4. Mehrdimensionale Suchstrukturen

Anforderung an eine Datenstruktur für Punktmengen:
Bereichsanfragen müssen effizient durchführbar sein

gegeben: Menge D von n Punkten im $\mathbb{R}^k$
Ordnung auf $\mathbb{R}^k$? Suchstruktur?

erster Ansatz: lexikografische Ordnung (s. Kap. 2)

$(p_1, ..., p_k) <_{lex} (q_1, ..., q_k)$
$:\Leftrightarrow \ \exists i \in \{0, ..., k-1\} : p_j = q_j \ \ \forall 1 \leq j \leq i$ und $p_{i+1} < q_{i+1}$
$\rightarrow$ eindimensionale Suchstruktur

Nachteile:

- balancierter Suchbaum gemäß $<_{lex}$ hat Höhe $\log n$,
  aber jeder Vergleich kostet Zeit $O(k)$ $\rightarrow$ Suchzeit $O(k \log n)$
- mehrdimensionale Anfragen werden von 1-d–Suchbaum nicht unterstützt

(Keßler 1998)

4.1 Der k-d-Baum (k-dimensionaler Suchbaum)

gegeben: Menge D von n Punkten, zunächst im $\mathbb{R}^2$ (d.h. $k = 2$)
Vorläufig: Punkte in D unterscheiden sich in jeder Koordinate

- Der Raum wird sukzessive durch achsenparallele Hyperebenen unterteilt
- die Koordinatenachse, zu der die Splithyperebene senkrecht verläuft ("Splitkoordinate"), wechselt bei jedem Unterteilungsschritt (zyklisch: 1, 2, ..., *k*, 1, 2, ...)

konstruiere k-d-Baum T für D:

Wähle *Splitkoordinate* x_d, $d \in \{1, ..., k\}$, z.B. x_1

Bestimme *Splitwert* $s \in \mathbb{R}$, der nicht in D als x_d–Koordinate vorkommt

$\rightarrow$ *Splitgerade* (für $k > 2$: Splithyperebene) $x_d = s$
teilt D in zwei Halbebenen (für $k > 2$: Halbräume)

$$D_{<s} = \{(x_1, ..., x_k) \in D : x_d < s\} \; = \; D \cap \{x_d < s\}$$

$$D_{>s} = \{(x_1, ..., x_k) \in D : x_d > s\} \; = \; D \cap \{x_d > s\}$$

falls $|D_{<s}| > 1$:
konstruiere k-d-Baum $T_{<s}$ für $D_{<s}$ als linken Unterbaum von T;

falls $|D_{>s}| > 1$:
konstruiere k-d-Baum $T_{>s}$ für $D_{>s}$ als rechten Unterbaum von T;

