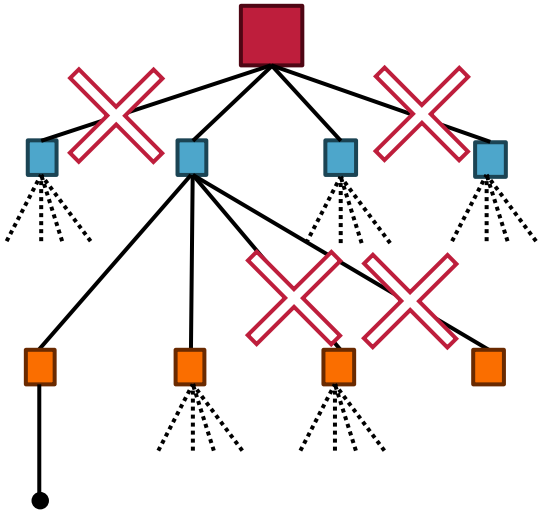
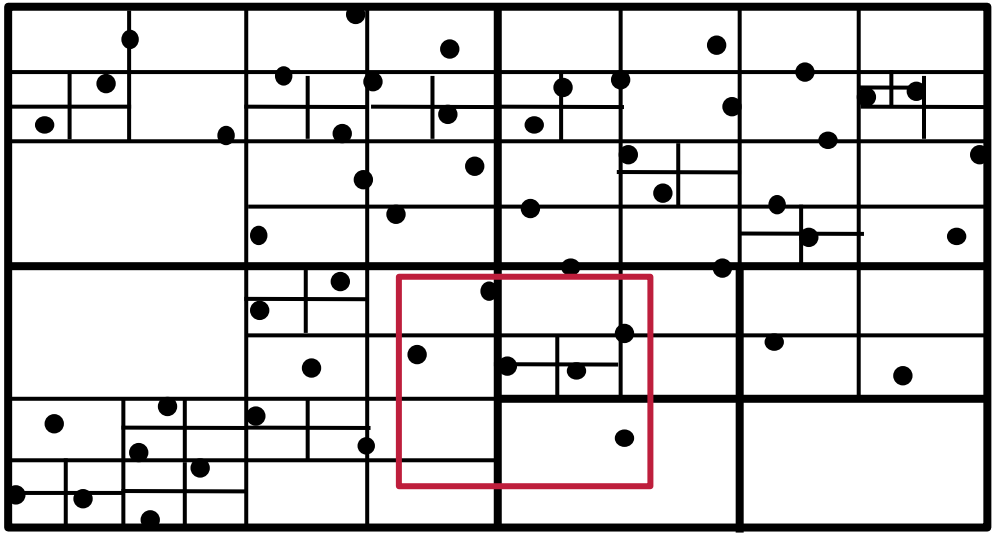
Technische Universität Braunschweig

Einführung in algorithmische Geometrie – KD-Bäume

Arne Schmidt

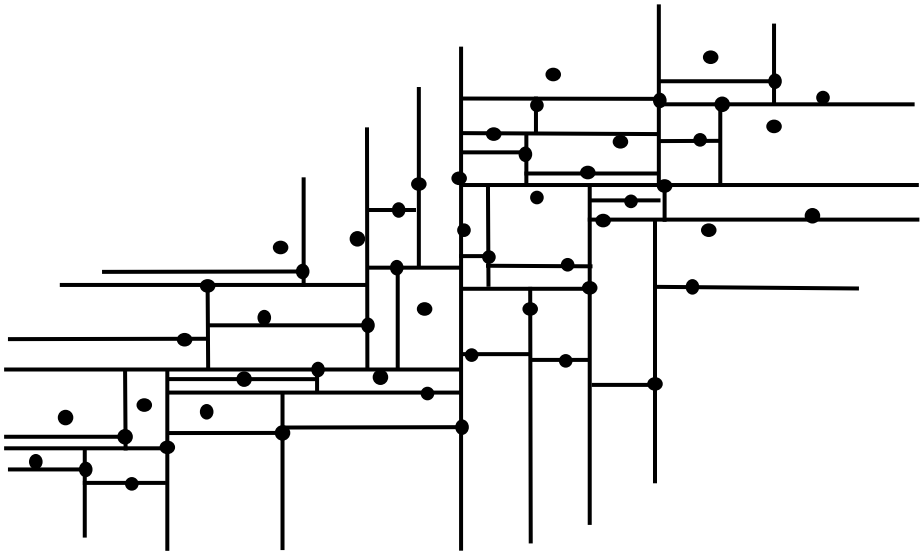
Range Query mit Quadtrees





Satz 2.7
Algorithmus 1.6 löst das Range-Query-Problem in Zeit $O(n + h)$, wobei n die Größe der Punktmenge ist und h die Höhe des Quadtrees.

Kapitel 2.1.2 – KD-Trees



Überlegungen

Wie teilen wir die Punkte auf?

Was können wir in 1D tun?

Was tun in 2D? 3D? 4D?...

