Kapitel V: Geometrische Datenstrukturen

5.1. Segmentbaum

5.1.1. Definitionen + Bemerkungen

Sei $S = \{s_1 .. s_n\}$ Menge von n horizontalen Liniensegmenten in der Ebene.

Wir bezeichnen mit:

$x_1(s_i)$: die x-Koord. des linken Endpkts von s_i

$x_2(s_i)$: die x-Koord. des rechten Endpkts von s_i

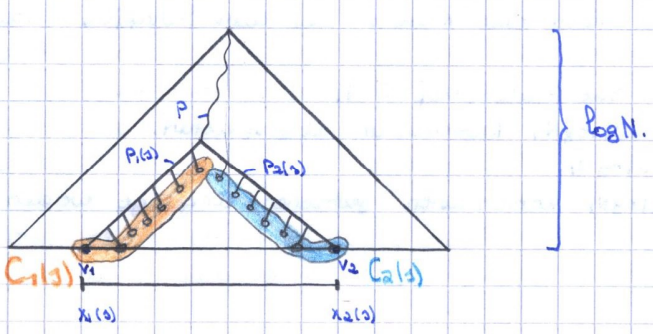
$y(s_i)$: die y-Koord von s_i.

Wir nehmen zunächst an, dass alle x-Koord. ganze Zahlen aus $\{1..N\}$ sind und die y-Koord. beliebige reelle Zahlen sind.
(→ sog. halbdynamischer Fall).

Ein Segmentbaum T zur Speicherung von S ist ein blattorientierter, binärer Suchbaum für die x-Koord. $\{1..N\}$ der Höhe $\log N$.
Jeder Knoten enthält zusätzlich eine Liste $NL(v)$ von Segmenten nach y-Koord. sortiert, die Knotenliste von v.
Bea: Beim Segmentbaum müssen die Blätter nicht verkettet sein.
Segmente von S werden wie folgt abgespeichert:
Sei $s \in S$.

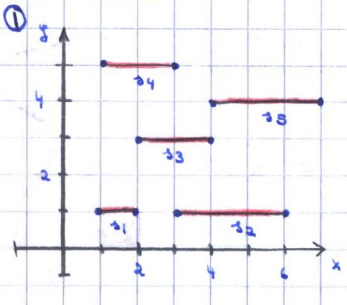
$P_1(s)$ = Suchpfad nach $x_1(s)$ \ Suchpfad nach $x_2(s)$
$P_2(s)$ = Suchpfad nach $x_2(s)$ \ Suchpfad nach $x_1(s)$
$C_1(s) = \{v : v$ rechtes Kind eines Knotens von $P_1(s)$ und $v \notin P_1(s)\} \cup \{v_1\}$
$C_2(s) = \{v : v$ linkes Kind eines Knotens von $P_2(s)$ und $v \notin P_2(s)\} \cup \{v_2\}$.
$C(s) = C_1(s) \cup C_2(s)$.

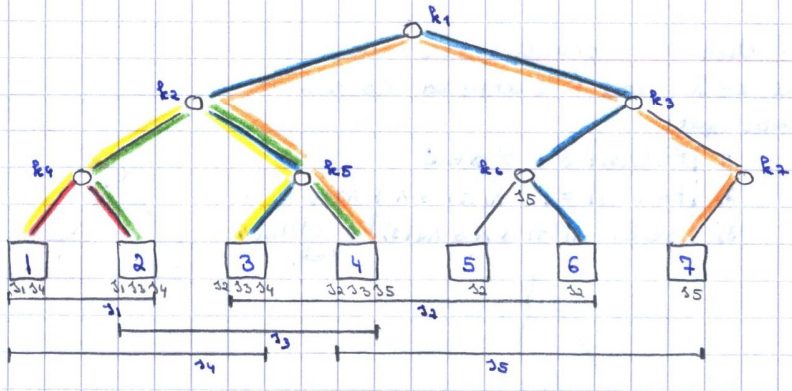
Dann wird s_i, $1 \le i \le n$ in alle Knotenlisten $NL(v)$ mit $v \in C(s)$ abgespeichert.
Skizze:



5.1.2. Beispiele:

①





$P_1(s_1)$ — $P_2(s_1)$ — $P_1(s_2)$ — $P_2(s_2)$ — $P_1(s_3)$ — $P_2(s_3)$ — $P_1(s_4)$ — $P_2(s_4)$ — $P_1(s_5)$ — $P_2(s_5)$

$C_1(s_1) = \{1\}$
$C_2(s_1) = \{2\}$
$C_1(s_2) = \{4,3\}$
$C_2(s_2) = \{5,6\}$
$C_1(s_3) = \{2\}$
$C_2(s_3) = \{3,4\}$
$C_1(s_4) = \{2,1\}$
$C_2(s_4) = \{3\}$
$C_1(s_5) = \{4\}$
$C_2(s_5) = \{k_6, 7\}$

$NL(1) = \{s_1, s_4\}$
$NL(2) = \{s_1, s_3, s_4\}$
$NL(3) = \{s_2, s_3, s_4\}$
$NL(4) = \{s_2, s_3, s_5\}$
$NL(5) = \{s_2\}$
$NL(6) = \{s_2\}$
$NL(7) = \{s_5\}$

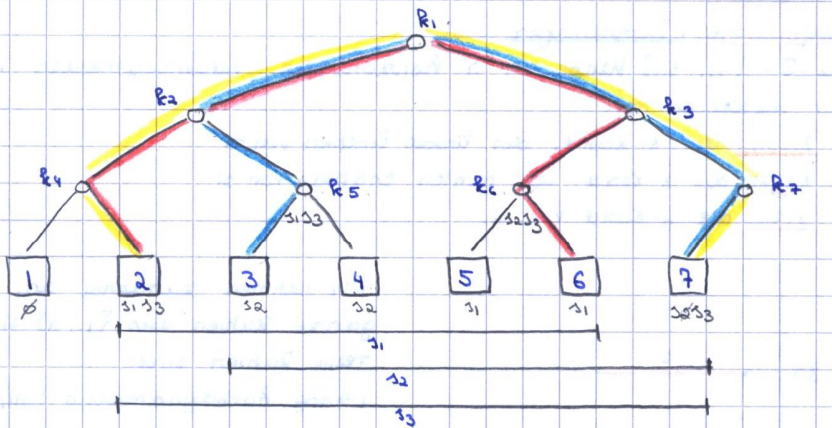
$NL(k_1 .. k_5) = \emptyset$
$NL(k_6) = \{s_5\}$
$NL(k_7) = \emptyset$

② $S = \{s_1, s_2, s_3\}$

$x_1(s_1) = 2,\ x_2(s_1) = 6$

$x_1(s_2) = 3,\ x_2(s_2) = 7$

$x_1(s_3) = 2,\ x_2(s_3) = 7.$



$P_1(s_1)$

$P_2(s_1)$ $C_1(s_1) = \{k_5, 2\}$

$P_1(s_2)$ $C_2(s_1) = \{5, 6\}$

$P_2(s_2)$ $C_1(s_2) = \{3, 4\}$

$P_1(s_3)$ $C_2(s_2) = \{k_6, 7\}$

$P_2(s_3)$ $C_1(s_3) = \{k_5, 2\}$

$C_2(s_3) = \{k_6, 7\}$

$NL(1) = \emptyset$

$NL(2) = \{s_1, s_3\}$

$NL(3) = \{s_2\}$

$NL(4) = \{s_2\}$

$NL(5) = \{s_1\}$

$NL(6) = \{s_1\}$

$NL(7) = \{s_2, s_3\}$

$NL(k_1) .. NL(k_4) = \emptyset$

$NL(k_5) = \{s_1, s_3\}$

$NL(k_6) = \{s_2, s_3\}$

$NL(k_7) = \emptyset$

5.1.3 Lemma:

✓ 1. $|C(s)| \leq 2 \cdot \log N \quad \forall s \in S.$

2) T hat Platzbedarf $O(N + n \cdot \log N)$

3) Einfügen eines Segmentes s kostet $O(\log N \cdot f(n))$, wobei $f(n)$ Kosten für Einfügen in Knotenliste der Länge n.

