Lehrstuhl für Informatik I
— Informationsstrukturen und
wissensbasierte Systeme —
Universität Würzburg

Prof. Dr. H. Noltemeier

Würzburg, 24. Juni 1998

# Übungen zur Vorlesung
# Praktische Informatik II (SS98)

## Musterlösung zum 4. Übungsblatt

Lösung zu Aufgabe 13: (Natürlicher Prioritätssuchbaum)

In der Vorlesung wurde ein Verfahren vorgestellt, das zur Speicherung von Punkten im $\mathbb{R}^2$ kein Skelett wie der statische *PST*, sondern den natürlichen Baum bzgl. der $x$-Koordinaten verwendet (sogenannter *natürlicher PST*, *nPST*). Dabei wurde folgende Vorgehensweise verwendet:

1) Start mit Wurzel $\frac{+\infty}{(+\infty,+\infty)}$ (sie repräsentiert die leere Punktmenge $S$).
2) Einfügen eines Punktes $P = (P.x, P.y)$:
   a) Suche gemäß $P.x$ einen Platz für $P$ (also ein Blatt)
   b) Erweitere den $nPST$ um zwei Söhne zu diesem Blatt
   c) Insert($P$, Wurzel($nPST$))

   Dabei gilt folgende Invariante ($*$): Jedes Nichtblatt in $nPST$ hat 2 Söhne.

(a) **Zu zeigen:**

Nach obigem Schema kann für jeden einzufügenden Punkt Schritt c) korrekt ausgeführt werden.

**Beweis:**

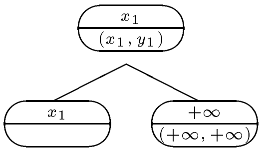
Es wird davon ausgegangen, daß die Punkte bezüglich ihrer $x$-Koordinaten disjunkt sind.

Wir zeigen nun mittels vollständiger Induktion, daß nach dem Einfügen einer Punktmenge $S := \{P_1, \ldots, P_n\} \subset \mathbb{R}^2$ in den $nPST$ folgende Aussagen gelten:

i) Für jeden Punkt $(x, y) \in S$ existiert ein Blatt im $nPST$ mit Splitwert $x$.
ii) Für alle Blätter $p$ mit Splitwert $p.x < +\infty$ gilt: Auf dem Pfad von der Wurzel des $nPST$ zum Blatt $p$ gibt es einen Knoten $p' \neq p$ mit Splitwert $p'.x = p.x$.
iii) Der $nPST$ ist ein binärer Suchbaum bezüglich der Splitwerte.
iv) Insert($P_n$, Wurzel($nPST$)) konnte korrekt ausgeführt werden.

Es folgt der Induktionsbeweis über $|S| = n$:

(IA) Wir fügen in den leeren $nPST$ den Punkt $P_1 = (x_1, y_1)$ ein und erhalten folgenden Baum:



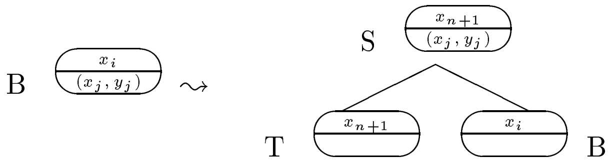

Man sieht leicht, daß für diesen $nPST$ die Aussagen i) bis iv) gelten.

(IV) Die Aussagen i) bis iv) gelten für ein $n \in \mathbb{N}$.

(IS) $n \to n + 1$

Wir wollen nun den Punkt $P_{n+1} = (x_{n+1}, y_{n+1})$ einfügen. Die Suche gemäß $x_{n+1}$ endet in einem Blatt $B$ mit Splitwert $x_i$. Eventuell enthält $B$ einen Punkt $(x_j, y_j)$.

Wir erweitern nun den $nPST$ wie folgt:




Da $B$ ein Blatt bleibt, für $x_{n+1}$ ein Blatt mit Splitwert $x_{n+1}$ hinzukommt und (IV) i) gilt, so gilt auch i) für $n + 1$.

Man sieht leicht, daß auch Aussage ii) für das neue Blatt $T$ korrekt ist. Auch für $B$ bleibt ii) gültig, da es nach (IV) ii) bereits einen solchen Pfad gab und dieser durch die Kante von $S$ nach $B$ erweitert werden kann. Für alle anderen Blätter gilt ii) weiterhin nach (IV).

$\Rightarrow$ Die Aussage ii) gilt für $n + 1$.