1D Punkte

Gegeben seien folgende Zahlen:
1, 51, 35, 22, 91, 13, 93, 44

Welches sind die nächstkleinere und größere Zahl zur 42?
→ 35 und 44

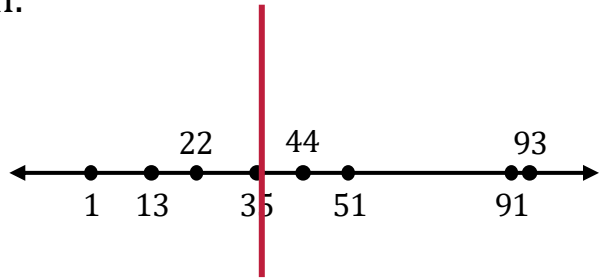
Welches sind die nächstkleinere und größere Zahl zur 72?
→ 51 und 91

Gesucht ist also das Intervall, in welchem die Zahl liegt!

Ohne Preprocessing kann das in Zeit $O(n)$ gelöst werden.

Partitionierung

Gegeben seien folgende Zahlen:
1, 51, 35, 22, 91, 13, 93, 44



Suche Zahl, welche die Zahlen in zwei Hälften teilt.
→ Median!

Das geht einfach, wenn Zahlen sortiert sind.

Sortieren (Recap)

Theorem 2.9:
Das Sortieren von n Elementen benötigt $O(n \log n)$ Zeit.

Nutze zum Sortieren bspw. Mergesort:

1. Sortiere die ersten $\frac{n}{2}$ Zahlen.
2. Sortiere die letzten $\frac{n}{2}$ Zahlen.
3. Füge beide sortierten Hälften zusammen.

Laufzeit:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$$

→ Mastertheorem liefert $T(n) \in \Theta(n \log n)$.

Mastertheorem (Recap)

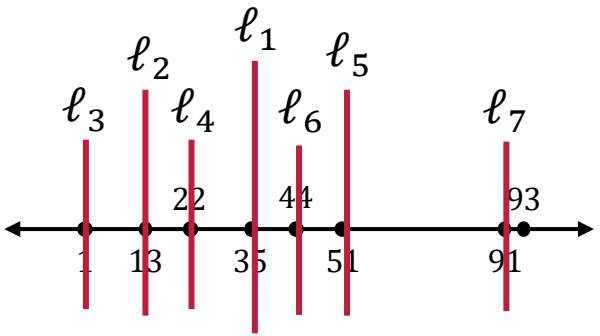
Theorem 2.10:
Sei $T: \mathbb{N} \to \mathbb{R}^+$ eine Funktion in der Form

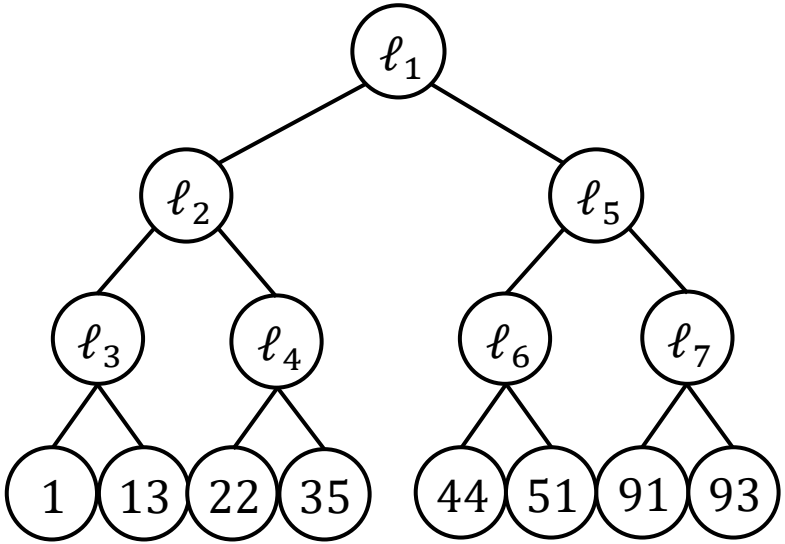
$$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + O(n^k)$$

Mit $m \in \mathbb{N}, k \in \mathbb{R}$ und $0 < \alpha_i < 1$ für alle $i \in \{1, \dots, m\}$.
Dann gilt

$$T(n) \in \begin{cases} \Theta(n^k), & \text{falls } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta(n^k \log n), & \text{falls } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c), & \text{falls } \sum_{i=1}^{m} \alpha_i^k > 1 \text{ mit } \sum_{i=1}^{m} \alpha_i^c = 1 \end{cases}$$

