

Norman May

B6, 29, Raum C0.05  
68131 Mannheim  
Telefon: (0621) 181-2517  
Email: norman@pi3.informatik.uni-mannheim.de

Matthias Brantner

B6, 29, Raum C0.05  
68131 Mannheim  
Telefon: (0621) 181-2517  
Email: msb@pi3.informatik.uni-mannheim.de

Algorithmen und Datenstrukturen  
Wintersemester 2004/05

8. Lösungsvorschlag  
24. Dezember 2004

Aufgabe 1

7 Punkte

Für 2 gegebene Strings  $a = a_0a_1 \dots a_p$  und  $b = b_0b_1 \dots b_q$ , wobei jedes  $a_i$  und jedes  $b_j$  in einem Zeichensatz enthalten sind, nennen wir  $a$  *lexikographisch kleiner* als String  $b$ , wenn eine der folgenden Bedingungen erfüllt ist:

1. Es gibt eine Zahl  $j$ , mit  $0 \leq j \leq \min(p, q)$ , mit  $a_i = b_i$  für alle  $i = 0, 1, \dots, j - 1$  und  $a_j < b_j$ , oder
2.  $p < q$  und  $a_i = b_i$  für alle  $i = 0, 1, \dots, p$

Wenn wir  $a$  und  $b$  als Bitstrings betrachten, dann ist  $10100 < 10110$  (Regel 1) und  $10100 < 101000$  (Regel 2).

Zum Speichern von Bitstrings kann ein *Radix-Baum* verwendet werden. In Abbildung ?? ist der Radix-Baum für die Bitstrings 1011, 10, 011, 100 und 0 gegeben. Weiße Knoten enthalten im Baum gespeicherte Bitstrings.

Zum Auffinden eines Schlüssels  $a = a_0a_1 \dots a_p$ , geht man in Tiefe  $i$  in den linken Teilbaum, wenn  $a_i = 0$  und nach rechts im anderen Fall.

Sei  $S$  eine Menge verschiedener Strings aus Binärzahlen mit der Gesamtlänge  $n$ . In den folgenden Teilaufgaben sollen Sie einen Algorithmus entwerfen, der mittels Radix-Bäumen  $S$  lexikographisch in  $\Theta(n)$  sortiert.

Aufgabe 1 a)

3 Punkte

Geben Sie in eine Funktion an, die einen String  $s$  in  $\Theta(|s|)$  in den Radixbaum einfügt. Begründen Sie, warum Ihr Algorithmus das Kriterium erfüllt.

Lösung

Da das Eingabealphabet nur aus 2 Elementen besteht, ist der Radix-Baum ein Binärbaum. Jeder Knoten im Baum enthält neben dem Zeiger zum linken und rechten Teilbaum ein Attribut *value*, das wahr ist, wenn der Knoten einen Bitstring darstellt, der in dem Radix-Baum gespeichert wurde.

