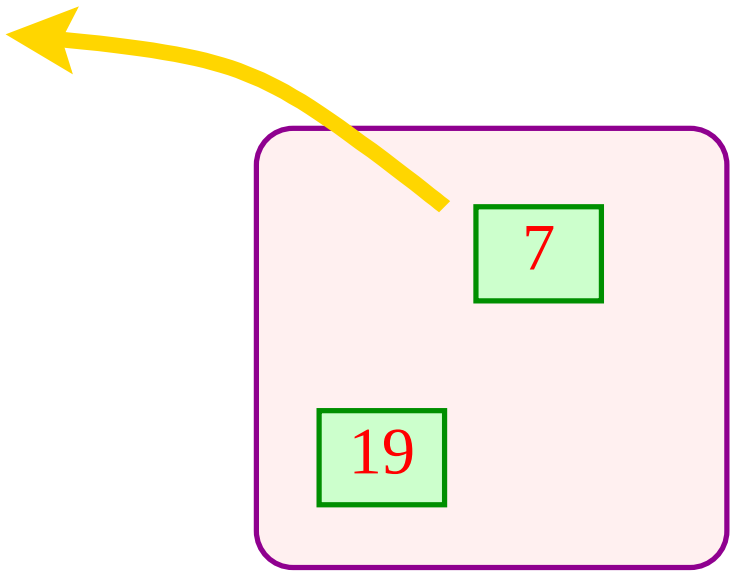
Manches erledigt sich von selbst ...



Operation: delete(task)

Anwendung: Schreibtisch

Anderes erledigen wir ...



Operation: extractMin()

Generelle Philosophie:

Um den Gesamtaufwand zu minimieren, wähle immer die momentan günstigste Alternative !

$\Rightarrow$ Weit verbreitete Lösungsmethode — etwa bei der Suche nach günstigsten Wegen von einem Startpunkt zu einem Ziel.

$\Rightarrow$ Grundlage von Navigationsgeräten

$\uparrow$Algorithmen und Datenstrukturen

Gesucht: Datenstruktur

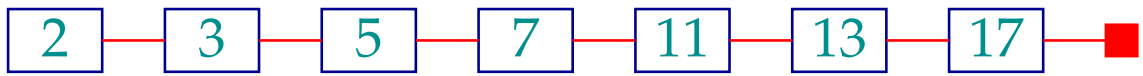
die folgende Methoden zur Verfügung stellt:

```
void insert (Task x);
void delete (Task x);
Task extractMin ();
Task extractMax ();

boolean isEmpty ();
List<Task> listOf ();
```

Zum Vergleich:

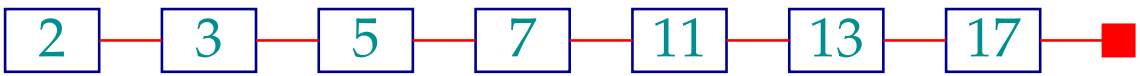
Sortierte Listen:



unterstützen effizient extractMin :-(

Zum Vergleich:

Sortierte Listen:



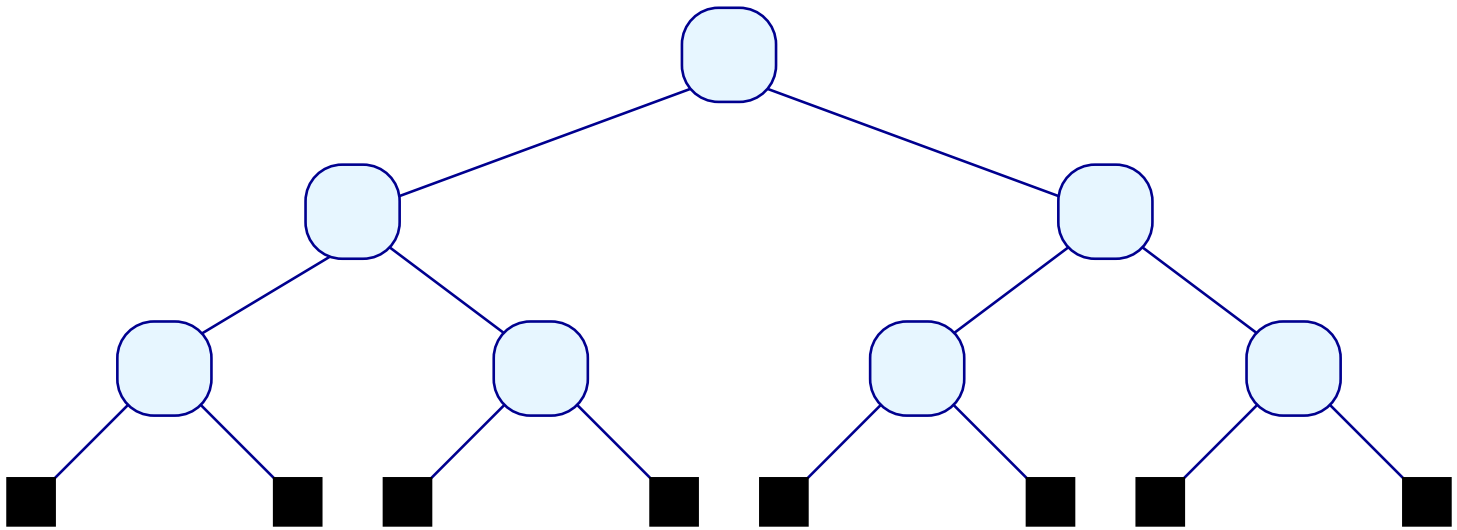
unterstützen effizient extractMin :-(

Sortierte Felder:

unterstützen effizient extractMin, extractMax, delete :-(

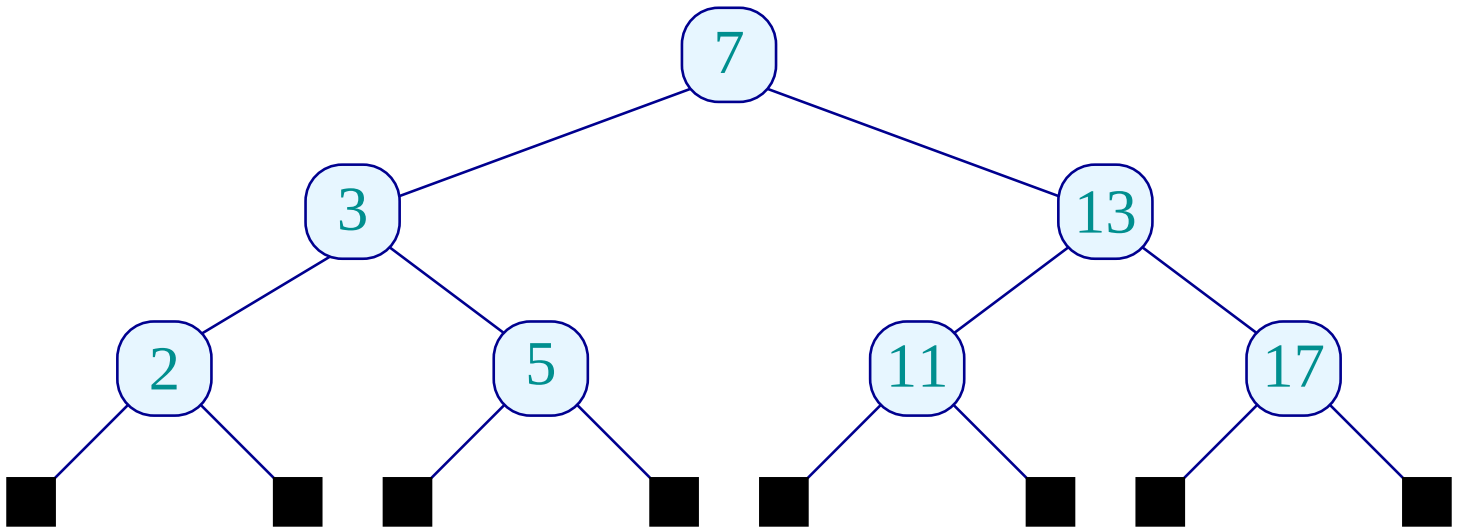
1. Idee:

Benutze balancierte Bäume ...



