Übung Algorithmische Geometrie

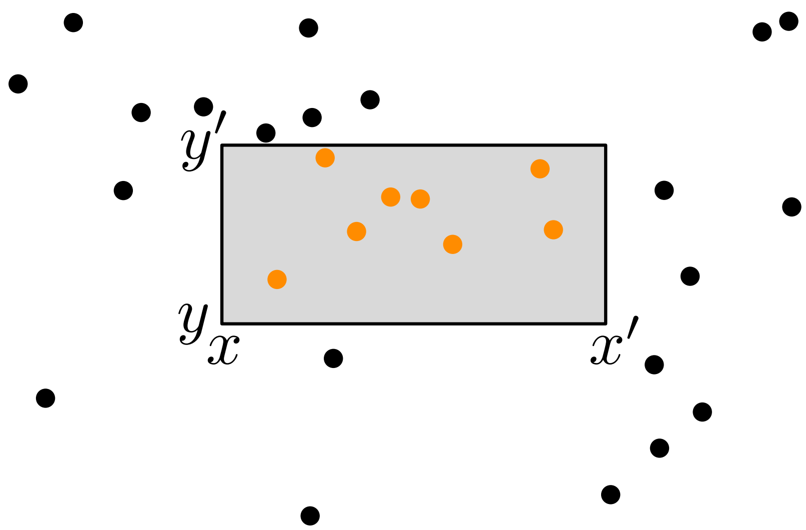
Bereichsabfragen II

LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Benjamin Niedermann

28.05.2014

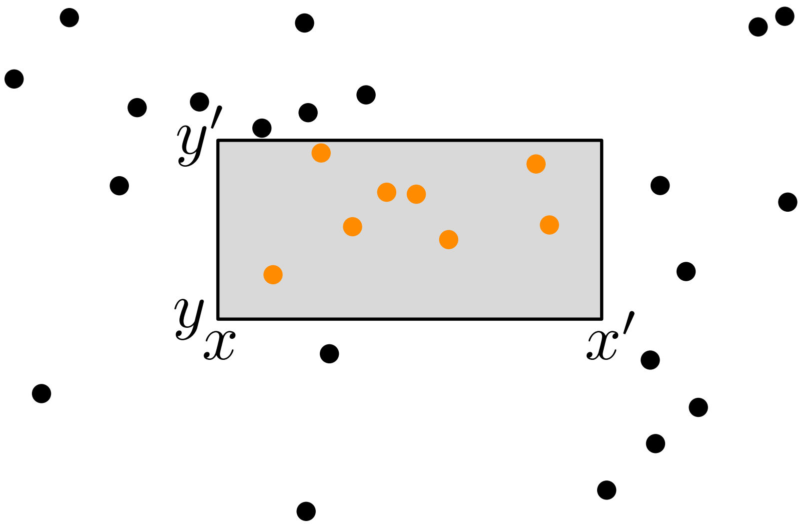
Objekttypen in Bereichsabfragen

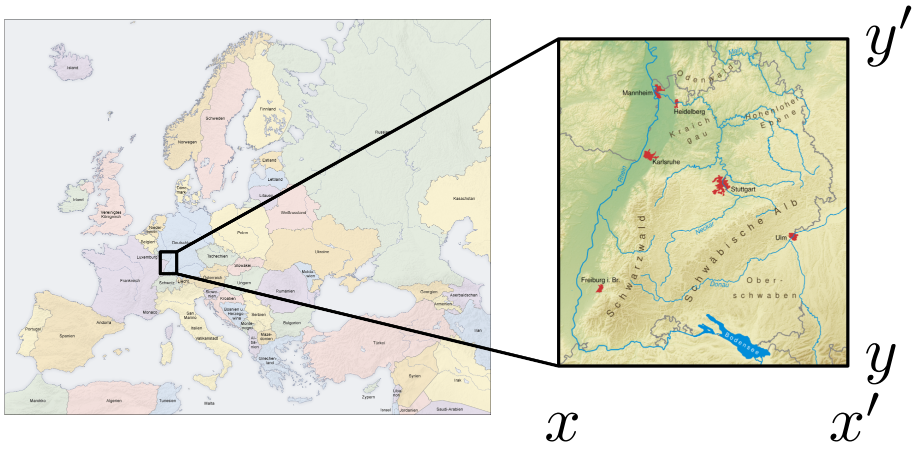


Bisher betrachteter Fall

- 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

Objekttypen in Bereichsabfragen





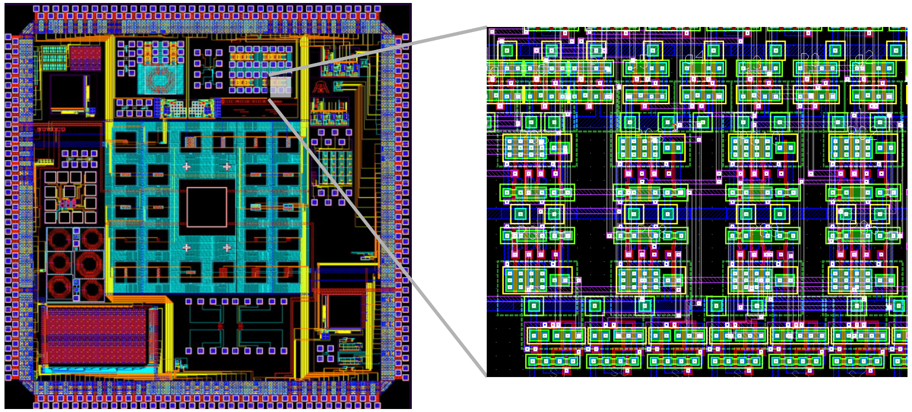
Bisher betrachteter Fall

- 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

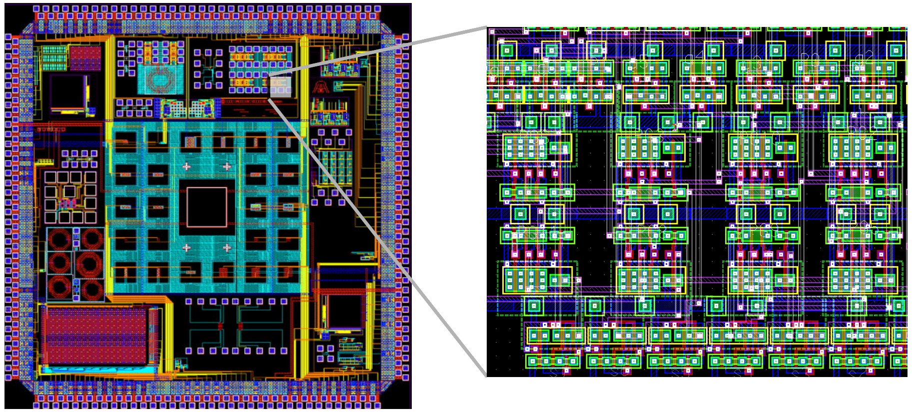
- Eingabe: Streckenmenge S (hier $S \subset \mathbb{R}^2$)
- Ausgabe: alle Strecken aus $S \cap [x, x'] \times [y, y']$
- Datenstrukturen: ?

Achsenparallele Strecken



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

Achsenparallele Strecken

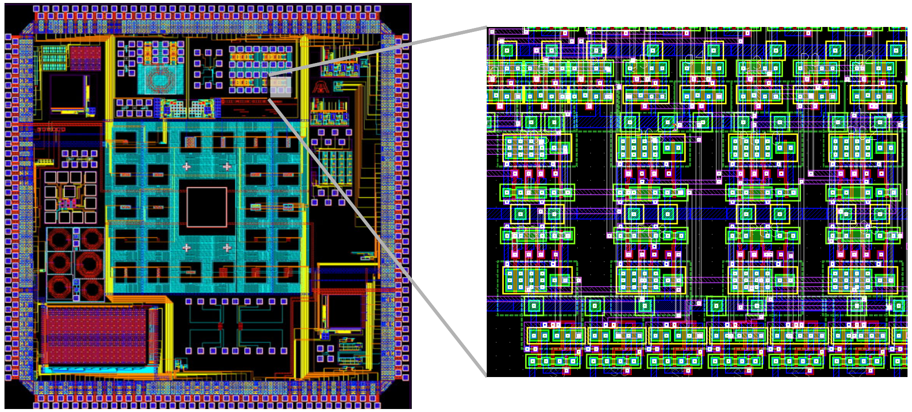


Spezialfall z.B. im VLSI Design: alle Strecken achsenparallel

Problemstellung:

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

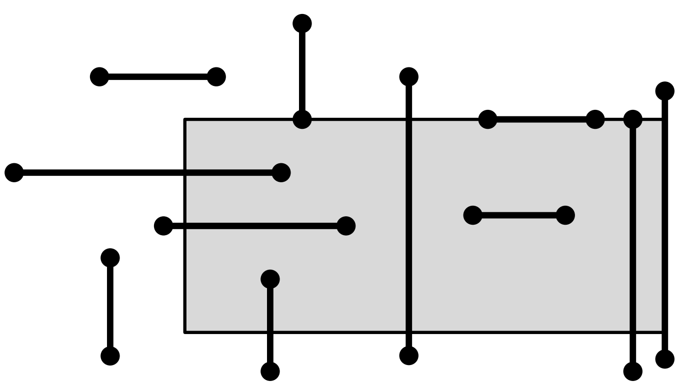
Achsenparallele Strecken



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

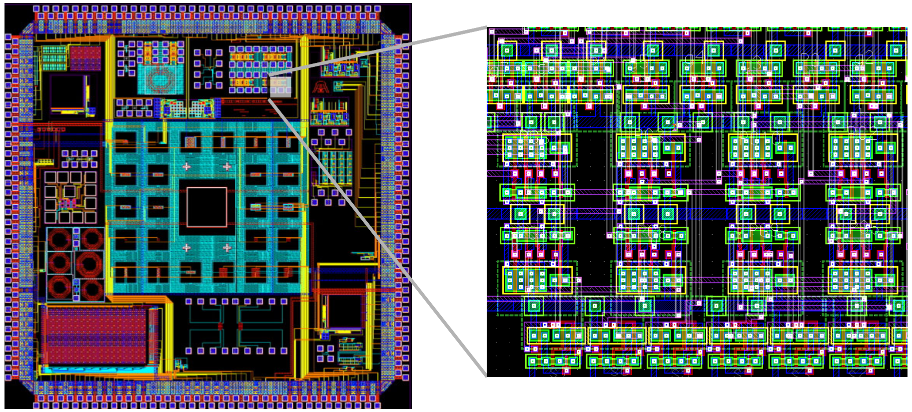
Problemstellung:

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



Wie könnte man vorgehen?

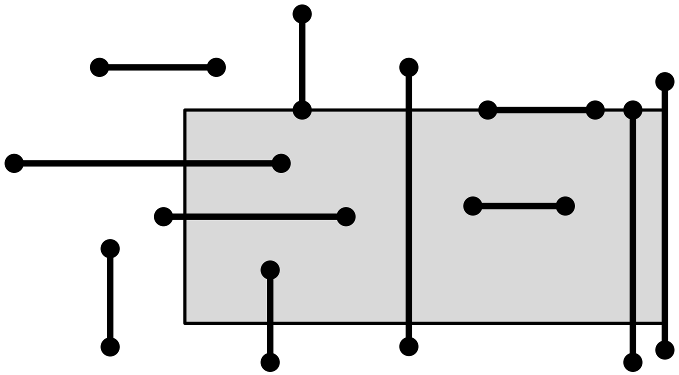
Achsenparallele Strecken



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.



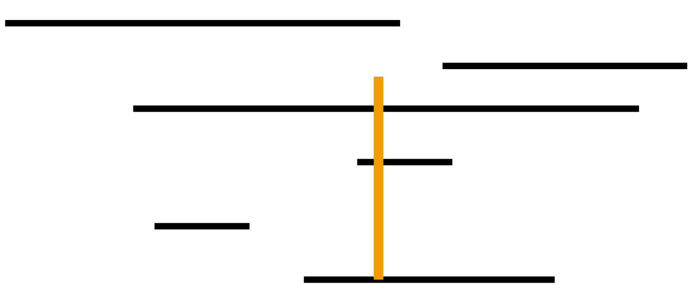
Fall 1: ≥ 1 Endpunkt in R
$\rightarrow$ Range Tree benutzen

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 s finde alle Strecken in H, die s schneiden (vertikale Strecken und horizontale Query analog).



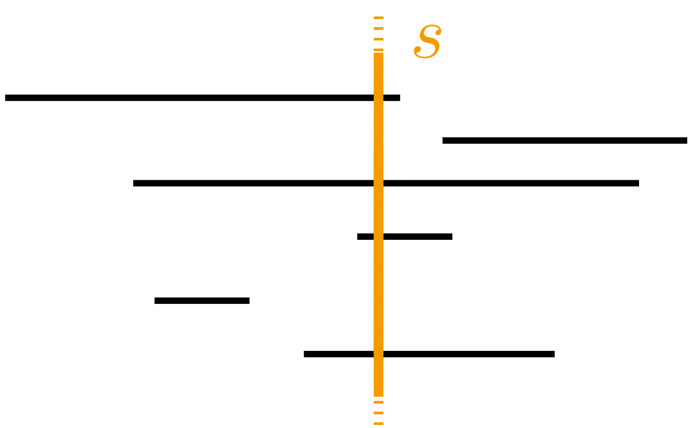
Fall 2 im Detail

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.