jeweils mit der nächsten Splitkoordinate $d \leftarrow (d+1)\%k$

innere Knoten von T entsprechen Splitgeraden

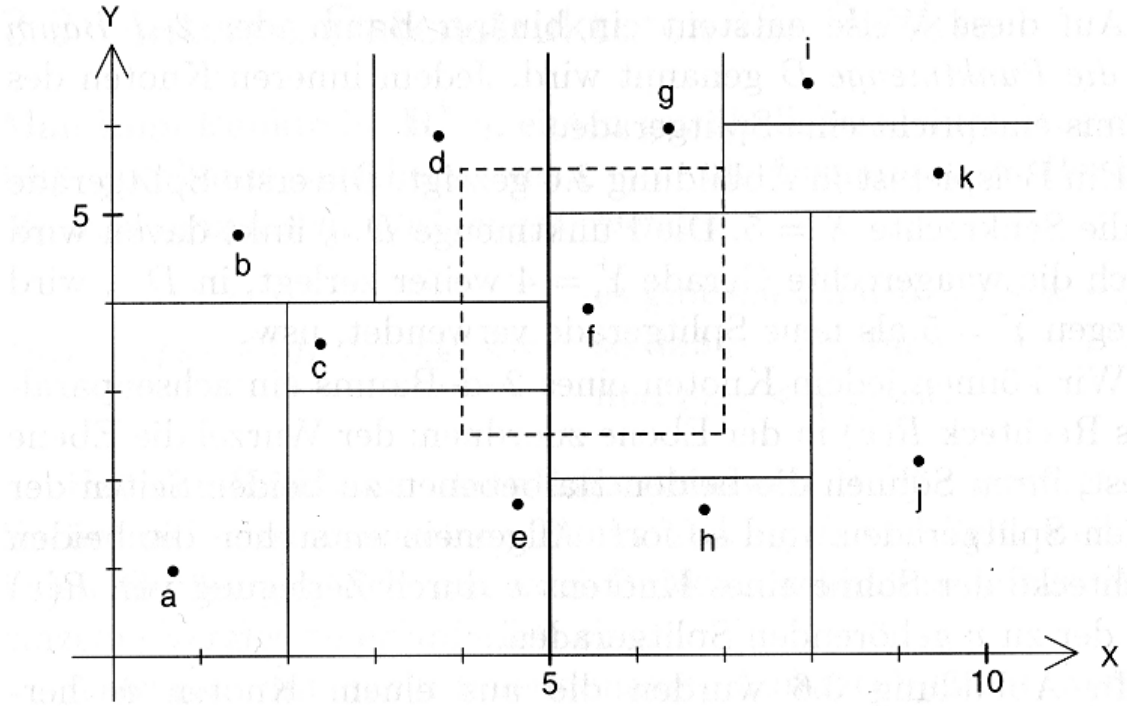
Blätter von T entsprechen Punkten von D

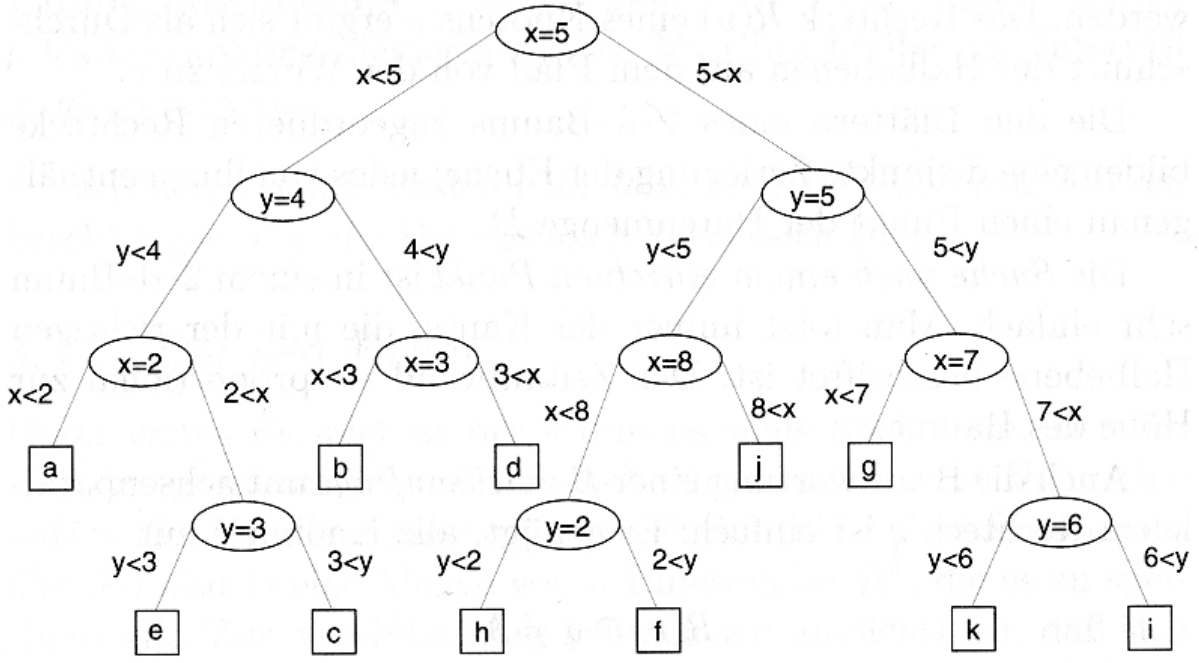
Kanten von T beschriftet mit Halbebene

Scope $R(v)$ eines Knotens v im 2-d-Baum:
achsenparalleles Rechteck, begrenzt durch Splitgeraden (bzw. $\pm\infty$),
das gerade alle Punkte im Unterbaum $T(v)$ von v enthält
$=$ Durchschnitt aller Halbebenen auf Pfad Wurzel$\rightarrow v$

Blattzahl $a(v)$ eines Knotens v im 2-d-Baum:
Anzahl aller Punkte in den Blättern von $T(v)$

Beispiel (d = 2):





(in den Knoten sind die Splitgeraden-Gleichungen angegeben)

(aus Klein 1997)

Suche nach Punkt $p \in D$ im k-d-Baum:

folge immer der Kante mit der richtigen Halbebene

Bereichsanfrage mit achsenparallelem Rechteck Q:

besuche alle Knoten $v \in T$ mit $R(v) \cap Q \neq \emptyset$

für diese v teste, ob v innerhalb Q liegt

Bemerkung: Falls $\exists\ v \in T$ mit $R(v) \cap Q \neq \emptyset$, so gilt dies auch für alle u auf Pfad Wurzel$\rightarrow v$, da $R(u) \supset R(v)$.

Satz:

Sei T ein 2-d-Baum der Höhe h für Menge D von n Punkten.

Bereichsanfrage mit achsenparallelem Rechteck Q geht in Zeit $O(2^{h/2} + a)$, wobei $a = |D \cap Q|$.

Beweis:

Zu besuchende Knoten v (d.h. $R(v) \cap Q \neq \emptyset$) sind zweierlei Typs:

Typ 1: $R(v) \subseteq Q$ (ganz enthalten)

Typ 2: $R(v) \not\subseteq Q$ (nur teilweise Überlappung)

Für Typ-1-Knoten v:

alle $a(v)$ Blätter im Unterbaum $T(v)$ ausgeben $\rightarrow$ Zeit $O(a(v))$

Für Typ-2-Knoten v:

Fall (2a): $Q \subseteq R(v)$ $\rightarrow$ bisher nur ein Suchpfad (Höhe $\leq k$)

Fall (2b): Mindestens eine Kante l von Q schneidet Rand von $R(v)$

$t_d :=$ # Typ-2-Knoten v der Tiefe d in T, für die der Rand von $R(v)$ von Kante l geschnitten wird

$t_1 \leq 1$ (Wurzel), $t_2 \leq 2$ (deren Söhne),

$t_{d+2} \leq 2t_d$ da l max. 2 von 4 Unterrechtecken schneiden kann

$\rightarrow t_d \leq 2^{d/2}$, $d = 1, 2, ..., h$

$\rightarrow$ Zeit für Besuch aller Typ-2-Knoten: $O(h + \Sigma_{d=1}^{h} 2^{d/2}) = O(2^{d/2})$

(deckt auch Suchzeit für Typ-1-Knoten ab, da jeder Typ-1-Knoten minimaler Tiefe einen Vater vom Typ 2 hat) □

Höhe h eines 2-d-Baumes mit n Blättern

Ein Binärbaum heißt *ausgeglichen*, falls für alle inneren Knoten v mit Söhnen v_1, v_2 gilt: $|a(v_1) - a(v_2)| \leq 1$

$\rightarrow h \leq \lceil \log n \rceil$

$\rightarrow$ Zeit für Bereichsanfrage: $O(\sqrt{n} + a)$

Konstruktion ausgeglichener 2-d-Bäume:

- sortiere D nach aufsteigenden x– und y–Koordinaten
- rekursiv unterteile D in gleich große Teilmengen $\rightarrow T$

Zeit je Teilungsschritt für Knoten v: $O(a(v))$

$\rightarrow$ Zeit für Aufbau: $O(n \log n)$

Bemerkung: (Dynamisierung)

Einfügen in Zeit $O(\log^2 n)$, Löschen in Zeit $O(\log n)$ möglich, falls Verlängerung der Suchzeit um Faktor $\log n$ akzeptiert wird (siehe z.B. Klein'97)

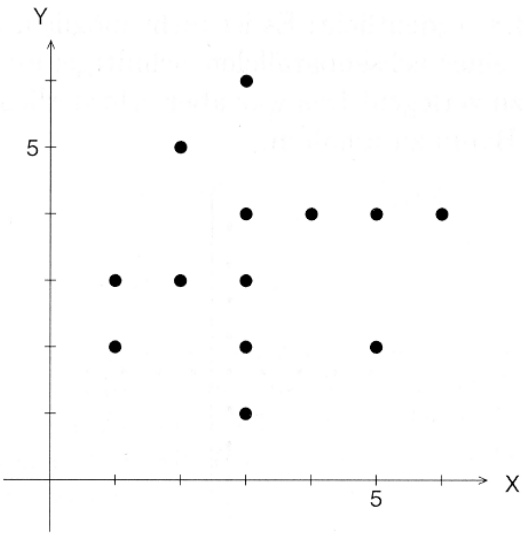
Beseitigung der Einschränkung

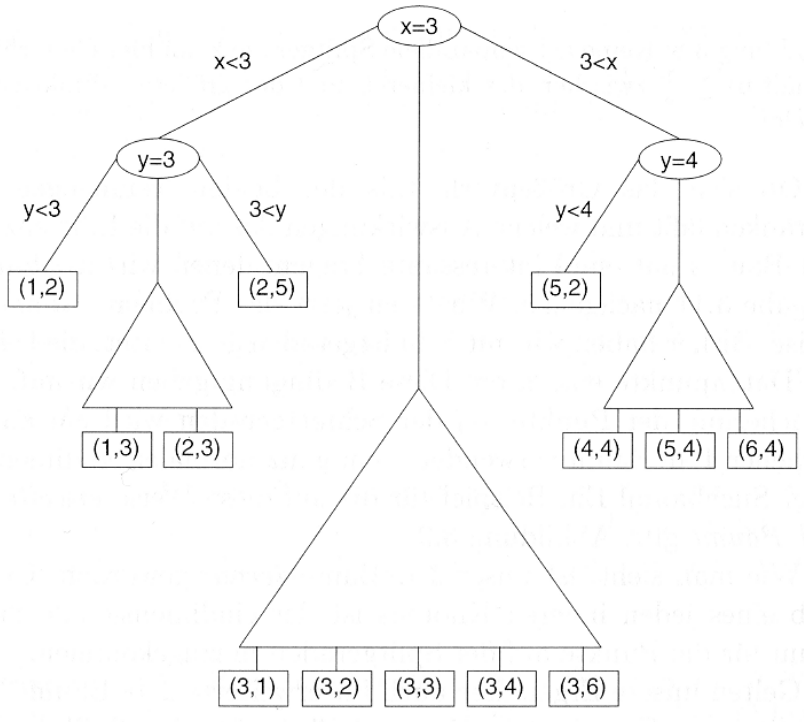
"verschiedene x/y–Koordinaten"

Problem: Aufteilung in zwei gleich große Teilmengen ist dann i.allg. nicht möglich

Abhilfe: Für Punkte *auf* Splitgerade verwende 1-d-Suchbäume
→ *erweiterter 2-d-Baum* (Knoten ternär)

Beispiel:





(aus Klein 1997)

Scope $R(v)$ für Knoten v auf Splitgerade
ist ein degeneriertes Rechteck
(Liniensegment, Intervall auf Splitgerade)

In Bereichsanfrage:

Behandlung analog, eindimensionales Suchen in $T(v)$ genügt

Erweiterter 2-d-Baum heißt $ausgeglichen$, falls für alle inneren Knoten v mit Söhnen v_1, v_2, v_3 gilt:
$a(v_1) \leq \frac{1}{2}a(v)$ und $a(v_2) \leq \frac{1}{2}a(v)$
$\rightarrow$ Höhe $O(\log n)$

k-d-Bäume für Dimensionen $k > 2$:

statt Splitgerade benutze Split-Hyperebene $x_d = s$,
$d = 1, 2, ..., k, 1, 2, ..., k, 1, 2, ...$
Punkte auf Splithyperebene in $(k-1)$-d-Baum gespeichert

Für ausgeglichenen k-d-Baum für n Punkte gilt:
Höhe $O(\log n)$
Konstruktionszeit $O((k)n \log n)$
Platz $O(n)$ (++)
Bereichsanfrage $O(n^{1-\frac{1}{k}} + a)$ (– –)
siehe Lee / Wong '77

