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

Prof. Dr. H. Noltemeier

Würzburg, 2. Juli 1998

Übungen zur Vorlesung
Praktische Informatik II (SS98)

Musterlösung zum 4. Übungsblatt

Lösung zu Aufgabe 15: (Bestimmung extremaler Punkte)

Sei $S = \{P_i = (x_i, y_i) | i = 1, \ldots, n\}$ eine Menge von n Punkten der reellen Ebene mit $x_i \neq x_j$ für $i \neq j$, $n \geq 3$ und $x_1 < x_2$; ferner sei $\hat{y}$ eine beliebige reelle Zahl. S sei durch einen Prioritätssuchbaum PST der Höhe h_{PST} repräsentiert.

Es war ein Algorithmus zu entwerfen, der in $\mathcal{O}(h_{PST})$ Zeit den folgenden Punkt $(x_*, y_*) \in S$ ermittelt – oder feststellt, daß es keinen derartigen Punkt in S gibt – wobei

$$x_* := \min\{x_i | x_1 \leq x_i \leq x_2 \text{ und es existiert ein } (x_i, y_i) \in S \text{ mit } y_i \leq \hat{y}\}$$

und y_* so klein wie möglich, so daß $(x_*, y_*) \in S$ gilt.

Ein Knoten v des PST habe dabei folgende Felder:

$v.split$	:	Splitwert des Knotens v
$v.point$	:	In v gespeicherter Punkt $P = (v.point.x, v.point.y)$. Es gilt $v.point = nil$, falls in v kein Punkt gespeichert ist.

Behauptung:

Das in den folgenden Algorithmen 1 und 2 beschriebene Verfahren besitzt die geforderten Eigenschaften.

Algorithmus 1 $\text{MinXinRectangle}(x_1, x_2, \hat{y}, PST)$

1: $v \leftarrow$ Wurzel(PST)
2: $w \leftarrow \text{NodeWithMinXinRectangle}(x_1, x_2, \hat{y}, v)$
3: **if** $w = nil$ **then**
4: return „Es existiert kein derartiger Punkt (x_*, y_*)"
5: **else**
6: return $w.point.x$

Beweis:

Definiere $xrange(v)$ für jeden Knoten v von PST wie folgt:

(i) $xrange(v) := \{v.split\}$
für jedes Blatt v von PST.

(ii) $xrange(v) := xrange(v.leftson) \cup xrange(v.rightson)$
für jeden inneren Knoten v von PST.

Algorithmus 2 NODEWITHMINXINRECTANGLE$(x_1, x_2, \hat{y}, v)$

if ($v.point$ ist undefiniert) **or** ($v.point.y > \hat{y}$) **then**
  $w \leftarrow nil$
else
  if v ist kein Blatt **then**
    if $x_1 \leq v.split$ **then**
      $w \leftarrow$ NODEWITHMINXINRECTANGLE$(x_1, x_2, \hat{y}, v.leftson)$ {–1–}
    if ($w = nil$) **and** ($v.split < x_2$) **then**
      $w \leftarrow$ NODEWITHMINXINRECTANGLE$(x_1, x_2, \hat{y}, v.rightson)$ {–2–}
  if ($x_1 \leq v.point.x \leq x_2$) **and** (($w = nil$) **or** ($v.point.x < w.point.x$)) **then**
    $w \leftarrow v$
return w

Für jedes Element s des Universums existiert in PST ein Blatt v mit Splitwert $v.split = s$. Der Splitwert eines inneren Knotens v ist das Maximum der Blattwerte im linken Teilbaum von v. Sucht man in PST nach dem Blatt mit Splitwert s, so werden genau die Knoten v durchlaufen, für die $s \in xrange(v)$ ist, da der PST bzgl. der Splitwerte einen binären Suchbaum bildet.