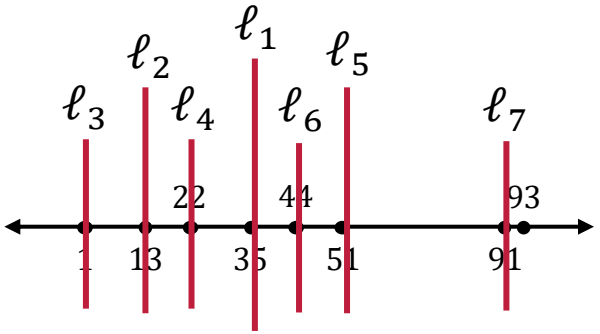
1D-Baum

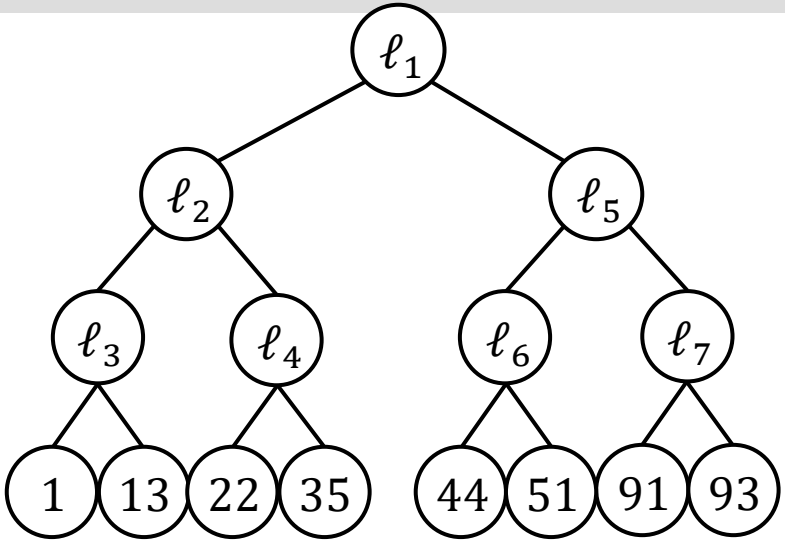




Die Struktur nennt sich auch 1D-Baum
Wie löst man auf dieser Struktur das Problem der nächstgrößeren / -kleineren Zahl?

Suchprobleme





Die nächstkleinere/-größere Zahl eines Blattes kann wie in einem binären Suchbaum gefunden werden!
Bekommt man eine Zahl, kann man so tun, als ob man sie in einen binären Suchbaum einfügen will.
Während des Prozesses landet man irgendwann an einer Stelle, die die nächstkleinere oder nächstgrößere Zahl beinhaltet.

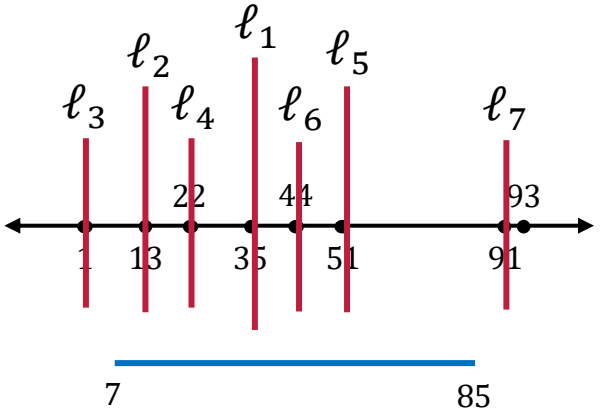
Range-Query in 1D-Bäumen

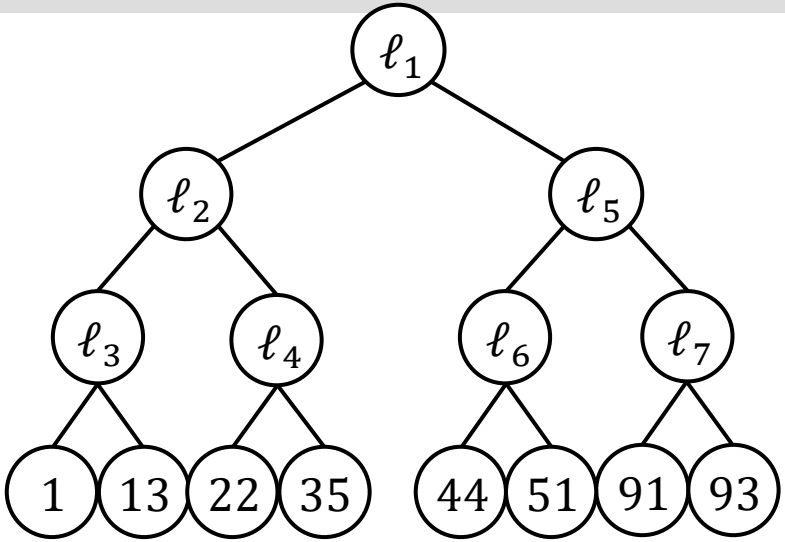
Eingabe: 1D-Baum einer Zahlenmenge $Z = \{z_1, \dots, z_n\}$, ein Intervall $I = [a, b]$.
Ausgabe: $Z' = Z \cap I$ mit $|Z'| := k$

1. Suche $z \in Z$ mit $z \geq a$ und z minimal.
2. Falls z nicht existiert, gib $\emptyset$ zurück.
3. Setze $Z' = \emptyset$
4. Solange $z \leq b$:
   a. Setze $Z' := Z' \cup \{z\}$
   b. Setze $z :=$ Nachfolger von z im Baum; falls nicht existent, setze $z := \infty$
5. Gib Z' zurück.

