Karlsruher Institut für Technologie

Algorithmische Geometrie

Real RAM, Word RAM, Point Location
Was ist eigentlich ein Computer?

Was kann dein Computer?

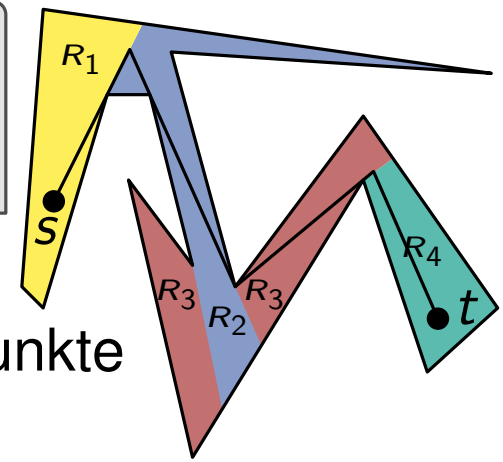
Berechnungsmodelle

- RAM (random access machine): $O(1)$-Speicherzugriff mittels Adresse
- real RAM
  - jeder Speicherplatz hält eine (beliebig große) reelle Zahl
  - arithmetische Operationen $(+, -, \cdot, /)$ in $O(1)$
  - kein Runden auf ganze Zahlen (sonst kann man recht kaputte Dinge tun)
  - in der algorithmischen Geometrie bietet sich dieses Modell an
    $\rightarrow$ man muss sich nicht mit numerischen Ungenauigkeiten ärgern
  - mögliches Problem: ggf. zu mächtig (mächtiger als dein Computer)
- word RAM
  - jeder Speicherplatz hält ein Wort bestehend aus w Bits
  - w ist ausreichend groß ($\geq \log n$)
  - arithmetische Operationen auf ganzen Zahlen (bis Größe 2^w) in $O(1)$
  - bitweise logische Operationen in $O(1)$
  - Bit-Shifts in $O(1)$

Kürzeste Wege in Polygonen

Problem: Minimum Link Path
Gegeben ein Polygon P, sowie Punkte s und t in P, berechne einen aus möglichst wenigen Strecken bestehenden st-Pfad, der P nicht verlässt.

Theorem **(ohne Beweis)**
Der Minimum Link Path zwischen zwei Punkten in einem Polygon der Größe n kann in $O(n)$ berechnet werden.



Grundsätzliches Vorgehen

- R_i = Menge der nach i Schritten von s erreichbaren Punkte
- berechne iterativ R_{i+1} aus R_i

Theorem **(ohne Beweis)**
Es gibt Instanzen, die mit $\Theta(n \log n)$ Bits codiert werden können, sodass die Repräsentation der Polygone $R_1, \ldots, R_n$ $\Theta(n^2 \log n)$ Bits benötigt.

Was stimmt denn nun?

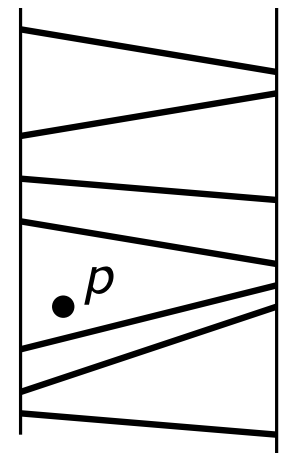
- das erste Theorem ist korrekt, wenn man eine real RAM annimmt
- eine Implementierung (z.B. mit double-Koordinaten) ist ggf. nicht robust

Point-Location in vertikalem Streifen (word RAM)

Problem
Gegeben sei eine Menge disjunkter Strecken S zwischen zwei vertikalen Geraden. Zwischen welchen zwei Strecken liegt ein Punkt p?