Weiter gilt

$$xrange(v.leftson) = \{x \in xrange(v) \mid x \leq v.split\},$$

$$xrange(v.rightson) = \{x \in xrange(v) \mid x > v.split\}, \text{ und}$$

$$v.point = (x, y) \Rightarrow x \in xrange(v),$$

falls die entsprechenden Knoten existieren.

Wir definieren weiter folgende Knotenmengen:

$$\begin{aligned} P &:= \{v \mid xrange(v) \cap [x_1, x_2] \neq \varnothing \text{ und } xrange(v) \not\subseteq [x_1, x_2]\} \\ C &:= \{v \mid xrange(v) \subseteq [x_1, x_2]\} \\ C_{max} &:= \{v \in C \mid father(v) \notin C\} \end{aligned}$$

Offenbar gilt $w \in P \cup C$ gdw. $xrange(w) \cap [x_1, x_2] \neq \varnothing$.

Schließlich sei mit $R := \{(x, y) \mid x_1 \leq x \leq x_2 \text{ und } y \leq \hat{y}\}$ der Halbstreifen bezeichnet, den es zu durchsuchen gilt.

Nun beweisen wir die Behautungen zur Laufzeit und Korrektheit des Algorithmus.

Laufzeit des Algorithmus:

Im folgenden verstehen wir unter „der Knoten v wird untersucht (oder: betrachtet)“, daß v als vierter Parameter bei einem Aufruf der Prozedur NODEWITHMINXINRECTANGLE auftritt.

(i) Während der Ausführung des Algorithmus werden nur Knoten v aus $P \cup C$ untersucht.
Beweis: Per Vollständiger Induktion nach der Rekursionstiefe der Aufrufe von NODEWITHMINXINRECTANGLE.

(IA) Die Wurzel w des PST wird beim ersten Aufruf von NodeWithMinXinRectangle betrachtet. Wegen $x_1, x_2 \in \{x_1, x_2, \dots, x_n\} = \mathit{xrange}(w)$ ist $\mathit{xrange}(w) \cap [x_1, x_2] \neq \varnothing$ und somit $w \in P \cup C$.

(IV) Der Knoten $v \in P \cup C$ des PST werde betrachtet.

(IS) Wegen $v \in P \cup C$ ist $\mathit{xrange}(v) \cap [x_1, x_2] \neq \varnothing$. Falls $x_1 \leq v.\mathit{split}$, so wird $v.\mathit{leftson}$ durch den Aufruf (1) untersucht; außerdem gilt $\mathit{xrange}(v.\mathit{leftson}) \cap [x_1, x_2] \overset{x_1 \leq v.\mathit{split}}{\neq} \varnothing$, also $v.\mathit{leftson} \in P \cup C$.
Wird in Aufruf (2) $v.\mathit{rightson}$ betrachtet, so gilt $v.\mathit{split} < x_2$ und damit dann auch $v.\mathit{rightson} \in P \cup C$.

Damit ist die Behauptung (i) gezeigt.

(ii) Die Knoten in P können in Zeit $\mathcal{O}(h_{PST})$ untersucht werden.
Beweis: Es ist $|P| \in \mathcal{O}(h_{PST})$, wie leicht zu erkennen ist, wenn man die Knoten $v_1, \dots, v_m$ auf einem Niveau des PST von links nach rechts durchläuft: Zunächst sind alle $x \in \mathit{xrange}(v_i)$ kleiner als x_1, bis man ein v_j mit $x_1 \in \mathit{xrange}(v_j)$ erreicht. Es folgen evtl. Knoten v_k mit $\mathit{xrange}(v_k) \subseteq [x_1, x_2]$, $k \geq j$. Schließlich wird ein Knoten x_l erreicht mit $x_2 \in \mathit{xrange}(v_l)$, und für alle weiteren Knoten v_m, $m \geq l$, gilt wieder $\mathit{xrange}(v_m) \cap [x_1, x_2] = \varnothing$. Also existieren auf jedem Niveau des PST höchstens zwei Knoten $v \in P$, und damit ist die Abschätzung gezeigt.