Man kann zeigen, dass dieser Algorithmus eine Laufzeit von $O(\log n + k)$ besitzt.
Allerdings nutzt dieser Algorithmus nicht die rekursive Eigenschaft eines 1D-Baumes aus.

Range-Query





Gehe vor wie bei Quadtrees:
Schneidet ℓ_i das Interval?

→ Falls ja, Rekursion auf beide Teilbäume.

→ Falls nein, Rekursion nur auf einen Teilbaum.

→ Schneidet die nächste Linie das Intervall wieder, muss die eine Hälfte komplett im Intervall liegen!

Range-Query in 1D-Bäumen (rekursiv)

Algorithmus 2.11: `(Query1D(`ℓ, I`))`
Eingabe: Wurzel ℓ eines 1D-Baumes einer Zahlenmenge $Z = \{z_1, \dots, z_n\}$, ein Intervall $I = [a, b]$.
Ausgabe: $Z' = Z \cap I$

1. Falls ℓ ein Blatt und $\ell \in [a, b]$, dann gib ℓ aus.
2. Falls ℓ ein Blatt und $\ell \notin [a, b]$, dann return.
3. Falls $\ell \geq a$, dann `Query1D(`left$(\ell), I$`)`
4. Falls $\ell < b$, dann `Query1D(`right$(\ell), I$`)`

Eine einfache Analyse zeigt, dass dieser Algorithmus eine Laufzeit von

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

besitzt. Also nach Mastertheorem $T(n) \in O(n)$. Das geht genauer!

Range-Query in 1D-Bäumen (rekursiv) – Analyse

Algorithmus 2.11: `(Query1D(`ℓ, I`))`
Eingabe: Wurzel ℓ eines 1D-Baumes einer Zahlenmenge $Z = \{z_1, \dots, z_n\}$, ein Intervall $I = [a, b]$.
Ausgabe: $Z' = Z \cap I$

1. Falls ℓ ein Blatt und $\ell \in [a, b]$, dann gib ℓ aus.
2. Falls ℓ ein Blatt und $\ell \notin [a, b]$, dann return.
3. Falls $\ell \geq a$, dann `Query1D(`left(ℓ), I`)`
4. Falls $\ell < b$, dann `Query1D(`right(ℓ), I`)`

Bis auf den Wurzelknoten wird nur bei einem Aufruf ein bestimmtes Element gesucht.
Bei dem anderen Aufruf (wenn dieser stattfindet), wird der gesamte Teilbaum mit k_i Knoten aufgerufen. Also ist die Laufzeit eher in der Form:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1) + O(k_i)$$

Also nach Mastertheorem $T(n) \in \Theta(\log n + \sum_i k_i) = \Theta(\log n + k)$.

Range-Query in 1D-Bäumen (rekursiv) – Analyse

Algorithmus 2.11: `(Query1D(`ℓ, I`))`
Eingabe: Wurzel ℓ eines 1D-Baumes einer Zahlenmenge $Z = \{z_1, \dots, z_n\}$, ein Intervall $I = [a, b]$.
Ausgabe: $Z' = Z \cap I$

1. Falls ℓ ein Blatt und $\ell \in [a, b]$, dann gib ℓ aus.
2. Falls ℓ ein Blatt und $\ell \notin [a, b]$, dann return.
3. Falls $\ell \geq a$, dann `Query1D(`left(ℓ), I`)`
4. Falls $\ell < b$, dann `Query1D(`right(ℓ), I`)`

Theorem 2.12:
Algorithmus 1.11 löst das Range-Query-Problem auf 1D-Punktmengen in Zeit $\Theta(\log n + k)$.

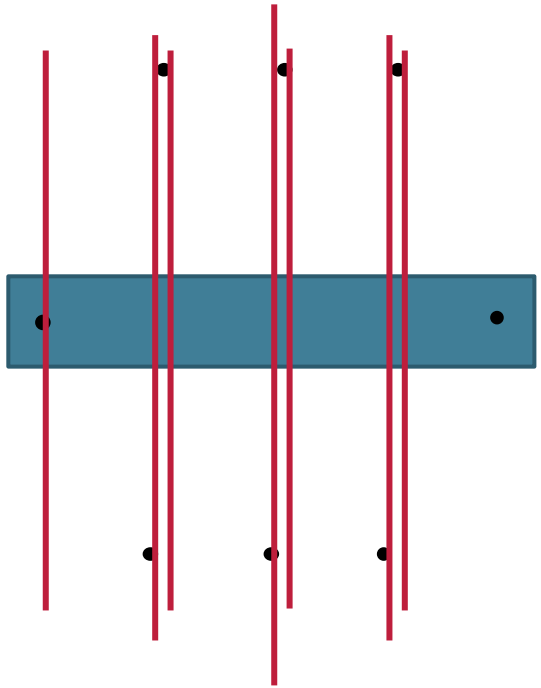
Überlegungen für 2D

Reicht es einfach
1D zu unterteilen?