Statische Variante

- S ist fest, viele Anfragepunkte p
- Ziel: Vorberechnung für möglichst schnelle Anfragen
  (bei annehmbaren Speicherverbrauch)



Erstmal 1-Dimensional

- Vorgängersuche in einer Liste von Zahlen
- Standard-Lösung binäre Suche: $O(\log n)$
- Ziel: nutze Eigenschaften der word RAM aus
  - Zahlen sind Ganzzahlig
  - Zahlen liegen im Intervall $[0, 2^w)$
  - arithmetische Operationen, bitweise logische Operationen und Shifts auf Wörtern der Länge w in $O(1)$
- Ziel: Laufzeit $o(\log n)$

Schneller als die binäre Suche erlaubt

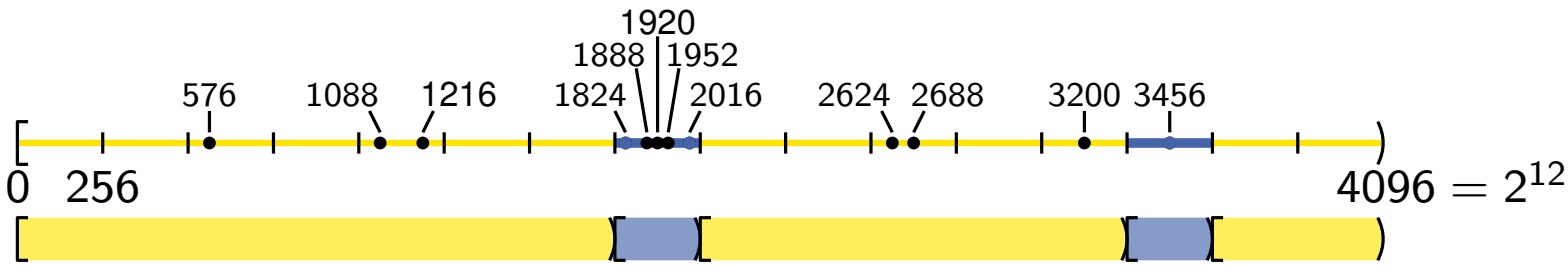
Grundsätzliche Idee der binären Suche

- laufe einen Rekursionsbaum herunter
- Entscheidung für linke/rechte Teilmenge der Zahlen: einen Vergleich
- pro Schritt: Anzahl möglicher Zahlen wird halbiert
- Rekursionstiefe: $\log_2(n)$

Ideen zur Verbesserung

- stärkere Verzweigung: schrumpfe Anzahl Zahlen um Faktor b pro Schritt
  - Rekursionstiefe: $\log_b(n)$ (sublogarithmisch, wenn b superkonstant)
  - Problem: Entscheidung für das richtige Teilintervall zu teuer
- Verkleinerung der Intervallgröße (zu Beginn liegen alle Zahlen in $[0, 2^w)$)
  - schrumpfe dieses Intervall pro Schritt um Faktor 2^h
  - Rekursionstiefe: w/h
  - Problem: Entscheidung ob Teilintervall noch Zahlen enthält
- Kombination der beiden
  - pro Schritt: schrumpfe Anzahl Zahlen oder Intervall
  - Entscheidung für Teilintervall: $O(1)$ mittels Bit-Operationen

Zerlegung in Teilintervalle



$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

Ein Zerlegungsschritt

- zerlege Intervall der Größe 2^ℓ in 2^h Zellen der Größe $2^{\ell-h}$
- markiere jede n/b-te Zahl
- markiere Zellen, die markierte Zahlen enthalten
- Partitionierung in Teilintervalle
  - jede markierte Zelle ist ein Teilintervall
  - jede maximale Sequenz von unmarkierten Zellen ist ein Teilintervall

Wortgröße: w
Anzahl Zahlen: n
Intervall: $[0, 2^\ell)$
Anzahl Zellen: 2^h
Verzweigungsgrad: b

Eigenschaften der Zerlegung

- wir haben $O(b)$ Teilintervalle
- die Grenzen der Teilintervalle sind Vielfache von $2^{\ell-h}$
- jedes Teilintervall hat Länge $2^{\ell-h}$ oder enthält maximal n/b Zahlen