(iii) Sei $v \in C$. Dann erzeugt bei Betrachtung des Knotens v höchstens einer der beiden rekursiven Aufrufe (1) und (2) in Algorithmus 2 einen zeitlichen Aufwand größer als $\mathcal{O}(1)$.
Beweis: Die Funktion NodeWithMinXinRectangle werde für den Knoten $v \in C$ aufgerufen.

Sei $v.\mathit{leftson}.\mathit{point} \in R$.
Dann liefert der rekursive Selbstaufruf (1) einen gültigen Knoten $w \neq \mathit{nil}$ zurück und Aufruf (2) wird nicht ausgeführt.

Sei $v.\mathit{leftson}.\mathit{point}$ undefiniert.
Dann bricht der Aufruf (1) sofort ab und liefert *nil* zurück, läuft also in Zeit $\mathcal{O}(1)$.

Sei $v.\mathit{leftson}.\mathit{point}$ definiert, aber $\notin R$.
Wegen $v \in C$ gilt $\mathit{xrange}(v.\mathit{leftson}) \subseteq \mathit{xrange}(v) \subseteq [x_1, x_2]$, und da $v.\mathit{leftson}.\mathit{point} \notin R$ folgt $v.\mathit{leftson}.\mathit{point}.y > \hat{y}$. Daher bricht der Aufruf (1) nach Zeit $\mathcal{O}(1)$ ab.

Also liegt der Aufwand für die Untersuchung eines Knotens $v \in C$ in $\mathcal{O}(h_{PST})$. Dies gilt insbesondere auch für $v \in C_{max} \subseteq C$.

(iv) Für höchstens einen Knoten $v \in C_{max}$ wird während seiner Untersuchung die Prozedur NodeWithMinXinRectangle rekursiv aufgerufen.
Beweis: Sei v ein Knoten aus C_{max}, bei dessen Untersuchung die Prozedur NodeWithMinXinRectangle rekursiv aufgerufen wird. Dann gilt $v.\mathit{point} \neq \mathit{nil}$ und $v.\mathit{point} \in R$. Sei $w \in C_{max}$ ein Knoten rechts von v in PST und z der am weitesten von der Wurzel entfernte gemeinsame Vorfahre von v und w. Da v im linken und w im rechten Teilbaum von z liegt, liefert der Aufruf (1) bei der Untersuchung von z einen gültigen Knoten, und (2) bleibt unausgeführt, d.h. w wird nicht untersucht.

Weiter kann es echt links von v in PST keinen weiteren Knoten $u \in C_{max}$ mehr geben, bei dessen Unteruchung rekursive Aufrufe gestartet werden, da v echt rechts von u läge und dann nach dem oben gesagten nicht betrachtet würde.

Also werden für höchstens einen Knoten $v \in C_{max}$ bei dessen Untersuchung Söhne von v rekursiv betrachtet.

(v) Die Anzahl der untersuchten Knoten aus C ist aus $\mathcal{O}(h_{PST})$.
Beweis: Es gilt $|C_{max}| \leq 2 \cdot |P|$, da der Vater von $v \in C_{max}$ in P liegt, also folgt mit (ii), daß $|C_{max}| \in \mathcal{O}(h_{PST})$. Aus (iii) und (iv) folgt, daß aus $C \backslash C_{max}$ nicht mehr als $\mathcal{O}(h_{PST})$ Knoten betrachtet werden. Insgesamt werden also höchstens $\mathcal{O}(h_{PST})$ Knoten aus C untersucht.

Also werden bei der Ausführung des Algorithmus $\mathcal{O}(h_{PST})$ Knoten betrachtet, woraus direkt die zu zeigende Laufzeitabschätzung folgt.

Korrektheit des Algorithmus:

Sei v ein Knoten des PST mit $v.point = (x_*, y_*)$. Sei $w_0, w_1, \ldots, w_k$ der Pfad von der Wurzel w_0 des PST zu $v = w_k$.