Falls nicht, was
können wir tun?

Wir nehmen an, die Punkte liegen
in allgemeiner Lage
Hier: Keine kolinearen Punkte

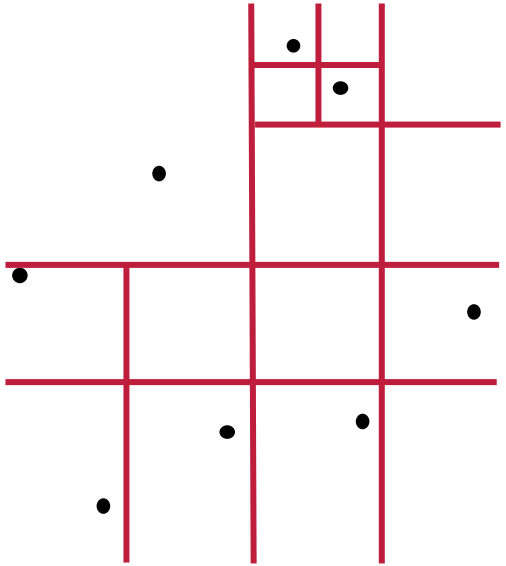
2D Punkte



Das blaue Rechteck enthält nur zwei Punkte, schneidet aber alle Unterteilungen!

→ Immer $O(n)$ Zeit für Range-Query.

2D Punkte – Quadtrees revisited

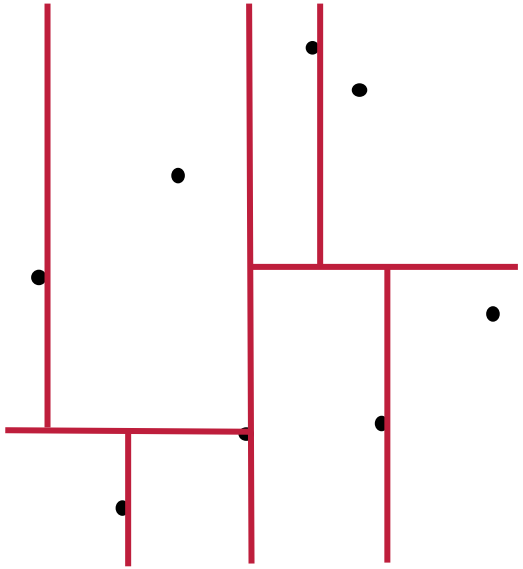


Orientieren wir uns an Quadtrees.

→ Passe Unterteilungen so an, dass die Punktmenge halbiert wird.

→ Betrachte Unterteilungen nach x- und y-Koordinaten getrennt.

2D Punkte – Quadtrees revisited

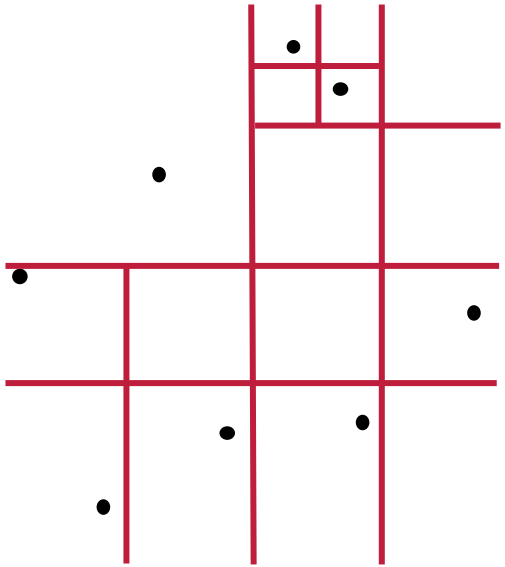


Orientieren wir uns an Quadtrees.

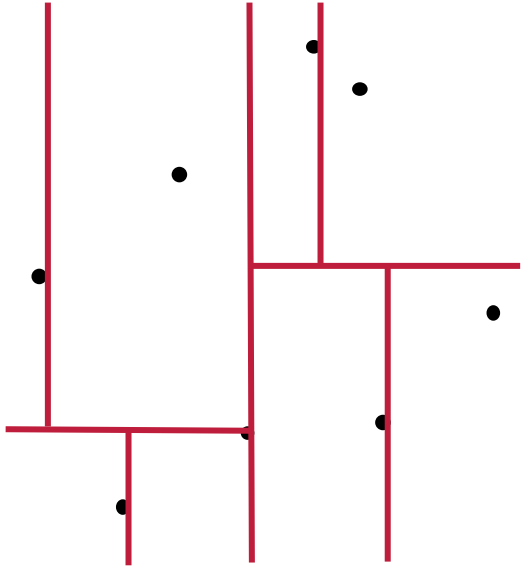
→ Passe Unterteilungen so an, dass die Punktmenge halbiert wird.