Betrachten wir nun den Splitwert $x_i$ von $B$, so muß $x_i > x_{n+1}$ gelten, denn $x_i \neq x_{n+1}$, da die $x$-Werte aller Punkte disjunkt sind. Wäre $x_i < x_{n+1}$, dann wäre nach (IV) ii) auf dem Suchpfad nach $B$ bereits ein Knoten mit dem Splitwert $x_i$ durchlaufen worden und wir hätten $B$ nie erreichen dürfen, da nach (IV) iii) der $nPST$ ein Suchbaum war.

$\Rightarrow$ $x_i > x_{n+1}$ und der Teilbaum aus den Knoten $S$, $T$ und $B$ ist ein binärer Suchbaum.

$\Rightarrow$ Die Aussage iii) gilt für $n + 1$.

D.h. vor dem Schritt $\textsc{Insert}(P_n, \text{Wurzel}(nPST))$ gelten die Aussagen i) bis iii) bereits. Aussage i) garantiert, daß beim *trickle-down* für jeden Punkt $(x_i, y_i) \in S$ ein Blatt mit Splitwert $x_i$ existiert. Umgekehrt gilt, da die $x$-Werte disjunkt sind, daß es für jedes Blatt mit Splitwert $x_i$ genau einen Punkt $(x_i, y_i)$ gibt. Aussage iii) gewährleistet, daß alle Punkte sich auf ihr zugehöriges Blatt „zubewegen", falls sie beim *trickle-down* verdrängt werden.

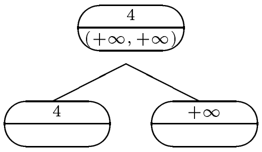
Somit gilt, daß alle Punkte gemäß ihrer Priorität ausweichen können und es gilt Aussage iv) auch für $n + 1$.

Aus der Aussage iv) ergibt sich sofort die Behauptung der Aufgabenstellung. $\square$

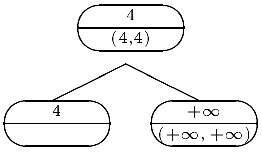
(b) Nun wird die Punktmenge $S := \{(4,4),(5,2),(3,5),(2,3),(7,1)\}$ in einen zunächst leeren *nPST* eingefügt.

INSERT(4,4):

- Erweiterung des Baumes:

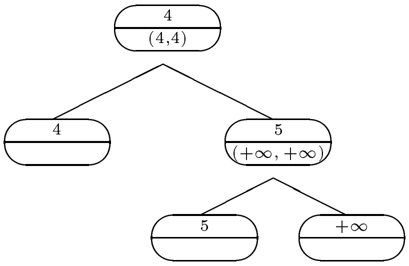



- Einfügen mit *trickle-down*:

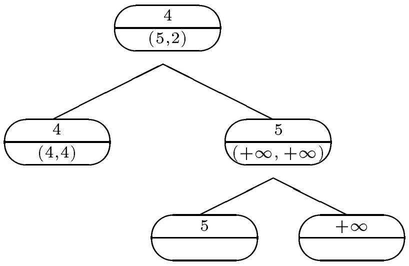



INSERT(5,2):

- Erweiterung des Baumes:

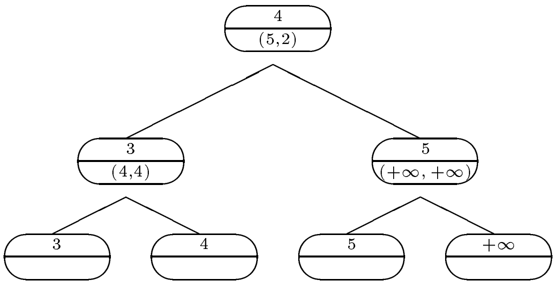



- Einfügen mit *trickle-down*:

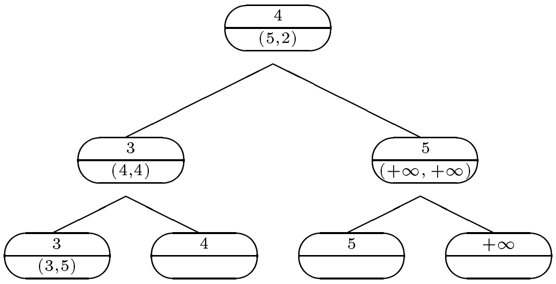



INSERT(3,5):

- Erweiterung des Baumes:

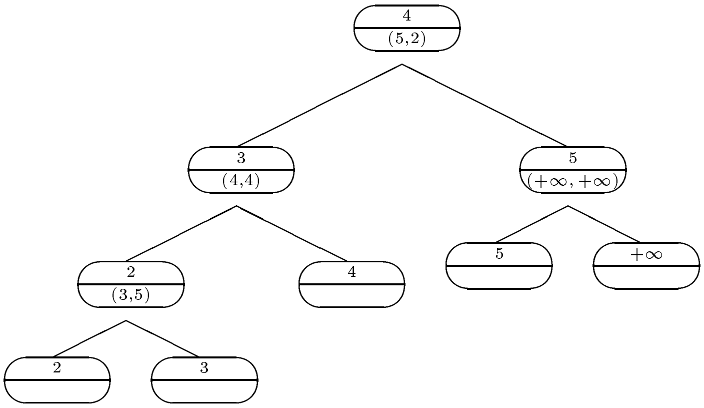


- Einfügen mit *trickle-down*:

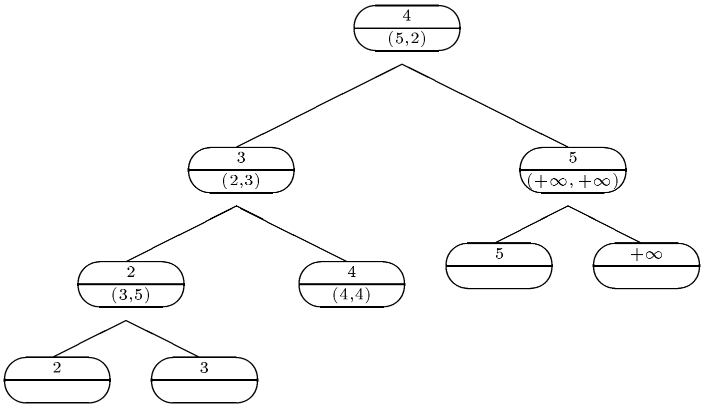



Insert(2,3):

- Erweiterung des Baumes:

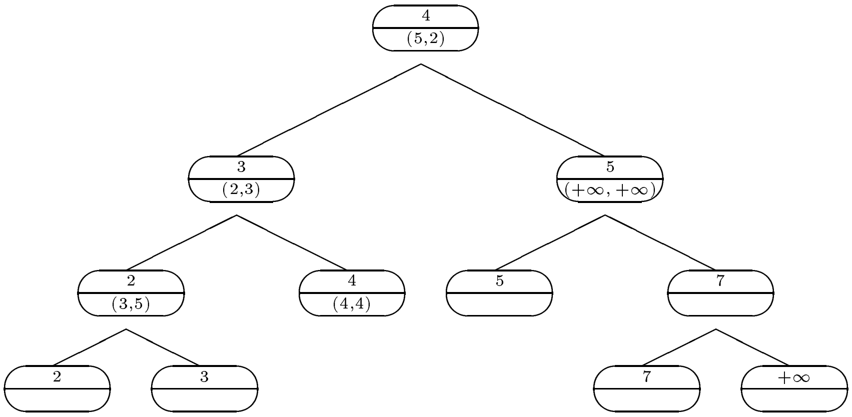



- Einfügen mit *trickle-down*:

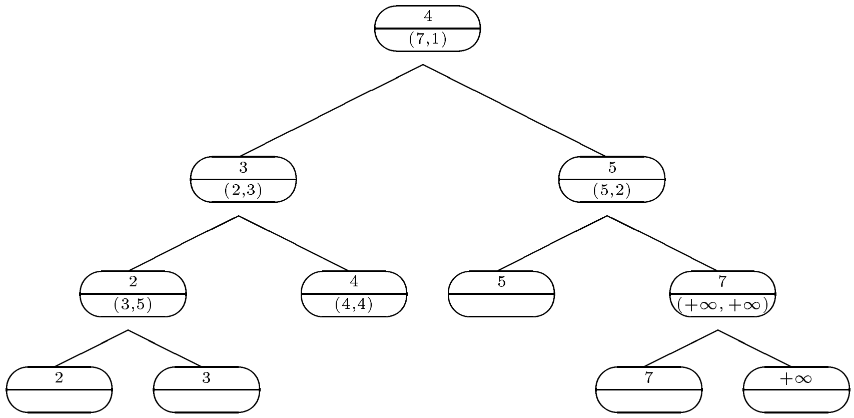



Insert(7,1):

- Erweiterung des Baumes:



- Einfügen mit *trickle-down*:




(c) **Zu zeigen:**

Bei Gleichverteilung der Punkte in $S$ mit $S \subset \{1, \ldots, N\} \times \{1, \ldots, N\}$ und $|S| \ll N$ ist der Erwartungswert für die Höhe des *nPST* aus $\mathcal{O}(\log |S|)$.