1. Idee:

Benutze balancierte Bäume ...



Diskussion:

- Wir speichern unsere Daten in den inneren Knoten :-)
- Zum Auffinden eines Elements müssen wir mit allen Elementen auf einem Pfad vergleichen ...

Diskussion:

- Wir speichern unsere Daten in den inneren Knoten :-)
- Zum Auffinden eines Elements müssen wir mit allen Elementen auf einem Pfad vergleichen ...
- Die Tiefe eines Baums ist die maximale Anzahl innerer Knoten auf einem Pfad von der Wurzel zu einem Blatt.
- Ein vollständiger balancierter Binärbaum mit $n = 2^k - 1$ inneren Knoten hat Tiefe $k = \log(n+1)$:-)

Diskussion:

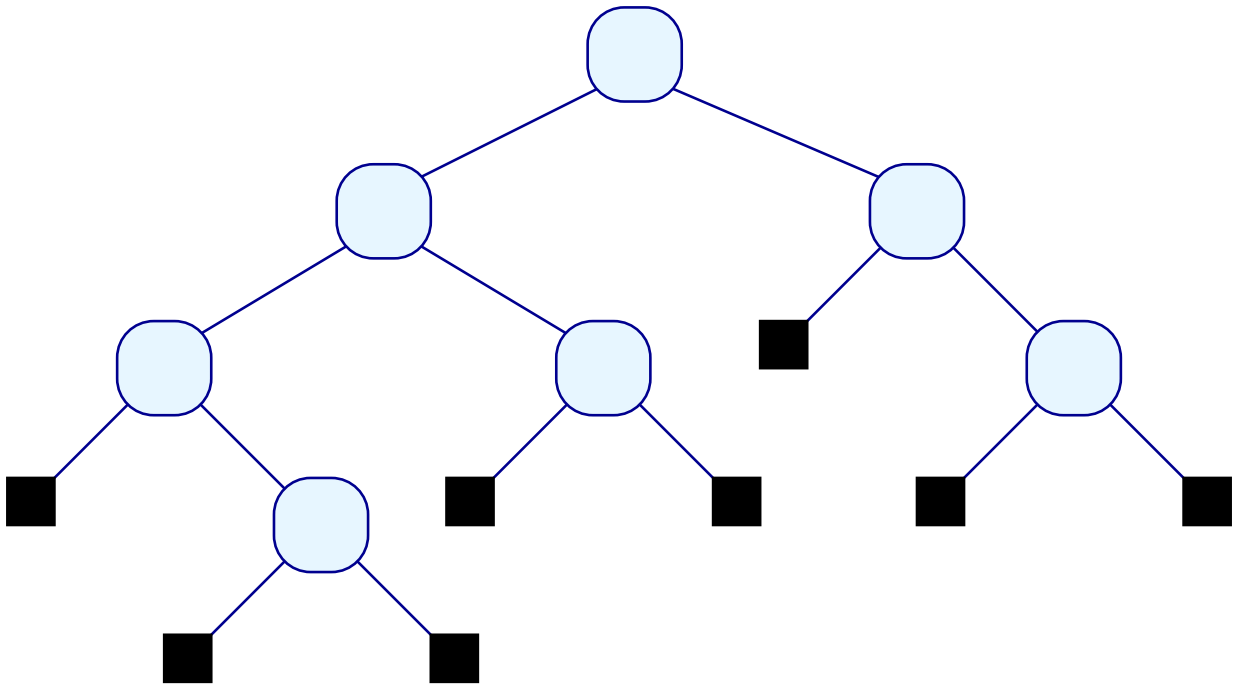
- Wir speichern unsere Daten in den inneren Knoten :-)
- Zum Auffinden eines Elements müssen wir mit allen Elementen auf einem Pfad vergleichen ...
- Die Tiefe eines Baums ist die maximale Anzahl innerer Knoten auf einem Pfad von der Wurzel zu einem Blatt.
- Ein vollständiger balancierter Binärbaum mit $n = 2^k - 1$ inneren Knoten hat Tiefe $k = \log(n+1)$:-)
- Wie fügen wir aber weitere Elemente ein ??
- Wie können wir Elemente löschen ???

2. Idee:

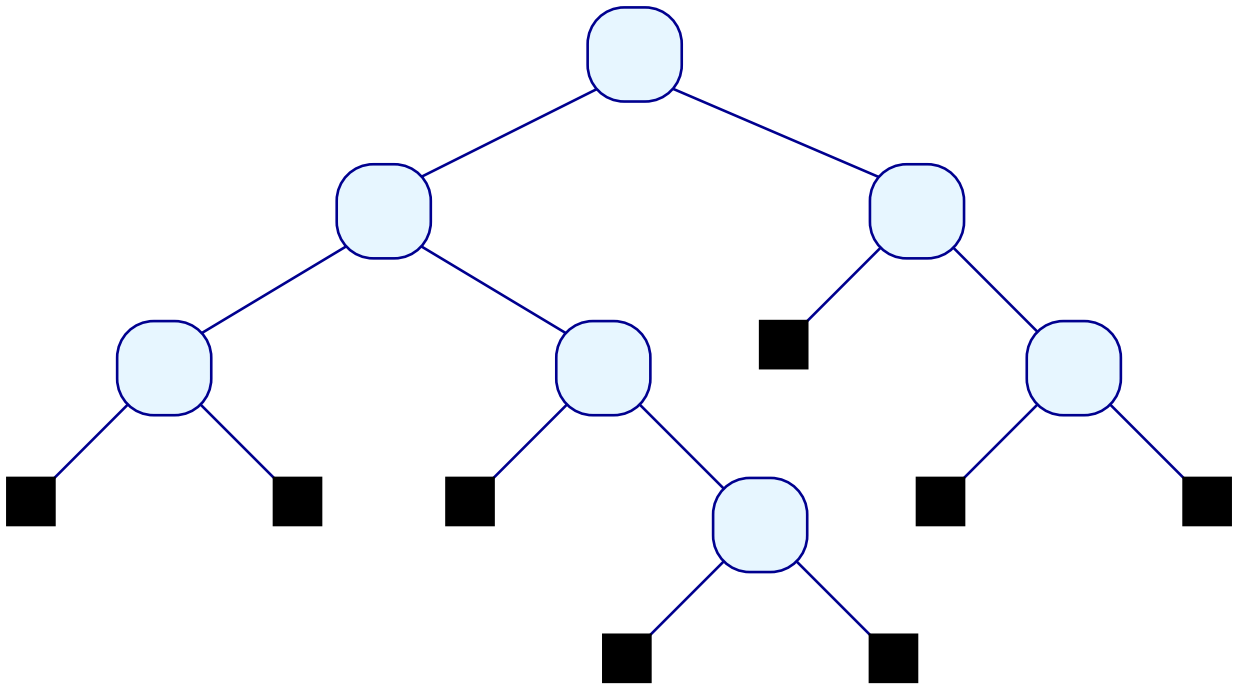
- Statt balancierter Bäume benutzen wir fast balancierte Bäume

