Vorlesung Algorithmische Geometrie

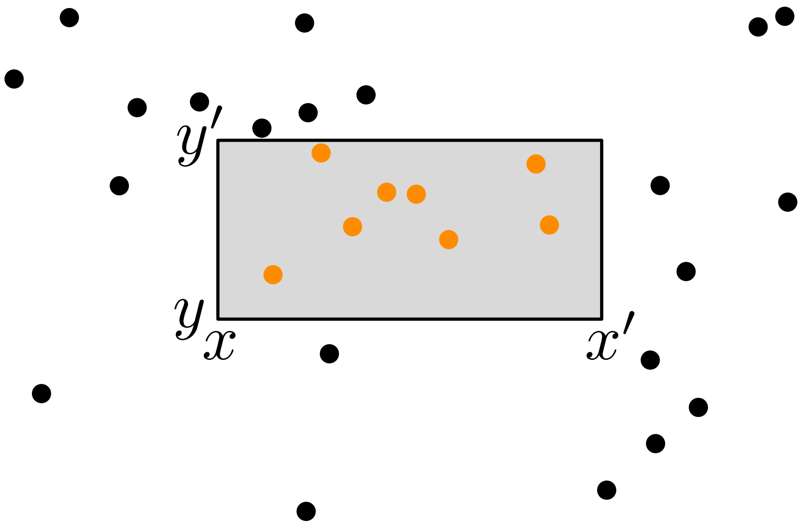
Bereichsabfragen II

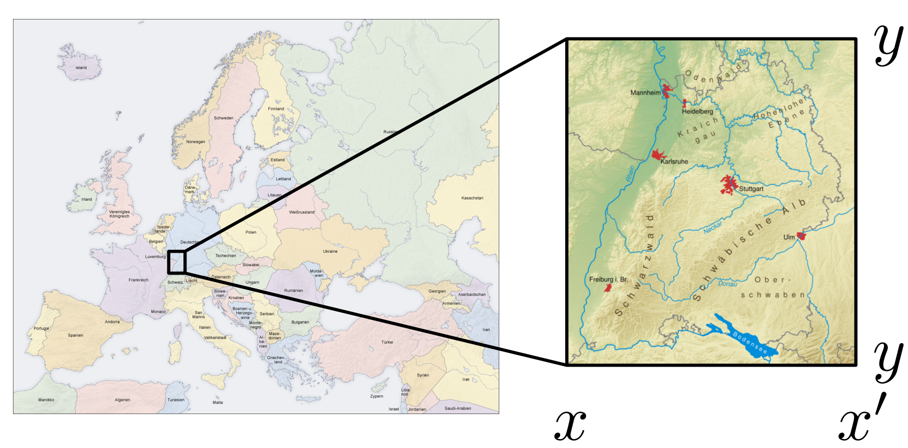
INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg

20.05.2014

Objekttypen in Bereichsabfragen





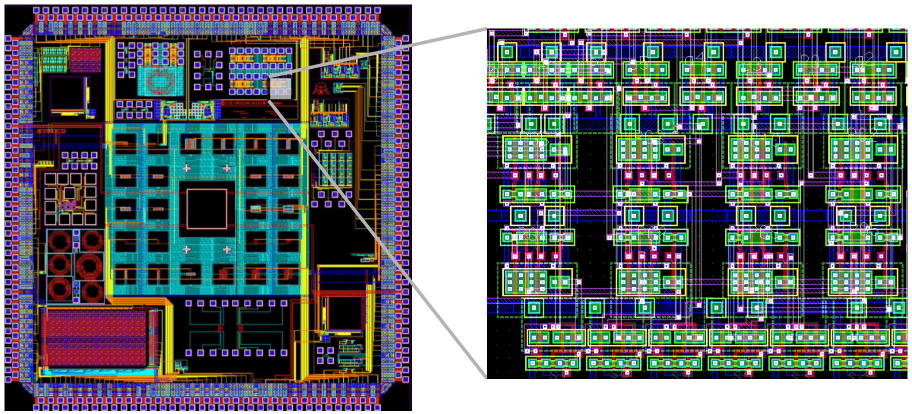
Bisher betrachteter Fall

- Eingabe: Punktmenge P (hier $P \subset \mathbb{R}^2$)
- Ausgabe: alle Punkte aus $P \cap [x, x'] \times [y, y']$
- Datenstrukturen: kd-Trees oder Range Trees

Weitere Variante

- Eingabe: Streckenmenge S (hier $S \subset \mathbb{R}^2$)
- Ausgabe: alle Strecken aus $S \cap [x, x'] \times [y, y']$
- Datenstrukturen: ?