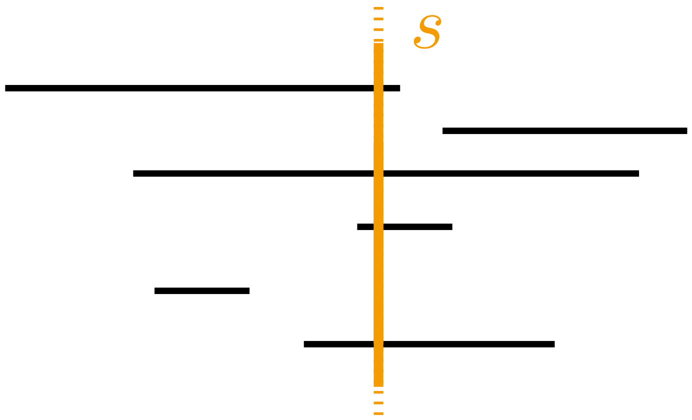
Fall 2 im Detail

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

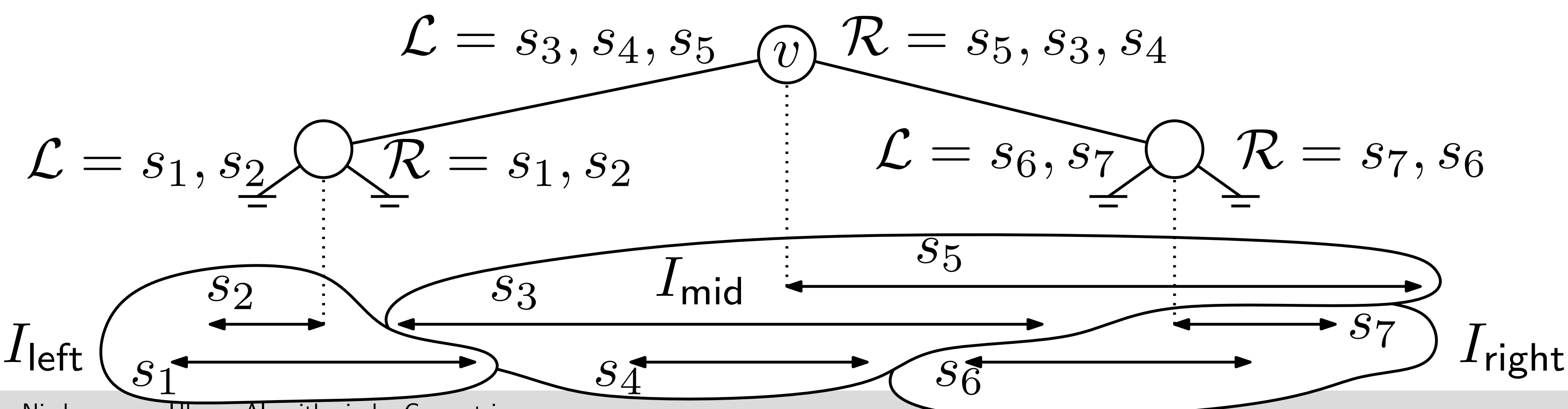
Konstruktion eines Intervall-Baums $\mathcal{T}$

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

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

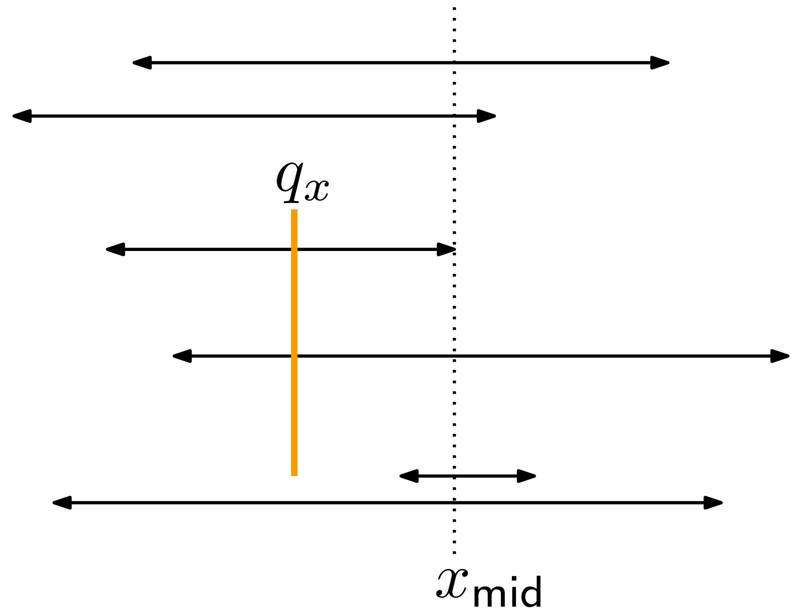
$\mathcal{T}$ besteht aus Knoten v für x_{mid}

- 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}



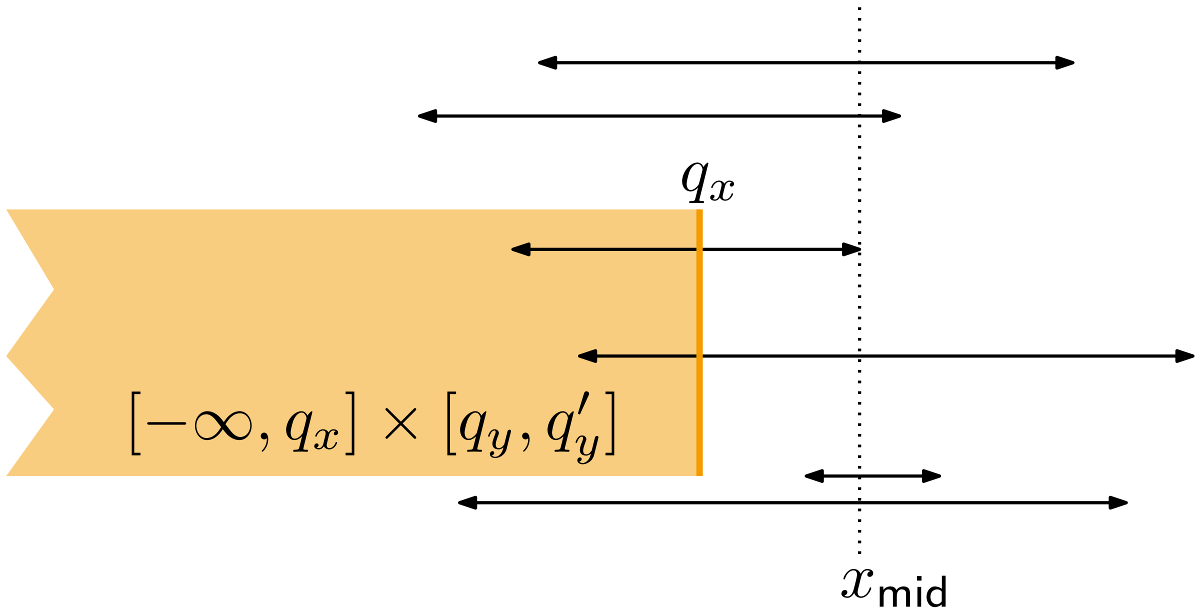
Von Geraden zu Strecken

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



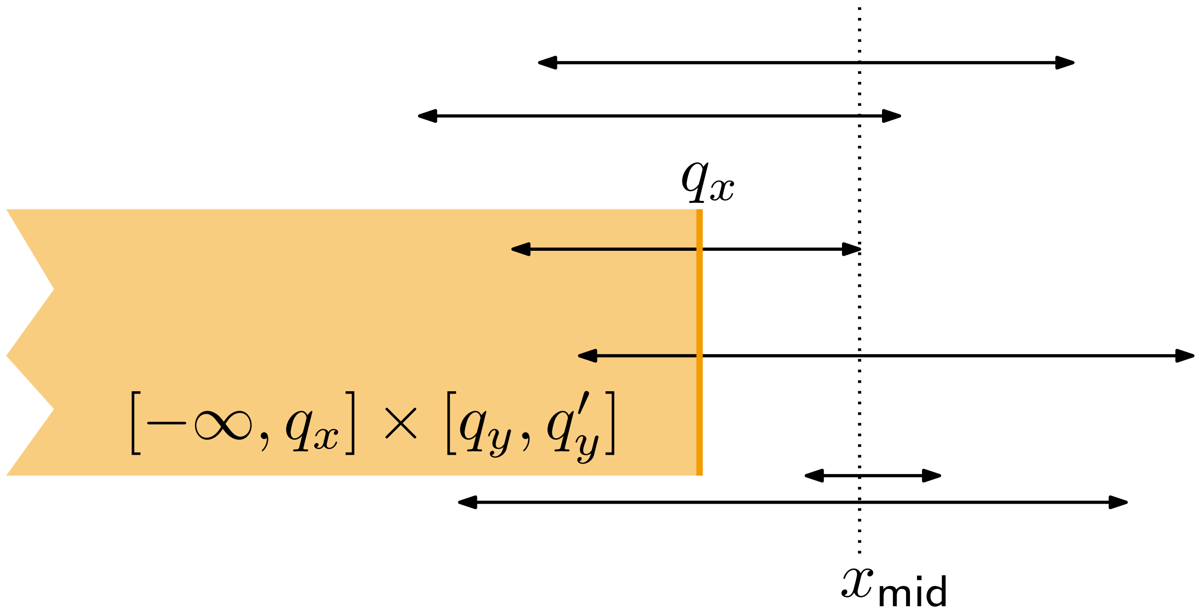
Von Geraden zu Strecken

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



Von Geraden zu Strecken

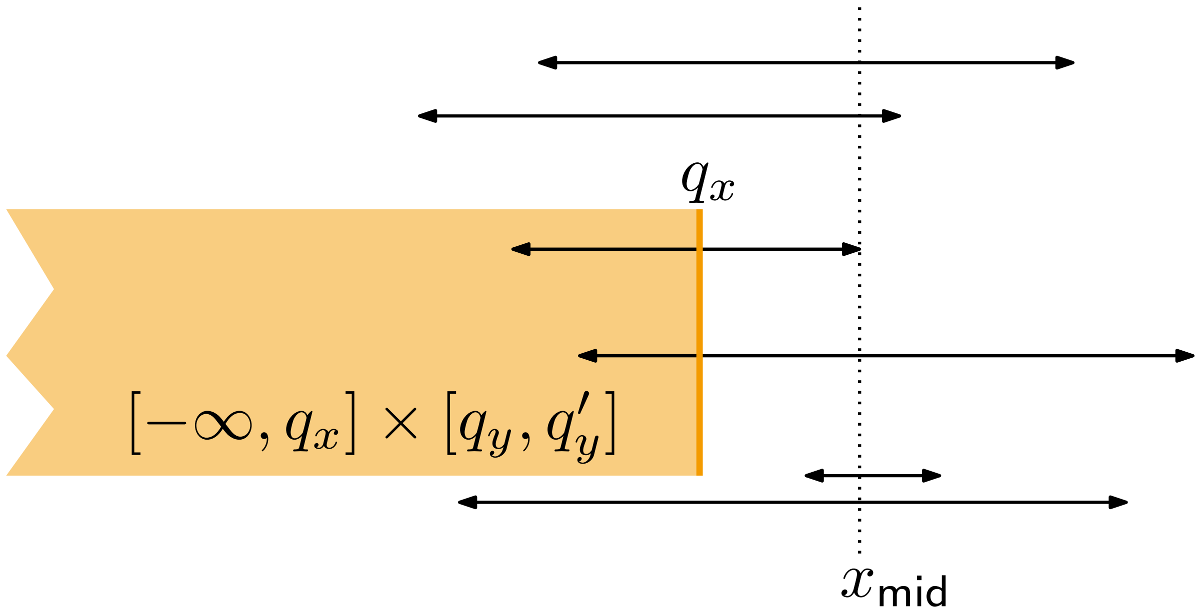
Wie lässt sich die Idee des Intervall-Baums abwandeln für Strecken $q_x \times [q_y, q_y']$ statt Geraden $x = q_x$?



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

Von Geraden zu Strecken

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

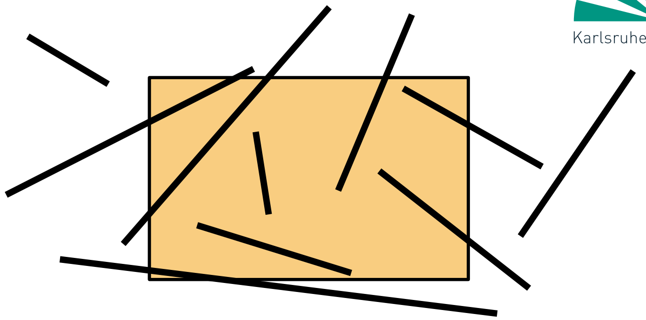


Die korrekten Strecken in I_{mid} lassen sich leicht mit einem Range Tree anstelle einfacher Listen finden!