  ...
- An jedem Knoten soll die Tiefe des rechten und linken Teilbaums ungefähr gleich sein :-)
- Ein AVL-Baum ist ein Binärbaum, bei dem an jedem inneren Knoten die Tiefen des rechten und linken Teilbaums maximal um 1 differieren:

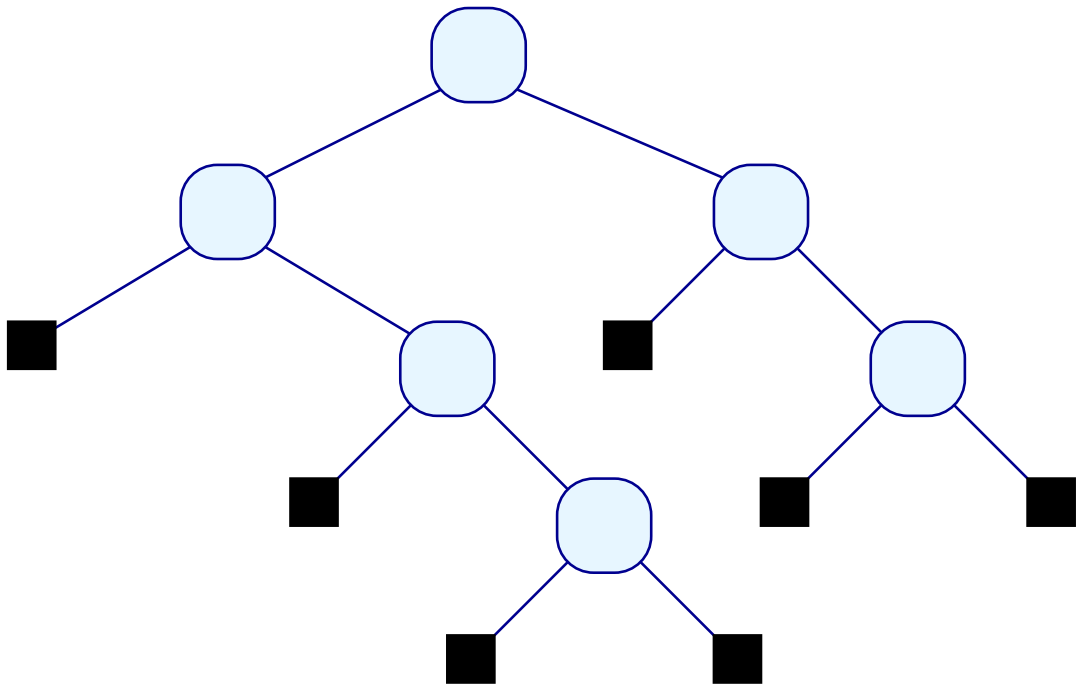
Ein AVL-Baum:



Ein AVL-Baum:



Kein AVL-Baum:



G.M. Adelson-Velskij, 1922

E.M. Landis, Moskau, 1921-1997

Wir vergewissern uns:

(1) Jeder AVL-Baum der Tiefe $k > 0$ hat mindestens

$$\mathsf{fib}(k) \quad \geq \quad A^{k-1}$$

Knoten wobei:

$$\begin{array}{lcll} \mathsf{fib}(k) & = & & k\text{-te Fibonacci-Zahl} \\ A & = & \frac{\sqrt{5}+1}{2} & \text{der goldene Schnitt} \end{array}$$

Wir vergewissern uns:

(1) Jeder AVL-Baum der Tiefe $k > 0$ hat mindestens

$$\mathsf{fib}(k) \quad \geq \quad A^{k-1}$$

Knoten wobei:

$$\begin{array}{lcll} \mathsf{fib}(k) & = & & k\text{-te Fibonacci-Zahl} \\ A & = & \frac{\sqrt{5}+1}{2} & \text{der goldene Schnitt} \end{array}$$

(2) Jeder AVL-Baum mit $n > 0$ inneren Knoten hat Tiefe maximal

$$\frac{1}{\log(A)} \cdot \log(n) + 1$$

Leonardo da Pisa, genannt Fibonacci, ca. 1180-1241

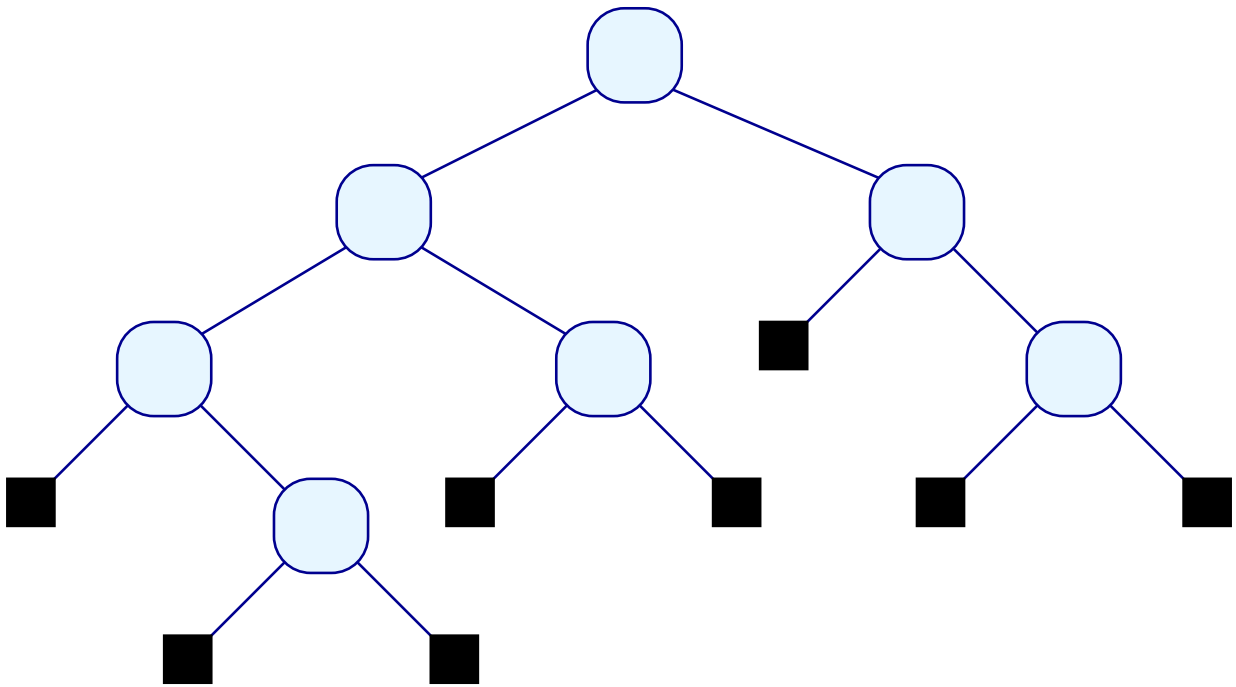
Die Fibonacci-Zahlen:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1.597, \ldots$$