**Beweis:**

Die Höhe des *nPST* ist eine Eigenschaft seiner Struktur. Da diese Struktur allein von den $x$-Koordinaten der einzufügenden Punkte und deren Reihenfolge abhängt (siehe Algorithmus), hängt auch die erwartete Höhe des *nPST* nur von der Verteilung der $x$-Koordinaten ab.

Sei $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \{1, \ldots, N\} \times \{1, \ldots, N\}$ eine zufällig ausgewählte Punktmenge mit $n \ll N$. Da die Punkte in $\{1, \ldots, N\} \times \{1, \ldots, N\}$ gleichverteilt sind, treten unter der Bedingung, daß die Punktmenge $S$ gezogen wird, alle Permutationen von $S$ mit derselben Wahrscheinlichkeit auf. Genauer:

$$
\begin{aligned}
& \texttt{Prob}[(x_1, y_1), \ldots, (x_n, y_n) \text{ werden gezogen}] \\
= \; & \texttt{Prob}[(x_{\pi(1)}, y_{\pi(1)}), \ldots, (x_{\pi(n)}, y_{\pi(n)}) \text{ werden gezogen}]
\end{aligned}
$$

für jede Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, wobei $\texttt{Prob}[e]$ die Wahrscheinlichkeit des Ereignisses $e$ bezeichnet.

Sei $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ die Reihenfolge, in der die Punkte gezogen wurden. Die Punkte werden nun in dieser Reihenfolge in einen zu Beginn leeren *nPST* eingefügt. Dann gilt nach der $i$-ten Einfügung (für $i = 0, \ldots, n$) die folgende

**Behauptung:** Streicht man aus dem *nPST* sämtliche Blätter und ersetzt jeden inneren Knoten durch einen Knoten, der nur noch den Splitwert enthält, so erhält man genau den natürlichen binären Suchbaum, der nach Einfügen der Werte $x_1, x_2, \ldots, x_i$ in dieser Reihenfolge aus dem ursprünglich leeren Baum entsteht.

**Beweis:** Vollständige Induktion über $i$.

**(IA)** $i = 0$

Der *nPST* besteht aus einem einzigen Blatt mit Splitwert $+\infty$ und Punkt $(+\infty, +\infty)$. Wird dieses gestrichen, so erhält man einen leeren *nPST*. Die Änderung des Knotentyps liefert einen leeren natürlichen binären Suchbaum.

**(IV)** Sei die Behauptung für die ersten $i$ Einfügungen bereits gezeigt.

**(IS)** $i \to i+1$

Sei $T_{nPST}$ der $nPST$, der durch Einfügen der ersten $i$ Punkte entstanden ist, und $T_{nST}$ der natürliche binäre Suchbaum, der durch Einfügen der Werte $x_1, \ldots, x_i$ entstanden ist. Für diese beiden Bäume gilt nach (IV) die Behauptung. Um den Punkt $(x_{i+1}, y_{i+1})$ in $T_{nPST}$ einzufügen wird für diesen Punkt gemäß $x_{i+1}$ und der angetroffenen Splitwerte ein Blatt gesucht. Dieses Blatt, dessen Splitwert mit $s$ bezeichnet sei, wird durch einen Knoten mit Splitwert $x_{i+1}$ ersetzt, der als linken Sohn ein Blatt mit Splitwert $x_{i+1}$ und als rechten Sohn ein Blatt mit Splitwert $s$ erhält.

Die Suche nach dem Blatt $p$ für $(x_{i+1}, y_{i+1})$ in $T_{nPST}$ verläuft auf einem Pfad, der der Suche nach einem Platz für $x_{i+1}$ in $T_{nST}$ entspricht; genauer ist der Suchpfad in $T_{nST}$ das Bild des Suchpfades in $T_{nPST}$ unter der angegebenen Baumtransformation, da $x_{i+1}$ mit den jeweiligen Splitwerten der besuchten Knoten verglichen wird.