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.

Aufgabe 1: Wie kann man Intervall-Bäume verwenden?

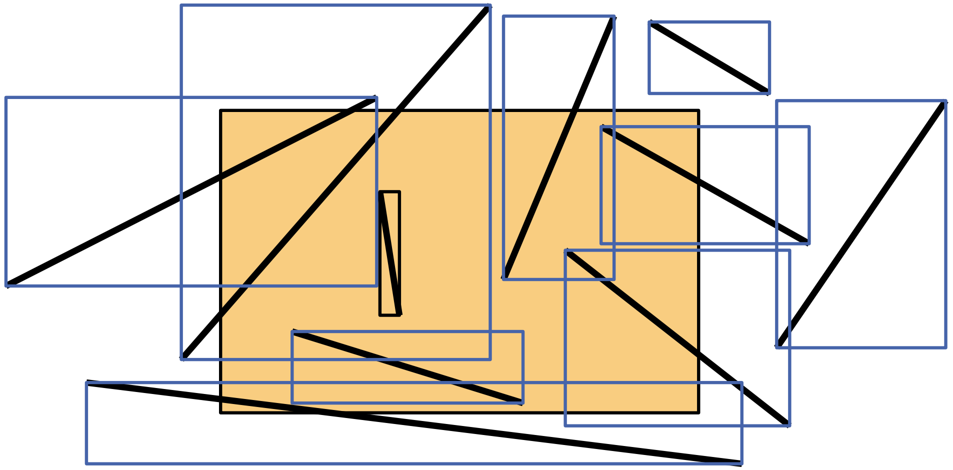
Lösung

Verwende Bounding Box der Segmente

1. Intervall-Baum auf Strecken der Bounding-Boxes.

2. Wenn Segmente von Bounding-Box Anfragebereich schneiden:

> Überprüfe ob enthaltene Strecke Anfragebereich schneidet.



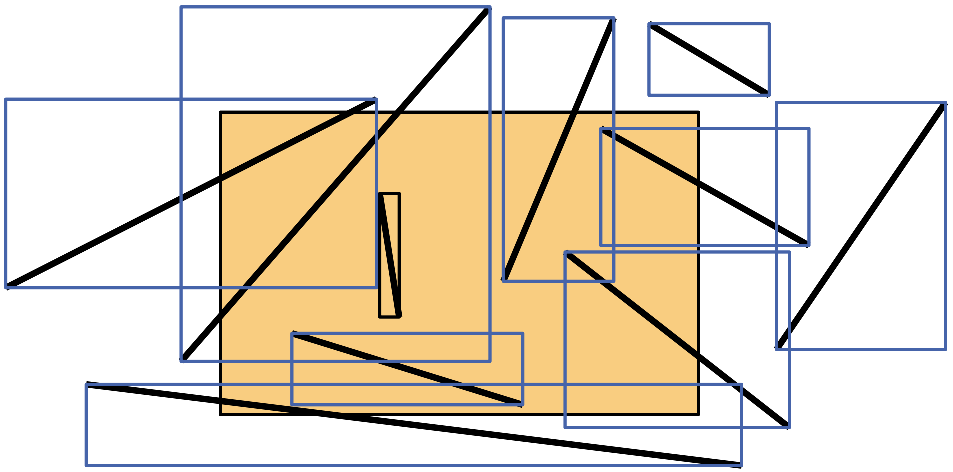
Lösung

Verwende Bounding Box der Segmente

1. Intervall-Baum auf Strecken der Bounding-Boxes.

2. Wenn Segmente von Bounding-Box Anfragebereich schneiden:

   Überprüfe ob enthaltene Strecke Anfragebereich schneidet.



Korrekt, denn:

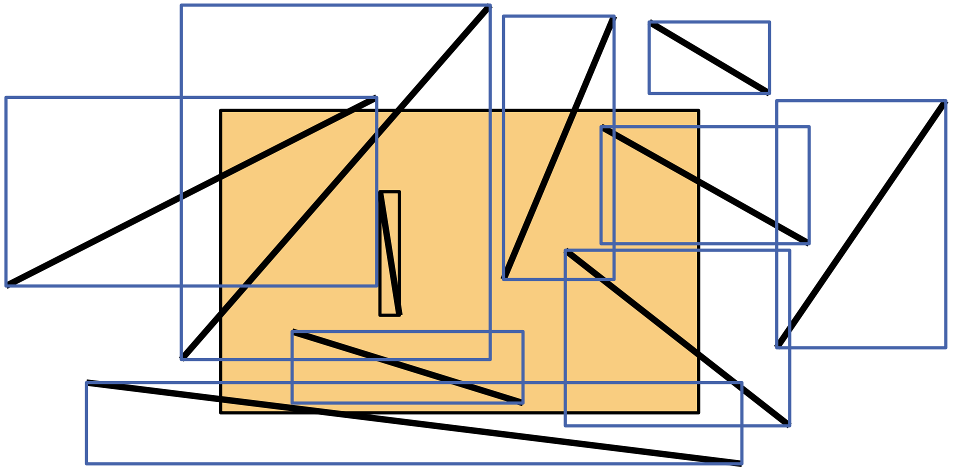
Lösung

Verwende Bounding Box der Segmente

1. Intervall-Baum auf Strecken der Bounding-Boxes.

2. Wenn Segmente von Bounding-Box Anfragebereich schneiden:

   Überprüfe ob enthaltene Strecke Anfragebereich schneidet.



Korrekt, denn:

Wenn Strecke Anfragebereich schneidet, dann auch entsprechende Bounding-Box.

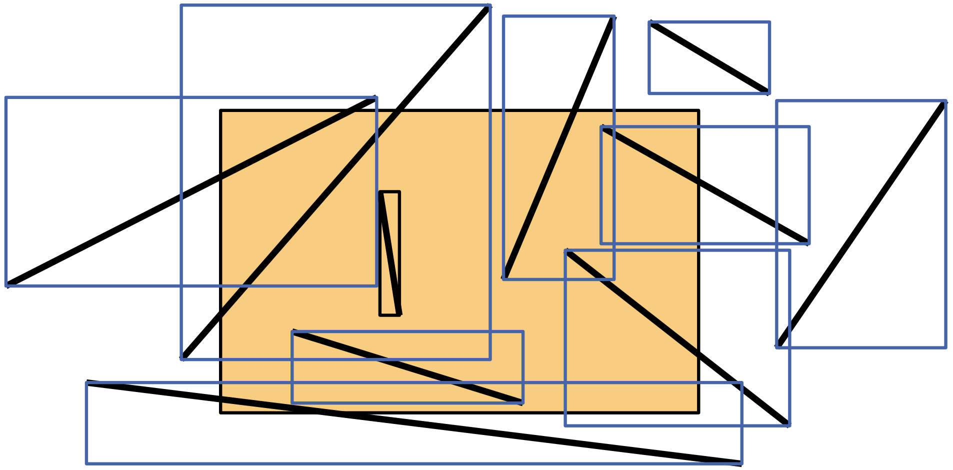
Lösung

Verwende Bounding Box der Segmente

1. Intervall-Baum auf Strecken der Bounding-Boxes.

2. Wenn Segmente von Bounding-Box Anfragebereich schneiden:

   Überprüfe ob enthaltene Strecke Anfragebereich schneidet.



Korrekt, denn:

Wenn Strecke Anfragebereich schneidet, dann auch entsprechende Bounding-Box.

Problem:

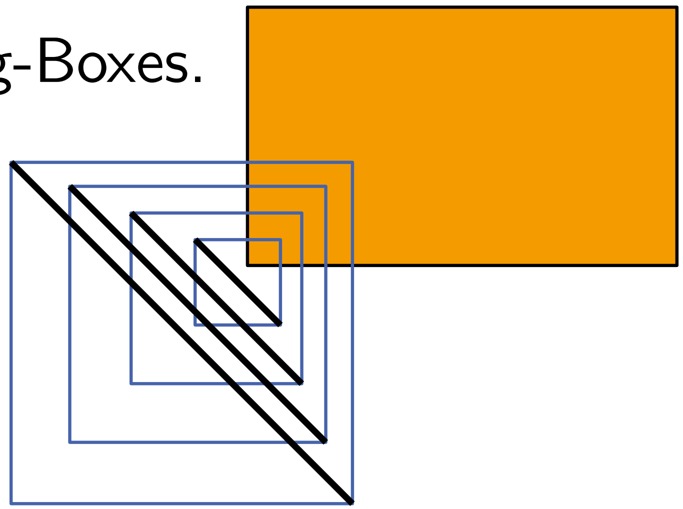
Lösung

Verwende Bounding Box der Segmente

1. Intervall-Baum auf Strecken der Bounding-Boxes.

2. Wenn Segmente von Bounding-Box Anfragebereich schneiden:

   Überprüfe ob enthaltene Strecke Anfragebereich schneidet.



Korrekt, denn:

Wenn Strecke Anfragebereich schneidet, dann auch entsprechende Bounding-Box.

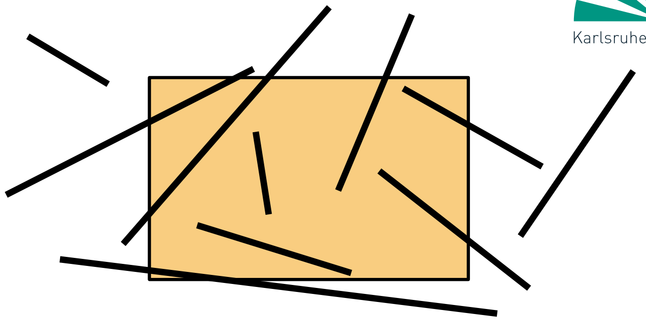
Problem: Es können mehr Strecken als nötig betrachtet werden.

denn es gilt nicht:

~~Wenn Bounding-Box Anfragebereich schneidet, dann auch enthaltene Strecke.~~

Beliebige Strecken

Daten in Landkarten enthalten Strecken beliebiger Orientierung.



Problemstellung:

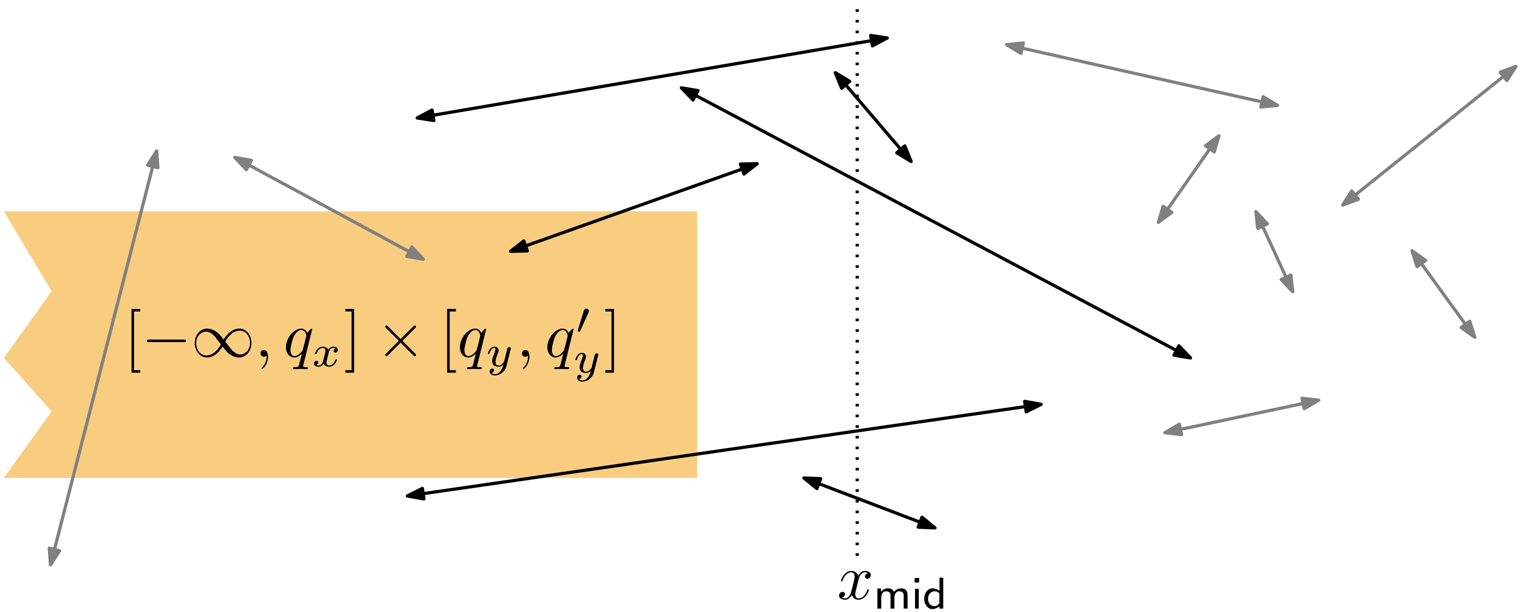
Gegeben n disjunkte Strecken und ein achsenparalleles Rechteck $R = [x, x'] \times [y, y']$ finde alle Strecken, die R schneiden.

Wie könnte man hier vorgehen?