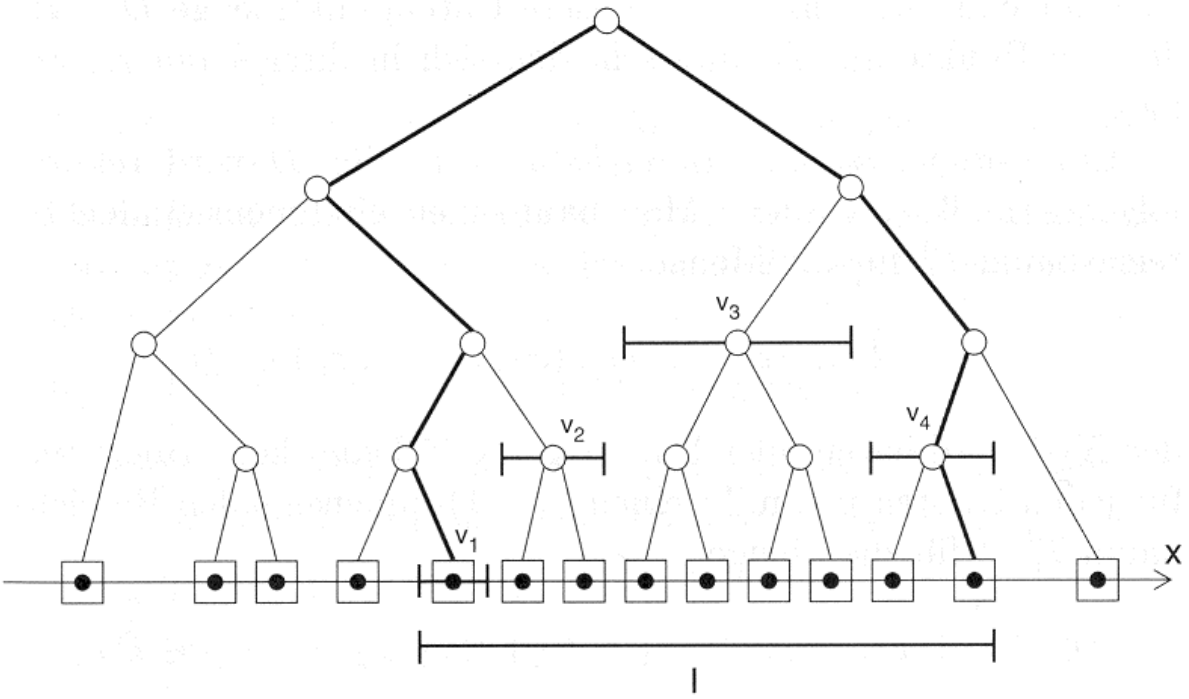
Bemerkung: Bereichsanfragen können beschleunigt werden, wenn man mehr als linearen Platz investiert
($\rightarrow$ Bereichsbäume)

(Keßler 1998)

4.2 Der Bereichsbaum (range tree) (Keßler 1998, Klein 1997)

betrachte zunächst eindimensionalen Fall:

eindimensionaler Bereichsbaum T^1 für n Punkte im $\mathbb{R}^1$ (Zahlen)
ist ein ausgeglichener binärer Suchbaum.
Jedes Blatt v enthält einen Punkt $a(v) \in D$
Jeder innere Knoten v enthält Trennelement s,
z.B. $s = \max_{u \in T(lson(v))} a(u)$, (Maximum des linken Teilbaums)
$\rightarrow$ Suche in Zeit $O(\log n)$ möglich



Scope jedes $v \in T^1$ ist ein Intervall

$$I(v) = [\min_{u \in T(v)} a(u), \max_{u \in T(v)} a(u)]$$

Für Blätter v ist $I(v) = a(v)$ (eine Zahl)

Bereichsanfrage mit Intervall Q:
suche alle am weitesten oben gelegenen Knoten v in T^1 mit $I(v) \subseteq Q$

Satz:

Seien $v_1, ..., v_\ell$ diese obersten Knoten mit $I(v_i) \subseteq Q$, $i = 1, ..., \ell$.

(a) $\cup_{i=1}^{\ell} I(v_i)$ enthält dieselben Punkte aus D wie Q

(b) Die $I(v_i)$ sind paarweise disjunkt.

(c) Finden aller v_i, $i = 1, ..., \ell$, ist in Zeit $O(\log n)$ möglich.

Bereichsanfrage geht in Zeit $O(\log n + a)$, mit $a = |D \cap Q|$.

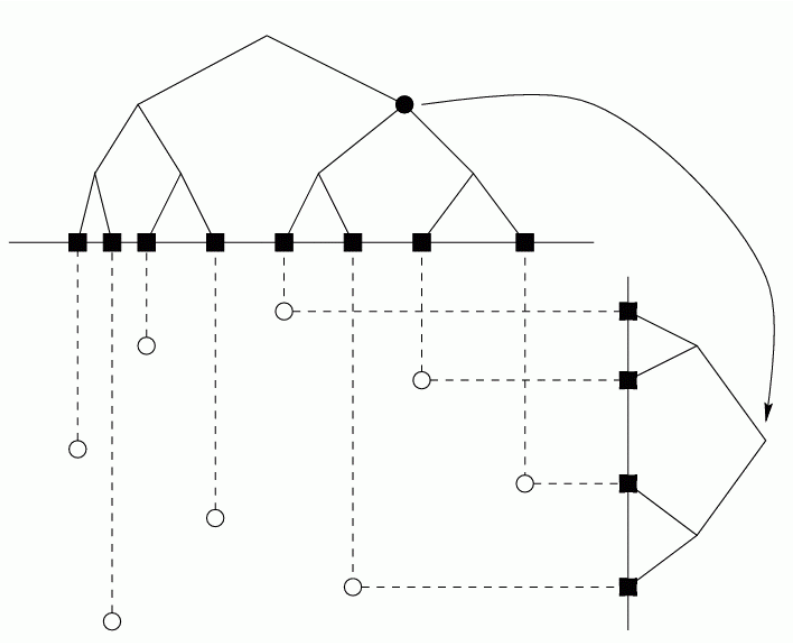
(hier ohne Beweis)

