5.1.9. Realisierung der Knotenlisten.

→ durch Binäre Balancierte Suchbäume. (Keine Gitterbäume!).

Zb. rot-schwarz Bäume, BB[α]-Bäume, AVL-Bäume, ...

Dann gilt für die Knotenlisten der Länge n:

- Platz: $O(n)$
- Einfügen: $f(n) = O(\log n)$
- Streichen: $g(n) = O(\log n)$
- Suchen: $h(n) = O(\log n)$.

5.1.10 Satz: Sei S eine Menge von n horizontalen Liniensegmenten in der Ebene mit x-Koord. aus $\{1..N\}$.

Dann gilt:

a). Ein Segmentbaum für S braucht Platz $O(N + n \log N)$.

b). Einfügen / Streichen eines Segments kostet: $O(\log n \log N)$

c) Suchen nach allen von einem vertikalen Suchsegment geschnittenen Segmenten kostet $O(\log n \cdot \log N + m)$

($\log n$: Suchen in Knotenliste; $\log N$: Pfad; m: Ausgabe.)

5.1.11. Bemerkungen:

1) ∃ auch voll dynamische Segmentbäume. Bei denen ist der zugrundeliegende Suchbaum für die x-Koordinaten nicht statisch, sondern ein bel. Baum. (BB[α]-Baum).
   Platz: $O(n \log n)$ ← Da keine Gitterbäume mehr ⇒ Höhe: $\log n$
   Einfügen / Streichen: $O(\log^2 n)$
   Suchen: $O(\log^2 n + m)$
2) ∃ ähnliche Datenstrukturen für bel. (nicht notwendig horizontale) Segmente → Partition Trees.

5.2 Range-Tree (Bereichsabfragebaum).

5.2.1. Def: Range-Tree speichert Menge von Pkten im $\mathbb{R}^d$

Query: für jede Dimension ein Intervall.

$[x_1^{(1)}, x_2^{(1)}], [x_1^{(2)}, x_2^{(2)}], \dots, [x_1^{(d)}, x_2^{(d)}]$.

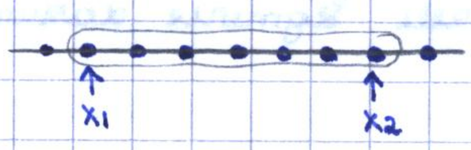
Liste alle Pkte in S auf, dessen Koordinaten jeweils in den entsprechenden Intervallen liegen.

5.2.2 Beispiele:

→ 5.2.2.1. d=1

Geg: S=Menge von Zahlen

Ges: $\{x \in S : x_1 \le x \le x_2\}$



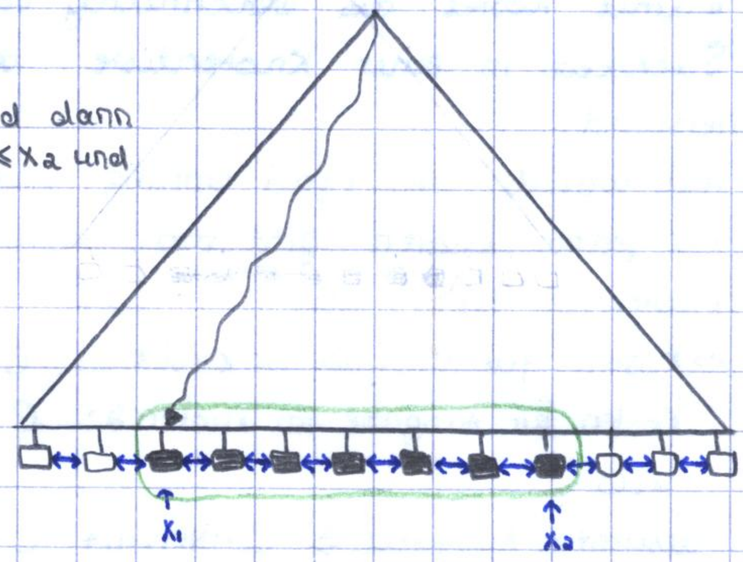
Datenstruktur: blattorientierter Suchbaum, wobei Blätter verkettet. (Hier kein Gitterbaum!)
(= 1-dim. Range-Tree).

Platz: $O(n)$ weil bin. Baum.

Zeit Query: $O(\log n + k)$ wandere Pfad nach x_1 ($\log n$) und dann wandere über die verketteten Blätter solange $x \le x_2$ und gebe sie aus (k).

$n = |S|$

Insert / Delete: $O(\log n)$



Bsp: $S = \{2, 5, 3, 8, 11, 7\}$