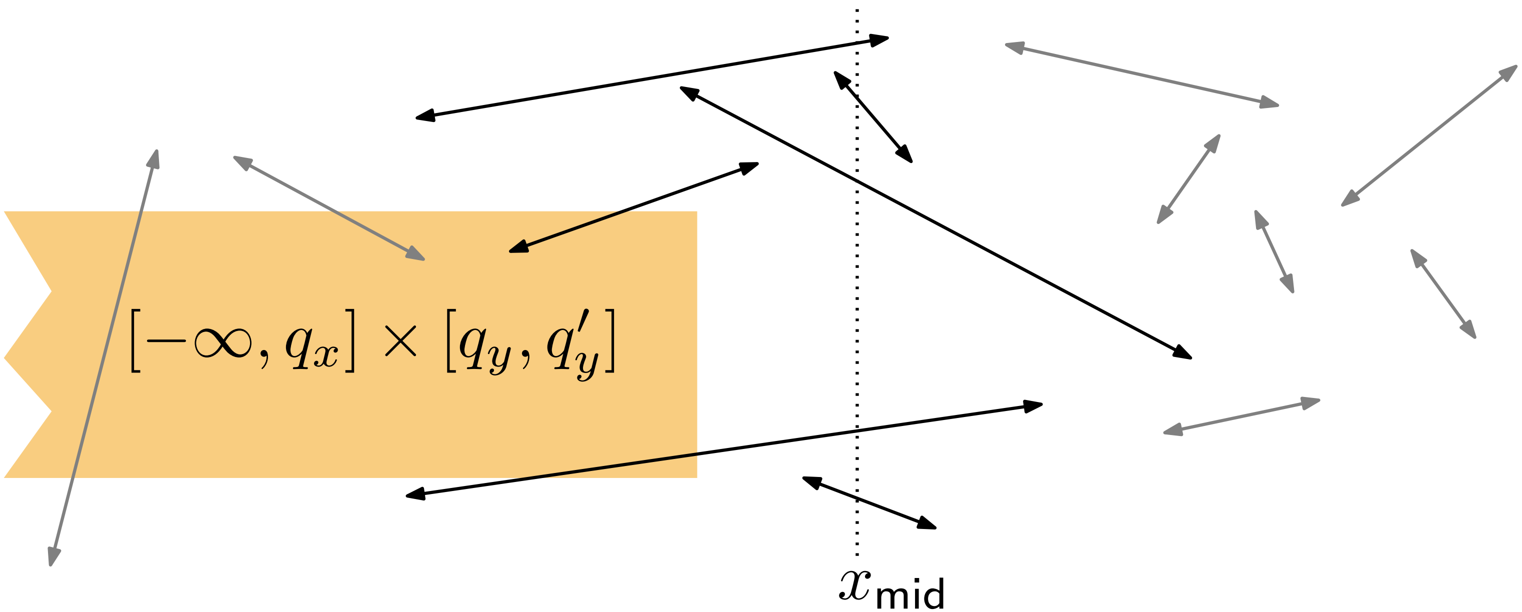
Fall 1: ≥ 1 Endpunkt in $R \rightarrow$ Range Tree benutzen

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

Zerlegung in Elementarintervalle



Zerlegung in Elementarintervalle



Gleiches 1d-Basisproblem:

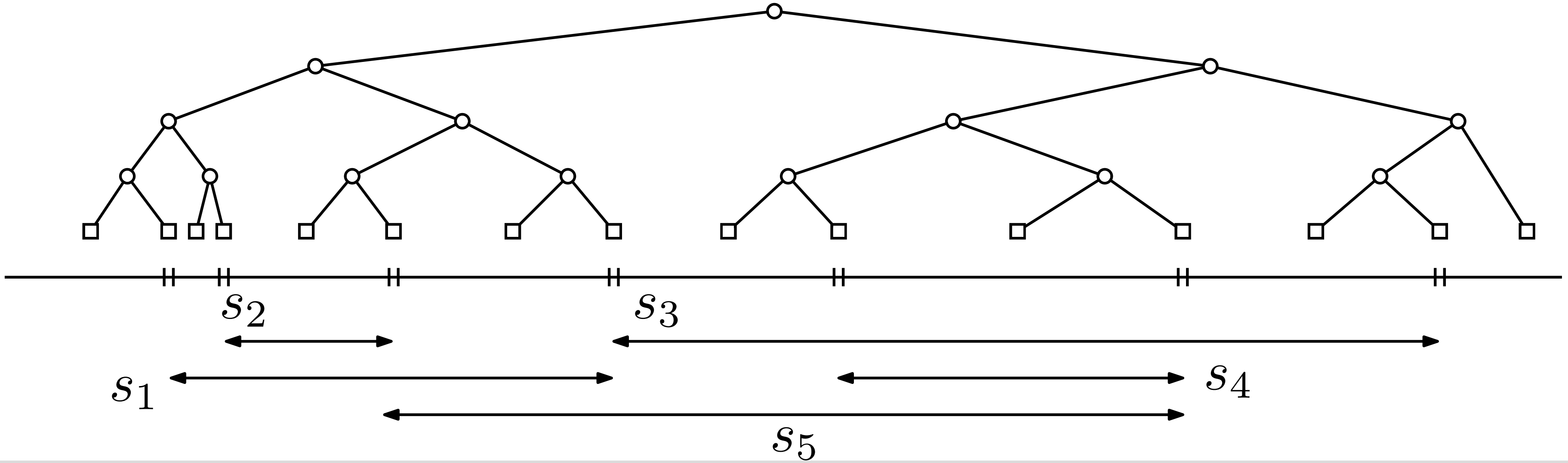
Gegeben n Intervalle $I = \{[x_1, x'_1], [x_2, x'_2], \ldots, [x_n, x'_n]\}$ und einen Punkt q_x, finde alle Intervalle, die q_x enthalten.

- 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

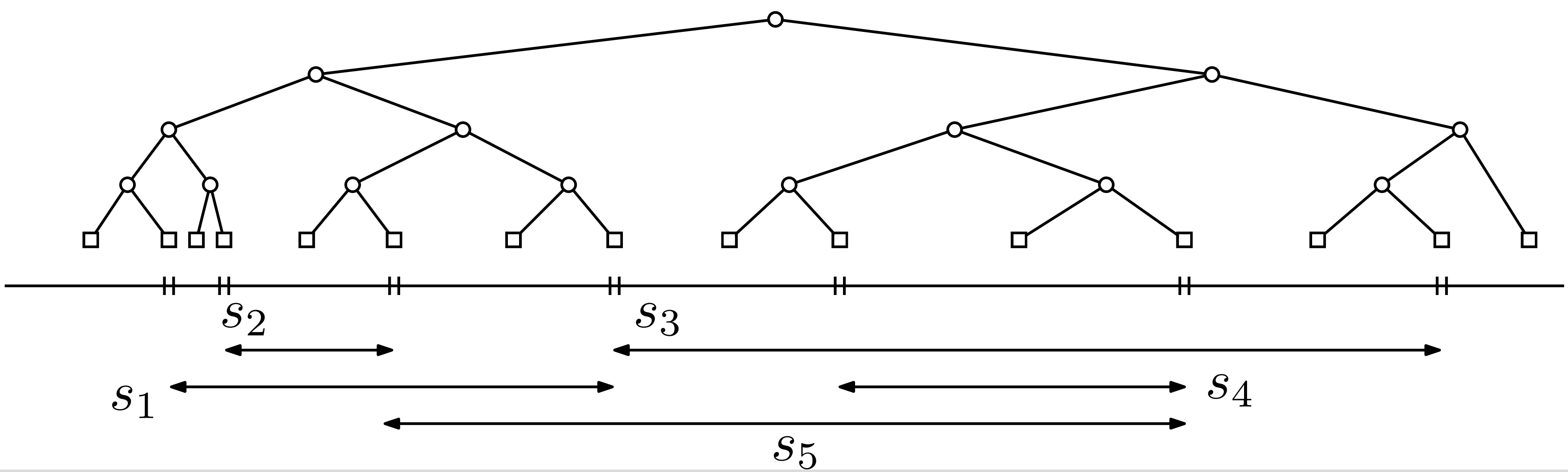


Segment-Bäume

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 dabei?



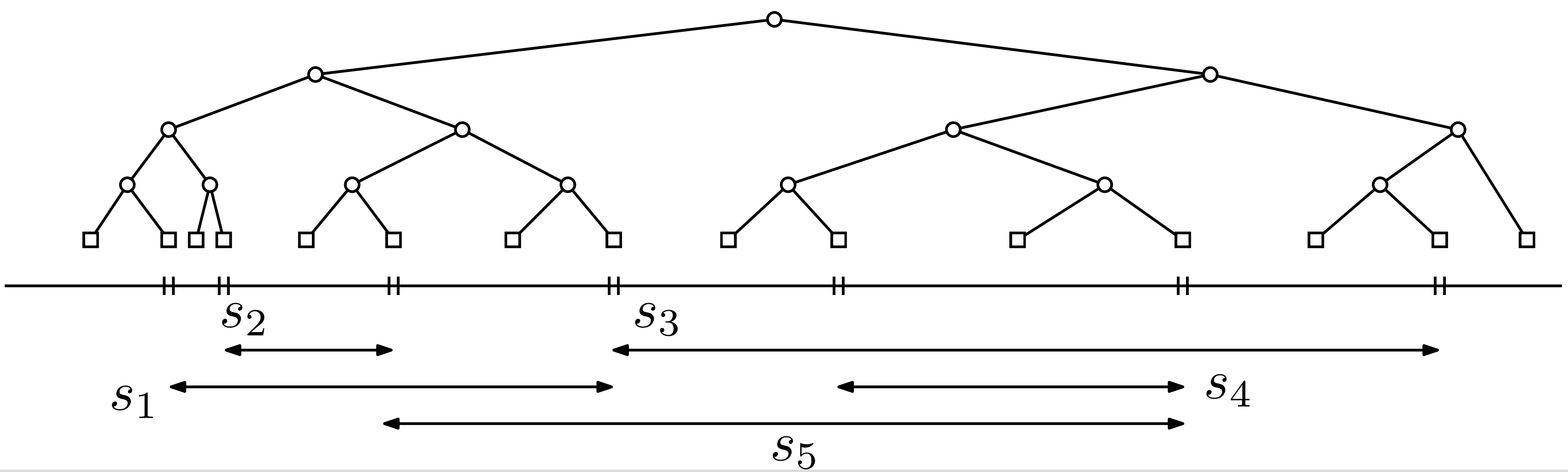
Segment-Bäume

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



Segment-Bäume

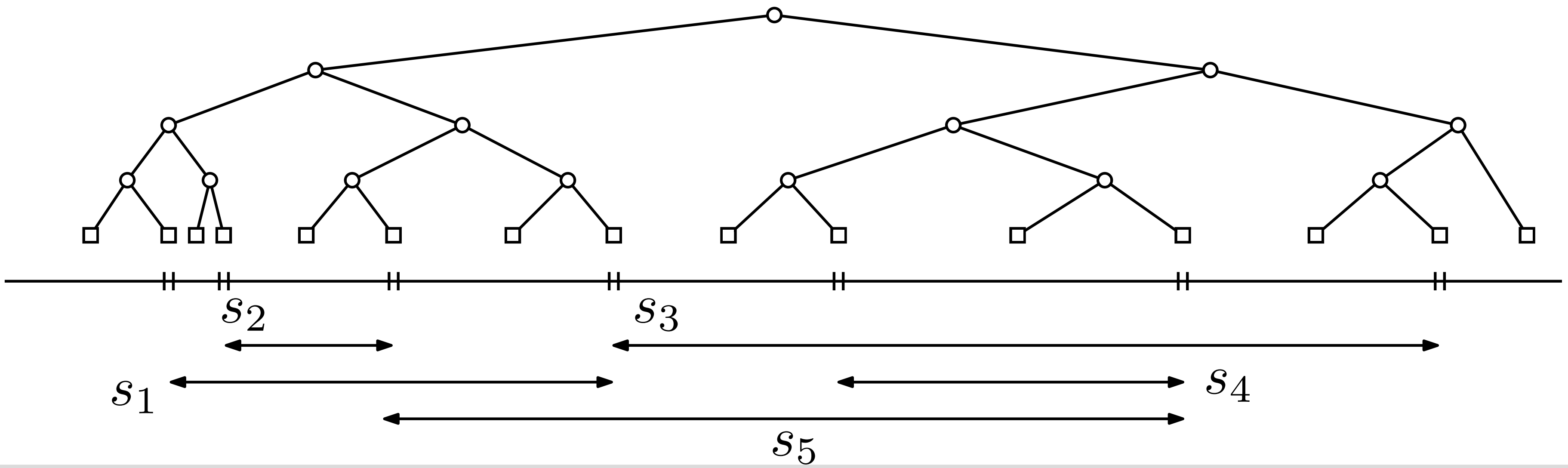
Idee für Datenstruktur:

- 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 $e(\text{parent}(v)) \not\subseteq s_i$