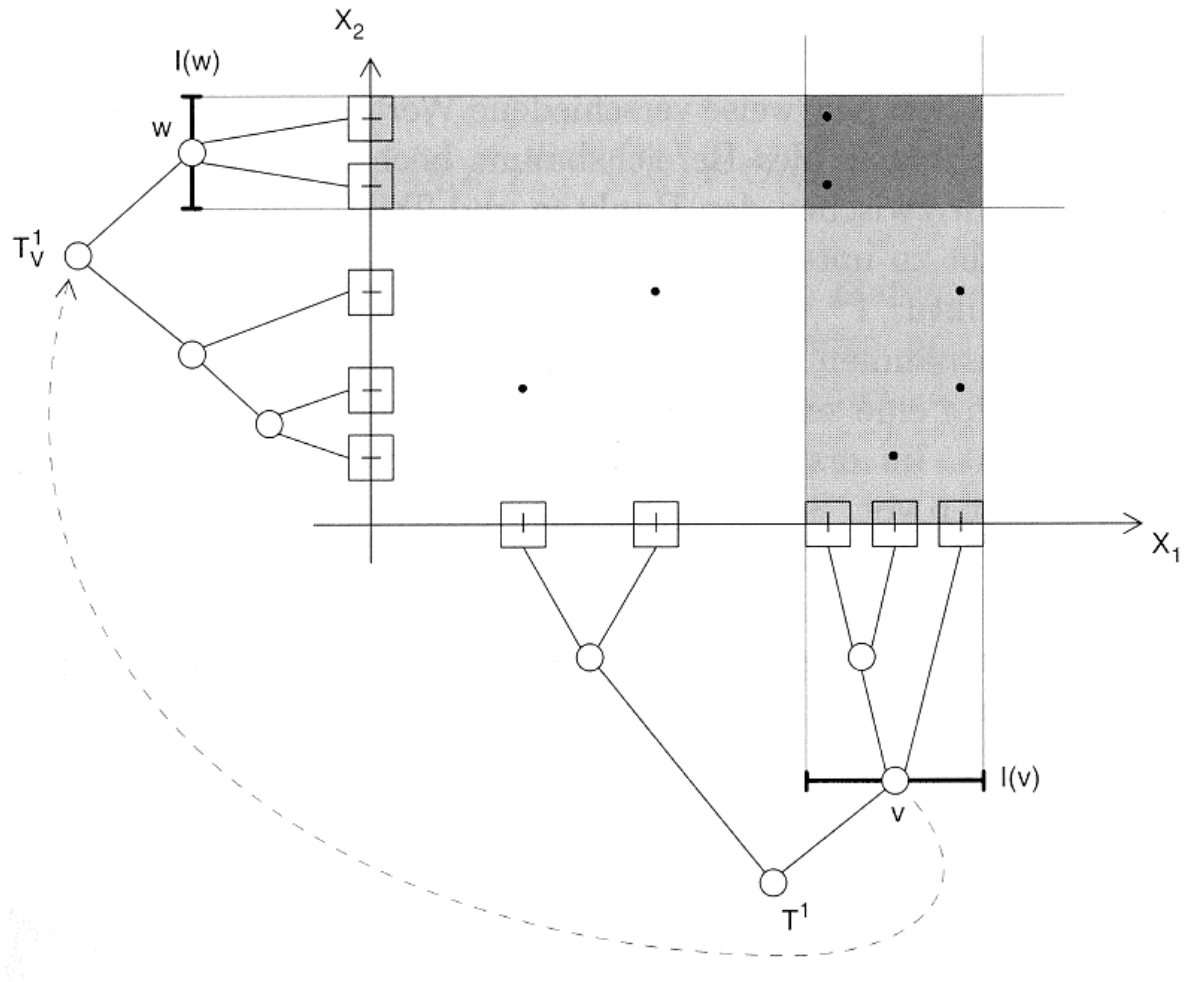
k-dimensionaler Bereichsbaum T^k für D

Annahme: Je zwei Punkte aus D unterscheiden sich in ihrer k–ten Komponente

Konstruktion von T^k für D:

- baue T^1 für $D^1 = \{x_1 \mid \exists x_2, ..., x_k : (x_1, x_2, ..., x_k) \in D\}$
- für alle Knoten $v \in T^1$
  konstruiere rekursiv (k-1)-dimensionalen Bereichsbaum T_v^{k-1}
  für $D_v^{k-1} := \{(x_2, ..., x_k) \mid \exists x_1 \in I(v) : (x_1, x_2, ..., x_k) \in D\}$.
  Ein Pointer in v verweist auf T_v^{k-1}.
- Rückgabe T^1





(hier in beiden Beispielen nur für *einen* Knoten der zugehörige x2-Bereichsbaum eingezeichnet)

Satz:

Sei $k \geq 2$. Der Speicherplatzbedarf für einen k-dimensionalen Bereichsbaum zur Speicherung von n Punkten im R^k (die die Menge D bilden) ist $O(n (\log n)^{k-1})$.

Beweis: oBdA seien alle $p \in D$ in allen Koordinaten verschieden

Datenpunkt $p \in D$ *kommt* in einem der T^1 von T^k *vor*, falls eine seiner Koordinaten in einem Blatt dieses T^1 gespeichert ist.

Im top-level T^1 kommt jeder Punkt $p \in D$ vor.

Seine x_1–Koordinate ist in $\log n$ Intervallen $I(v)$ enthalten.

Für jeden T_v^{k-1} gilt selbiges für die x_2–Koordinate usw.

$\rightarrow$ jeder Punkt $p \in D$ kommt in

$$\sum_{i=0}^{k-1} \log^i n \;=\; \frac{\log^k n - 1}{\log n - 1} \;\in\; \Theta(\log^{k-1} n)$$

vielen T^1 vor

$\rightarrow$ Platz $O(n \log^{k-1} n)$.

Satz: Unter den Voraussetzungen des vorigen Satzes ist die Konstruktionszeit des Bereichsbaumes $O(n (\log n)^{k-1})$.

Beweis:

Sortiere Punkte in D nach allen Komponenten

$\rightarrow$ Zeit $O((k)n \log n)$

Induktion über k:

Aufbau des top-level T^1 in Zeit $O(n)$ mit $2n-1$ Knoten v_i

$\rightarrow 2n-1$ Bäume $T_i := T_{v_i}^{k-1}$, und $|T_i| = n_i$.