→ Betrachte Unterteilungen nach x- und y-Koordinaten getrennt.

2D Punkte – Quadtrees revisited

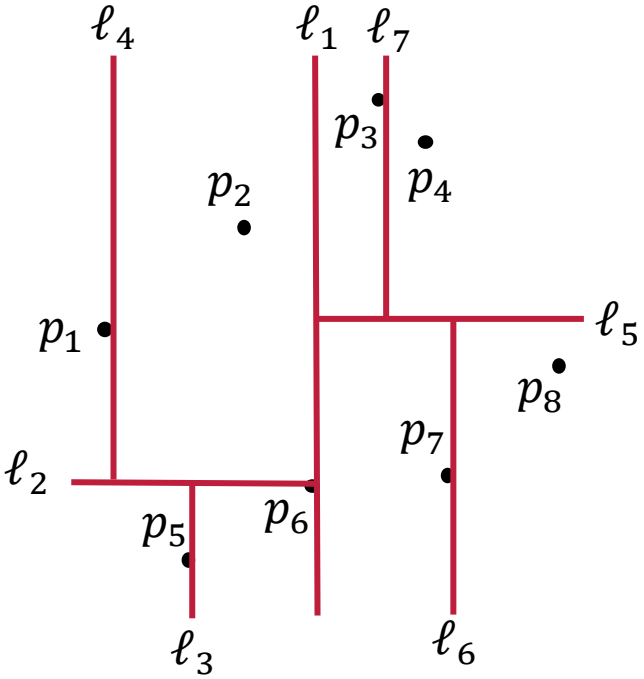


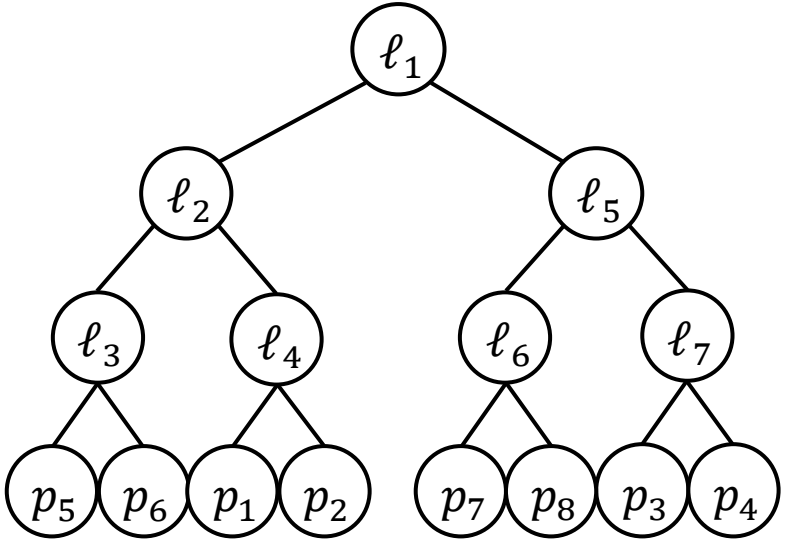
→ Quadtree



→ 2D-Baum

2D-Baum


ℓ_4
ℓ_1
ℓ_7
p_3
p_2
p_4
p_1
ℓ_5
p_8
p_7
ℓ_2
p_5
p_6
ℓ_3
ℓ_6


ℓ_1
ℓ_2
ℓ_5
ℓ_3
ℓ_4
ℓ_6
ℓ_7
p_5
p_6
p_1
p_2
p_7
p_8
p_3
p_4

2D-Baum-Algorithmus

Algorithmus 2.13 (BUILDKDTREE)
Eingabe: Punktmenge P, Rekursionstiefe *depth*
Ausgabe: Wurzel eines k-d-Baums

1. **if** ($|P| = 1$)
2. **return** Blatt mit diesem Punkt
3. **else if** (*depth* ist gerade)
4. Teile in zwei Teilmengen P_1 ($\leq \ell$) und P_2 ($> \ell$) an vertikaler Median-Linie ℓ
5. **else**
6. Teile in zwei Teilmengen P_1 ($\leq \ell$) und P_2 ($> \ell$) an horizontaler Median-Linie ℓ
7. Setze v_{left} := BUILDKDTREE(P_1, *depth+1)*
8. Setze v_{right} := BUILDKDTREE(P_2, *depth+1)*
9. Erzeuge Knoten v für ℓ mit v_{left} und v_{right} als Kinderknoten
10. **return** v

KD-Bäume - Laufzeit

Theorem 2.14:
Algorithmus 2.13 (BUILDKDTREE) konstruiert einen 2D-Baum in Zeit $O(n \log n)$.