Also ergibt die Transformation von $T_{nPST}$ nach dem Einfügen von $(x_{i+1}, y_{i+1})$ genau den natürlichen binären Suchbaum, der aus $T_{nST}$ durch Einfügen von $x_{i+1}$ entsteht, da die eigentliche Einfügeoperation weder die Struktur noch die Splitwerte des $nPST$ verändert, und das Blatt $p$ zu einem inneren Knoten von $T_{nPST}$ und dessen Bild somit ein neuer Knoten in $T_{nST}$ wurde.

Damit ist die Zwischenbehauptung gezeigt.

Hieraus läßt sich schließen, daß die Höhe eines $nPST$, der durch Einfügen der Punkte $(x_1, y_1), \ldots, (x_n, y_n)$ entsteht, um genau 1 größer ist als die Höhe des natürlichen binären Suchbaumes, der durch Einfügen der Werte $x_1, \ldots, x_n$ entsteht.

Da die erwartete Höhe eines natürlichen binären Suchbaumes, der durch Einfügen einer Wertemenge $S$ in zufälliger Reihenfolge entsteht (wobei jede Reihenfolge gleich wahrscheinlich ist), bekanntlich aus $\mathcal{O}(log|S|)$ ist[1], ist der Erwartungswert für die Höhe des $nPST$ unter den gegebenen Bedingungen ebenfalls aus $\mathcal{O}(log|S|)$. □

## Lösung zu Aufgabe 14: (Löschen im natürlichen Prioritätsbaum)

(a) Es war eine DELETE-Prozedur zu entwerfen, die einen Punkt aus dem $nPST$ löscht, wobei die erwartete logarithmische Höhe des $nPST$ und die Invariante $(*)$ aus Aufgabe 13 gewahrt bleiben sollen.

Sei $A = (A.x, A.y)$ für einen Punkt $A$. Ein Knoten $p$ habe die Felder $p.sw$ für den Splitwert, $p.punkt$ für den gespeicherten Punkt und $p.rightson$ und $p.leftson$ für seine Söhne. Weiter gelte $undef > n$ für jedes $n \in \mathbb{N}$

[1] Beweise dieser Eigenschaft finden sich z.B. in T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990, H. Noltemeier, *Informatik III*, Kapitel 3.2.6, S. 49/50 und Th. Ottmann und P. Widmayer, *Algorithmen und Datenstrukturen*, BI-Wissenschaftsverlag, 1993

**Algorithmus 1** DeletePoint(Ecke $p$, Punkt $A$)

```
if p.punkt.y = undef then
  return false
else if A = p.punkt then
  if p.leftson.punkt.y > p.rightson.punkt.y then
    p.punkt ← p.leftson.punkt
    DeletePoint(p.leftson, p.leftson.punkt)
    return true
  else
    p.punkt ← p.rightson.punkt
    DeletePoint(p.rightson, p.rightson.punkt)
    return true
else if A.x > p.punkt.x then
  return DeletePoint(p.rightson, A)
else
  return DeletePoint(p.leftson, A)
```

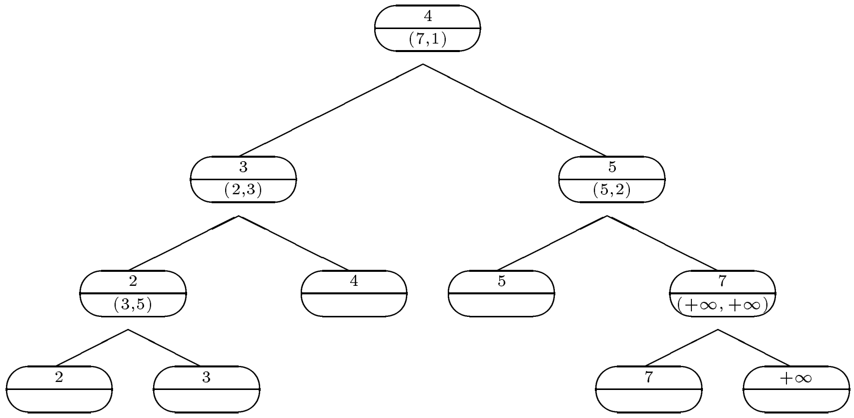
**Algorithmus 2** Delete(Ecke $p$, Punkt $A$)

```
ret ← DeletePoint(p, A)
if not ret then
  return false {Punkt nicht vorhanden}
else
  Lösche das Blatt mit Splitwert A.x
  Suche den inneren Baumknoten b mit Trennwert A.x
  if b.leftson ≠ undef then
    Bestimme rechtesten Baumknoten a des linken Sohnes von b {a ist
    symmetrischer Vorgänger von b; a hat keinen rechten Sohn, also geht
    folgendes:}
    Linke a aus dem Baum
  else
    Bestimme linkesten Baumknoten a des rechten Sohnes von b
    Linke a aus dem Baum
  b.sw ← a.sw
  if a.punkt.x ≠ undef then
    Insert(b, a.punkt)
```