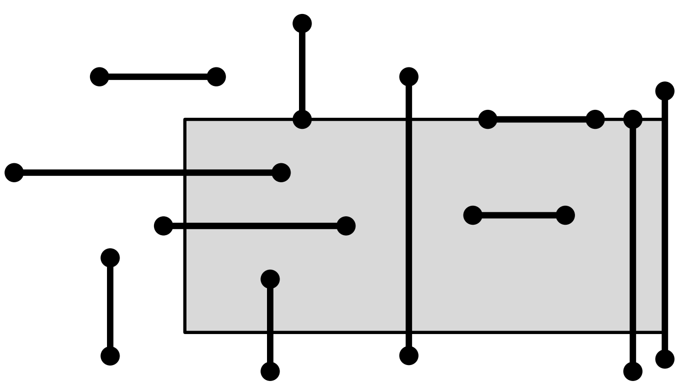
Achsenparallele Strecken



Spezialfall z.B. im VLSI Design: alle Strecken achsenparallel

Problemstellung:

Gegeben n vertikale und horizontale Strecken und ein achsenparalleles Rechteck $R = [x, x'] \times [y, y']$ finde alle Strecken, die R schneiden.



Fall 1: ≥ 1 Endpunkt in R
$\rightarrow$ Range Tree benutzen

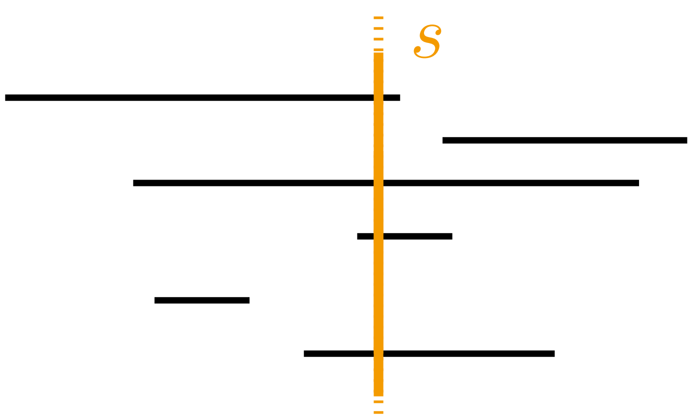
Fall 2: beide Endpunkte $\notin R$
$\rightarrow$ schneiden linke oder obere Kante von R

Fall 2 im Detail

Problemstellung:
Gegeben eine Menge H von n horizontalen Strecken und eine vertikale Query-~~Strecke~~ Gerade s finde alle Strecken in H, die s schneiden (vertikale Strecken und horizontale Query analog).

Eine Stufe einfacher: vertikale Gerade $s := (x = q_x)$

Gegeben n Intervalle $I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$ und einen Punkt q_x, finde alle Intervalle, die q_x enthalten.



Wie könnte eine geeignete Datenstruktur aussehen?