Segment-Bäume

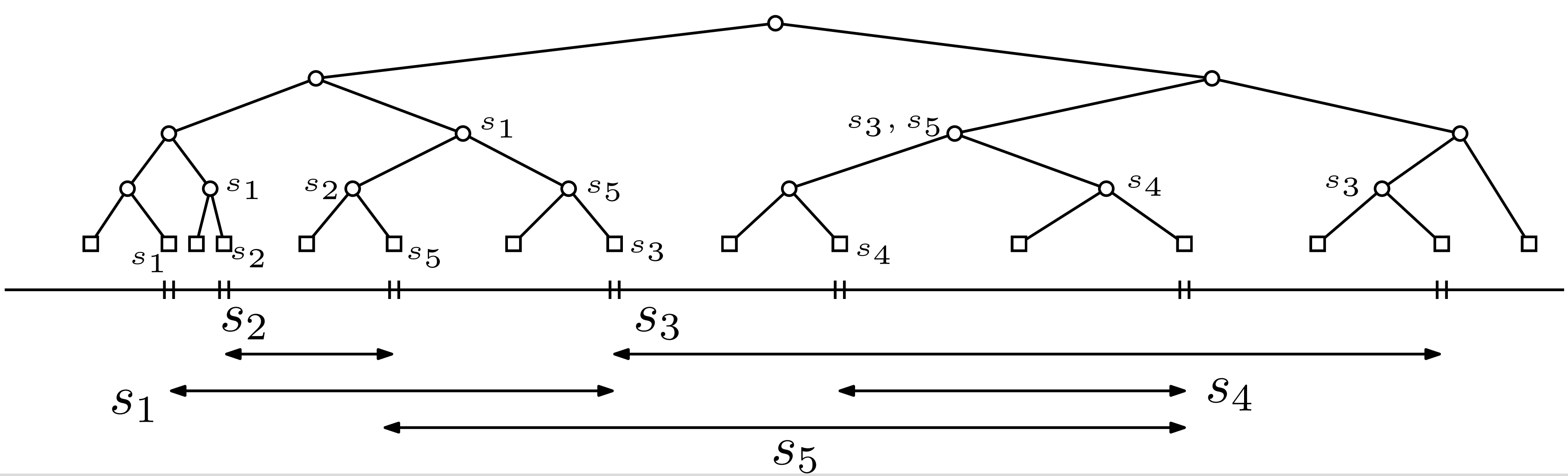
Idee für Datenstruktur:

- 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

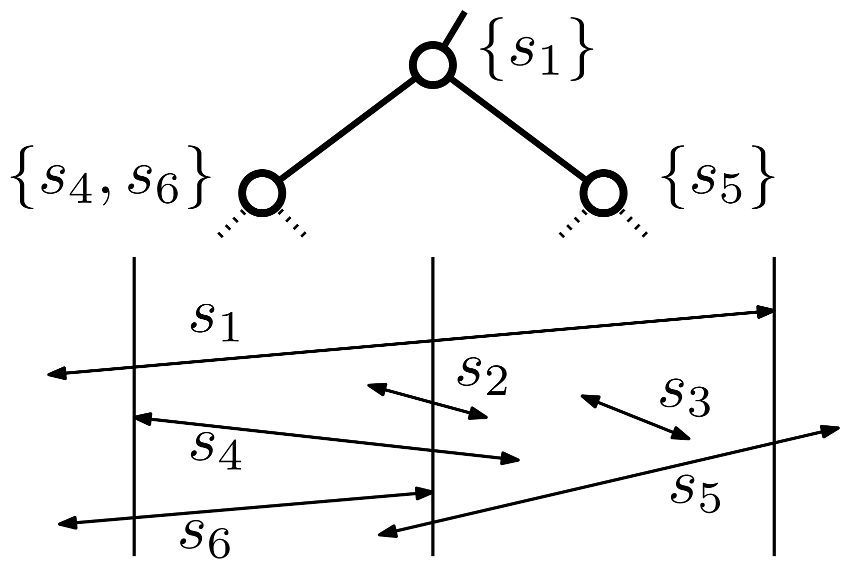
$\rightarrow$ speichere Intervalle so hoch wie möglich im Baum

- 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 $e(\text{parent}(v)) \not\subseteq s_i$



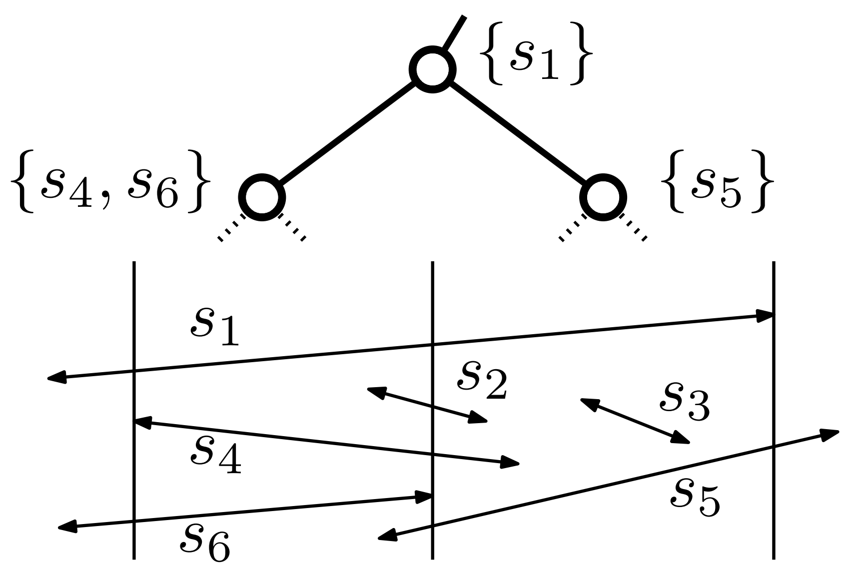
Zurück zu beliebigen Strecken

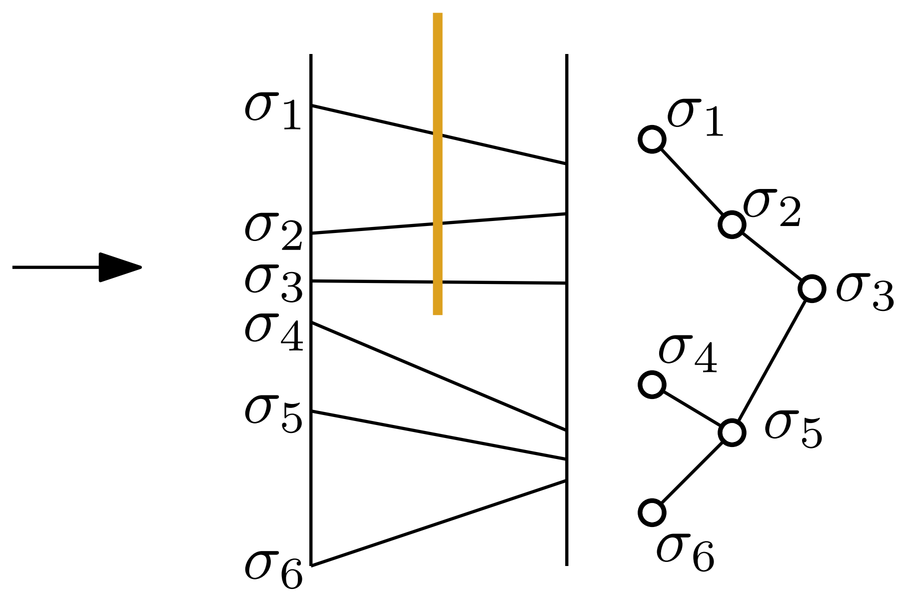
- erstelle Segment-Baum für die x-Intervalle der Strecken
- jeder Knoten v entspricht vertikalem Streifen $e(v) \times \mathbb{R}$
- Strecke s ist in $I(v)$ gdw. s den Streifen von v kreuzt, aber nicht den Streifen von parent(v)
- an jedem Knoten v im Suchpfad für vertikale Strecke $q_x \times [q_y,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



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 $q_x \times [q_y, q_y']$ 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.

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.

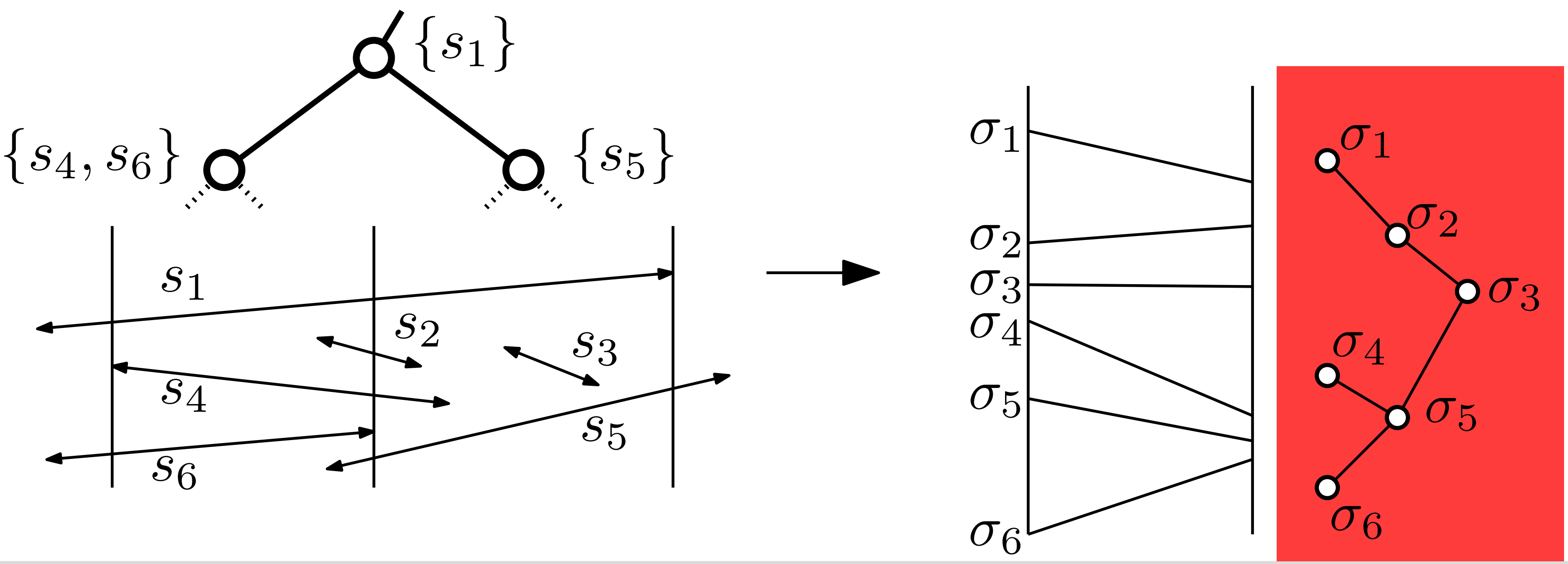
Bemerkung:
Die Zeit zum Aufbau der Datenstruktur kann auf $O(n \log n)$ verkürzt werden ($\rightarrow$ Übungsblatt)

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)

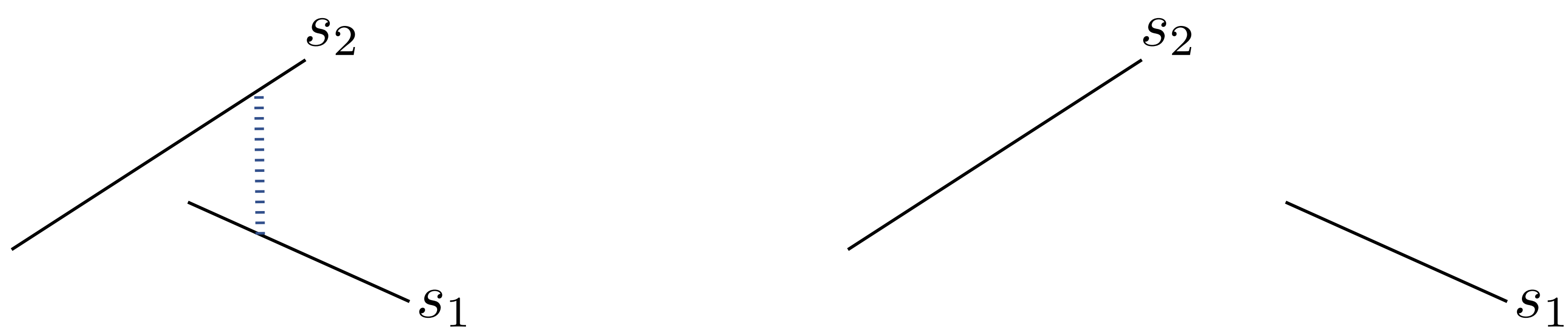
Problem: Aufbau der Hilfssuchbäume



Lösung

Seien s_1, s_2 zwei Strecken.

s_1 liegt *unterhalb* von s_2 ($s_1 \prec s_2$), falls es Punkte $p_1 \in s_1$ und $p_2 \in s_2$ mit $x(p_1) = x(p_2)$ und $y(p_1) < y(p_2)$ gibt.