4) Streichen eines Segments s kostet $O(\log N \cdot g(n))$, wobei $g(n)$ Kosten für Streichen aus Knotenliste der Länge n.

Beweis:

1). Jeder Knoten $v \in C(s)$ ist Kind eines Knotens auf dem Suchpfad nach $x_1(s)$ bzw $x_2(s)$. im schlimmsten Fall: $|C_1(s)| = \log N = |C_2(s)| \Rightarrow |C(s)| \leq 2 \log N.$

2). Das Gerüst des Baums hat Platzbedarf $O(N)$ ← Großer Baum aus

1) ⇒ Jedes Segment ist in $O(\log N)$ Knotenlisten abgespeichert. denn s ist nur in den Knoten $v \in C(s)$ gespeichert $|C(s)| \leq 2 \log N \Rightarrow s$ ist in $O(\log N)$ Knoten gespeichert.

⇒ Platzbedarf: $O(N + n \cdot \log N)$. ← Ganzer Baum

3)/4). s muss aus / in $O(\log N)$ Knotenlisten gestrichen / eingefügt werden.

$f(n) \hat{=}$ Kosten für ein-einfügen im kleinen Baum } $2 \cdot \log N \cdot f(n)$ Kosten, um s in alle kleinen (erforderlichen (gesamt $\leq 2 \log N$)) Bäume einzufügen

Insgesamt in $\leq 2 \cdot \log N$ Bäume einfügen }

⇒ $O(\log N \cdot f(n))$

Für Streichen analog: $O(\log N \cdot g(n))$.

5.1.4 Suchen in Segmentbäumen:

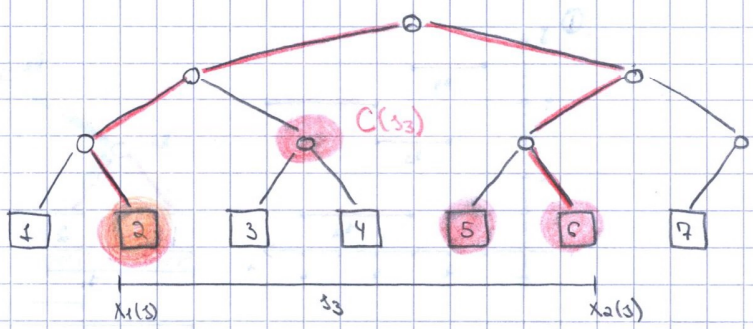
Sei $x \in \mathbb{R}$ und P Suchpfad nach x

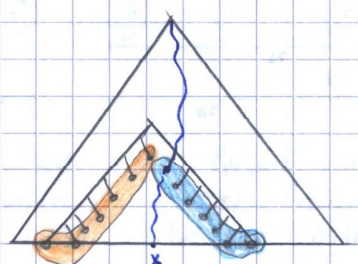
Dann gilt:

1) $|P \cap C(s)| \leq 1 \quad \forall s \in S$

2) $|P \cap C(s)| = 1 \quad x_1(s) \leq x \leq x_2(s)$

3) $\{s \in S : x_1(s) \leq x \leq x_2(s)\} = \bigcup_{v \in P} NL(v)$





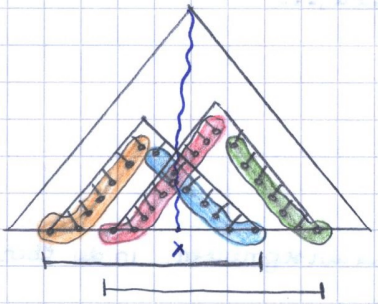
Bew: Ein Pfad ist eindeutig bestimmt

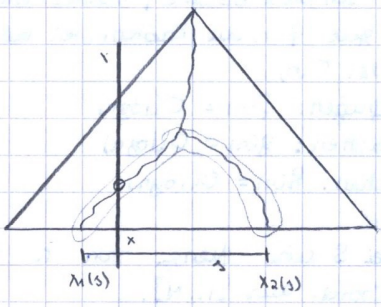
$\forall s \in S$ gilt: jedes x mit $x_1(s) \leq x \leq x_2(s)$ liegt unterhalb $C(s)$ und der Pfad von der Wurzel bis zu x geht durch genau einen Knoten von $C(s)$. ⇒ $|P \cap C(s)| = 1$

falls $x > x_2(s)$ oder $x < x_1(s)$ so schneidet der Pfad von der Wurzel bis zu x keinen Knoten aus $C(s)$

⇒ $|P \cap C(s)| = 0$

dh. die Knotenlisten auf dem Suchpfad nach x enthalten die Segmente, die von der vertikalen Geraden durch x geschnitten werden.





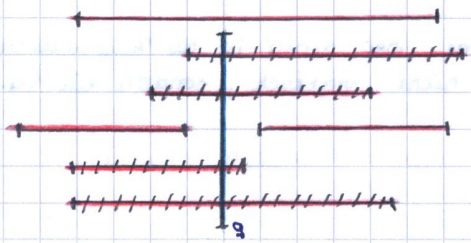
5.1.5. Laufzeit: $S(x) := \{s \in S : x_1(s) \le x \le x_2(s)\}$ kann in Zeit $O(\log N + m)$ berechnet werden, wobei $m := |S(x)|$, dh. die Größe der Ausgabe.

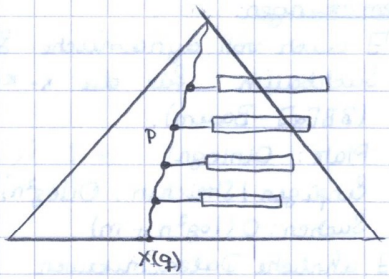
den Pfad durchlaufen ↑ in jedem Knoten in den kleineren Baum gehen und alle Knoten des kleineren Baumes (= die gesuchten Segmente) ausgeben. Für jede Ausgabe Const. Zeit Gesamt m Ausgaben ⇒ $O(\log N + m)$.

5.1.6. Problem:

Geg: das vertikale Segment q

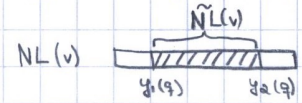
Ges: berechne $S(q) = \{s \in S : s \cap q \neq \emptyset\}$.





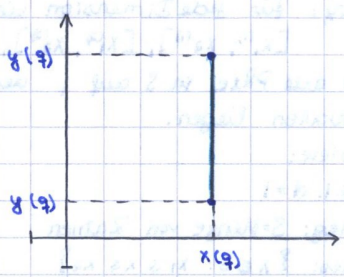
$\tilde{NL}(v) := \{s \in NL(v) : y_1(q) \le y(s) \le y_2(q)\}$

Dann ist $S(q) = \{s \in S : x_1(s) \le x(q) \le x_2(s) \text{ und } y_1(q) \le y(s) \le y_2(q)\} = \bigcup_{v \in P} \tilde{NL}(v)$



5.1.7 Algorithmus zur Berechnung des Problems:

1. P ← Suchpfad nach x(q)
2. Forall v ∈ P do
3. Lokalisiere $y_1(q)$ und $y_2(q)$ in NL(v)
4. gib alle Segmente davor aus
5. od.



5.1.8. Laufzeit:

Damit kostet die Berechnung von S(q): $O(\log N \cdot f(n) + m)$ Zeit, wobei $f(n)$ die Suchzeit in einer Knotenliste der Länge n ist.