Intervall-Bäume

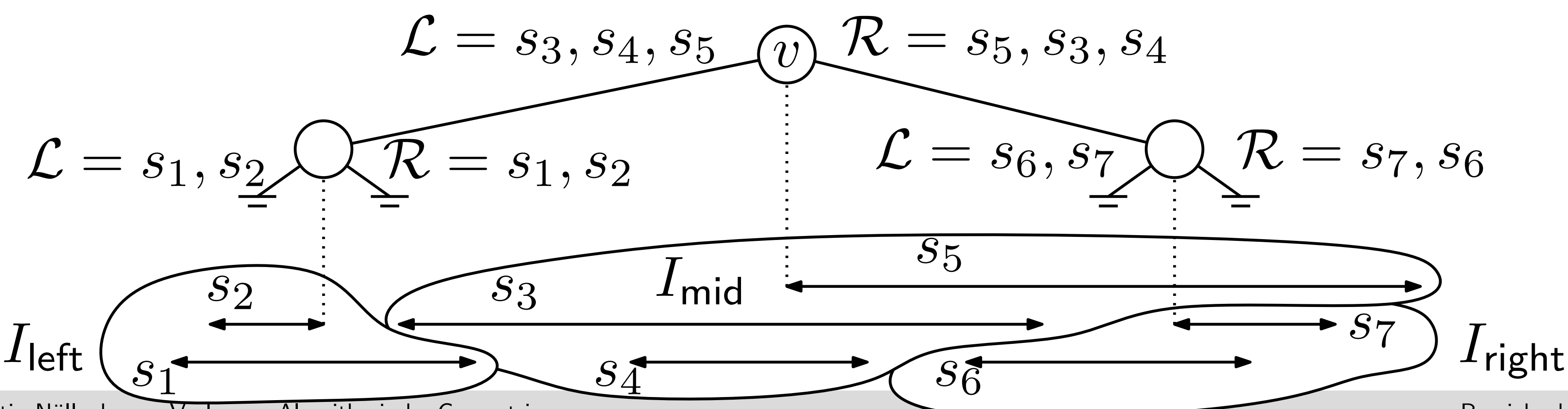
Konstruktion eines Intervall-Baums $\mathcal{T}$

- für $I = \emptyset$ ist $\mathcal{T}$ ein Blatt
- sonst sei x_{mid} Median der Endpunkte von I und definiere

$$\begin{aligned} I_{\text{left}} &= \{[x_j, x'_j] \mid x'_j < x_{\text{mid}}\} \\ I_{\text{mid}} &= \{[x_j, x'_j] \mid x_j \le x_{\text{mid}} \le x'_j\} \\ I_{\text{right}} &= \{[x_j, x'_j] \mid x_{\text{mid}} < x_j\} \end{aligned}$$

$\mathcal{T}$ besteht aus Knoten v für x_{mid}

- Listen $\mathcal{L}(v)$ und $\mathcal{R}(v)$ für I_{mid} sortiert nach linken bzw. rechten Intervallgrenzen
- linkes Kind von v ist Intervall-Baum für I_{left}
- rechtes Kind von v ist Intervall-Baum für I_{right}



Eigenschaften Intervall-Bäume

Lemma 1: Ein Intervall-Baum für n Intervalle benötigt $O(n)$ Platz und hat Tiefe $O(\log n)$. Er kann in $O(n \log n)$ Zeit erstellt werden.

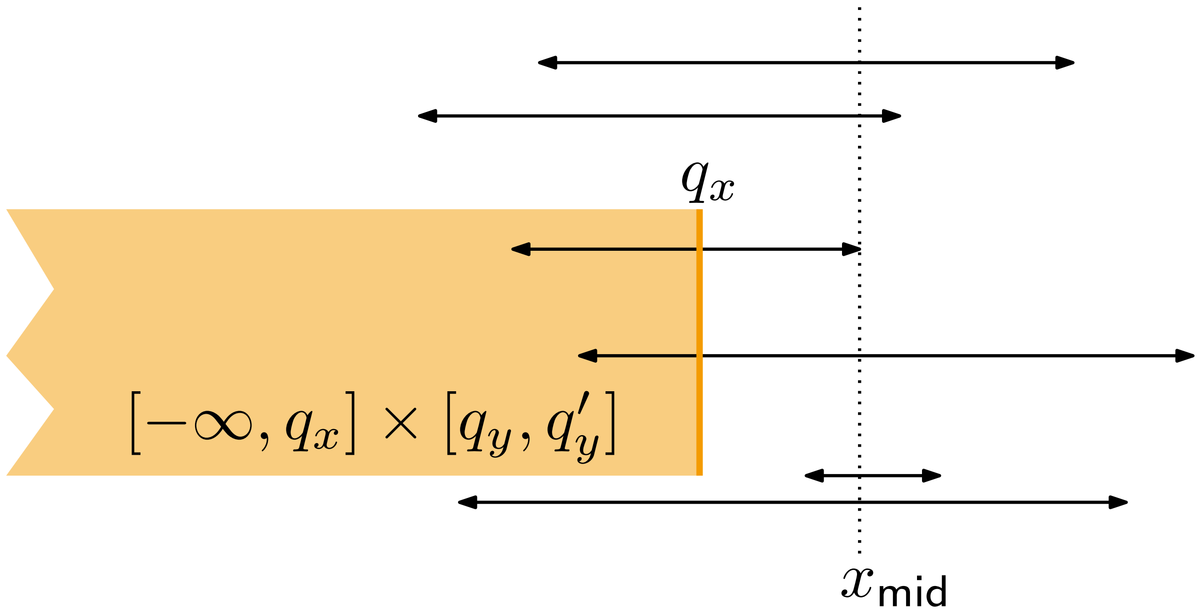
QueryIntervalTree(v, q_x)