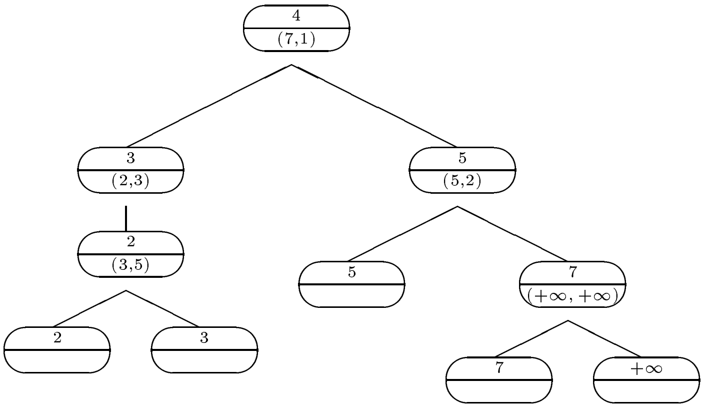
Aus einem $nPST$ wird ein Punkt $A$ gelöscht, indem Algorithmus 2 (DELETE) auf die Wurzel $p$ des $nPST$ und den Punkt $A$ angewendet wird.

(b) Nun wird mittels der in (a) angegebenen Prozedur den Punkt $(4,4)$ aus dem in Aufgabe 13 (b) entwickelten $nPST$ gelöscht.

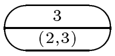
- Anwendung des Algorithmus DELETEPOINT liefert folgenden Baum:

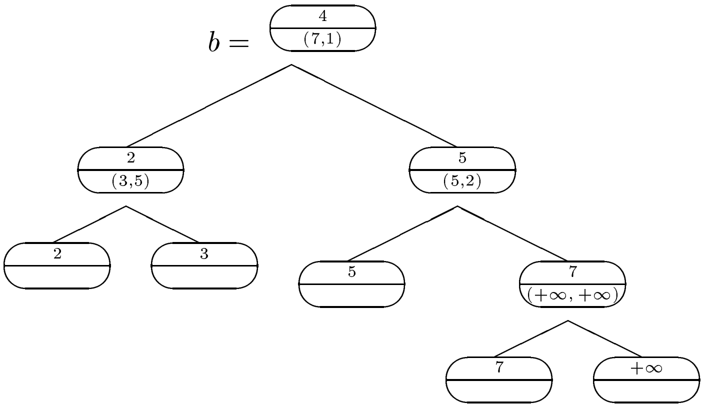



- Algorithmus DELETE, Zeile 5, liefert



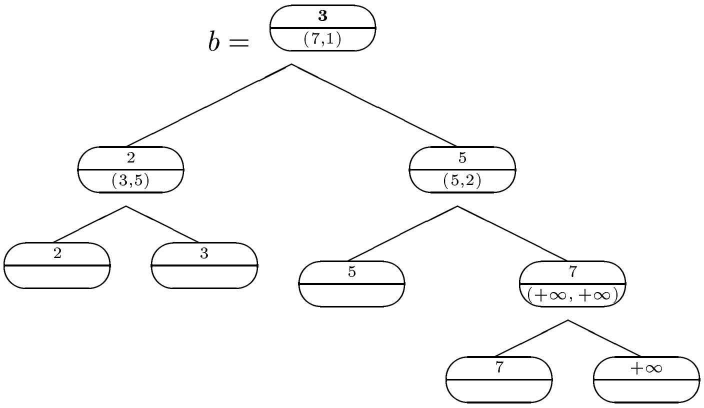

- Algorithmus DELETE, Zeilen 8 und 9, liefern

$a =$ 

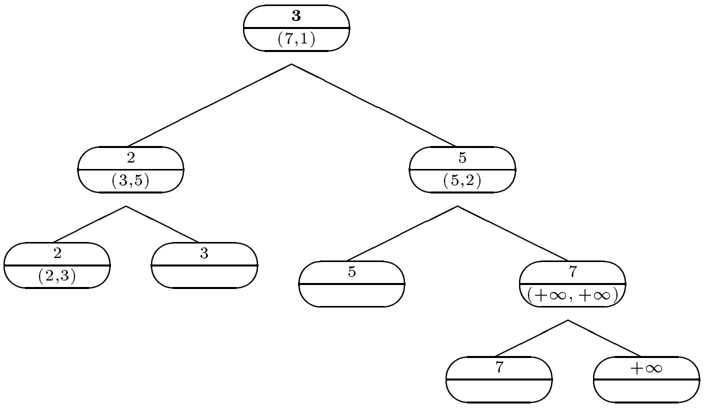

$b =$ 

- Algorithmus Delete, Zeile 13, liefert
  $a.punkt = (2, 3)$

$b =$




- Algorithmus Delete, Zeilen 15, liefert




Lösung zu Aufgabe 16: (Zur Flächen- und Konturbestimmung von Mengen achsenparalleler Rechtecke)

Gegeben sei eine Menge $S = \{[a_i, b_i] \mid a_i < b_i \text{ für } i = 1, \ldots, n\}$ von $n$ Intervallen der reellen Zahlengeraden und ein Intervall $[a, b]$ mit $a < b$.

Gesucht ist ein Algorithmus, der die $m$ maximalen Teilintervalle $[c_j, d_j]$ mit $c_j < d_j$ und $1 \leq j \leq m$ von $[a, b]$ bestimmt, deren Inneres nicht von $S$ geschnitten wird (d.h. für die $(c_j, d_j) \cap [a_i, b_i] = \varnothing$ für $i = 1, \ldots, n$ gilt).

**Behauptung:**

Der folgenden Sweepline-Algorithmus (Algorithmus 3) löst die Aufgabe korrekt und vollständig.

**Beweis:**

1. **Korrektheit** Der Algorithmus ist korrekt, d.h. die zurückgelieferten Intervalle haben alle die geforderte Eigenschaft.

   Wir zeigen hierzu per Induktion nach der Anzahl $n$ der Schleifendurchgänge, daß $V$ stets die Anzahl der an der zuletzt inspizierten Stelle geöffneten Intervalle enthält.

   (IA) $n = 0$
   Offenbar wird an der ersten Inspektionsstelle genau ein Intervall geöffnet.

**Algorithmus 3** Maximale ungeschnittene Teilintervalle

1: Sammle alle Intervalle $[a_i, b_i] \in S$ mit $a < b_i$ und $a_i < b$ in der Menge $S'$ auf.
2: Sortiere die $a_i$, $b_i$ schwach monoton wachsend. Diese sortierte Menge von maximal $2n$ reellen Zahlen bildet die Ereignisstruktur $Q$
3: $q \leftarrow Q.\mathit{first}()$
4: **if** $q > a$ **then**
5:     $R \leftarrow \{[a, q]\}$
6: **else**
7:     $R \leftarrow \varnothing$
8: $V \leftarrow 1$ {Die Statusstruktur wird durch die Variable $V$ repräsentiert, die stets die Zahl der Intervalle enthält, die den zuletzt inspizierten Punkt enthalten.}
9: **while** $Q$ ist nicht leer **do**
10:     $q \leftarrow Q.\mathit{next}(q)$
11:     **if** $q$ ist Endpunkt eines Intervalles **then**
12:         $V \leftarrow V - 1$
13:     **else**
14:         $V \leftarrow V + 1$
15:     **if** $V = 0$ **then**
16:         $p \leftarrow Q.\mathit{next}(q)$
17:         **if** $p = \mathit{nil}$ **then**
18:             $p \leftarrow b$
19:         $p \leftarrow \min\{p, b\}$
20:         **if** $q < p$ **then**
21:             $R \leftarrow R \cup \{[q, p]\}$
22: Gib die Menge $R$ der Ergebnisintervalle zurück.

(IV) Sei die Behauptung für $n \in \mathbb{N}$ gezeigt.

(IS) $n \to n + 1$

Ist die Inspektionsstelle $q$ ein Endpunkt, so gilt: Nach (IV) ist $V \geq 1$ und $V$ enthält die Anzahl der geöffneten Intervalle. Da $q$ ein Endpunkt ist, wird nun ein Intervall geschlossen und $V \leftarrow V - 1$ gesetzt, und die Behauptung gilt für $n + 1$.

Ist die Inspektionsstelle $q$ ein Anfangspunkt, so wird analog $V \leftarrow V + 1$ gesetzt, und die Behauptung gilt für $n + 1$.