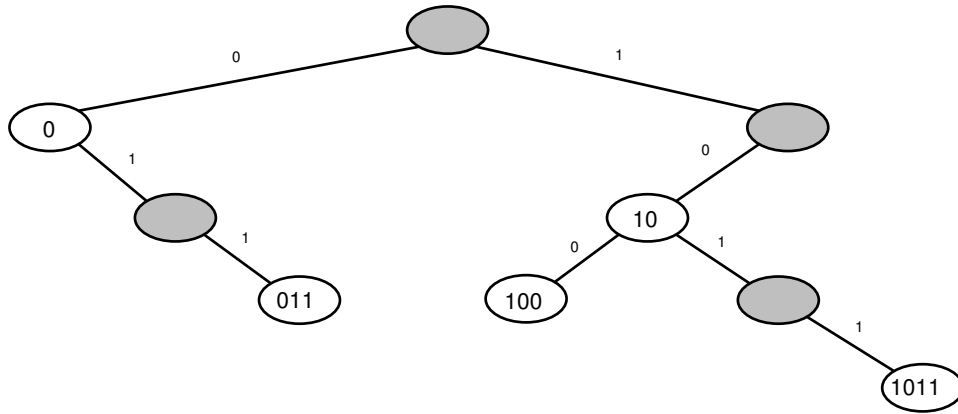


Abbildung 1: Radix-Tree für das Beispiel

▷ insert string  $s$  into the current sub tree

```

insert( $s$ )
  if head( $s$ ) ==  $\epsilon$ 
    value  $\leftarrow$  true
    return
  if head( $s$ ) == '0'
    if left == NULL
      left  $\leftarrow$  new node ▷ create new node
      left.insert(tail( $s$ ))
    else
      if right == NULL
        right  $\leftarrow$  new node ▷ create new node
        right.insert(tail( $s$ ))

```

Bei einem String  $s$  der Länge  $|s|$  wird die Methode **insert**  $|s|$ -mal rekursiv aufgerufen. Daraus folgt direkt die Komplexitätsklasse  $\Theta(|s|)$ .

(Code 2.5 Punkte, Komplexität: 0.5)

Aufgabe 1 b)

3 Punkte

Geben Sie in eine Funktion an, die alle Strings im Radixbaum  $O(|S|)$  ausdrückt. Begründen Sie, warum Ihr Algorithmus das Kriterium erfüllt.

Lösung

▷ print all strings in the subtree routed at the current node

▷  $S$  the bitstring representing the current node

```

printSorted( $s$ )
  if this == NULL
    return

```

```

if value == true
    print(S)

left.printSorted(S+'0')
right.printSorted(S+'1')

```

Der schlechteste Fall tritt ein, wenn nur Strings eingefügt wurden, die keine Präfixe gemeinsam haben (das kann nur bei maximal 2 eingefügten Strings der Fall sein). Dann muss der Pfad zu den Blättern dieser Strings traversiert werden. Die Summe der Länge dieser Pfade entspricht genau  $n$ .

(Code 1.5 Punkte, Komplexität: 1.5)

### Aufgabe 1 c)

1 Punkte

Verwenden Sie die Lösungen aus den ersten beiden Teilaufgaben, um zu zeigen, dass die gewünschte Gesamtlauzeit erreicht wird, wenn zuerst alle Strings in  $S$  in den Radixbaum eingefügt und anschliessend sortiert ausgedruckt werden.

### Lösung

Jeder eingefügte String  $s_i \subseteq S$  der Länge  $|s_i|$  wird in  $\Theta(|s_i|)$  Schritten in den Radix-Baum eingefügt. Insgesamt folgt daraus die Gesamtkomplexität für das Einfügen  $T_1(n) = \sum_i |s_i| = n$ .

Das Ausdrucken aller eingefügten Strings wird in  $T_2(n) = O(n)$  ausgeführt.

Daraus folgt die Gesamtkomplexität  $T(n) = T_1(n) + T_2(n) = \Theta(n)$

---

### Aufgabe 2

7 Punkte

---

Sei  $b_n$  die Anzahl der verschiedenen geordneten Binärbäume mit  $n$  Knoten. In dieser Aufgabe wird eine Formel untersucht, die die Anzahl der strukturell verschiedenen Binärbäume bestimmt.

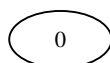
### Aufgabe 2 a)

3 Punkte

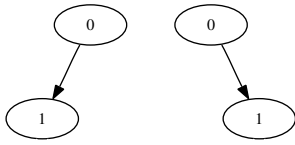
Geben Sie für  $n = 1, 2, 3, 4$  die strukturell verschiedenen Binärbäume an.