if v kein Blatt **then**
- **if** $q_x < x_{\mathsf{mid}}(v)$ **then**
  - suche von links in $\mathcal{L}$ Intervalle, die q_x enthalten
  - QueryIntervalTree($lc(v), q_x$)
- **else**
  - suche von rechts in $\mathcal{R}$ Intervalle, die q_x enthalten
  - QueryIntervalTree($rc(v), q_x$)

Lemma 2: Mit einem Intervall-Baum können alle k Intervalle, die einen Punkt q_x enthalten in $O(\log n + k)$ Zeit ausgegeben werden.

Von Geraden zu Strecken

Wie lässt sich die Idee des Intervall-Baums abwandeln für Strecken $q_x \times [q_y, q'_y]$ statt Geraden $x = q_x$?

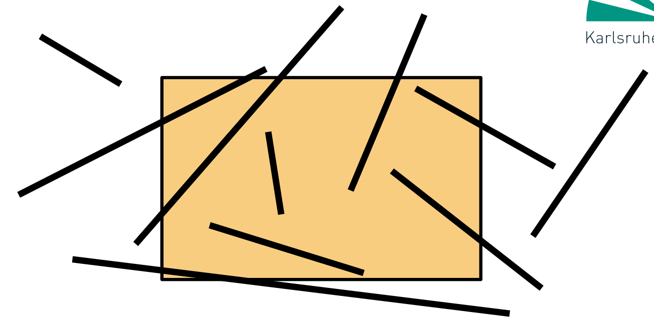


Die korrekten Strecken in I_{mid} lassen sich leicht mit einem Range Tree anstelle einfacher Listen finden!

Satz 1: Sei S eine Menge achsenparalleler Strecken in der Ebene. Alle k Strecken, die ein achsenparalleles Rechteck R schneiden lassen sich in $O(\log^2(n) + k)$ Zeit finden. Die Datenstruktur benötigt $O(n \log n)$ Platz und Aufbauzeit.

Beliebige Strecken

Daten in Landkarten enthalten Strecken beliebiger Orientierung.



Problemstellung:

Gegeben n disjunkte Strecken und ein achsenparalleles Rechteck $R = [x, x'] \times [y, y']$ finde alle Strecken, die R schneiden.

