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!

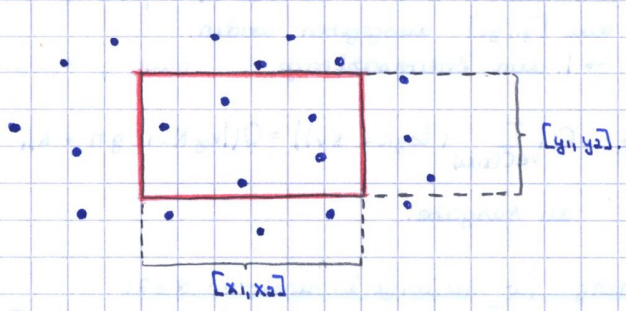
Jeder Pkt wird in log N Knoten gespeichert. $|S| = n \Rightarrow n \cdot \log N$.

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:

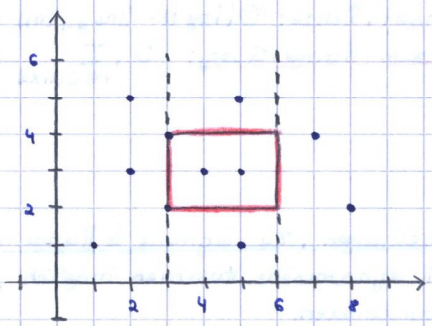
1. Schritt: Betrachte Pkte im unendlichen Streifen zw. x_1, x_2.

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

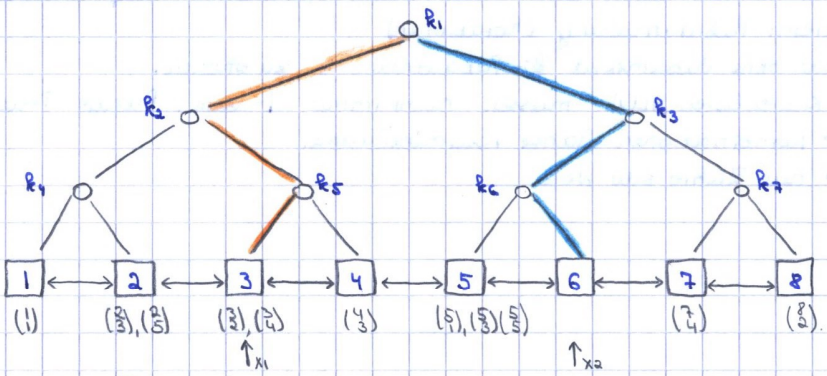
Bsp. zu Veranschaulichung:

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

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



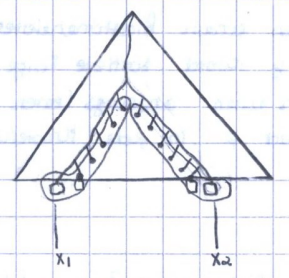
$p_1 = S$
$p_2 = \{(1,1), (3,3), (3,5), (5,3), (5,5), (7,4), (4,3)\}$
$p_3 = \{(5,1), (5,3), (5,5), (7,4), (8,2)\}$
$p_4 = \{(1,1), (3,3), (3,5)\}$
$p_5 = \{(3,3), (5,3), (4,3)\}$
$p_6 = \{(5,1), (5,3), (5,5),$
$p_7 = \{(7,4), (8,2)\}$



$C(x_1, x_2) = \{3, 4, 5, 6\}$
⇒ 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

5.3. Priority - Search - Tree:

5.3.1. Def: Sei $S \subset \mathbb{R}^2$

Ein Priority - Search - Tree (PST) T ist primärorientierter Suchbaum nach den x-Koord. der Pkte aus S.

5.3.2. Speichern der Pkte:

Ann: paarw. verschiedene x-Koordinaten.

$p \in S$ wird in einem Knoten auf Suchpfad nach p_x gespeichert und zwar so, dass T bzgl. der y-Koord. einen Maximumsheap bildet.

⇒ Auf jedem Pfad werden y-Koord. kleiner

⇒ Wurzel enthält Knoten mit max. y-Koord.

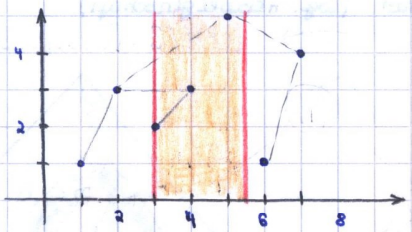
PST hat zwei Eigenschaften:
- Suchbaum nach x-Koord. der Pkte
- Heap nach y-Koord. der Pkte.