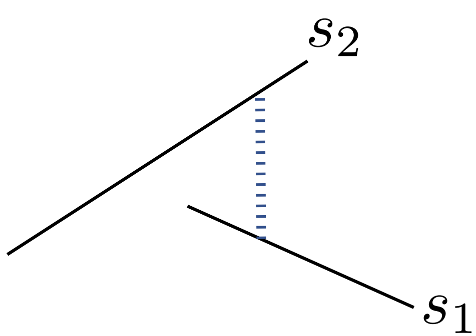
$s_1 \prec s_2$ $\qquad$ $s_1 \not\prec s_2$

1. Zeige, dass Relation $\prec$ auf S azyklisch ist.
   $\longrightarrow$ Es gibt topologische Sortierung
2. Berechne topologische Sortierung auf S
3. Verwende toplogische Sortierung, um Hilfssuchbäume zu erstellen.

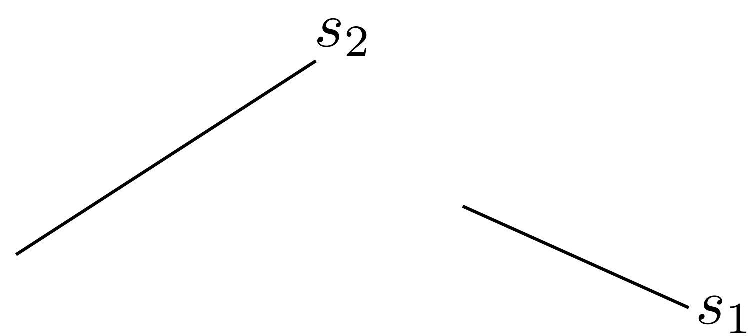
Lösung

Seien s_1, s_2 zwei Strecken.

s_1 liegt *unterhalb* von s_2 ($s_1 \prec s_2$), falls es Punkte $p_1 \in s_1$ und $p_2 \in s_2$ mit $x(p_1) = x(p_2)$ und $y(p_1) < y(p_2)$ gibt.



$s_1 \prec s_2$



$s_1 \not\prec s_2$

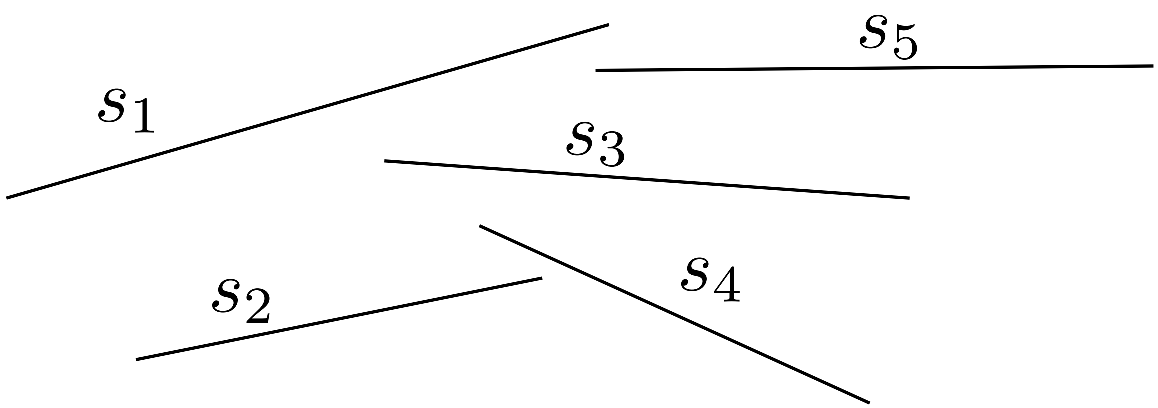
1. Zeige, dass Relation $\prec$ auf S azyklisch ist.

   $\longrightarrow$ Es gibt topologische Sortierung

2. Berechne topologische Sortierung auf S

3. Verwende toplogische Sortierung, um Hilfssuchbäume zu erstellen.

Berechnung der topologischen Sortierung



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

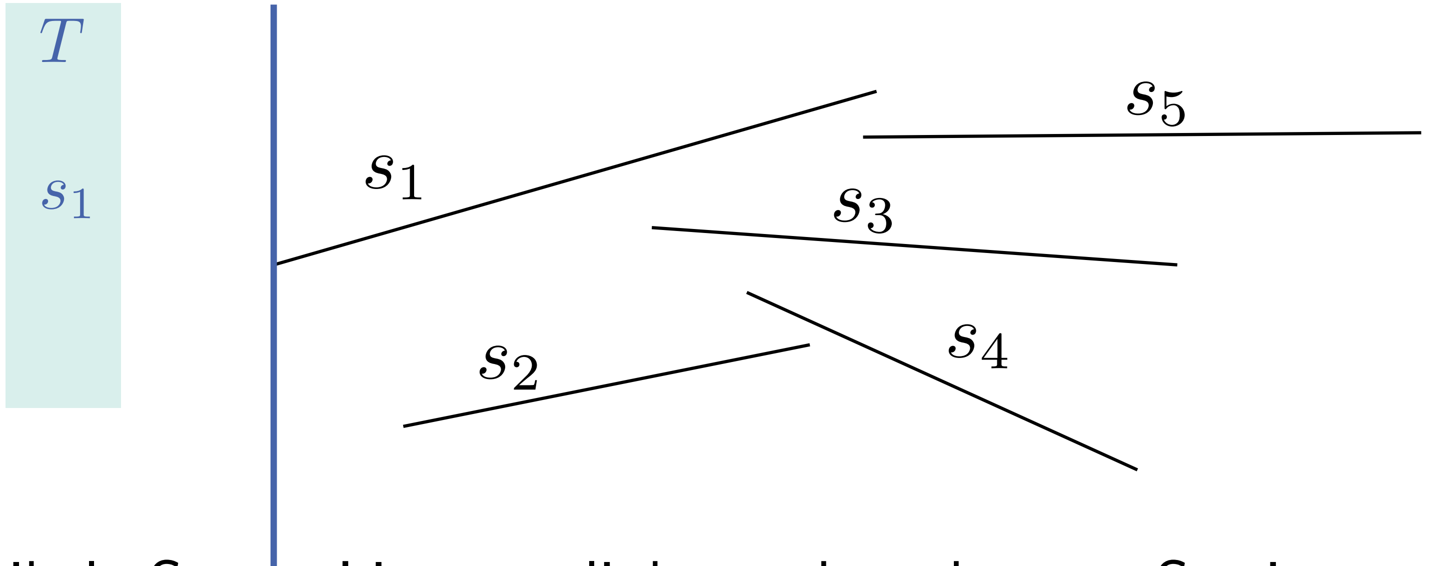
Abarbeitung der Events p:

p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

Berechnung der topologischen Sortierung



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

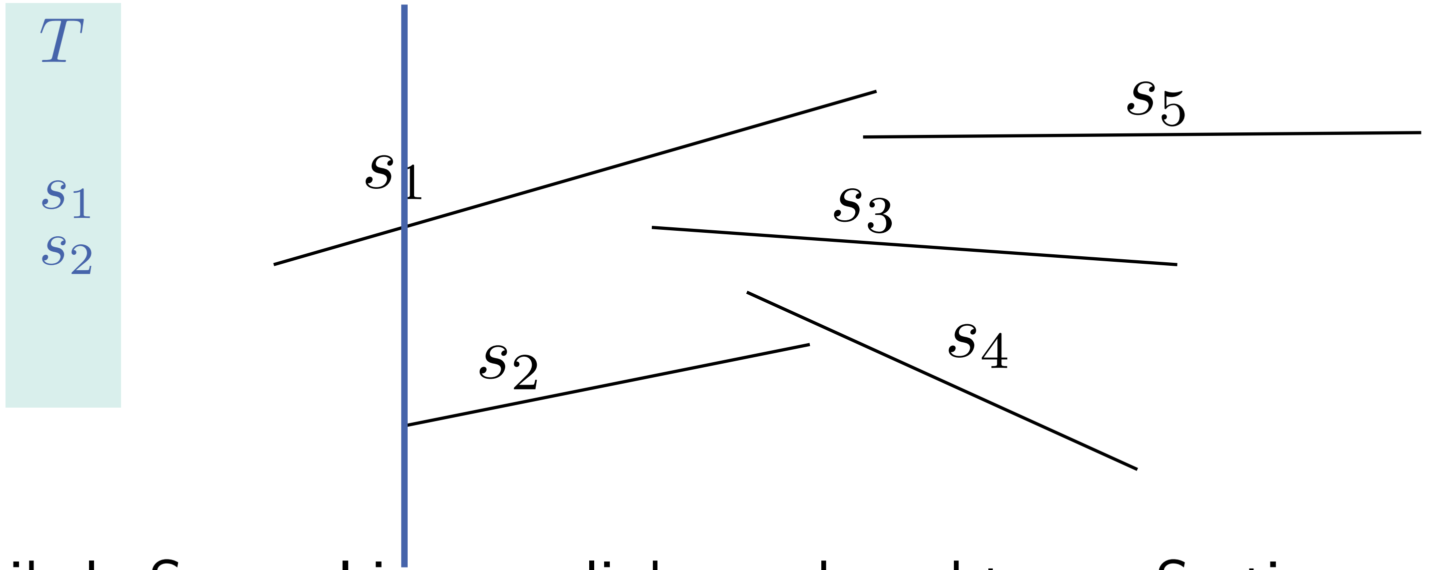
Abarbeitung der Events p:

p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

Berechnung der topologischen Sortierung



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

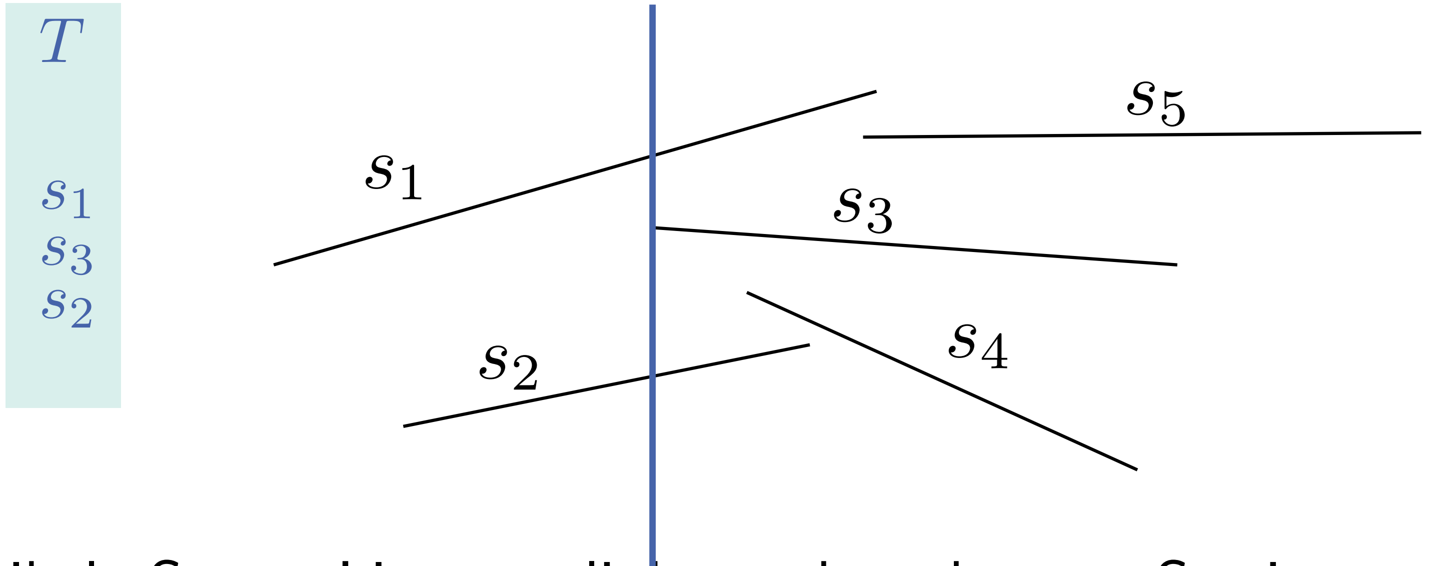
Abarbeitung der Events p:

p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

Berechnung der topologischen Sortierung



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

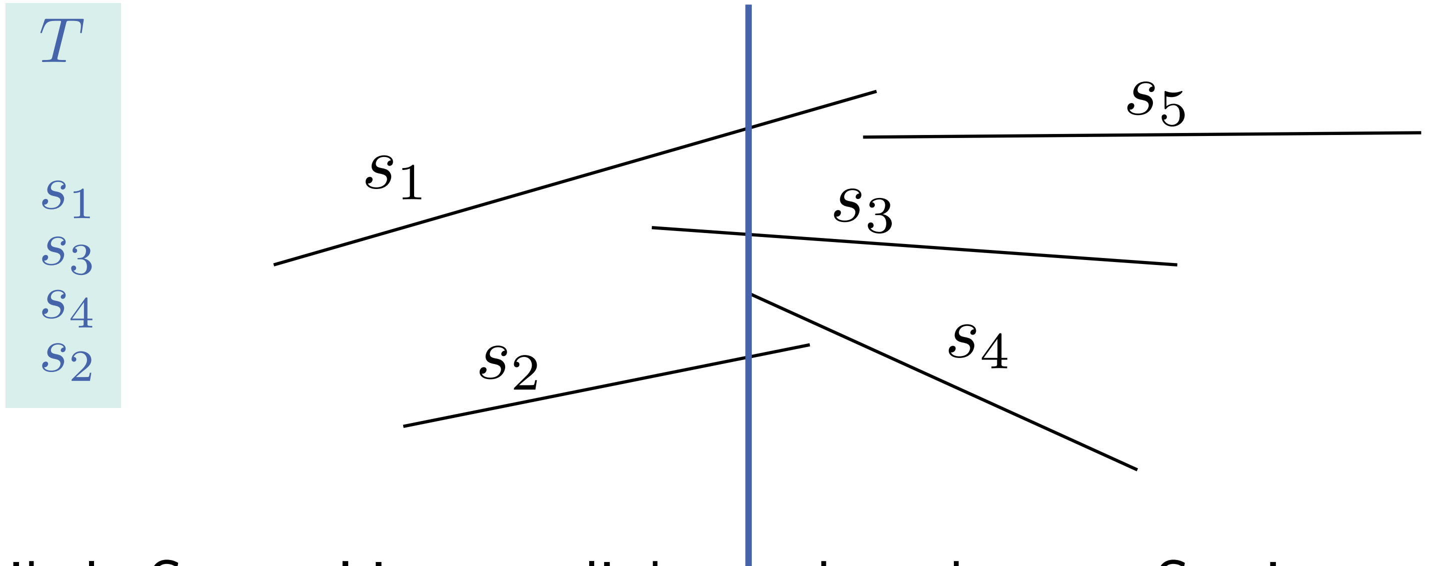
Abarbeitung der Events p:

p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

Berechnung der topologischen Sortierung



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

Abarbeitung der Events p:

p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

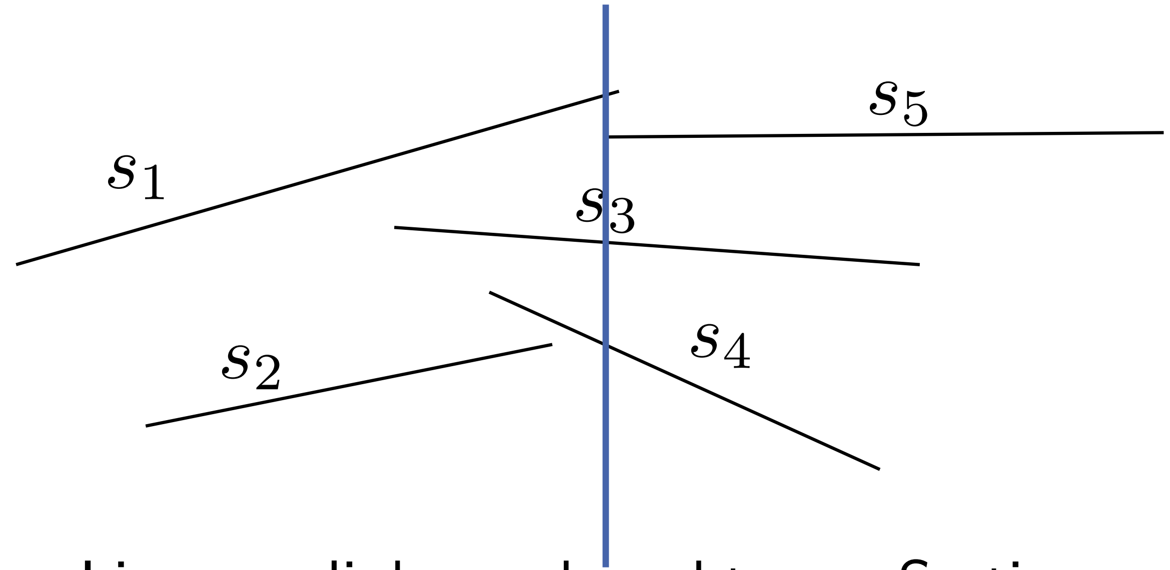
s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

Berechnung der topologischen Sortierung

T

s_1
s_5
s_3
s_4
s_2



Vertikale Sweep-Line von links nach rechts um Sortierung T zu erhalten:

Events: Endpunkte der Strecken.

Sweep-Line-Zustand: Strecken, die von Sweep-Line geschnitten werden. Repräsentation als Binärbaum.

Abarbeitung der Events p:

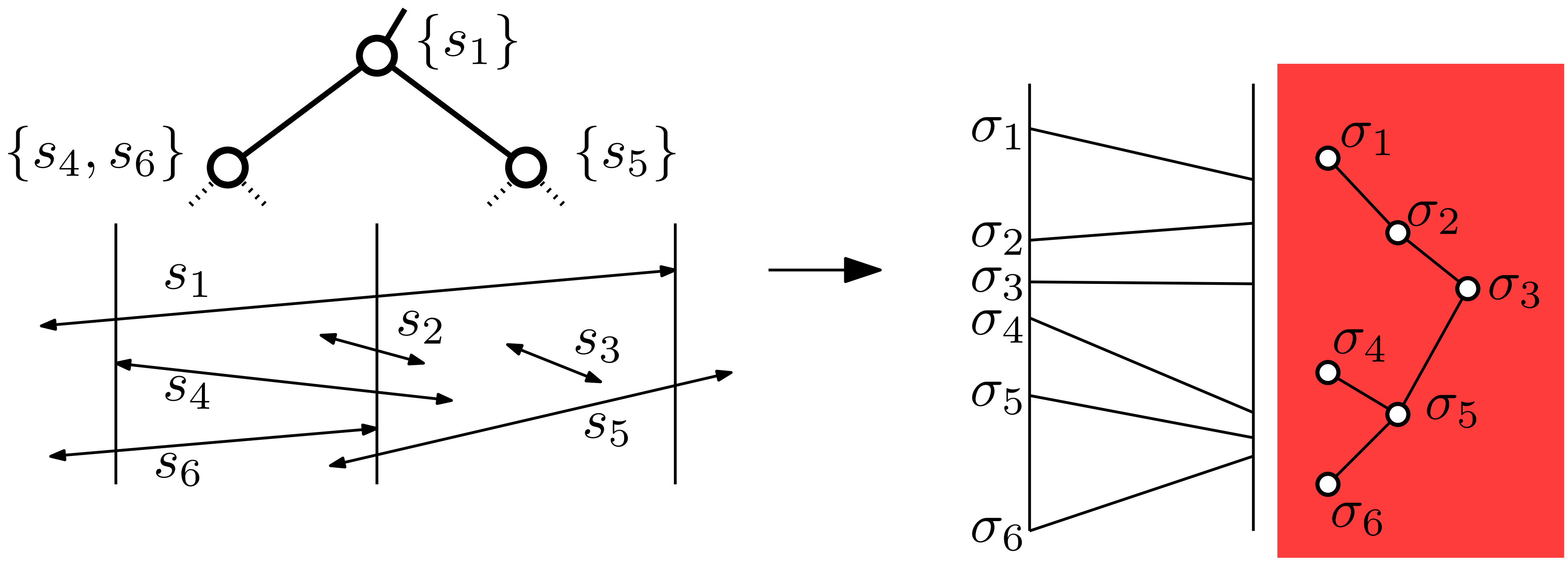
p ist linker Endpunkt einer Stecke s_i: s_i wird in S einsortiert.

s_i wird entsprechend seiner zwei Nachbarn in S in T eingefügt.

p ist rechter Endpunkt einer Stecke s_i: s_i wird aus S entfernt.

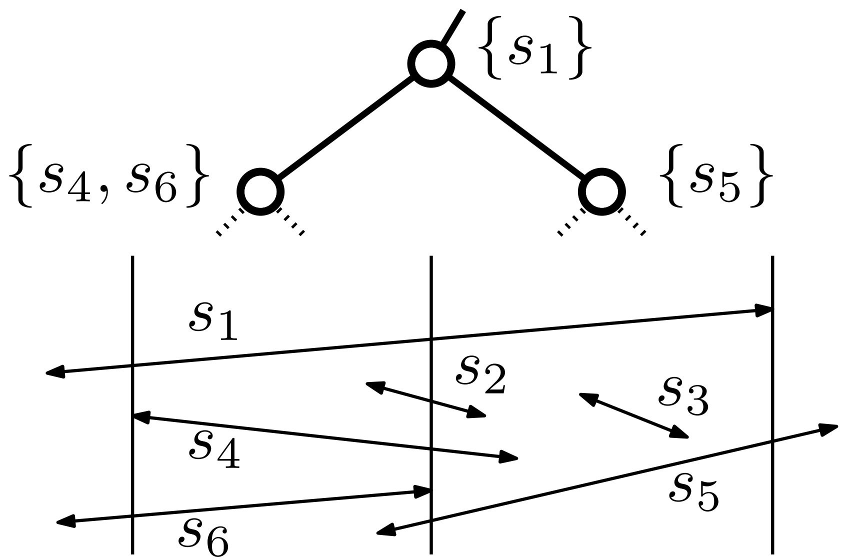
Aufbau der Bäume

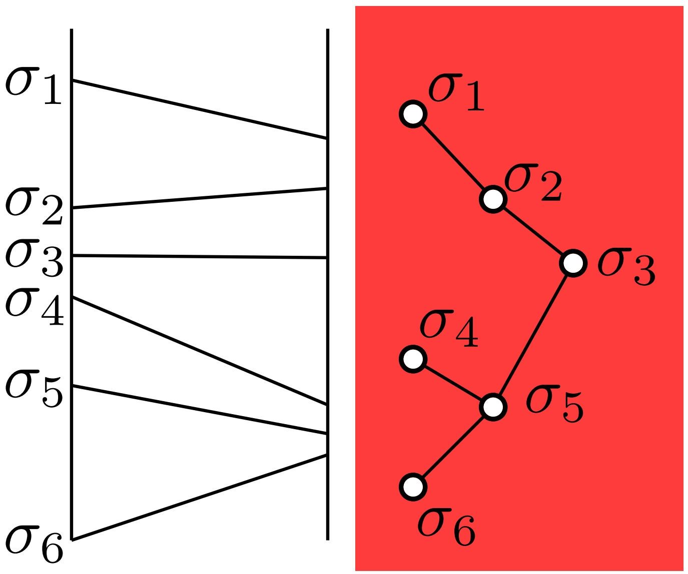
Verwende topologische Sortierung:



Aufbau der Bäume

Verwende topologische Sortierung:





Topologische Sortierung entspricht in einem Streifen der gesuchten vertikalen Sortierung.

→ Füge Strecken in $I(v)$ entsprechend topologischer Sortierung ein.

→ Baue binären Baum von $I(v)$ in $|I(v)|$ Zeit auf.

→ $O(n)$ Zeit insgesamt.

Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Intervallbäumen:

Intervall-Bäume

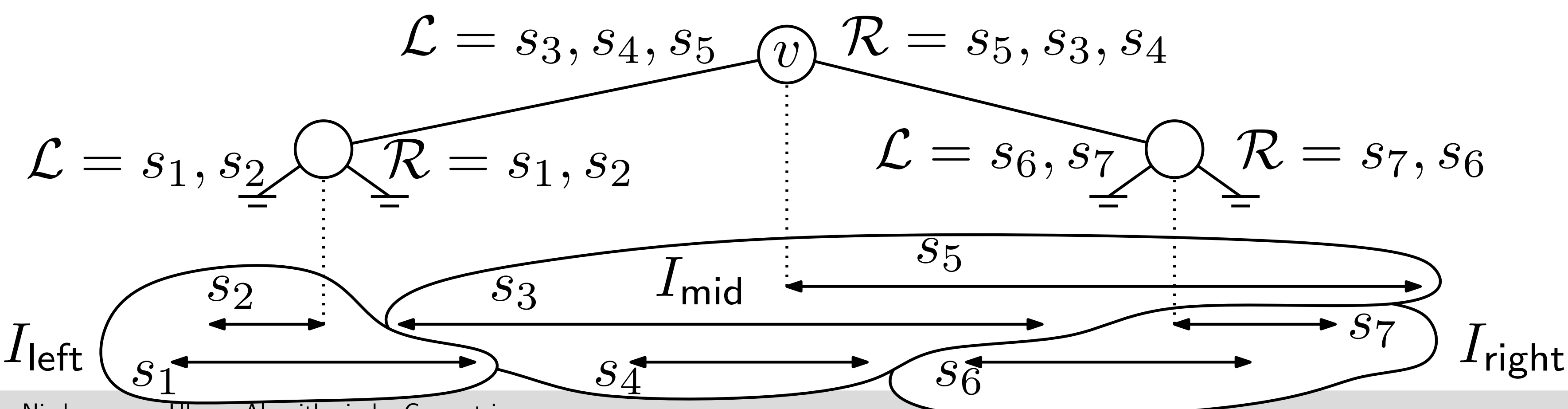
Konstruktion eines Intervall-Baums $\mathcal{T}$

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

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

$\mathcal{T}$ 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}



Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Intervallbäumen:

QIT(v, q_x)

```
if v kein Blatt then
    if q_x < x_mid(v) then
        return QIT(lc(v), q_x)+Anzahl Intervalle in L, die q_x enthalten
    else
        return QIT(rc(v), q_x)+Anzahl Intervalle in R, die q_x enthalten
return 1
```

Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Intervallbäumen:

QIT(v, q_x)

if v kein Blatt **then**
  if $q_x < x_{\text{mid}}(v)$ **then**
    return QIT$(lc(v), q_x)$+Anzahl Intervalle in $\mathcal{L}$, die q_x enthalten
  else
    return QIT$(rc(v), q_x)$+Anzahl Intervalle in $\mathcal{R}$, die q_x enthalten
return 1

Kann mit binärer Suche in $O(\log n)$ Zeit implementiert werden.