Rekursive Zerlegung

Wortgröße: w
Anzahl Zahlen: n
Intervall: $[0, 2^\ell)$
Anzahl Zellen: 2^h
Verzweigungsgrad: b

Karlsruher Institut für Technologie

Lemma (schöne (h, b)-Zerlegung)

Geg. n Zahlen in $I = [0, 2^\ell)$. I kann in $O(b)$ Teile zerlegt werden, sodass:

- jedes Teilintervall hat Länge $2^{\ell-h}$ oder enthält maximal n/b Zahlen
- die Grenzen der Teilintervalle sind Vielfache von $2^{\ell-h}$

Rekursiver Entscheidungsbaum

- für jedes Teilintervall in (h, b)-Zerlegung:
  - berechne Entscheidungsbaum rekursiv
  - hänge Ergebnis an Wurzelknoten
- Abbruch der Rekursion, wenn man nur noch wenige Zahlen hat
- Höhe des Baumes: maximal $\log_b(n) + w/h$ Warum?

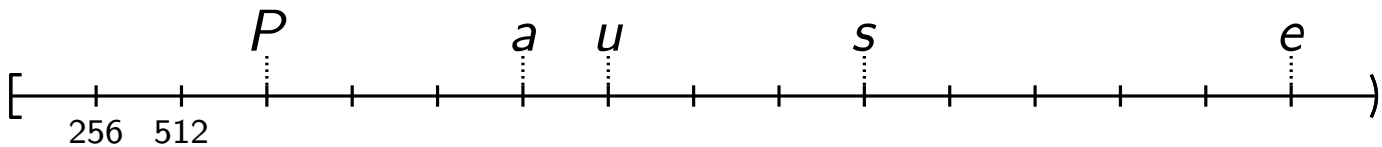
Bemerkung: Damit das Teilintervall im rekursiven Aufruf wieder die Form $[0, 2^\ell)$ hat, muss man es ggf. zur 0 verschieben und zur nächsten 2er-Potenz vergrößern.

Suche nach dem Vorgänger einer Zahl q

- finde bei der Wurzel startend iterativ das Teilintervall, das q enthält
- Laufzeit: $\log_b(n) + w/h$ mal das richtige Teilintervall finden
- $O(\log_b(n) + b)$ für $w/h = b/c$
- $O(\log n / \log\log n)$ für $b = \sqrt{\log n}$

geht in $O(1)$ wenn $bh/c \leq w$ für geeignete Konstante c (wir werden gleich sehen wie)

Wie viele Bits brauchst du?



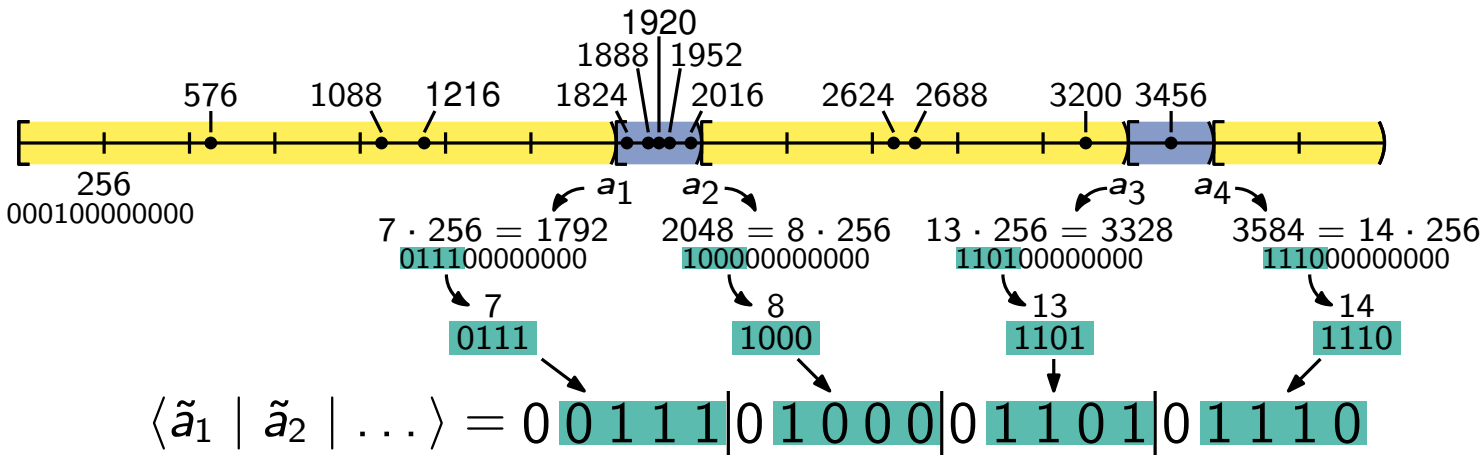
Wie viele Bits sind insgesamt nötig, um die Zahlen P, a, u, s, e zu kodieren?