### Lösung

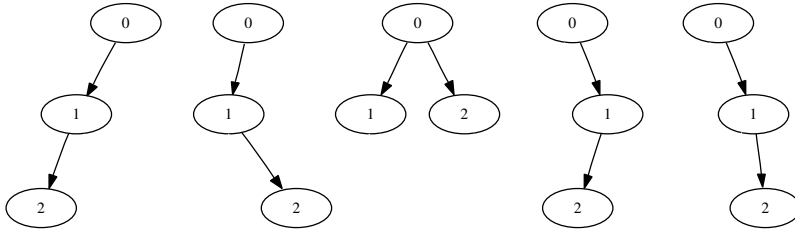
Baum mit einem Knoten:



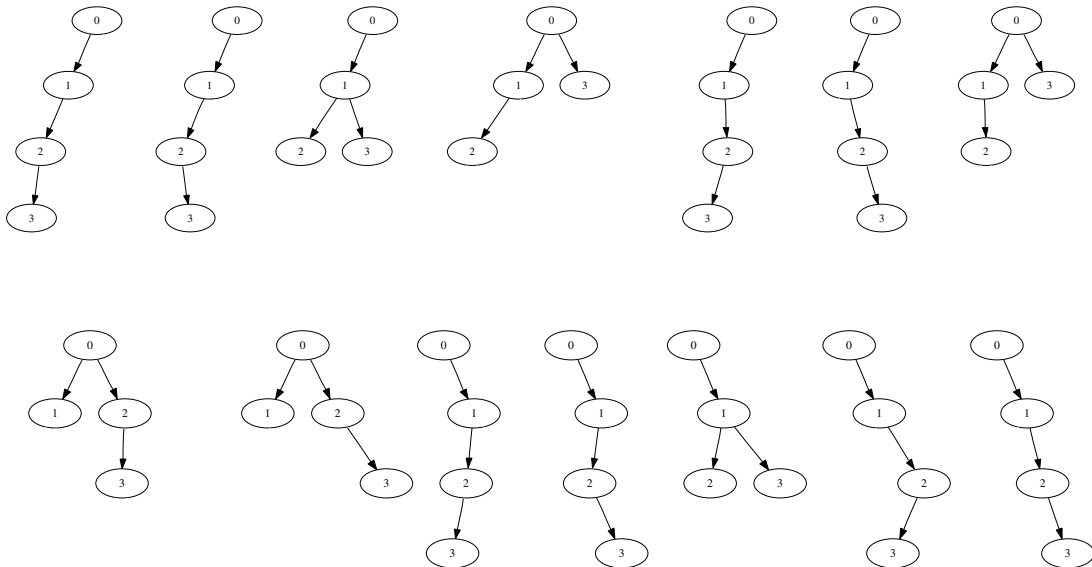
Bäume mit 2 Knoten:



Bäume mit 3 Knoten:



Bäume mit 4 Knoten:



Die Anzahl der strukturell verschiedenen Binärbäume mit  $n$  Knoten entspricht der  $n$ -ten Catalan'schen Zahl ( $C(n) = b_n$ ).

(1 und 2 Knoten: je 0.5 Punkte; 3 und 4 Knoten je 1 Punkt)

Aufgabe 2 b)

4 Punkte

Die Anzahl der strukturell verschiedenen Binärbäume kann mit folgender geschlossener Formel berechnet werden:

$$b_n = \frac{1}{1+n} \binom{2n}{n}$$

Zeigen Sie, daß:

$$b_n = \Theta\left(\frac{4^n}{\sqrt{\pi n^{3/2}}}\right)$$

Falls Sie keinen Lösungsweg finden, zeigen Sie, daß  $b_n = \Omega(2^n)$ .

Lösung

zu zeigen:  $b_n = \Theta\left(\frac{4^n}{\sqrt{\pi n^{3/2}}}\right)$

$$\begin{aligned} b_n &= \frac{1}{1+n} \binom{2n}{n} \\ &= \frac{1}{n+1} \frac{(2n)!}{n!n!} \\ &= \frac{\sqrt{4\pi n} \left(\frac{2n}{e}\right)^{2n} (1 + \Theta(1/n))}{(n+1) (\sqrt{2\pi n} \frac{n}{e} (1 + \Theta(1/n)))^2} \\ &= \frac{4^n}{(n+1) \sqrt{\pi n} (1 + \Theta(1/n))^2} \\ &= \Theta\left(\frac{4^n}{\sqrt{\pi n^{3/2}}}\right) \end{aligned}$$