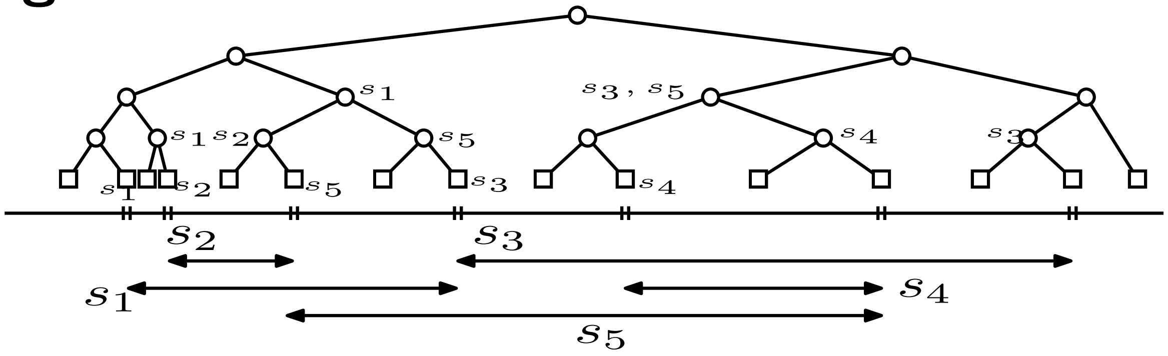
Laufzeit: $O(\log^2 n)$

Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Segmentbäumen:

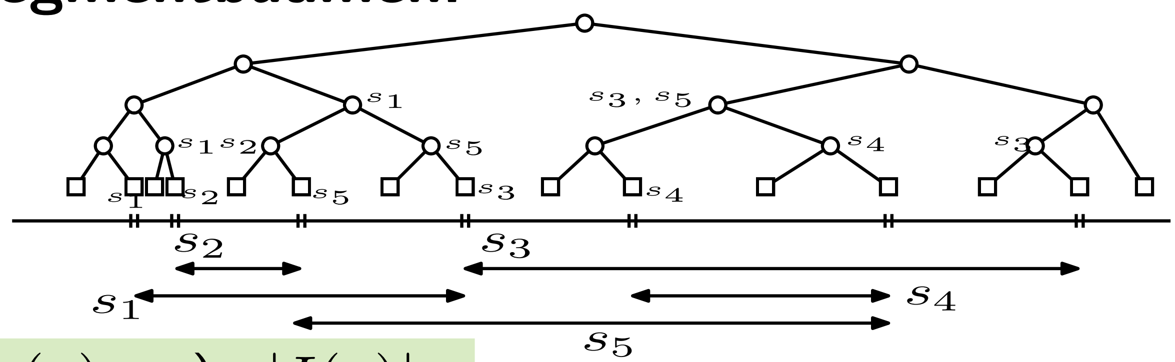


Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Segmentbäumen:



QuerySegmentTree(v, q_x)

if v kein Blatt **then**

  if $q_x \in e(lc(v))$ **then**

    QuerySegmentTree$(lc(v), q_x) + |I(v)|$

  else

    QuerySegmentTree$(rc(v), q_x) + |I(v)|$

return 1

Speichere nicht $I(v)$ sondern $|I(v)|$ an den Knoten.

Anfragezeit $O(\log n)$ Zeit und $O(n)$ Speicher.

Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

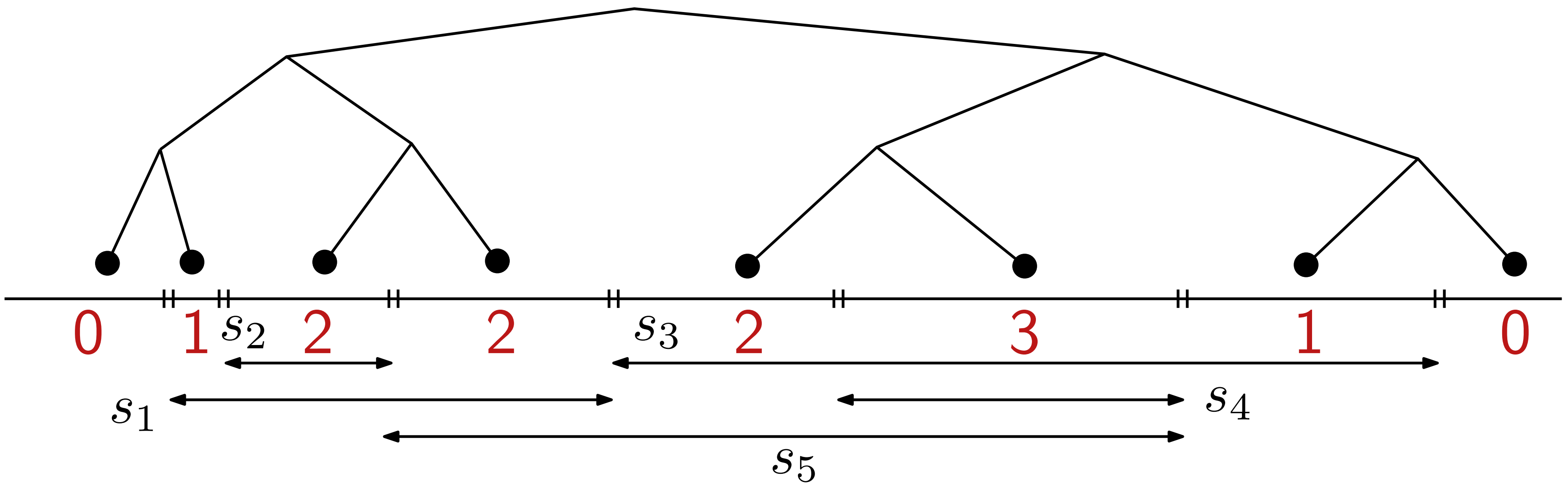
Datenstruktur basiert auf Binärbaum:

Aufgabe 3

gegeben: Menge I bestehend aus n Intervallen

gesucht: Datenstruktur, mit deren Hilfe man möglichst effizient Anzahl Intervalle bestimmen kann, in denen Punkt $p \in \mathbb{R}$ liegt.

Datenstruktur basiert auf Binärbaum:



1. Teile Intervalle in elementare Intervalle auf.
2. Speichere für jedes elem. Intervall, in wie vielen Interv. es vorkommt.
3. Baue Binärbaum basierend auf den Intervallgrenzen auf.

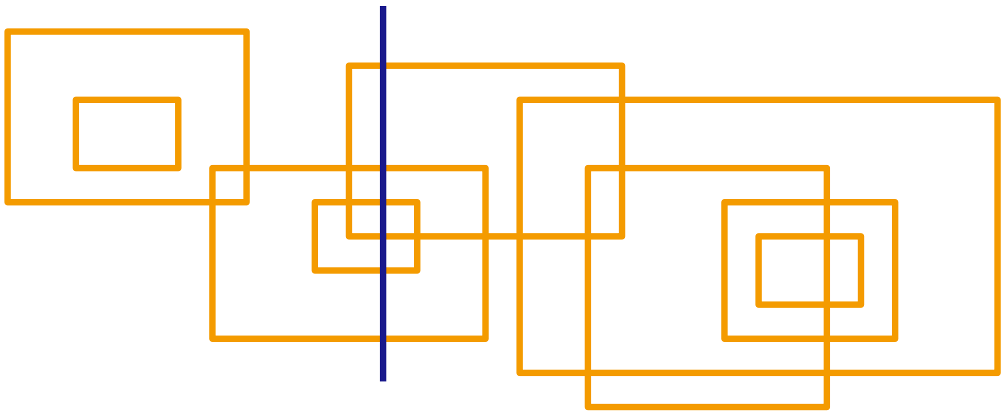
$\longrightarrow$ Anfrage $O(\log n)$, Speicher $O(n)$

Aufgabe 4

geg.: Menge $\mathcal{R}$ an achsenparallelen Rechtecken

ges.: Verfahren, dass in $O(n \log n)$ Zeit $\max_{p \in \mathbb{R}} w_{\mathcal{R}}(p)$ bestimmt.

Für $p \in \mathbb{R}$ gibt $w_{\mathcal{R}}(p)$ Anzahl Rechtecke aus $\mathcal{R}$ an, in denen p liegt.

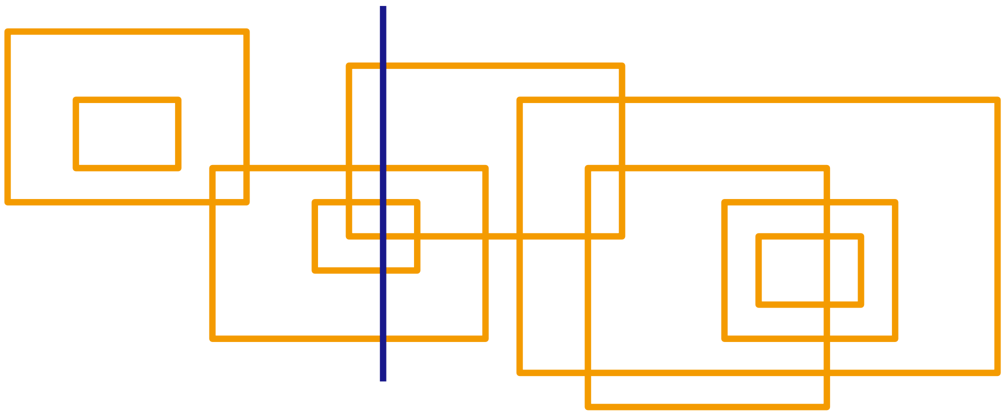


Aufgabe 4

geg.: Menge $\mathcal{R}$ an achsenparallelen Rechtecken

ges.: Verfahren, dass in $O(n \log n)$ Zeit $\max_{p \in \mathbb{R}} w_{\mathcal{R}}(p)$ bestimmt.

Für $p \in \mathbb{R}$ gibt $w_{\mathcal{R}}(p)$ Anzahl Rechtecke aus $\mathcal{R}$ an, in denen p liegt.



Sweep-Line: von links nach rechts.

SL-Zustand: Segmentbaum T, speichert horizontale Kanten als Intervalle.

Events: vertikale Segmente der Rechtecke.

linke vert. Strecke $\overline{pq}$:

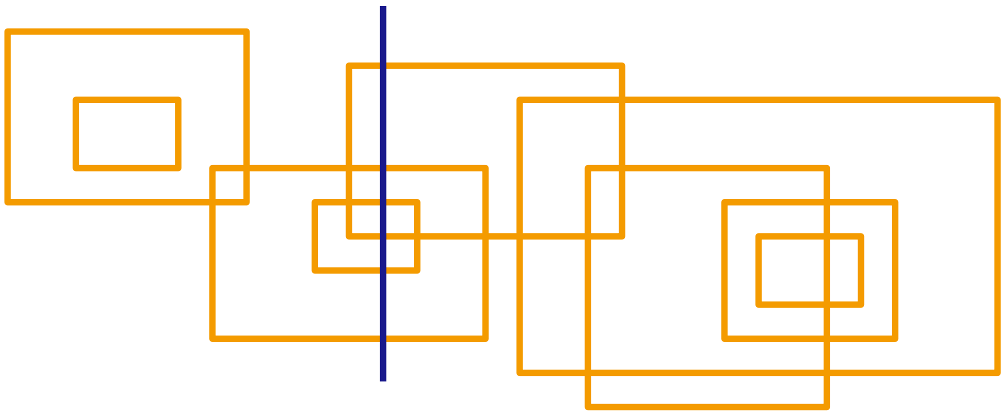
1. Bestimme wie viele Intervalle in T von $[y(p), y(q)]$ geschnitten werden.
   $\longrightarrow$ update von bisherigen $\max w_{\mathcal{R}}(p)$
2. Füge $[y(p), y(q)]$ in T ein.

Aufgabe 4

geg.: Menge $\mathcal{R}$ an achsenparallelen Rechtecken

ges.: Verfahren, dass in $O(n \log n)$ Zeit $\max_{p \in \mathbb{R}} w_{\mathcal{R}}(p)$ bestimmt.

Für $p \in \mathbb{R}$ gibt $w_{\mathcal{R}}(p)$ Anzahl Rechtecke aus $\mathcal{R}$ an, in denen p liegt.



Sweep-Line: von links nach rechts.

SL-Zustand: Segmentbaum T, speichert horizontale Kanten als Intervalle.

Events: vertikale Segmente der Rechtecke.

rechtes vert. Strecke $\overline{pq}$: Entferne Intervall $[y(p), y(q)]$.