$$
\begin{aligned}
P = 3 \cdot 256 &\longrightarrow 0011\,00000000 \\
a = 6 \cdot 256 &\longrightarrow 0110\,00000000 \\
u = 7 \cdot 256 &\longrightarrow 0111\,00000000 \\
s = 10 \cdot 256 &\longrightarrow 1010\,00000000 \\
e = 15 \cdot 256 &\longrightarrow 1111\,00000000
\end{aligned}
\qquad 0011 \mid 0110 \mid 0111 \mid 1010 \mid 1111
$$

20 Bits

Repräsentation der Teilintervalle

Wortgröße: w
Anzahl Zahlen: n
Intervall: $[0, 2^\ell)$
Anzahl Zellen: 2^h
Verzweigungsgrad: b



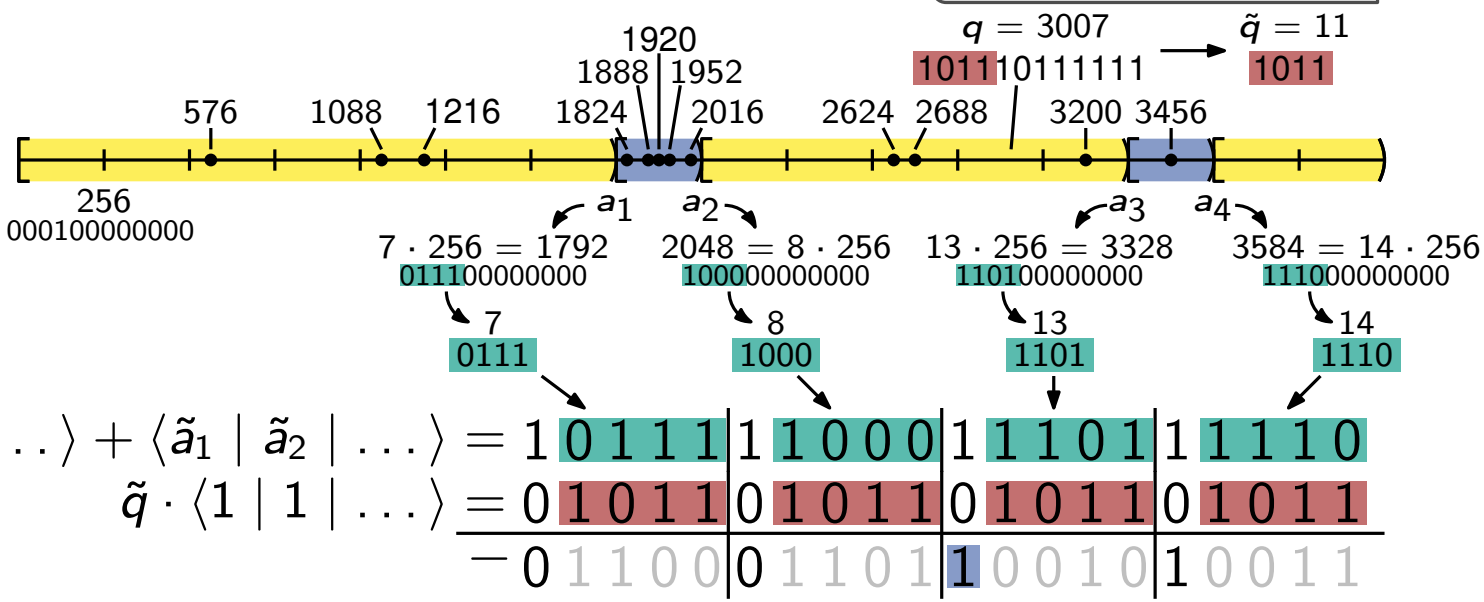
$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