(4 Punkte)

alternativ: zu zeigen:  $b_n = \Omega(2^n)$

$$\begin{aligned} b_n &= \frac{1}{1+n} \binom{2n}{n} \\ &= \frac{1}{n+1} \frac{(2n)!}{n!n!} \\ &= \prod_{i=2}^n \frac{n+i}{i} \\ &\geq 2^{n-1} = \Omega(2^n) \end{aligned}$$

(2.5 Punkte)

---

Aufgabe 3

6 Punkte

---

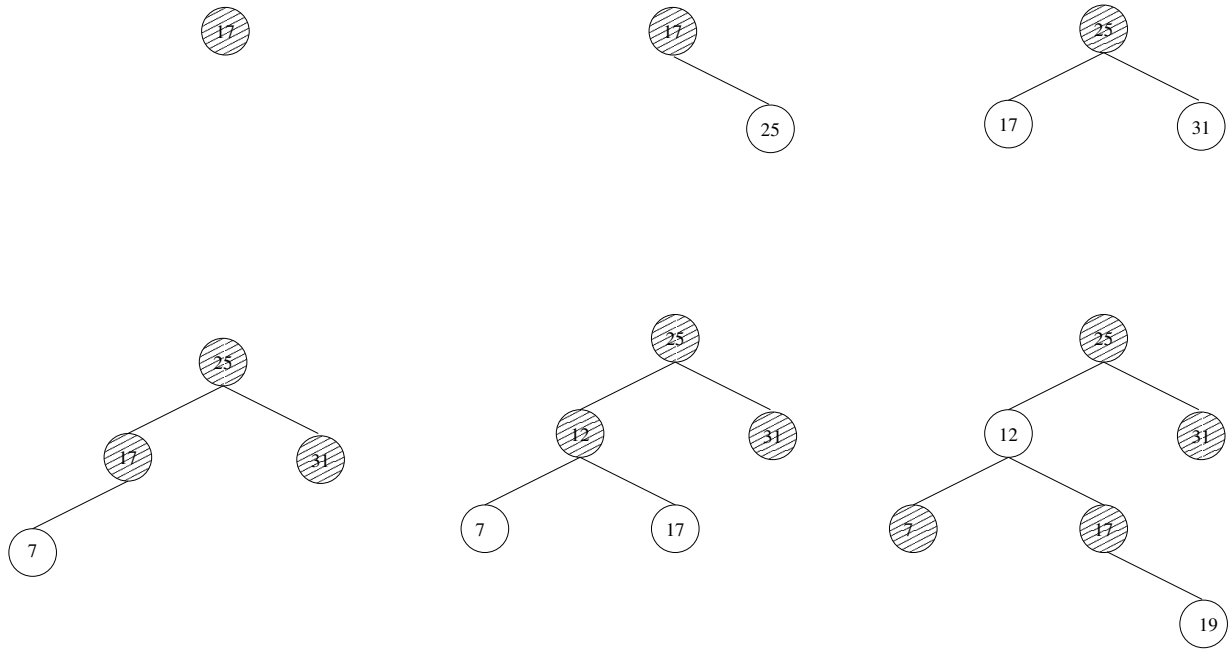
Aufgabe 3 a)

3 Punkte

Geben Sie den RS-Baum an, der nach dem Einfügen der Schlüssel 17, 25, 31, 7, 12, 19 in einem leeren Baum, entsteht.

Lösung

Die schraffierten Knoten stellen die schwarzen und die nicht schraffierten die roten Knoten dar.