Dies gilt insbesondere, da $Q$ die Anfangs- und Endpunkte aufsteigend sortiert enthält.

Hieraus folgt die Korrektheit des Algorithmus:

(a) Offenbar ist das in den Schritten 4–7 gemeldete Intervall ein Intervall mit der gewünschten Eigenschaft.

(b) In der Schleife (Schritte 9–21) gilt: Ein Intervall wird nur dann gemeldet, wenn an der aktuellen Stelle alle Intervalle geschlossen werden und entweder der nächste Anfnagspunkt echt größer ist oder, falls kein weiterer Punkt in $Q$ vorhanden ist, $b$ echt größer ist. Also ist auch dieses Intervall von der gewünschten Eigenschaft. Insbesondere kann das Intervall nicht größer gewählt werden.

**2. Vollständigkeit** Der Algorithmus ist vollständig, d.h. es werden alle Intervalle mit der geforderten Eigenschaft gefunden.

(a) Die in Schritt 1 entfernten Intervalle ändern offenbar nichts am Gesamtergebnis.

(b) Es gibt drei Möglichkeiten für die Lage eines Intervalles $[c_j, d_j]$

(i) $[a, p_1] = [c_i, d_i]$ mit $p_1 := \min_{i=1,\dots,n} \{a_i\}$ falls $p_1 > a$. Diese Intervall wird in den Schritten 4–5 gefunden.

(ii) $[c_i, d_i]$ mit $c_i \geq p_1 := \min_{i=1,\dots,n} \{a_i\}$, $d_i \leq \min\{\max_{i=1,\dots,n} \{b_i\}, b\}$. Liegt dieser Fall vor, so ist wegen der Maximalität an der betreffenden Stelle ein Endpunkt eines Intervalles $[a_i, b_i]$ zu finden, d.h. $V = 0$, und der nächste Anfangspunkt muß echt größer als dieser Endpunkt sein. Dieses Intervall wird in der Schleife gefunden.

(iii) $[c_i, d_i]$ mit $c_i = \max_{i=1,\dots,n} \{b_i\}$, $d_i = b$. Auch dieses Intervall wird in der Schleife gefunden.

Somit ist der Algorithmus korrekt und vollständig. □

**Behauptung:**

Der Aufwand im *worst-case* hinsichtlich Rechenzeit ist aus $\mathcal{O}(n \log n)$, der für den Speicherbedarf ist aus $\mathcal{O}(n)$.

**Beweis:**

**1. Rechenzeit** :

- Schritt 1 kann in $\mathcal{O}(n)$ durchegführt werden.
- Schritt 2 kann mit einem entsprechenden Sortierverfahren in Zeit $\mathcal{O}((2n)\log(2n)) = \mathcal{O}(n \log n)$ durchgeführt werden.
- Der Rechenzeitaufwand für Schritt 3 ist konstant, da wir die Elemente aus $Q$ nur durchlaufen.
- Die Schritte 4–8 laufen in konstanter Zeit.
- Die Schleife wird maximal $(2n-1)$-mal durchlaufen. In jedem Durchlauf ist der Aufwand konstant, also ist der Aufwand hierfür aus $\mathcal{O}(n)$.
- Es kann maximal $n + 1$ Intervalle geben, also liegt der Aufwand der Ergebnisausgabe auch in $\mathcal{O}(n)$.

Insgesamt wird die Laufzeit also durch den Sortierschritt dominiert und liegt damit wie behauptet in $\mathcal{O}(n \log n)$.

**2. Speicherbedarf** :

- $S$ und $S'$ können maximal $n$ Intervalle enthalten, $Q$ speichert höchstens $2n$ reelle Zahlen. Unter der Annahme, daß wir reelle Zahlen mit endlich viel Speicherplatz darstellen können, liegt der Speicherplatzbedarf hierfür in $\mathcal{O}(n)$.
- Verwendet man ein *in-situ*-Sortierverfahren, also ein Verfahren, das nur konstant viel zusätzlichen Speicherplatz benötigt, wird der Speicherplatzbedarf nur um $\mathcal{O}(n)$ vergrößert.

- Es können maximal $n+1$ Intervalle gefunden werden, nämlich dann, wenn die $[a_i, b_i]$ keine gemeinsamen Punkte haben und alle im Inneren von $[a, b]$ liegen. Also ist der Speicherplatzbedarf für $R$ auch in $\mathcal{O}(n)$

Somit liegt der Speicherplatzbedarf insgesamt in $\mathcal{O}(n)$. $\square$