Dann gilt $w_i.point.y \leq y_* \leq \hat{y}$ für alle $i = 0, \ldots, k$, da der PST bzgl. der y-Koordinaten einen Heap bildet.

Weiter zeigen wir, daß der Knoten v durch den Algorithmus untersucht wird, indem wir dies induktiv für alle Knoten w_i, $i = 0, \ldots, k$, zeigen.

Die Wurzel w_0 wird offenbar als erster Knoten untersucht.

Angenommen, der Knoten w_i wird betrachtet. Dann unterscheiden wir die folgenden beiden Fälle:

Fall 1: $w_{i+1} = w_i.leftson$
Dann wird auf jeden Fall auch w_{i+1} untersucht, wie ein Blick auf Algorithmus 2 zeigt.

Fall 2: $w_{i+1} = w_i.rightson$
Dann ist $w_i.split < x_*$. Nach Definition von x_* muß daher für alle Knoten z im linken Teilbaum von w_i gelten, daß $z.point.x < x_1$ oder $z.point.y > \hat{y}$ ist. Also verläuft die Untersuchung von $w_i.leftson$ erfolglos, und $w_{i+1} = w.rightson$ wird untersucht.

Also wird v gefunden.

Existiert kein derartiger Knoten v in PST mit $v.point = (x_*, y_*)$, so bricht die Prozedur NodeWithMinXinRectangle nach $\mathcal{O}(h_{PST})$ Schritten mit dem Rückgabewert *nil* ab, und MinXinRectangle gibt korrekt über die Nichtexistenz eines solchen Knotens Auskunft. □

Lösung zu Aufgabe 15: (alternative Lösung)

Gesucht ist ein Algorithmus, der in $\mathcal{O}(h_{PST})$ den folgenden Punkt $(x_*, y_*) \in S$ ermittelt – oder feststellt, daß es keinen derartigen Punkt in S gibt. Hierbei sei

$x_* := min\{x_i \mid x_1 \leq x_i \leq x_2$ und es existiert ein $(x_i, y_i) \in S$ mit $y_i \leq \hat{y}\}$.

Lösung:

Folgender Algorithmus SuchePunkt dient zur Suche des Punktes p mit minimalem $p.x$, unter der Bedingung, daß $x_1 \leq p.x$ und $p.y \leq \hat{y}$ ist. Als Rückgabewert liefert die Routine einen Punkt p_2 und eine boolesche Variable *erfolg*. Hierbei bedeutet *erfolg==true*, daß ein solcher Punkt p_2, wie oben gefordert, in dem Baum mit Wurzel p gefunden wurde, sonst wäre *erfolg==false* und $p_2 == nil$. In der Routine Extremal wird abschließend noch geprüft, ob für den gefundenen Punkt auch $p.x \leq x_2$ gilt.

Definition der Routinen:

Algorithmus 3 (p_2,*erfolg*) SuchePunkt(p)

```
Punkt p';
boolean erfolg;
if p.left ≠ nil and p.left.y ≤ ŷ and p.sp ≥ x1 then
   (p', erfolg) = SuchePunkt (p.left)
if erfolg==false and p.right ≠ nil and p.right.y ≤ ŷ then
   (p', erfolg) = SuchePunkt (p.right)
if erfolg==true then
   if p.x ≤ p'.x and p.x ≥ x1 then
      return (p, true)
   else
      return (p', true)
else
   if p.x ≥ x1 then
      return (p, true)
   else
      return (nil, false);
```

Algorithmus 4 (*Punkt*) Extremal(Wurzel)

```
Punkt best;
boolean erfolg;
(best, erfolg) = SuchePunkt(Wurzel);
if erfolg==true and best.x ≤ x2 then
   return best;
else
   return nil;
```

Korrektheit:

Der Algorithmus durchsucht beginnend bei der Wurzel für jeden Knoten zuerst den linken Teilbaum. Falls dort ein geeigneter Punkt gefunden wurde, muß der rechte Teilbaum nicht mehr durchsucht werden, da alle Punkte des rechten Teilbaums größere x-Koordinaten besitzen, als die Punkte des linken Teilbaums (Aufteilung bzgl. des Splitwertes!).