(0.5 Punkte je korrekter Operation)

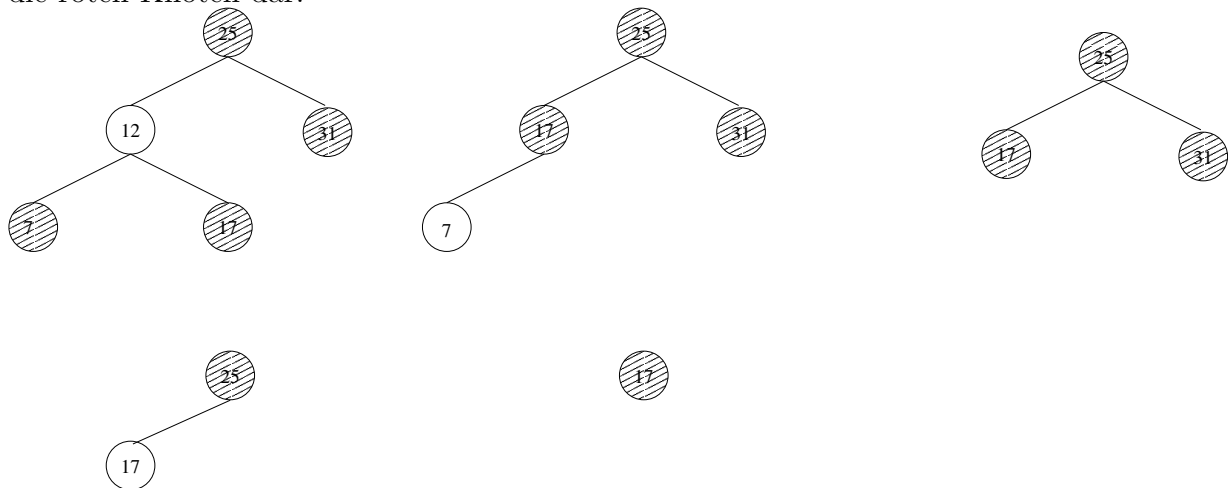
**Aufgabe 3 b)**

**3 Punkte**

Es sei der RS-Baum aus Aufgabe 3 gegeben, geben Sie die RS-Bäume an, die beim sukzessiven Löschen der Schlüssel 19, 12, 7, 31, 25, 17 entstehen.

**Lösung**

Die schraffierten Knoten stellen die schwarzen Knoten und die nicht schraffierten die roten Knoten dar.



(0.5 Punkte je korrekter Operation)

---

**Aufgabe 4**

**7 Punkte**

---

Seien  $i$  und  $i'$  in einem Intervall-Baum  $T$  gespeicherte Intervalle. Das Intervall  $i'$  ist das Intervall mit dem kleinsten Anfangspunkt aus der Menge der  $i$  überlappenden Intervalle in  $T$ . Geben Sie einen effizienten Algorithmus in Pseudo-Code an, der zu einem gegebenen Intervallbaum  $T$  und einem Intervall  $i$ , das Intervall  $i'$  bestimmt. Falls die Menge der überlappenden Intervalle leer ist, soll der Algorithmus *NIL* als Rückgabewert liefern.

### Lösung

Sei  $x$  ein Knoten in einem Intervallbaum und  $i$  ein Intervall. Die Funktion Interval-Successor liefert zu  $x$  und  $i$  denjenigen Knoten des Teilbaums, dessen Wurzel von  $x$  gebildet wird, mit einem  $i$  überlappenden Intervall und einer minimalen unteren Schranke. Voraussetzung ist,  $max[x] \geq low[i]$  und  $low[int[x]] \leq high[i]$ .

Interval-Successor( $x, i$ )

```

if  $x = nil$ 
  then return  $x$ 
 $y \leftarrow x$ 
while  $left[y] \neq nil$  and  $max[left[y]] \geq low[i]$ 
  do  $y \leftarrow left[y]$ 
if  $int[y]$  overlaps  $i$ 
  then return  $y$ 
  else return Interval-Successor( $right[y], i$ )

```

(4 Punkte)

Mit Interval-Successor kann der gesuchte Algorithmus angegeben werden.

ML-Interval-Search( $T, i$ )

```

 $x \leftarrow$  Interval-Search ( $T, i$ )
if  $x \neq nil$ 
  then
    if  $left[x] \neq nil$  and  $max[left[x]] \geq low[i]$ 
      then  $y \leftarrow$  Interval-Successor ( $left[x], i$ )
      return  $int[y]$ 
    else return  $int[x]$ 
  else return nil

```

(3 Punkte)