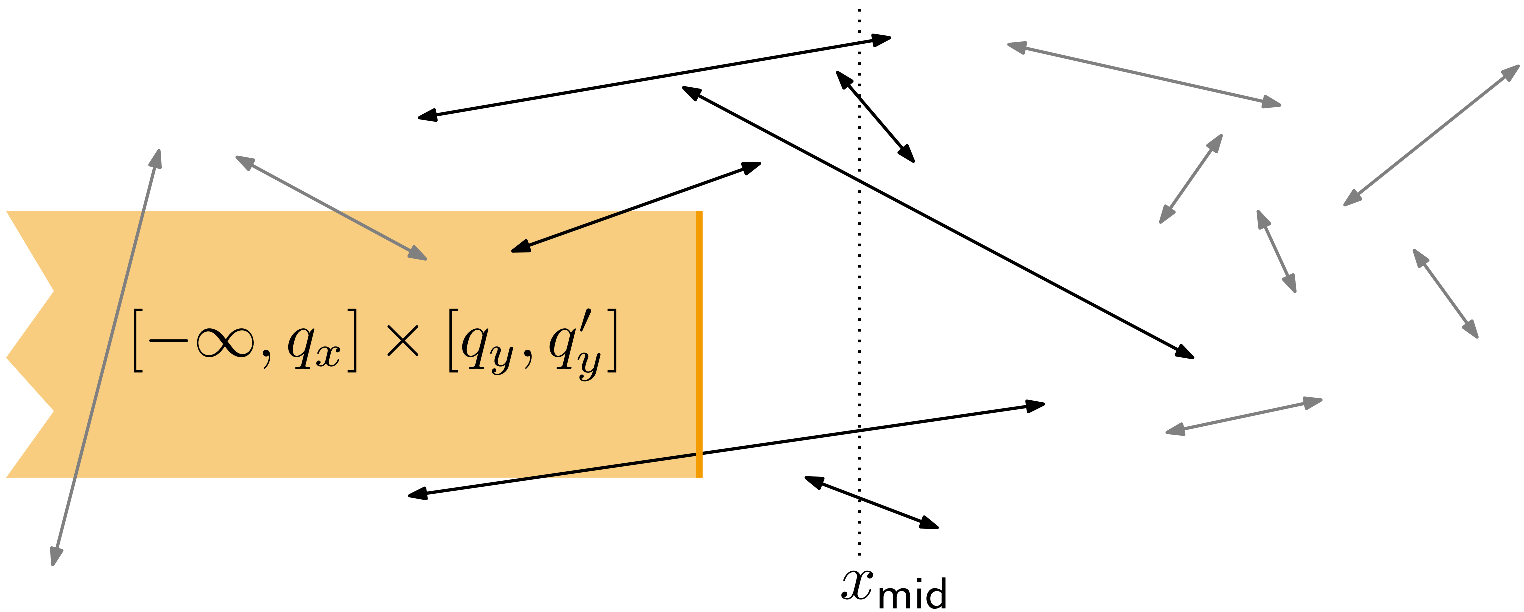
Wie könnte man hier vorgehen?

Fall 1: ≥ 1 Endpunkt in $R \rightarrow$ Range Tree benutzen

Fall 2: beide Endpunkte $\notin R \rightarrow$ schneiden mindestens eine Kante von R

Zerlegung in Elementarintervalle

Intervall-Bäume helfen diesmal nicht wirklich



Gleiches 1d-Basisproblem:

Gegeben n Intervalle $I = \{[x_1, x'_1], [x_2, x'_2], \ldots, [x_n, x'_n]\}$ und einen Punkt q_x, finde alle Intervalle, die q_x enthalten.

- sortiere alle x_i und x'_i in Liste $p_1, \ldots, p_{2n}$
- bilde sortierte Elementarintervalle $(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, [p_{2n}, p_{2n}], (p_{2n}, \infty)$

Segment-Bäume

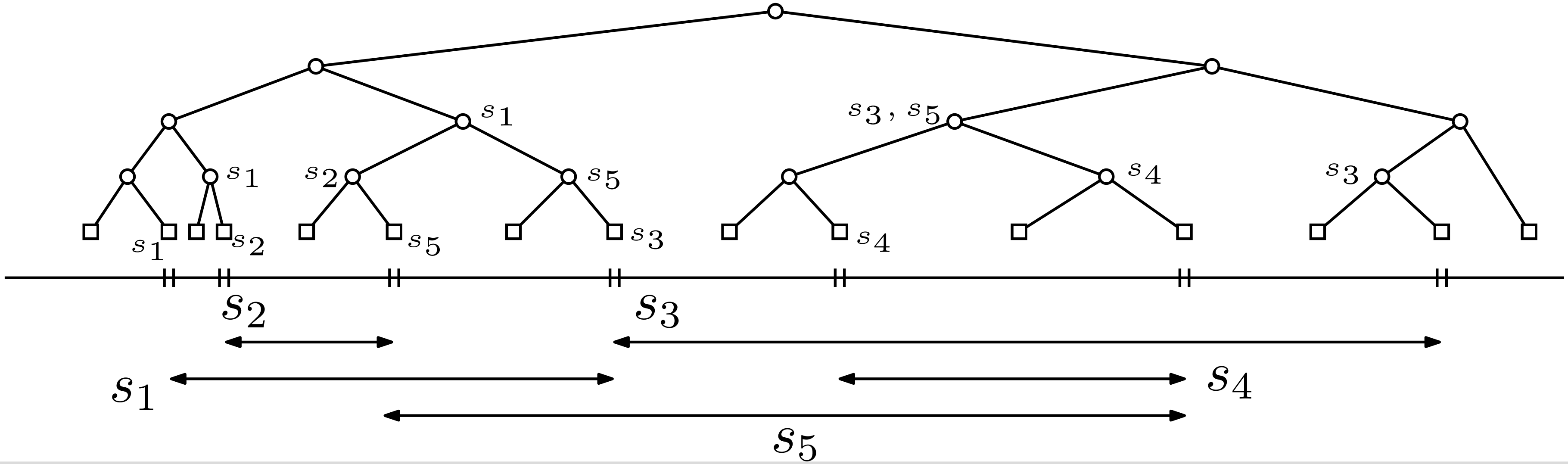
Idee für Datenstruktur:

- erstelle binären Suchbaum mit Elementarintervalle in den Blättern
- für alle Punkte q_x in einem Elementarintervall ist die Antwort gleich
- Blatt μ für Elementarintervall $e(\mu)$ speichert Intervallmenge $I(\mu)$
- Abfrage benötigt $O(\log n + k)$ Zeit

Problem: Speicherbedarf ist im schlimmsten Fall quadratisch

$\rightarrow$ speichere Intervalle so hoch wie möglich im Baum

- Knoten v repräsentiert Intervall $e(v) = e(lc(v)) \cup e(rc(v))$
- Eingabeintervall $s_i \in I(v) \Leftrightarrow e(v) \subseteq s_i$ und $e(\text{parent}(v)) \not\subseteq s_i$



Eigenschaften Segment-Bäume

Lemma 3: Ein Segment-Baum für n Intervalle benötigt $O(n \log n)$ Platz und kann in $O(n \log n)$ Zeit erstellt werden.

Beweisskizze:

InsertSegmentTree($v, [x, x']$)

if $e(v) \subseteq [x, x']$ **then**
  speichere $[x, x']$ in $I(v)$
else
  if $e(lc(v)) \cap [x, x'] \neq \emptyset$ **then**
    InsertSegmentTree($lc(v)), [x, x']$)
  if $e(rc(v)) \cap [x, x'] \neq \emptyset$ **then**
    InsertSegmentTree($rc(v)), [x, x']$)

Abfrage in Segment-Bäumen

QuerySegmentTree(v, q_x)

```
gib alle Intervalle in I(v) aus
if v kein Blatt then
    if q_x ∈ e(lc(v)) then
        QuerySegmentTree(lc(v), q_x)
    else
        QuerySegmentTree(rc(v), q_x)
```