Aufbau (rekursiv):

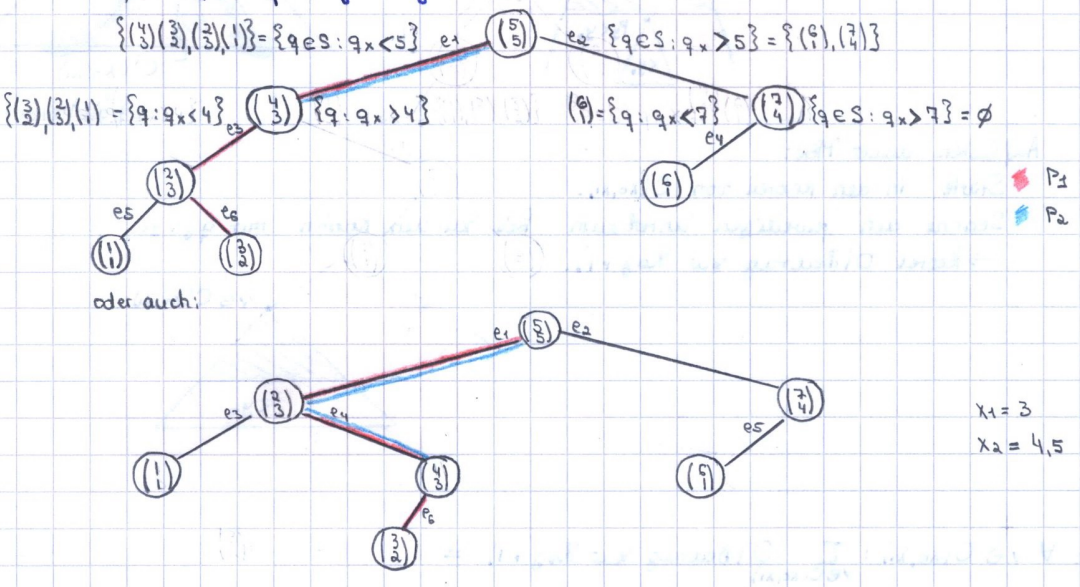
Wurzel w speichert $p \in S$ mit max. y-Koord.
- linker Unterbaum T_ℓ von w ist PST für $\{q \in S : q_x < p_x\}$
- rechter Unterbaum T_r von w ist PST für $\{q \in S : q_x > p_x\}$.

5.3.3. Beispiel:

$S = \{(1),(\frac{3}{2}),(\frac{2}{3}),(\frac{4}{3}),(\frac{6}{1}),(\frac{5}{5}),(\frac{7}{4})\}$.



Im Streifen: $(\frac{3}{2}),(\frac{4}{3}),(\frac{5}{5})$



oder auch:

$x_1 = 3$

$x_2 = 4{,}5$

5.3.4. Problem1 und Lösung:

Wo liegen alle Pkte aus einem vertikalen Streifen?

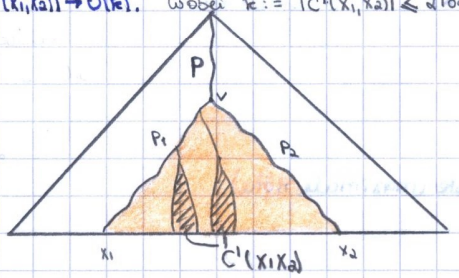
Antwort: auf P_1, P_2, P und allen Knoten zwischen P_1 und P_2!

Achtung: Auf den Pfaden P, P_1, P_2 können auch andere Pkte liegen.

Berechnung der Pkte:
- Filtere die Pkte auf Pfaden nach $[x_1, x_2]$ in Zeit $2 \cdot \log n + k$.
- Gib alle Pkte, die in Knoten zwischen P_1, P_2 gespeichert sind, aus $|C'(x_1,x_2)| \to O(\tilde{k})$. wobei $\tilde{k} := |C'(x_1,x_2)| \leq 2 \log n$

$2(\text{Höhe}(v) + 1 + O(k_1) + 1 + O(k_2) + \dots) =$
$= 2 \cdot (\text{Höhe}(v) + O(k_1) + O(k_2) + \sum 1) =$
$= 2 \cdot (\log n + \tilde{k})$

$\tilde{k}$:= # Knoten in orange.



Im Bsp: 1 Bild ⇒ $\{(\frac{5}{5}),(\frac{4}{3}),(\frac{2}{3}),(\frac{3}{2})\} \supset$ gesuchte Menge

2 Bild ⇒ $\{(\frac{5}{5}),(\frac{2}{3}),(\frac{4}{3}),(\frac{3}{2})\} \supset$ gesuchte Menge

Platz: $O(n)$!