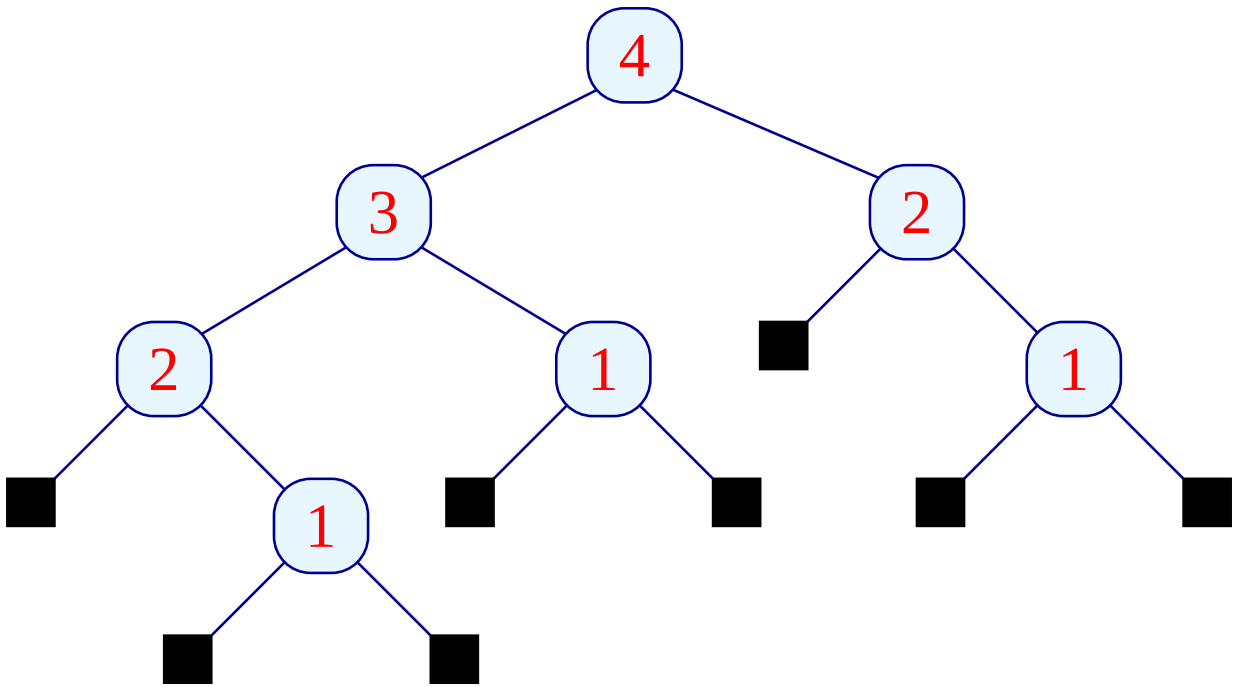
2. Idee (Fortsetzung)

- Fügen wir ein weiteres Element ein, könnte die AVL-Eigenschaft verloren gehen :-(
- Entfernen wir ein Element, könnte die AVL-Eigenschaft verloren gehen :-(
- Dann müssen wir den Baum so umbauen, dass die AVL-Eigenschaft wieder hergestellt wird :-)
- Dazu müssen wir allerdings an jedem inneren Knoten wissen, wie tief die linken bzw. rechten Teilbäume sind ...

Repräsentation:



Repräsentation:

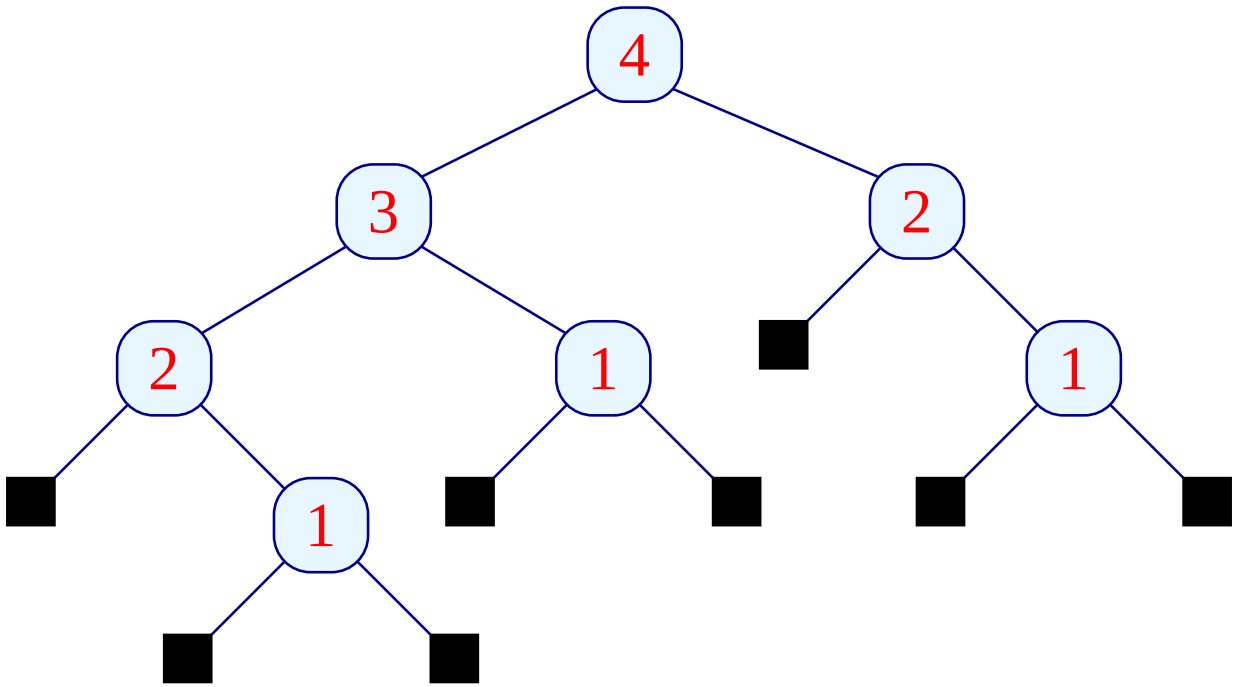


3. Idee:

- Anstelle der Tiefen speichern wir an jedem Knoten nur, ob die Differenz der Tiefen der Teilbäume negativ, positiv oder ob sie gleich sind !!!

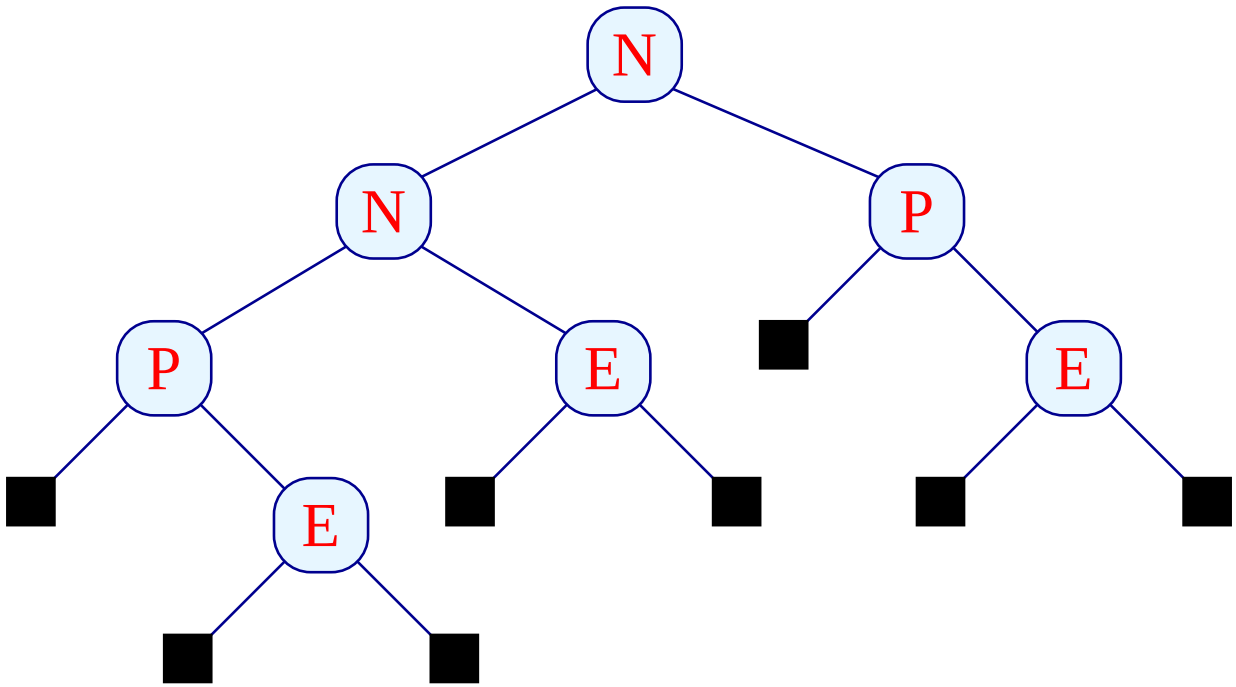
3. Idee:

- Anstelle der Tiefen speichern wir an jedem Knoten nur, ob die Differenz der Tiefen der Teilbäume negativ, positiv oder ob sie gleich sind !!!