Nach Induktionsannahme kann T_i in Zeit $\leq C \cdot n_i \log^{k-2} n$ konstruiert werden.

Jedes v_i kommt in $\log n$ vielen T_i's vor

$$\rightarrow \quad \sum_{i=1}^{2n-2} n_i \ \leq \ n \log n$$

$\rightarrow$ Gesamtaufbau in Zeit

$$C \sum_{i=1}^{2n-2} n_i \log^{k-2} n_i$$

$$\leq C \log^{k-2} n \cdot \sum_{i=1}^{2n-2} n_i$$

$$\leq Cn \log^{k-1} n$$

Bereichsanfrage mit achsenparallelem Hyper–Rechteck $Q \subset \mathbb{R}^k$:

(0) falls $k = 1$, beantworte Anfrage wie im 1-D-Fall

(1) Stelle Q dar als $Q = I \times Q'$
mit Intervall I und Hyper–Rechteck $Q' \subset \mathbb{R}^{k-1}$

(2) Wie im 1-D-Fall bestimme oberste Knoten $v_1, ..., v_\ell$ in T^1,
deren Intervalle $I(v_i)$ zusammen I abdecken

(3) Für $1 \leq i \leq \ell$: beantworte Bereichsanfrage Q' an $T_{v_i}^{k-1}$.

Satz:
Unter den Voraussetzungen der vorigen beiden Sätze kann eine achsenparallele Bereichsanfrage Q in der Zeit $O((\log n)^k + a)$, $a = |\, Q \cap D \,|$, beantwortet werden.

Beweis:

In (2) werden $2\lceil \log n \rceil$ viele Knoten $v_j \in T^1$ bestimmt, deren $I(v_j)$ das Anfrageintervall Q abdecken.

Nach Induktionsannahme gilt für $T^{k-1}_{v_j}$

Zeit $C \log^{k-1} n_j + a_j$ zum Berichten der a_j Punkte aus Q'

Gesamtzeit:

$$C \log n + C \sum_{j=1}^{2\lceil \log n \rceil} (\log^{k-1} n_j + a_j) \ \in \ O(\log^k n + a)$$

4.3 Der Prioritätssuchbaum (priority search tree)

(McCreight 1981; Klein 1997; Keßler 1998)

$=$ Kombination aus T^1 (1-d-Bereichsbaum) und Heap für Punkte im $\mathbb{R}^2$

gegeben: Menge D von n Punkten $p_i = (x_i, y_i)$, $i = 1, ..., n$

Vorläufige Annahme: $x_i \neq x_j$ für $i \neq j$

baue T^1 (1-d-Bereichsbaum, ausgeglichen)

jeder Knoten in T^1 hat Platz für 1 Punkt

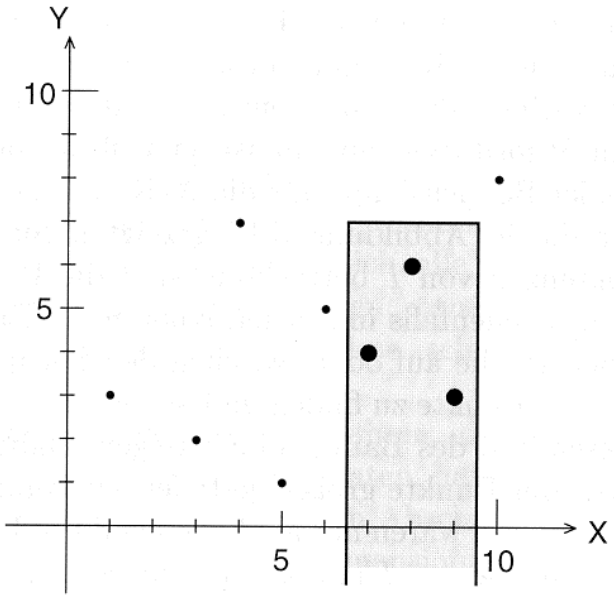
trage die $p_i \in D$ in die $Knoten$ des T^1 ein, so daß gilt:

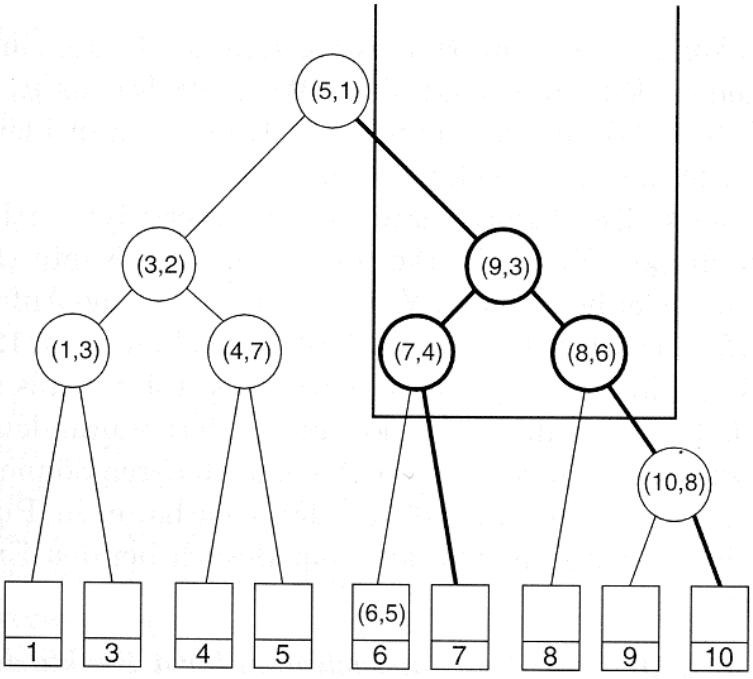
(1) jeder $p_i = (x_i, y_i)$ steht auf Suchpfad für x_i

(2) entlang Pfad Wurzel→Blatt: aufsteigende y–Koordinaten
$(heap–Eigenschaft)$

(3) jeder p_i steht so dicht bei der Wurzel wie möglich.