Beweis:
Suche Median-Linien mit Laufzeit $O(n)$, um die Punkte aufzuteilen. Damit ergibt sich folgende Gesamtlaufzeit:

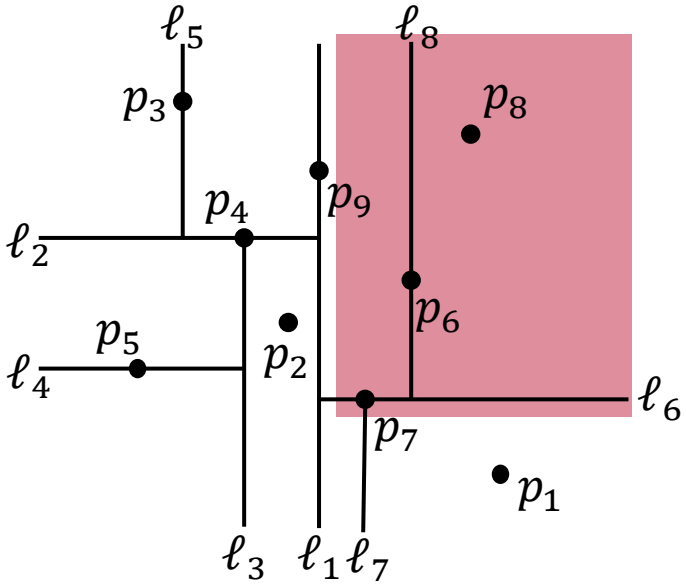
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

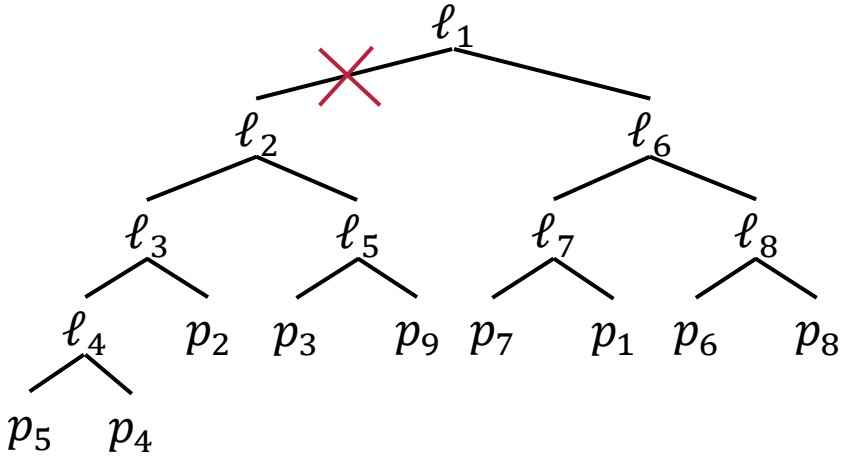
Mit Mastertheorem ergibt sich $T(n) \in O(n \log n)$.

Theorem 2.15:
Ein 2D-Baum benötigt $O(n)$ Speicherplatz.

Der Algorithmus kann für beliebig viele Dimensionen angepasst werden. Die asymptotische Laufzeit und der Speicherplatz ändert sich dabei nicht!

2D-Bäume – Range Query





Welche Punkte liegen im Rechteck?

Antwort: p_7 p_6 p_8

Range-Query – 2D-Trees

Algorithmus 2.16 (SEARCHKDTREE)
Eingabe: Wurzel v eines (Teil-)KD-Baums, Rechteck R
Ausgabe: Knoten unterhalb v, die in R liegen

```
1.  if (v ist ein Blatt)
2.     Gib v aus, falls v in R
3.  else
4.     if (Region(left(v)) ganz in R)
5.        REPORTSUBTREE(letft(v))
6.     else if (Region(left(v)) schneidet R)
7.        SEARCHKDTREE(left(v), R)
8.     if (Region(right(v)) ganz in R)
9.        REPORTSUBTREE(right(v))
10.    else if (Region(right(v)) schneidet R)
11.       SEARCHKDTREE(right(v), R)
```