Unterschied:

1). Man wandert den Pfad entlang und bei jedem Knoten gibt man seine Knotenliste NL(v) aus.
⇒ Kosten für Ausgabe im Knoten 1: $C \cdot |NL(v_1)|$
Kosten für Ausgabe im Knoten 2: $C \cdot |NL(v_2)|$
usw.
Gesamt m Ausgaben ⇒ $C \cdot |NL(v_1)| + C \cdot |NL(v_2)| + \ldots = C \cdot \underbrace{(|NL(v_1)| + |NL(v_2)| + \ldots)}_{= m} = C \cdot m$
weil alle ausgeg. werden!
⇒ $O(\log N + m)$
// Bem: nicht kennen!!!
// Anzahl der Elemente i.d Knotenlist i.a. verschieden
// ⇒ $\log N \cdot \underbrace{C_i \cdot |NL(v)|}_{=: f(n)}$ geht nicht!!! geht nur wenn
// Dann aber nicht mehr genau und nicht mehr output sensitiv und viel größere obere Schranke!!!

2). Man wandert den Pfad entlang und bei jedem Knoten sucht man nach entspr. Segmenten in Zeit $f(n)$ (= die größte Zeit, die es geben kann)
Höhe $\log N$, in jedem Knoten $f(n)$
⇒ $\log N \cdot f(n)$!

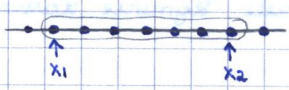
5.1.9. Realisierung der Knotenlisten.
→ durch Binäre balancierte Suchbäume. (Keine Gitterbäume!)
z.B. rot-schwarz Bäume, BB[α]-Bäume, AVL-Bäume, ...
Dann gilt für die Knotenlisten der Länge n:
- Platz: O(n)
- Einfügen: $f(n) = O(\log n)$
- Streichen: $g(n) = O(\log n)$
- Suchen: $R(n) = O(\log n)$.

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
Kosten $O(\log n \cdot \log N + m)$
(Suchen in Knotenliste · Pfad · Ausgabe.)

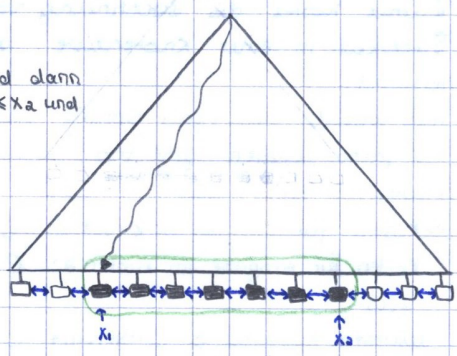
5.1.11. Bemerkungen:
1) ∃ auch voll dynamische Segmentbäume. Bei denen ist der zugrundeliegende Suchbaum für die x-Koordinaten nicht statisch, sondern ein bel. Baum. (BB[α]-Baum).
Platz: $O(n \log n)$ ← Da keine Gitterbäume mehr ⇒ Höhe: $\log n$
Einfügen / Streichen: $O(\log^2 n)$
Suchen: $O(\log^2 n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

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.
$[x_1^{(1)}, x_2^{(1)}], [x_1^{(2)}, x_2^{(2)}], \ldots, [x_1^{(d)}, x_2^{(d)}]$.
Liste alle Pkte in S auf, dessen Koordinaten jeweils in den entsprechenden Intervallen liegen.

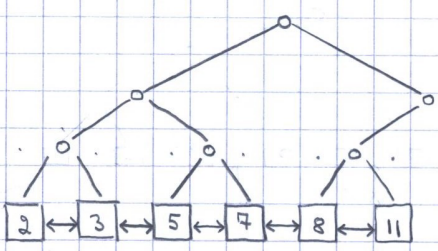
5.2.2 Beispiele:
→ 5.2.2.1. d=1
Geg: S=Menge von Zahlen
Ges: $\{x \in S : x_1 \le x \le x_2\}$



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



5.2.2.2. d = 2.

Geg: $S \subset \mathbb{R}^2$ von n Pkten.

Ges: $\{q \in S : x_1 \le q_x \le x_2 \wedge y_1 \le q_y \le y_2\}$

Datenstruktur: blattorientierter Suchbaum, wobei Blätter verkettet.

