Übungsblatt 8

Julius Auer, Alexa Schlegel

Aufgabe 1 (Vorverarbeitungszeit für Bereichsbäume):

Der Algorithmus zur Konstruktion - der $T(n)$ Zeit benötigt - ist straight-forward:

(1) Lege Knoten an und konstruiere sekundäre Struktur - im 2D-Fall ist die sekundäre Struktur ein „einfacher“ binärer Baum mit n Knoten, der in $O(n \cdot \log n)$ Zeit konstruiert werden kann.
(2) Finde Median der Punkte und teile deren Menge in zwei möglichst gleich große Teile. Mit geeignetem Median-Algorithmus (z.B. BFPRT) ist das in $O(n)$ möglich.
(3) Konstruiere die beiden resultierenden Teilbäume rekursiv in $O(T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil))$.

Der Einfachheit halber sei n im Folgenden eine 2er-Potenz. Es gilt:

$$T(1) = 1$$

Somit ergibt sich für die Laufzeit:

$$\begin{aligned}
T(n) &= \overbrace{2 \cdot T\left(\frac{n}{2}\right)}^{(3)} + \overbrace{n \cdot \log n}^{(1)} + \overbrace{n}^{(2)} \\
&= 2 \cdot T\left(\frac{n}{2}\right) + n \cdot (\log n + 1) \\
&= 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2} \cdot \left(\log \frac{n}{2} + 1\right)\right) + n \cdot (\log n + 1) \\
&= 2 \cdot \left(2 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4} \cdot \left(\log \frac{n}{4} + 1\right)\right) + \frac{n}{2} \cdot \left(\log \frac{n}{2} + 1\right)\right) + n \cdot (\log n + 1) \\
&\dots \\
&= 2^k \cdot T\left(\frac{n}{2^k}\right) + n \cdot \sum_{i=0}^{k-1} \left(\log \frac{n}{2^i} + 1\right)
\end{aligned}$$

Die Primärstruktur hat eine Höhe von maximal $\log n$:

$$\begin{aligned}
T(n) &= 2^{\log n} \cdot T\left(\frac{n}{2^{\log n}}\right) + n \cdot \sum_{i=0}^{\log n - 1} \left(\log \frac{n}{2^i} + 1\right) \\
&= n + n \cdot \left(\log \frac{n}{n-1} + \log n - 1\right) \\
&= n \cdot \log \frac{n}{n-1} + n \cdot \log n \\
&\in n \cdot O(1) + n \cdot \log n \\
&\in O(n \cdot \log n)
\end{aligned}$$

□

Aufgabe 2 (dynamische Segmentsbäume):

Um die Ausgeglichenheit eines Segmentbaums zu gewährleisten, sind unterschiedliche Ansätze denkbar (habe vergessen ob/welcher Ansatz in der Vorlesung vorgeschlagen wurde). Z.B. könnte man eine andere ausgeglichene, binäre Baumstruktur Zugrunde legen (z.B. einen AVL-Baum) und die Rebalancierungs-Operationen (ehm... ich meine natürlich „ Ausgleichungs-Operationen“) derart anpassen, dass die Knotenlisten insbesondere der inneren Knoten valide bleiben.

Ein einfacherer Ansatz - der sich zumindest bzgl. der asymptotischen Laufzeit nicht von eben genanntem unterscheidet - wäre, die Anzahl Knoten des „Skeletts“' des Segmentbaums zu verdoppeln, sobald ein Intervall eingefügt werden soll, dessen Grenzen außerhalb des bislang größten Intervalls des Baums liegen. Es wird hierzu mit einem Skelett initialisiert, dessen Anzahl an Elementarintervallen eine 2er-Potenz ist. Sobald ein Intervall eingefügt werden soll, bei dem zumindest eine der Grenzen in einem ∞-Intervall liegt, wird der Baum zur linken/rechten Seite hin verdoppelt (wenn eine Grenze im linken/rechten ∞-Intervall liegt) und eine neue Wurzel erzeugt. Der bestehende Baum muss hierbei nicht angepasst werden. Da nicht gefragt (?) wird die Initialisierung des Baum-Skeletts hier keiner genaueren Betrachtung unterzogen.

Es bleibt, das Einfügen/Streichen in einen bestehenden Baum anzugeben. Hierfür seien $i = (l, r)$ ein Intervall das eingefügt/gelöscht werden soll, $root$ die Wurzel des Baumes, $w(v)$ der Vergleichswert in einem Knoten v, $leaf(v)$ die Eigenschaft eines Knotens v ein Blatt zu sein, $left(v)/right(v)$ das linke/rechte Kind eines Knoten v und $I(v)$ die Knotenliste für den Knoten v. Der Einfachheit halber seien die Knotenlisten als Sets implementiert, die keine Duplikate erlauben.