Beispiel eines Prioritätssuchbaumes für 9 Punkte in der Ebene (eingezeichnet ist außerdem eine Halbstreifen-Anfrage):





Bemerkung: (1)–(3) stets erfüllbar ($\rightarrow$ Übung)

Punkte $p_i = (x_i, y_i)$ eintragen in Reihenfolge steigender y–Koordinaten (vorsortieren $\rightarrow O(n \log n)$)

- laufe entlang Suchpfad zu x_i nach unten
- sobald freier Knoten v gefunden, trage p_i in v ein

Konstruktionszeit: $O(n \log n)$, Platz $O(n)$

Anfragen an den Prioritätssuchbaum:

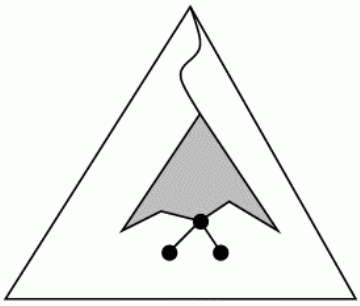
1-dim. Anfrage mit Intervall *I* auf der *x*-Achse:
gelöst mittels T^1 in der Zeit $O(\log n + a)$ (a = Anzahl der zu berichtenden Punkte).

Halbstreifenanfrage

bestimme alle $(x_i, y_i) \in D$ mit $x_i \in I$ und $y_i \leq y_0$

d.h. $D \cap (I \times]-\infty, y_0])$

beim Abstieg im T^1 gemäß Intervall I:
stoppe Abstieg, sobald y–Koordinaten größer als y_0 werden, und kehre um
Für jeden Knoten, der zur Ausgabe beiträgt, werden höchstens 2 Knoten (Söhne) betrachtet, die nicht beitragen.
Zeit: $O(\log n + a)$



für Rechteckanfragen ist diese Vorgehensweise weniger günstig (aber siehe unten).

Annahme $x_i \neq x_j$ für $i \neq j$ kann entfallen:
benutze lexikographische Ordnung auf (x_i, y_i)–Paaren
für Konstruktion / Suchen im T^1.

Anwendungsbeispiel: Speicherung von 1-d-Intervallen

gegeben:

Menge von Intervallen $I_i = [a_i, b_i] \subseteq \mathbb{R}$, $i = 1, ..., n$

Zahl a_0 bzw. Intervall $I_0 = [a_0, b_0]$

gesucht:

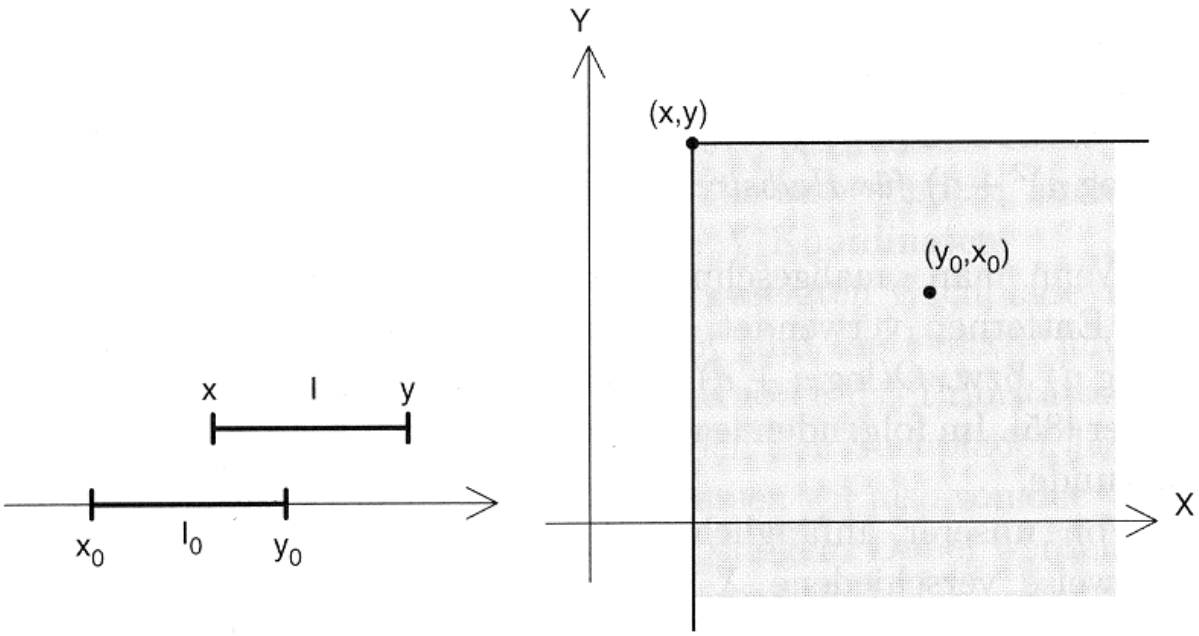
(1) Menge aller I_i mit $a_0 \in I_i$ (Aufspießanfrage)

(2) Menge aller I_i mit $I_0 \cap I_i \neq \emptyset$ (Überlappanfrage)

Bemerkung: (1) ist Spezialfall von (2)

Idee: Ordne jedem $I_i = [a_i, b_i]$ einen Punkt $p_i := (b_i, a_i) \in \mathbb{R}^2$ zu
$(geometrische\ Transformation)$

Das Problem "Intervallüberlappung" wird mit dieser Transformation in eine Bereichsanfrage transformiert:



Klar: $I_i = [a_i, b_i$ überlappt mit $I_0 = [a_0, b_0]$

$\iff a_0 \leq b_i$ und $a_i \leq b_0$

$\iff p_i = (b_i, a_i)$ liegt rechts unterhalb von $p_0 = (a_0, b_0)$

$\rightarrow$ Lösung von (2):

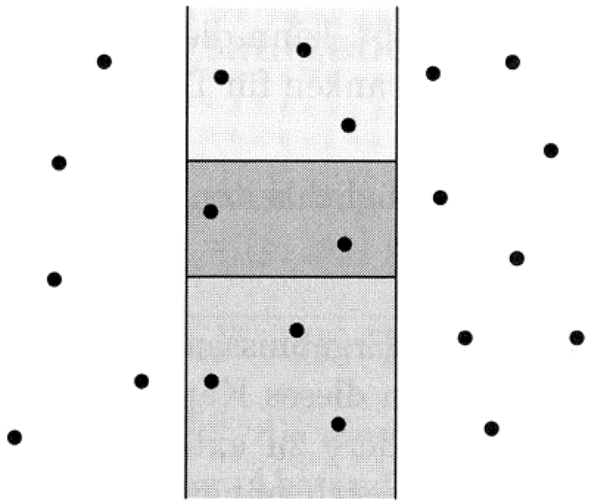
1. baue Prioritätssuchbaum T für $D = \{p_i = (b_i, a_i) : \ i = 1, ..., n\}$
2. für Anfrageintervall $I_0 = [a_0, b_0]$:

   finde in T alle $p_i = (b_i, a_i)$

   in Viertelebene rechts unterhalb von $p_0 := (a_0, b_0)$

   $=$ Halbstreifenanfrage $D \cap ([a_0, \infty[\times] - \infty, b_0])$

Zeit für Überlappanfrage: $O(\log n + a)$



Rechteck–Anfrage an Prioritätssuchbaum

naiv: Rechteck Q als Durchschnitt zweier Halbstreifen H_1, H_2:
schlecht, falls $|D \cap Q| \ll |D \cap H_1| + |D \cap H_2|$

Streifenmethode:

Zerlege Ebene in waagerechte Streifen S_i der Breite h

Bis zu n Streifen S_i sind nichtleer (enth. $n_i \geq 1$ Punkt aus D)

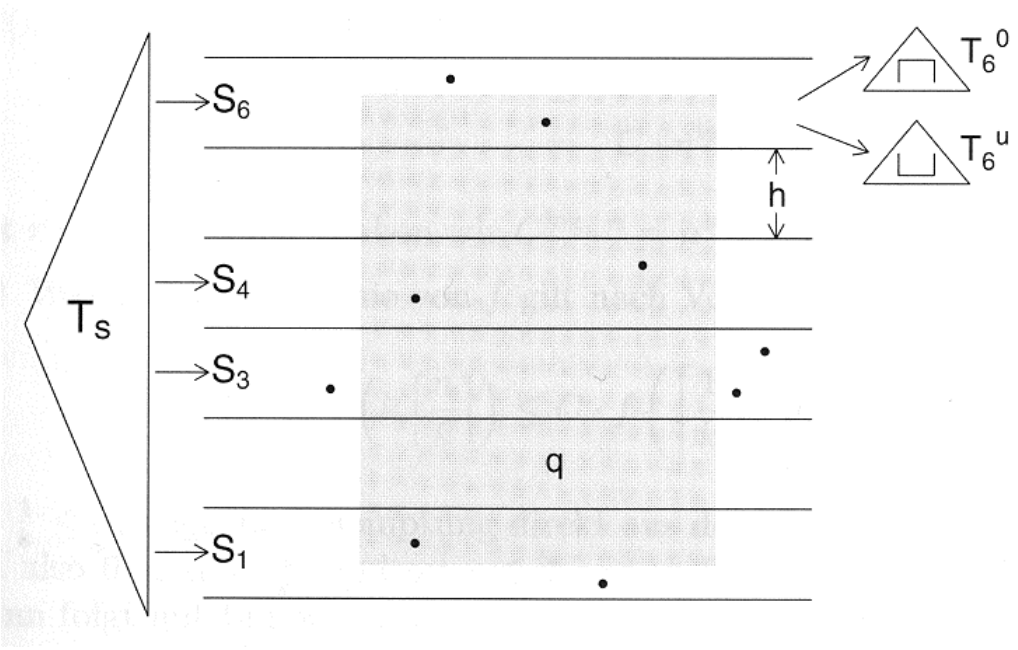
Merke Indizes i der nichtleeren Streifen S_i in einem ausgeglichenen binären Suchbaum T_S

In jedem nichtleeren Streifen S_i halte

- Prioritätssuchbaum T_i^o für nach oben beschränkte Halbstreifen
- Prioritätssuchbaum T_i^u für nach unten beschränkte Halbstreifen

(für T_i^u: umgekehrte Ordnung bei Heap-Eigenschaft)

Jedes achsenparallele Anfragerechteck Q mit Höhe $\leq h$ schneidet max. 2 aufeinanderfolgende Streifen S_i



(Indizes der nichtleeren S_i durch 1-d-Bereichsanfrage an T_S)
Für diese max. 2 nichtleeren Streifen S_i, S_{i+1} mit $Q \cap S_i \neq \emptyset$
ex. Halbstreifen Q_i, nach unten beschränkt, mit $Q \cap S_i = Q_i \cap S_i$
und Halbstreifen Q_{i+1}, nach oben beschränkt, mit $Q \cap S_{i+1} = Q_{i+1} \cap S_{i+1}$
$\rightarrow$ $\leq$ 2 Halbstreifenanfragen Q_i an T_i^u und Q_{i+1} an T_{i+1}^o
liefern alle Beiträge zu $D \cap Q$.

Gesamtzeit: $O(\log n + a)$