Zunächst x-Koord. aus $\{1..N\}$. ⇒ Gitterbaum.

Ein 2 dim. Range-Tree besteht aus einem blattorientierten Suchbaum T für $\{1..N\}$.
Jeder Knoten v speichert eine nach y-Koord. sortierte Folge NL(v) (Knotenliste) von Pkten.
Die Pkte aus S werden wie folgt in einen anfangs leeren Baum T eingefügt:
Sei $p \in S$ mit $p = (p_x, p_y)$, dann wird p in jede Knotenliste NL(v) auf dem Suchpfad nach p_x gespeichert.

↑ Hier auch: blattorientierte Suchbäume, aber keine Gitterbäume!

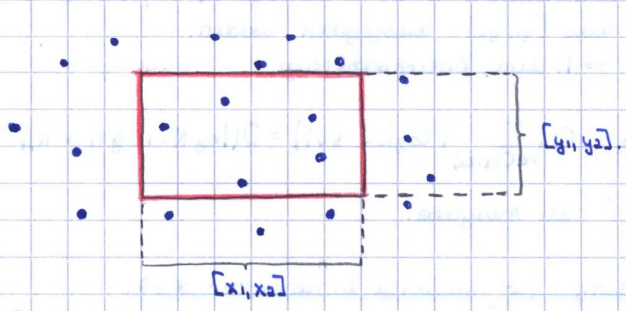
Jeder Pkt wird in log N Knoten gespeichert. $|S| = n \Rightarrow n \cdot \log N$.

Platz: $O(N + n \log N)$

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

↑ jeder NL(v) ist balancierter Baum (1. dim. Range Tree).

2-dim. Range Query:



gib alle Pkte $p \in S$ aus mit:
$x_1 \le p_x \le x_2 \wedge y_1 \le p_y \le y_2$

Vorgehen:

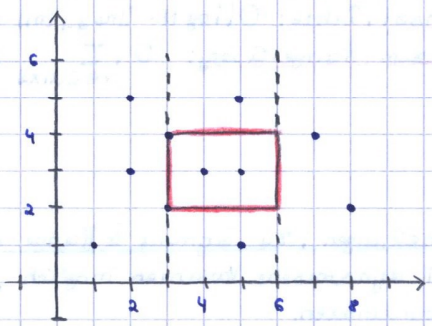
1. Schritt: Betrachte Pkte im unendlichen Streifen zw. x_1, x_2.

Beh: Die Knotenlisten NL(v) mit $v \in C(x_1, x_2)$ enthalten genau die gesuchten Pkte.

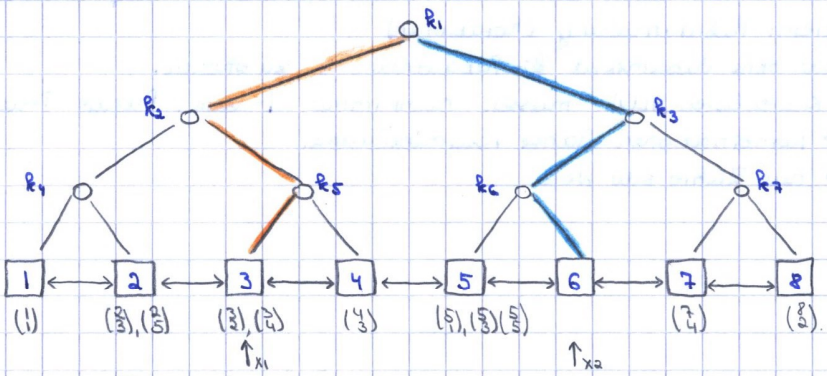
Bsp. zu Veranschaulichung:

$S = \{(1,1), (3,3), (3,5), (5,3), (5,5), (7,4), (4,3), (8,2)\}$.

$x_1 = 3$, $x_2 = 6$, $y_1 = 2$, $y_2 = 4$.