3. Idee:

- Anstelle der Tiefen speichern wir an jedem Knoten nur, ob die Differenz der Tiefen der Teilbäume negativ, positiv oder ob sie gleich sind !!!



Einfügen:

- Ist der Baum ein Blatt, erzeugen wir einen neuen inneren Knoten mit zwei neuen leeren Blättern.
- Ist der Baum nicht-leer, vergleichen wir den einzufügenden Wert mit dem Wert an der Wurzel.

  → Ist er größer, fügen wir rechts ein.

  → Ist er kleiner, fügen wir links ein.

Einfügen:

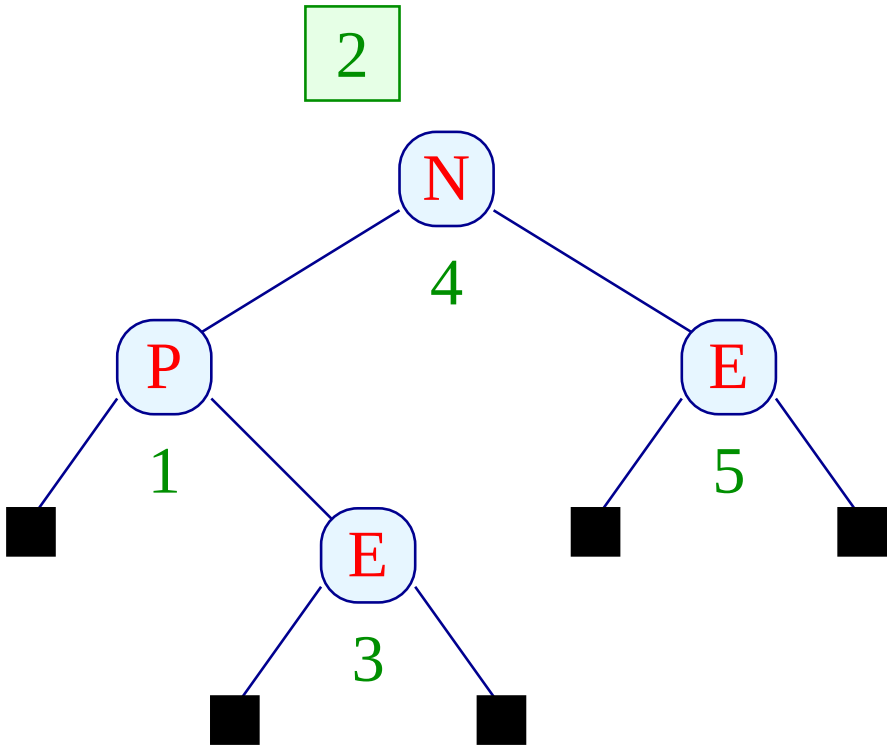
- Ist der Baum ein Blatt, erzeugen wir einen neuen inneren Knoten mit zwei neuen leeren Blättern.
- Ist der Baum nicht-leer, vergleichen wir den einzufügenden Wert mit dem Wert an der Wurzel.

  → Ist er größer, fügen wir rechts ein.

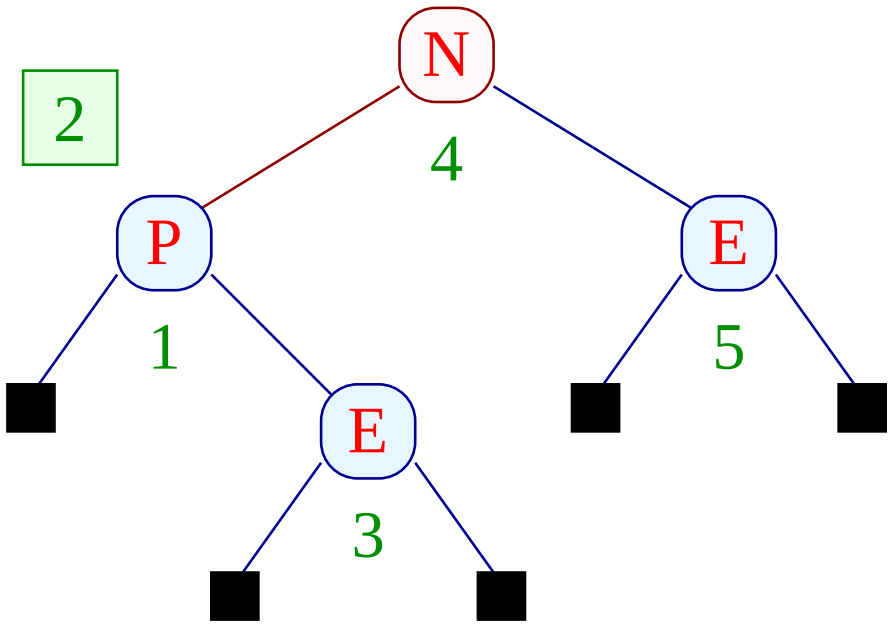
  → Ist er kleiner, fügen wir links ein.

- Achtung: Einfügen kann die Tiefe erhöhen und damit die AVL-Eigenschaft zerstören !
- Das müssen wir reparieren ...

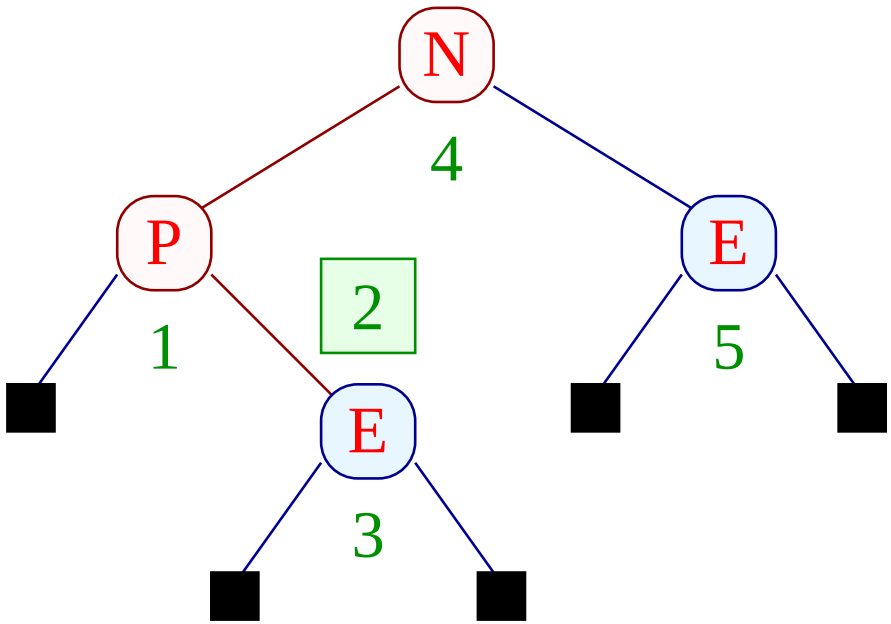
Einfügen:



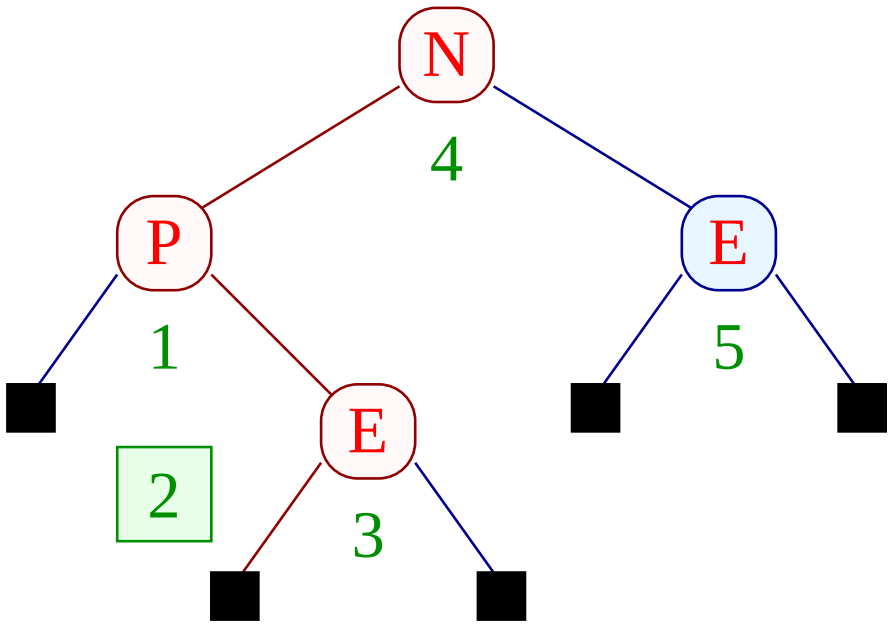
Einfügen:



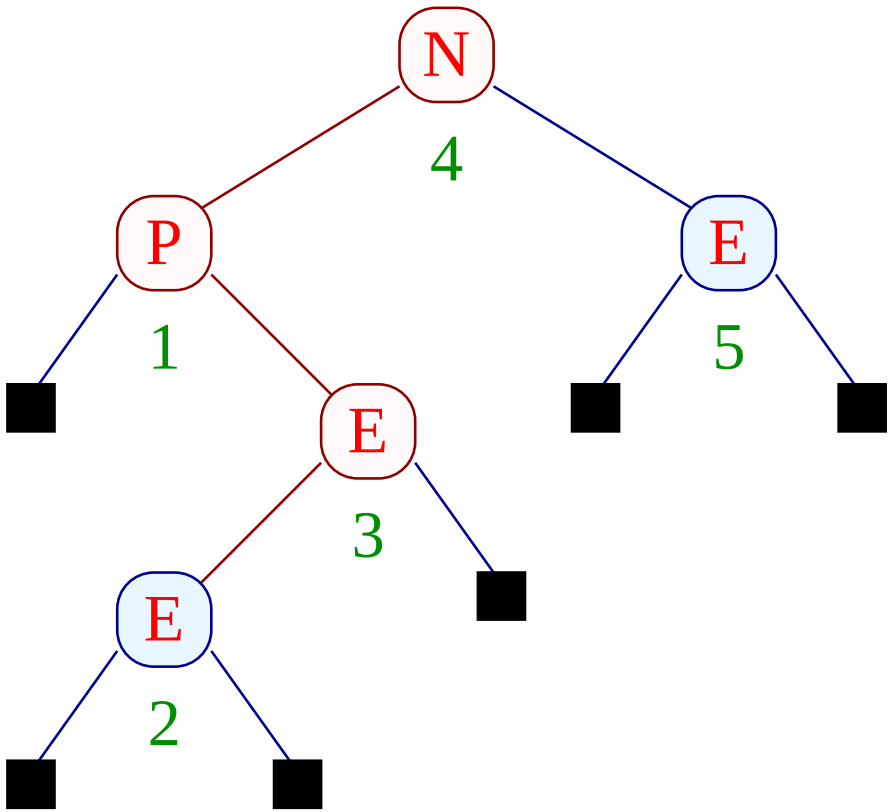
Einfügen:



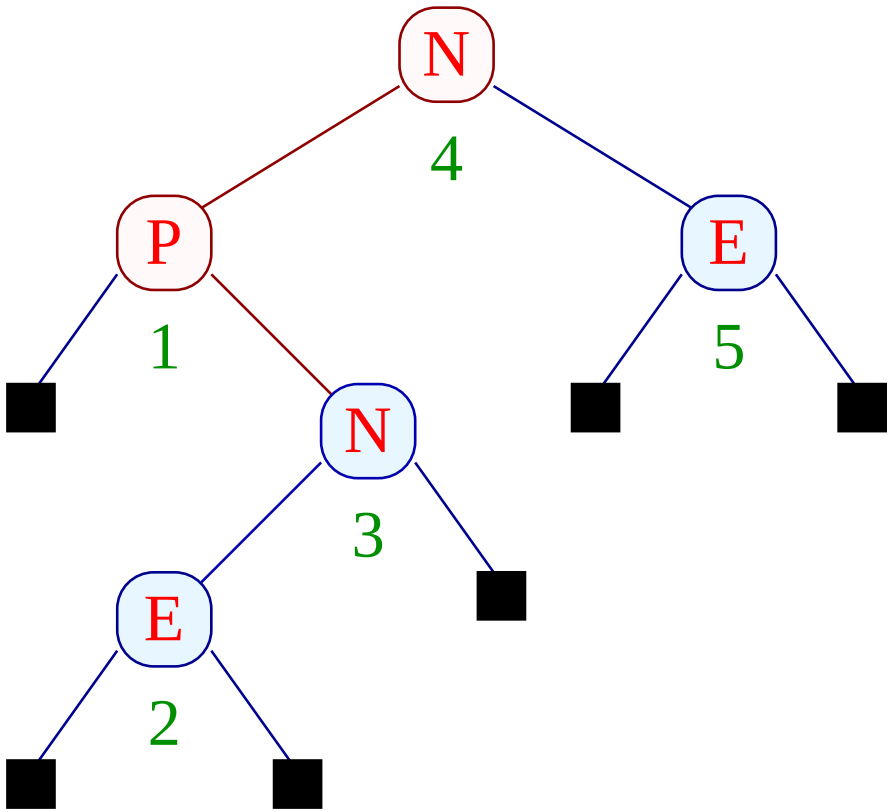
Einfügen:



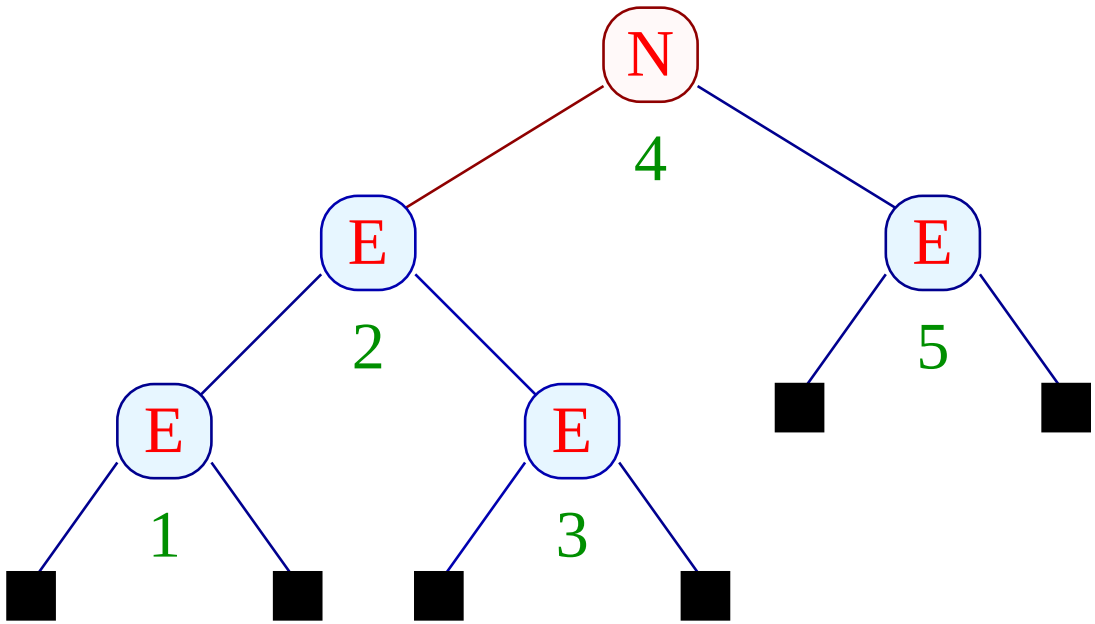
Einfügen:



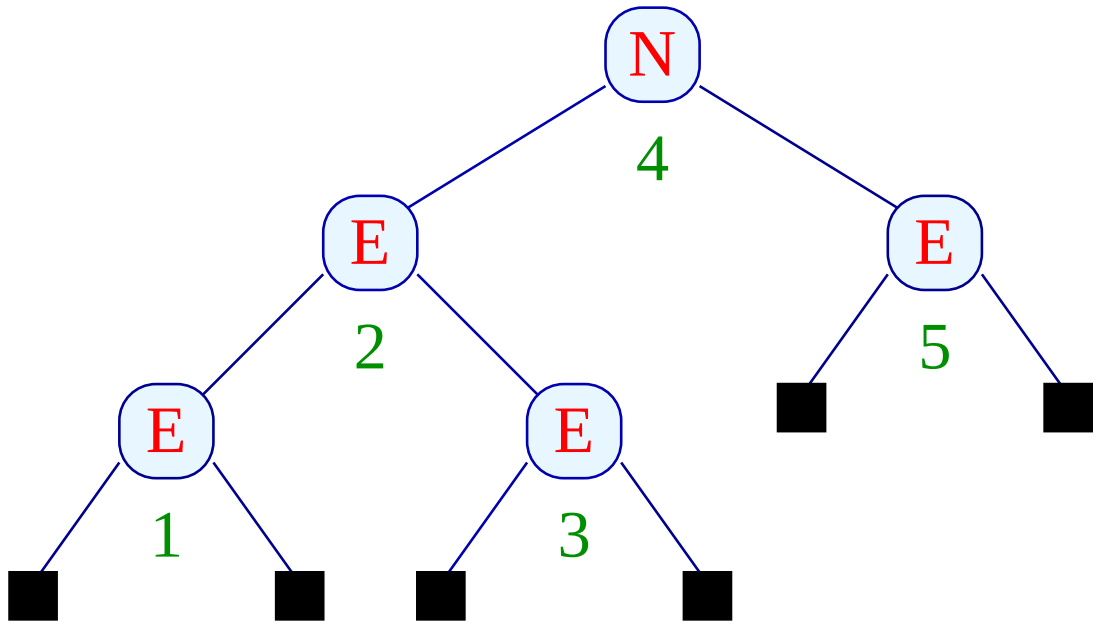
Reparieren:



Reparieren:



Reparieren:



Beobachtung:

- Die Reparatur erfolgt nur entlang des Pfads zu dem Blatt, an dem eingefügt wurde.
- Um zu wissen, ob eine Reparatur erforderlich ist, müssen wir für jeden fertigen AVL-Baum feststellen, ob sich seine Tiefe vergrößert hat.

Beobachtung:

- Die Reparatur erfolgt nur entlang des Pfads zu dem Blatt, an dem eingefügt wurde.
- Um zu wissen, ob eine Reparatur erforderlich ist, müssen wir für jeden fertigen AVL-Baum feststellen, ob sich seine Tiefe vergrößert hat.
- Erhöht sich die Tiefe für einen Teilbaum nicht, muss für die darüber liegenden Knoten keine Korrektur mehr vorgenommen werden :-)
- Muss eine Reparatur vorgenommen werden, dann hängt die zu ergreifende Maßnahme allein von den Markierungen der Knoten ab ...

Maßnahmen:

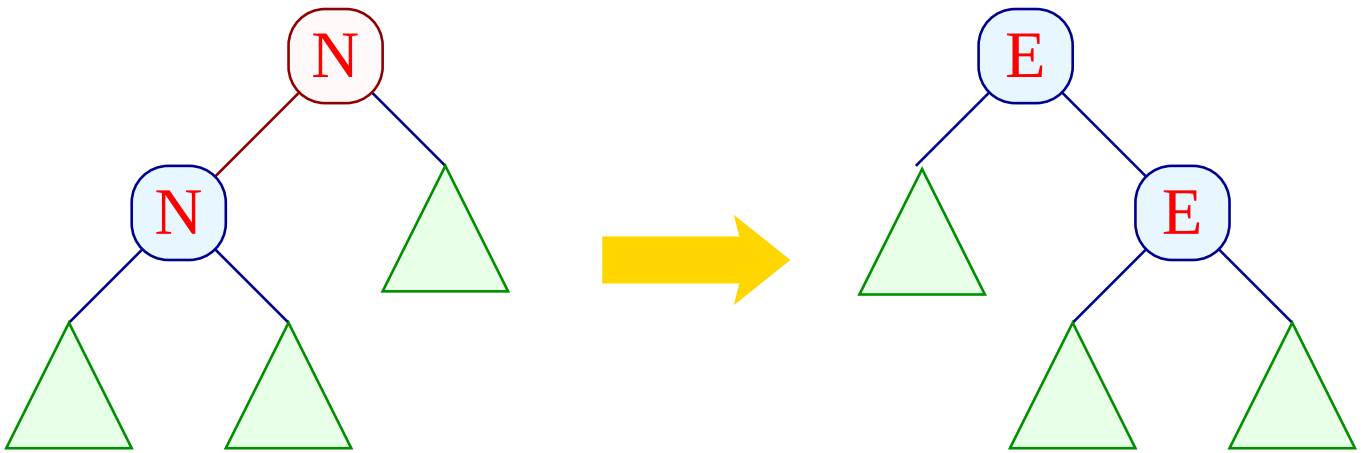
- Einfügen in den flacheren Teilbaum erhöht die Gesamttiefe nie :-)

  Gegebenenfalls werden aber beide Teilbäume gleich tief.