Gesamtlaufzeit: $2\log n + k + O(\log n + \tilde{k}) =$
$= O(\log n + K)$

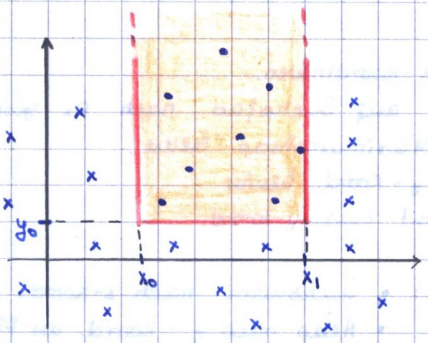
K = # Ausgaben. $(= k + \tilde{k})$

5.3.5 Problem 2 + Lösung:

Wozu Heap-Eigenschaft nach y-Koordinaten?
→ 1½-dimensionale Range Abfragen
(Halb offene oder 3 seitige).

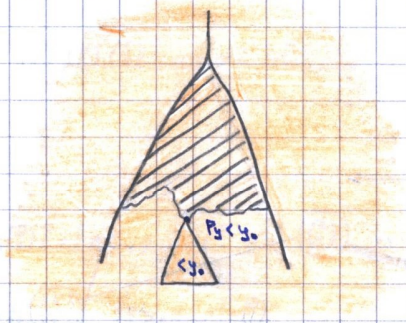
Geg: $x_0, x_1, y_0, \ldots, S \subset \mathbb{R}^2$

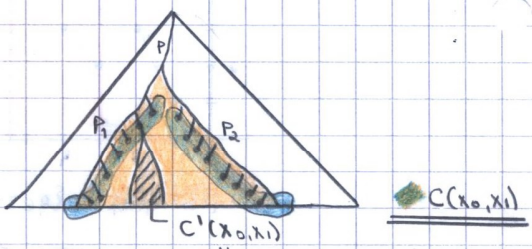
Ges: alle $p \in S$ mit
$x_0 \leq p_x \leq x_1$ und
$p_y \geq y_0$



Lösung:

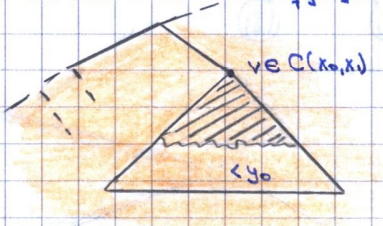
- Filtern der Pfade: $\forall$ Knoten $v \in P_1 \cup P_2 \cup P$ gib enthaltenen Pkt p aus, falls $x_0 \leq p_x \leq x_2 \wedge p_y \geq y_0$
  → Kostet Zeit $O(\log n)$. Mit Ausgabe kostet dies $O(\log n + k)$
- Rest der Ausgabe (normalerweise müssen alle $v \in P_1 \cup P_2 \cup P$ und zwischen P_1 und P_2 betrachtet werden. Oben nur $v \in P_1 \cup P_2 \cup P$, hier also die restlichen, dh. also: v zwischen P_1 und P_2) ist im oberen Bereich von $C'(x_0, x_1)$ gespeichert (wg. Heapeigenschaft)





{"alle kleinen Bäume"}

Auflisten dieser Pkte:
Starte in den Knoten von $C(x_0, x_1)$.
Scanne den jeweiligen Unterbaum bis zu den Knoten mit $p_y < y_0$
→ Kostet O(Beitrag zur Ausg + 1).



Laufzeit: $\forall v \in C(x_0, x_1)$: $\sum_{v \in C(x_0, x_1)} O$(Beitrag zur Ausg + 1) =

$= O(\log n + K')$, wobei K' = Beitrag zur Gesamtlösung von $C'(x_0 x_1)$

$\text{Höhe}(v) + 1 + O(\text{Beitr. zu Ausg}(v_1) + 1) + 1 + O(\ldots) + \ldots + \text{Höhe}(v) + 1 + O(\ldots) + \ldots = 2 \cdot \log n + \underbrace{\sum_{v \in C(x_0, x_1)} O(\text{Beitr. zu Ausg} + 1)}_{=: K'} = O(\log n + K')$

5.3.6. Satz (Zusammenfassung):

Der Priority-Search Tree verwaltet eine Menge von n Pkten im $\mathbb{R}^2$ unter folgenden Operationen und Laufzeiten:

- Insert / Delete: $O(\log n)$
- Drei-seitige Bereichsabfrage: $O(\log n + K)$, K = # Ausgaben.

und benötigt Speicherplatz: $O(n)$.

⇒ Gesamtlaufzeit: $a \log n + k + O(\log n + K') = \underline{O(\log n + \tilde{K})}$
$\tilde{K} = k + K'$, # Ausgaben.

5.3.7 Anwendung:

feste (maximale Höhe).

↑ Range-Abfragen mit PST (→ Übung).