Wird kein passender Punkt gefunden, so durchläuft der Algorithmus ebenfalls den rechten Teilbaum.

Anschließend wird noch geprüft, ob der Punkt im akt. Knoten eine Lösung mit kleinerer x-Koordinate darstellt.

Der linke Teilbaum wird nur durchsucht, falls der Splitwert des aktuellen Knotens noch größer gleich x_1 ist, da sonst alle Punkte des Teilbaums zu kleine x-Werte haben müssen.

Außerdem müssen bei einem Knoten, dessen y-Koordinate zu groß ist, keine Nachfolger mehr betrachtet werden (Heapeigenschaft!), d.h. weder der rechte noch der linke Teilbaum muß durchsucht werden.

Damit werden auf jeden Fall alle relevanten Knoten geprüft, wodurch sichergestellt ist, daß auch der Knoten, der die Lösung mit kleinster x-Koordinate darstellt, gefunden wird.

Begründung der Laufzeit:

1. Schritt: Zu Beginn durchläuft der Algorithmus SuchePunkt den Baum von der Wurzel in Richtung des Blattes mit dem Splitwert x_1, durch entsprechende rekursive Aufrufe des linken bzw. rechten Teilbaums.

Nun ist der Algorithmus bei einem Knoten k angekommen, der keine geeigneten Nachfolger mehr besitzt, d.h. k ist entweder Blatt, oder die Söhne von k besitzen y-Werte größer als $\hat{y}$.

Ist k ein korrekter Knoten (man beachte, daß in dem Algorithmus noch nicht bzgl. x_2 geprüft wird!), so gibt die Routine diesen zurück und setzt *erfolg=true*, wodurch alle Routinen ohne weitere rekursive Aufrufe enden. Daraus folgt, daß der Aufwand in $\mathcal{O}(h_{PST})$ liegt.

Ist k kein korrekter Knoten, so läuft der Algorithmus entlang des Pfades in Richtung Wurzel zurück (durch Beenden der einzelnen Routinen) und prüft die rechten Söhne der Knoten auf diesem Pfad bzgl. ihrer y-Werte. Die x-Werte dieser Söhne müssen größer als x_1 sein, da sie rechts vom Suchpfad zu x_1 liegen.

Falls ein passender Punkt vorhanden ist, so wird die Routine, bevor sie endet, SuchePunkt mit einem rechten Sohn r aufrufen, der *erfolg=true* setzt.

2. Schritt: Von diesem Knoten r aus wird nun die Suche in dem Baum fortgesetzt. Nun liegt der Fall vor, daß sich alle Nachfolger des Knotens r rechts vom Knoten k befinden, d.h. daß sie Splitwerte größer x_1 besitzen.

Somit liefert jeder weitere Aufruf der Routine SuchePunkt mit einem rechten oder auch linken Sohn, *erfolg==true* zurück. Hieraus ergibt sich, daß niemals mehr der rechte und linke Teilbaum eines Knotens durchsucht werden müssen, denn das Betreten eines linken Teilbaum, liefert immer *erfolg==true* zurück und somit wird der rechte Teilbaum (*if*-Abfrage) nicht mehr betreten.

D.h. der Algorithmus läuft vom Knoten r maximal bis zu einem Blatt. Der Aufwand hierfür liegt wieder in $\mathcal{O}(h_{PST})$.

Die einzelnen Routinen enden schließlich alle mit *erfolg==true*, wodurch weitere Aufrufe von SuchePunkt unterbunden werden.

Die Routine SUCHEPUNKT wird somit maximal $2 \cdot \mathcal{O}(h_{PST})$ oft aufgerufen. Die Routine selbst kann in konstanter Zeit abgearbeitet werden, ebenso wie die Routine EXTREMAL, die nur einmal aufgerufen wird. Somit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(h_{PST})$.