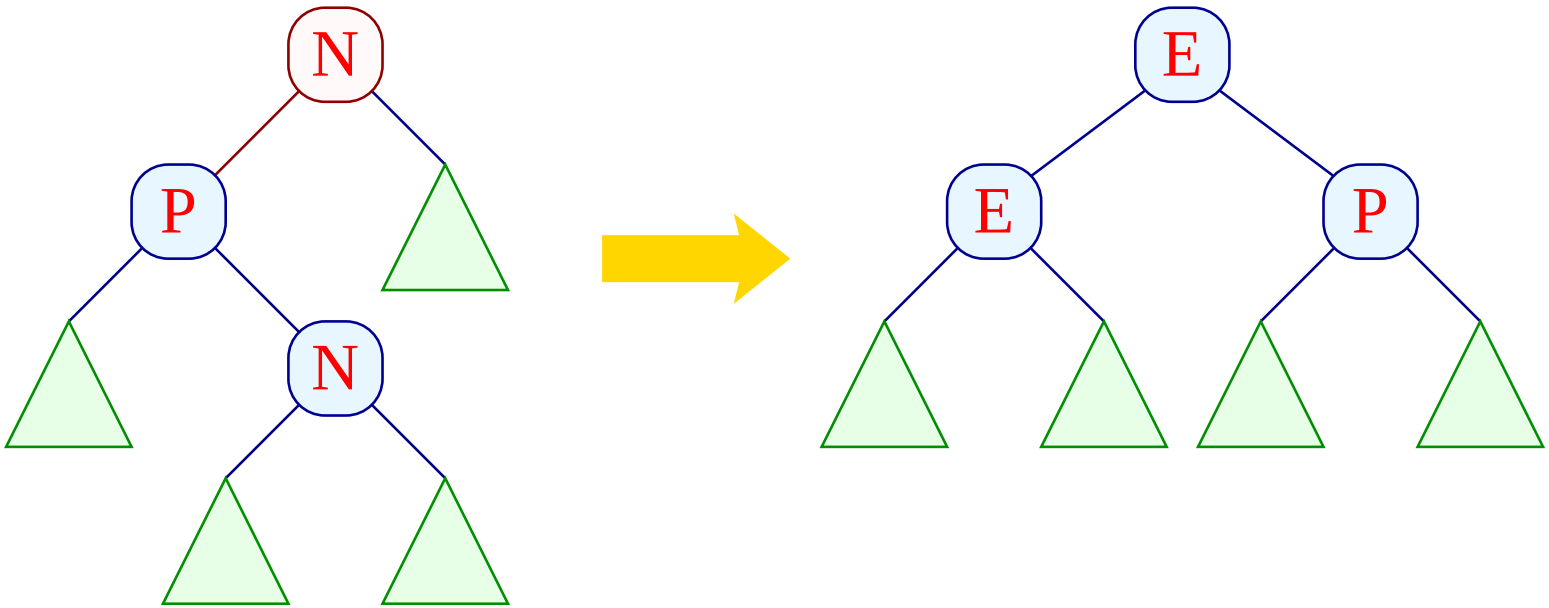
- Einfügen in den tieferen Teilbaum kann dazu führen, dass der Tiefenunterschied auf 2 anwächst :-(

  Dann rotieren wir Knoten an der Wurzel, um die Differenz auszugleichen ...

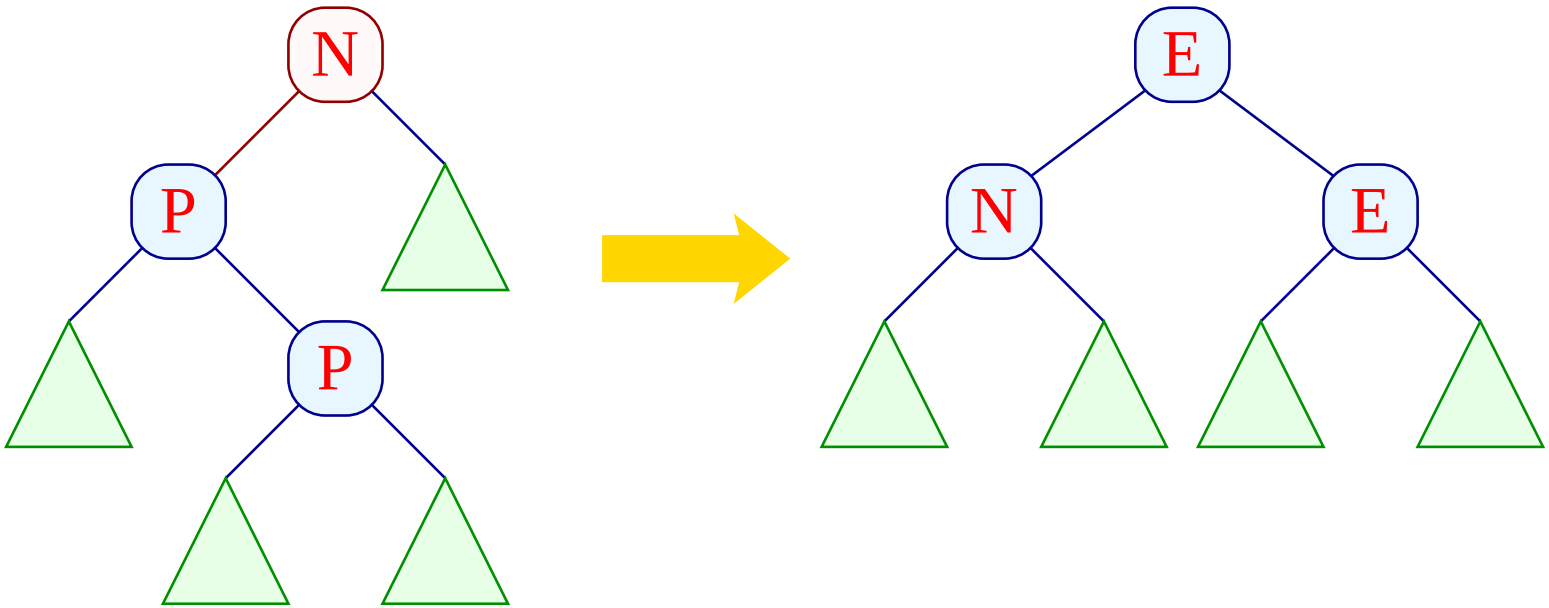
`rotateRight:`



`rotateRight:`



`rotateRight:`



Diskussion:

- Einfügen benötigt höchstens soviele Aufrufe von `insert` wie der Baum tief ist.
- Nach Rückkehr aus dem Aufruf für einen Teilbaum müssen maximal drei Knoten umorganisiert werden.
- Der Gesamtaufwand ist darum proportional zur Tiefe des Baums, d.h. zu $\log(n)$:-)

Extraktion des Minimums:

- Das Minimum steht am linksten inneren Knoten.
- Dieses finden wir mithilfe eines rekursiven Besuchens des jeweils linken Teilbaums :-)

  Den linksten Knoten haben wir gefunden, wenn der linke Teilbaum leer ist :-))

Extraktion des Minimums:

- Das Minimum steht am linksten inneren Knoten.
- Dieses finden wir mithilfe eines rekursiven Besuchens des jeweils linken Teilbaums :-)

  Den linksten Knoten haben wir gefunden, wenn der linke Teilbaum leer ist :-))
- Entfernen eines Blatts könnte die Tiefe verringern und damit die AVL-Eigenschaft zerstören.
- Nach jedem Aufruf müssen wir darum den Baum lokal reparieren ...

Diskussion:

- Insgesamt ist die Anzahl der rekursiven Aufrufe beschränkt durch die Tiefe. Zur Reparatur werden bei jedem Aufruf maximal drei Knoten umgeordnet.
- Der Gesamtaufwand ist damit wieder proportional zu $\log(n)$:-)

Diskussion:

- Insgesamt ist die Anzahl der rekursiven Aufrufe beschränkt durch die Tiefe. Zur Reparatur werden bei jedem Aufruf maximal drei Knoten umgeordnet.
- Der Gesamtaufwand ist damit wieder proportional zu $\log(n)$:-)
- Analog konstruiert man Funktionen, die das Maximum extrahieren.
- Nachdem wir verstanden haben, wie AVL-Bäume funktionieren, können wir sie nun in Java implementieren ...

Vielen Dank!