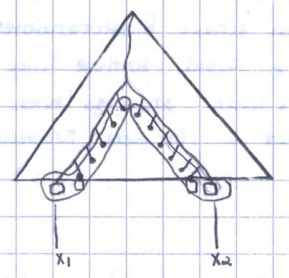
$p_1 = S$
$p_2 = \{(1,1), (3,3), (3,5), (5,3), (5,5), (7,4), (4,3)\}$
$p_3 = \{(5,1), (5,3), (5,5), (7,4), (8,2)\}$
$p_4 = \{(1,1), (3,3), (3,5)\}$
$p_5 = \{(3,3), (5,3), (4,3)\}$
$p_6 = \{(5,1), (5,3), (5,5),$
$p_7 = \{(7,4), (8,2)\}$



$C(x_1, x_2) = \{3, 4, 5, 6\}$
⇒ NL(3), NL(4), NL(5), NL(6) enthalten die gesuchten Pkte
⇒ $\{(3,3), (3,4), (4,3), (5,1), (5,3), (5,5)\}$

Begründung für die Behauptung:

Für beliebigen Knoten v enthält $NL(v)$ genau die Pkte, deren Suchpfad (nach x-Koord.) durch v geht, dh. alle Pkte im Bereich der Blätter des Unterbaumes von v.



Die Bereiche der Knoten $v \in C(x_1, x_2)$ bilden eine Partition des Intervalls $[x_1, x_2]$.

2. Schritt: Zur Beantwortung der eigentlichen Bereichsabfrage müssen wir nun die Knotenlisten $NL(v)$ $\forall v \in C(x_1, x_2)$ filtern, so dass nur Pkte mit y-Koord. aus $[y_1, y_2]$ ausgegeben werden.

→ 1. dim. Bereichsabfrage. nach y-Koord.!

⇒ Laufzeit: $O\left(\sum_{v \in C(x_1,x_2)} (\log n + K_v)\right) = O(\log N \cdot \log n + K)$, wobei $K := \sum_{v \in C(x_1,x_2)} K_v$ = Gesamtgröße der Ausgabe.

5.2.3 Verallgemeinerung für beliebige Dimensionen $d \geq 2$:

d-dim. Range-Tree besteht aus einem blattorientierten Suchbaum T für die erste Koordinate. Jeder Knoten v speichert $NL(v)$ als $(d-1)$-dim. Range-Tree. (Bzgl. Koord. 2..d).

Jeder Pkt $p \in S$ wird in allen $NL(v)$ für v auf dem Suchpfad nach 1. Koord. von p gespeichert.

Zuerst: Jede Dimension hat Koord. $\{1..N\}$ außer der letzten.

Platzbedarf: $O(\log N \cdot Platz_{d-1}(n) + N) = O(n \cdot \log^{d-1} N + N)$ ✓

Insert / Delete: $O(\log N \cdot Ins_{d-1}(n)) = O(\log^{d-1} N \cdot \log n)$

d-dim. Range-Query: $O\left(\sum_{v \in C(x_1,x_2)} (Query_{d-1}(n) + K_v)\right) = O(\log^{d-1} N \cdot \log n + K)$. ✓

5.2.4 Bemerkungen (zu Segment- & Range-Trees)

1). Voll dynamische Varianten möglich, dh. kein Gitter $\{1..N\}$ sondern beliebige, reelle Koordinaten.

Dazu: dynamischer Gerüstbaum T, dh. beim Einfügen und Löschen von Pkten (Segmenten) muss eventuell Blatt (mit entsprechender x-Koordinate) in T eingefügt oder entfernt werden. neben den Insert / Delete-Operationen auf Knotenlisten.

Problem: Rebalancierung (Rotationen)

Lösbar ohne Gesamtzeit für Rebalancierung zu erhöhen.

2). Die Knotenlisten $NL(v)$ müssen nicht immer 1-dim. Range-Trees (Suchbäume) sein.

- Kombinationen Range / Segmentbäume.
- nur Zähler oder Werte