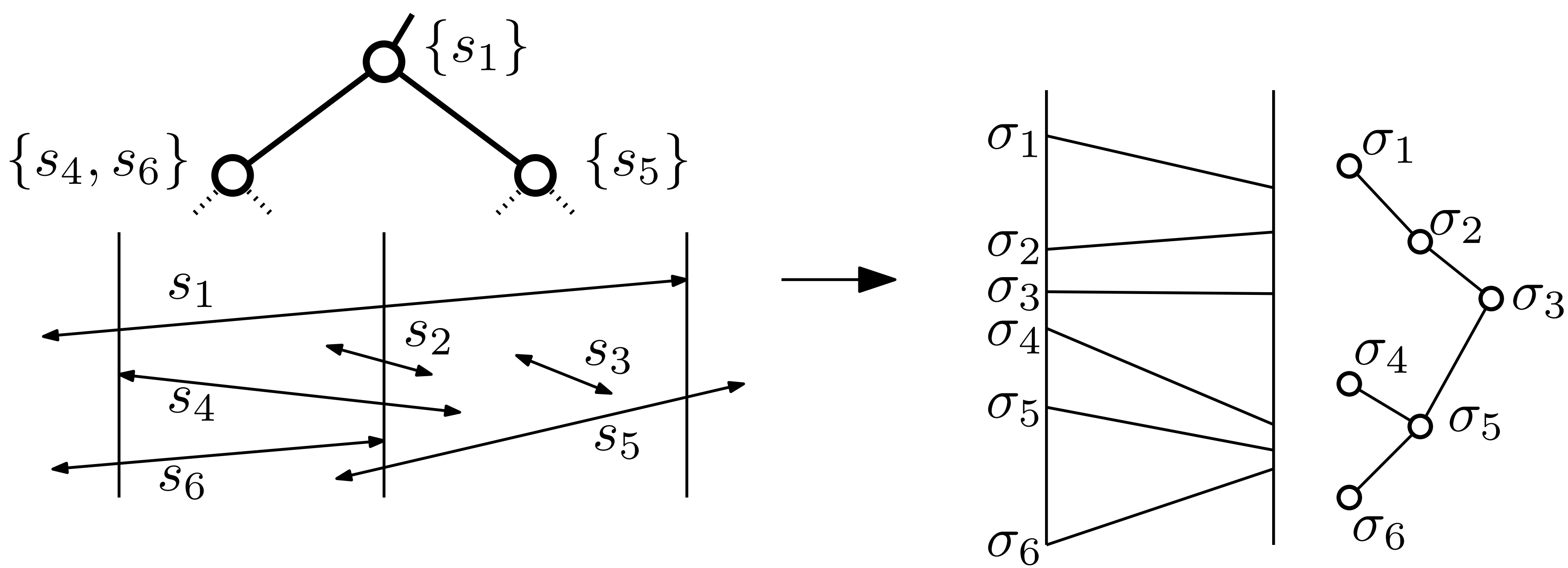
Lemma 4: Alle k Intervalle, die einen Punkt q_x enthalten können mit einem Segment-Baum in $O(\log n + k)$ Zeit ausgegeben werden.

Lemma 4 liefert zunächst das gleiche Resultat wie Intervall-Bäume.
Was ist anders?

$\rightarrow$ *alle* an einem positiven Knoten v gespeicherten Intervalle enthalten q_x
– in Intervall-Bäumen muss man dort noch weitersuchen

Zurück zu beliebigen Strecken

- erstelle Segment-Baum für die x-Intervalle der Strecken
- jeder Knoten v entspricht vertikalem Streifen $e(v) \times \mathbb{R}$
- Strecke s ist in $I(v)$ gdw. s den Streifen von v kreuzt, aber nicht den Streifen von parent(v)
- an jedem Knoten v im Suchpfad für vertikale Strecke $s' = q_x \times [q_y, q_y']$ kreuzen alle Strecken in $I(v)$ die x-Koordinate q_x
- suche Strecken im Streifen, die s' kreuzen über vertikal sortierten binären Hilfssuchbaum



Zusammenfassung

Satz 2: Sei S eine Menge im Inneren disjunkter Strecken in der Ebene. Alle k Strecken, die ein achsenparalleles Rechteck R schneiden lassen sich in $O(k + \log^2 n)$ Zeit finden. Die Datenstruktur benötigt $O(n \log n)$ Platz und $O(n \log^2 n)$ Aufbauzeit.

Bemerkung:
Die Zeit zum Aufbau der Datenstruktur kann auf $O(n \log n)$ verkürzt werden ($\rightarrow$ Übungsblatt)

Diskussion

Platzbedarf von Intervall-Bäumen

Als Hilfsstruktur in den Intervall-Bäumen hatten wir Range Trees mit $O(n \log n)$ Platz gesehen. Das lässt sich mit modifizierten Heaps auf $O(n)$ senken. (Kap. 10.2 in [BCKO'08])

Wie kann man effizient die geschnittenen Strecken zählen?

Segment- und Intervall-Bäume unterstützen durch leichte Modifikationen auch effiziente Zählabfragen (unabhängig von k) $\rightarrow$ Übungsblatt

Was macht man bei nicht-rechteckigen Abfrageregionen?

Durch Triangulierung eines Abfragepolygons kann man das Problem auf Dreiecke zurückführen. Datenstrukturen dazu finden sich z.B. in Kap. 16 von [BCKO'08].