Zum Einfügen wird zunächst der passende Pfad verfolgt, solange das einzufügende Intervall vollständig links oder rechts des betrachteten Knoten liegt. Anschließend wird der Baum einmal mit der linken Grenze des Intervalls durchlaufen wobei den Knotenlisten aller rechten Teilbäume das Intervall hinzugefügt wird, und einmal mit der rechten Grenze wobei den Knotenlisten aller linken Teilbäume das Intervall hinzugefügt wird. Löschen geschiet analog:

Algorithm 1 INSERT(i=(l,r))

$v \leftarrow root$
while $l \leq w(v)$ **and** $r \leq w(v)$ **or** $l > w(v)$ **and** $r > w(v)$ **do**
  if $isLeaf(v)$ **then**
    $I(v) \leftarrow i$
    return
  end if
  if $l \leq w(v)$ **and** $r \leq w(v)$ **then**
    $v := left(v)$
  else
    $v := right(v)$
  end if
end while
$v_l := v$
$v_r := v$
while not $isLeaf(v_l)$ **do**
  $I(right(v_l)) \leftarrow i$
  if $l \leq w(v_l)$ **then**
    $v_l := left(v_l)$
  else
    $v_l := right(v_l)$
  end if
end while
$I(v_l) \leftarrow i$
while not $isLeaf(v_r)$ **do**
  $I(left(v_r)) \leftarrow i$
  if $r \leq w(v_r)$ **then**
    $v_r := left(v_r)$
  else
    $v_r := right(v_r)$
  end if
end while
$I(v_r) \leftarrow i$

Algorithm 2 DELETE(i=(l,r))

$v_l := root$
$v_r := root$
while not $isLeaf(v_l)$ **do**
  $I(right(v_l)) := I(right(v_l)) \setminus i$
  if $l \leq w(v_l)$ **then**
    $v_l := left(v_l)$
  else
    $v_l := right(v_l)$
  end if
end while
$I(v_l) := I(v_l) \setminus i$
while not $isLeaf(v_r)$ **do**
  $I(left(v_r)) := I(left(v_r)) \setminus i$
  if $l \leq w(v_r)$ **then**
    $v_r := left(v_r)$
  else
    $v_r := right(v_r)$
  end if
end while
$I(v_r) := I(v_r) \setminus i$

Aufgabe 3 (Punkt-Rechteck-Anfragen):

Gegeben ist eine Menge R von n Rechtecken. Jedes Rechteck r besteht aus einem x- und einem y-Intervall. Wie in der Aufgabenstellung vorgeschlagen, wird als Primärstruktur ein Segmentbaum S auf Grundlage der x-Intervalle der Rechtecke aufgebaut. Ein Knoten in S wird, wie in der Vorlesung beschrieben, mit den Intervallen markiert, zusäzlich mit den zugehörigen Rechtecken. Somit wird pro Knoten k die Menge der potentiell in Frage kommenden Rechtecke R_k gespeichert (Knotenliste).

Pro Knoten k wird eine Sekundärstruktur aufgebaut. Diese ist wiederum ein Segmentbaum, welcher auf Grundlage der y-Intervalle der Recheckte R_k aufgebaut wird. Jeder Knoten wird wieder mit Intervallen und zugehörigen Rechtecken markiert.

Für einen Anfragepunkt $p = (x, y) \in \mathbb{R}^2$ wird nun folgendermaßen vorgegangen:

(1) Suche in der Primärstruktur nach x. Sammele auf dem Weg alle Knotenlisten ein.
(2) Für jede gefundene Knotenliste: suche in der Sekundärstruktur nach y und vereinige alle Knotenlisten (Liste von Rechtecken), die die Suche durchläuft.
(3) Vereinige alle Listen von Rechtecken, dies ist die Lösung.

Vorverarbeitungszeit Primärstruktur P Das Universum U_x min $|U_x| = N$ liegt sortiert vor. Der binäre Suchbaum für U_x kann in $O(N)$ aufgebaut werden. Das Einfügen der n Intervalle benötigt $O(n \cdot \log N)$ Zeit. Das Aufbauen der Primärstruktur benötigt also insgesamt

$$O(n \cdot \log N + N)$$

.

Vorverarbeitungszeit Sekundärstruktur S Stellt sich die Frage, wie lange eine Knotenliste ist. In der VL wurde $O(\log N)$ angegeben, da in jeder Ebene des Segmentbaumes jedes Intervall nur zu höchstens zwei Knotenlisten gehören kann. Das zugehörige Universum U_y hat nun nur noch maximal $2 \cdot \log N$ Elemente. Der zugehörige Binärbaum wird also in $O(\log N)$ aufgebaut. Das Einfügen der neuen Intervalle benötigt $\log N \cdot \log(\log N)$ Zeit. Das Aufbauen der Sekundärstruktur benötig damit insgesamt

$$O(\log N \cdot \log(\log N) + \log N)$$

Das kann man bestimmt irgendwie vereinfach oder besser abschätzen, evtl. ist das

$$O(\log^3 N + \log N)$$

.

Vorverarbeitungszeit insgesamt Für jeden Knoten in P, wird eine Sekundärstruktur aufgebaut. In einem vollständigen Baum ist die Anzahl der Blätter gleich der Anzahl der inneren Knoten minus 1. D.h in unserem Baum P gibt es $2N + 1$ Blätter (nach Definition des Segmentbaumes), damit insgesamt $4N + 1$ Knoten, also $O(N)$. Insgesamt ergibt sich:

$$O(n \cdot \log N + N + N \cdot (\log N \cdot \log(\log N) + \log(N))$$

Speicherbedarf Hier ist nur der Platzbedarf für die zusätzliche Sekundärstruktur interessant.

Anfragezeit Suche in der Primärstruktur in $O(\log N)$. Für jeden Knoten auf dem Suchpfad muss nocheinmal in der Sekundärstruktur gesucht werden, also $O(\log N \cdot \log(\log N)) + k)$. k ist dabei die Ausgabe der Liste der Rechtecke.