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. Dass

$$T(1) = 1$$

ist klar. Somit ergibt sich für die Laufzeit:

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

Die Primärstruktur hat natürlich 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))

```
1: v ← root
2: while l ≤ w(v) and r ≤ w(v) or l > w(v) and r > w(v) do
3:    if isLeaf(v) then
4:       I(v) ← i
5:       return
6:    end if
7:    if l ≤ w(v) and r ≤ w(v) then
8:       v := left(v)
9:    else
10:      v := right(v)
11:   end if
12: end while
13: v_l := v
14: v_r := v
15: while not isLeaf(v_l) do
16:   I(right(v_l)) ← i
17:   if l ≤ w(v_l) then
18:      v_l := left(v_l)
19:   else
20:      v_l := right(v_l)
21:   end if
22: end while
23: I(v_l) ← i
24: while not isLeaf(v_r) do
25:   I(left(v_r)) ← i
26:   if r ≤ w(v_r) then
27:      v_r := left(v_r)
28:   else
29:      v_r := right(v_r)
30:   end if
31: end while
32: I(v_r) ← i
```

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

```
1: v_l := root
2: v_r := root
3: while not isLeaf(v_l) do
4:    I(right(v_l)) := I(right(v_l)) \ i
5:    if l ≤ w(v_l) then
6:       v_l := left(v_l)
7:    else
8:       v_l := right(v_l)
9:    end if
10: end while
11: I(v_l) := I(v_l) \ i
12: while not isLeaf(v_r) do
13:   I(left(v_r)) := I(left(v_r)) \ i
14:   if l ≤ w(v_r) then
15:      v_r := left(v_r)
16:   else
17:      v_r := right(v_r)
18:   end if
19: end while
20: I(v_r) := I(v_r) \ i
```

Aufgabe 3 (Punkt-Rechteck-Anfragen):