Range-Query-Laufzeit

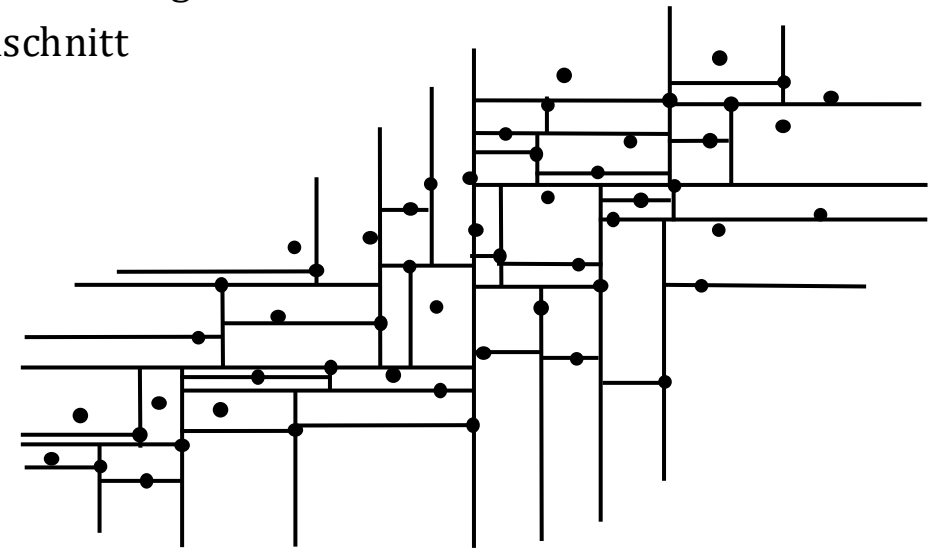
Theorem 2.17:
Algorithmus 2.16 (SEARCHKDTREE) hat eine Laufzeit von $O(\sqrt{n} + k)$ um k Punkte auszugeben.

Beweis: Tafel!

kd-Bäume – Höhere Dimensionen

Bisher nur 1D und 2D betrachtet. Laufzeiten für $k \geq 2$ Dimensionen:

- BuildKDTree: $O(n \log n)$
- SearchKDTree: $O(n^{1-\frac{1}{k}} + x)$, um x Elemente auszugeben
- FindNearestNeighbor: $O(\log n)$ im Durchschnitt



Randbemerkungen

Datenstruktur	Erstellen	Query-Zeit	Speicherbedarf
Quadtree	$O(n \log n)$	$O(n + h)$	$O(n \log N)$
Kd-Tree	$O(n \log n)$	$O\left(n^{1-\frac{1}{d}} + k\right)$	$O(n)$
Range-Trees	$O\left(n \log^{d-1} n\right)$	$O\left(\log^{d} n + k\right)$	$O\left(n \log^{d-1} n\right)$
Range-Trees mit Fractional Cascading	$O\left(n \log^{d-1} n\right)$	$O\left(\log^{d-1} n + k\right)$	$O\left(n \log^{d-1} n\right)$

Erlaubt man ein dynamisches Setting (einfügen und löschen von Punkten), benötigt man für n Einfüge-, Lösch-, und Queryoperationen $\Omega\left(n \log^{d} n\right)$ Zeit.

Randbemerkungen II – R-Trees

Idee:

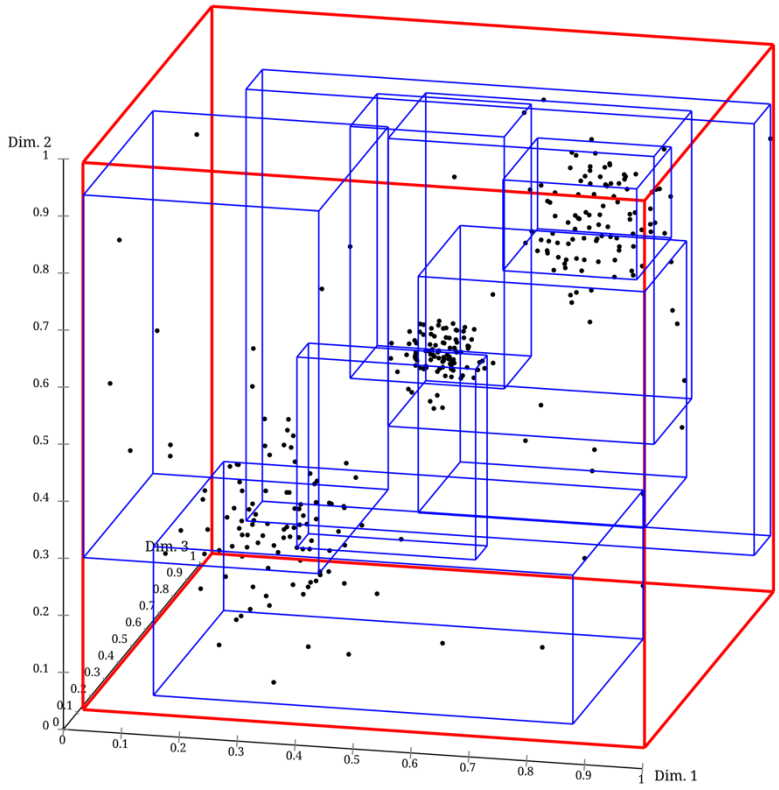
- Teile Punkt in M Boxen auf, die
  - Wenig freien Raum beinhalten.
  - Wenig überlappen
- Rekursion auf jede Box.
- Basierend auf B-Bäumen (siehe AuD)

Qualität

- Schlechte Worst-Case-Performance
- Sehr gut in der Praxis

Anwendung:

- SQLite (max 5 dimensionen)
- Oracle (max 4 Dimensionen)



Nächste Woche

Drehen wir das Spiel um...

Wir haben eine Karte gegeben und...

...fragen uns, in welchem Raum wir stehen.

Wie berechnen wir das?