Repräsentation der Unterteilung durch $\bar{b}$ Intervallgrenzen $a_1, \ldots, a_{\bar{b}}$

- die Intervallgrenzen sind vielfache von $2^{\ell-h}$
- Teile die Grenzen durch $2^{\ell-h} \rightarrow$ Ergebnisse sind maximal 2^h groß
- Konkatenation der Binärrepräsentationen: $h \cdot \bar{b}$ Bits
- spendiere ein extra-Bit pro Grenze $\rightarrow (h+1) \cdot \bar{b}$ Bits
- wir können annehmen, dass $(h+1) \cdot \bar{b} \leq w \rightarrow$ benötigt nur ein Wort
- bezeichne das resultierende Wort mit $\langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle$
- Speichere $\langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle$ an entsprechendem Knoten im Rekursionsbaum

Suche nach dem Teilintervall

Wortgröße: w
Anzahl Zahlen: n
Intervall: $[0, 2^\ell)$
Anzahl Zellen: 2^h
Verzweigungsgrad: b



$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

Anfrage: in welchem Teilintervall befindet sich q?

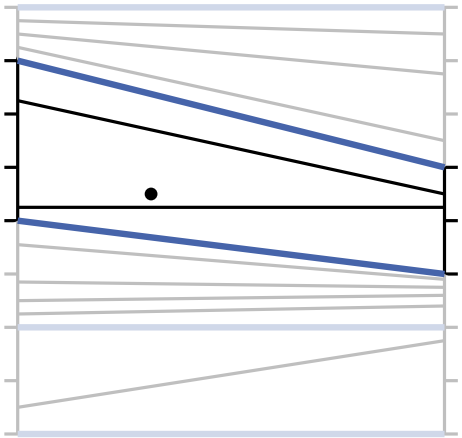
- betrachte $\tilde{q} = \lfloor q/2^{\ell-h} \rfloor$ (geht in konstanter Zeit dank Bit-Shift)
- es genügt $\tilde{q}$ in $\tilde{a}_1, \tilde{a}_2, \ldots$ zu finden (Division durch $2^{\ell-h}$ erhält Ordnung)
- es gilt außerdem: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \,\&\, 2^h = 0$ Warum?
- berechne nun: $(\underbrace{\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle}_{\text{vorberechnet}} - \tilde{q} \cdot \underbrace{\langle 1 \mid 1 \mid \ldots \rangle}_{\text{vorberechnet}}) \,\&\, \underbrace{\langle 2^h \mid 2^h \mid \ldots \rangle}_{\text{vorberechnet}}$
- Kosten: $1\times$ Subtraktion, $1\times$ Multiplikation, $1\times$ bitweise Verundung $\rightarrow O(1)$
- das Most Significant 1-Bit liefert i, sodass $a_{i-1} < q \leq a_i$
  (das MS1B kann man auch mittels elementarer Operationen bestimmen)

1D Suche → 2D Suche

1D: Pro Knoten des Suchbaums	Was hilft das?
■ $O(b)$ Grenzen ■ jede Grenze ist vielfaches von $2^{\ell-h}$	alle Grenzen passen zusammen in ein Wort (h Bits pro Grenze)
■ zwischen benachbarten Grenzen: ■ kurzes Teilintervall ($2^{\ell-h}$) ■ wenige Zahlen (n/b)	in jedem Knoten machen wir ausreichend Fortschritt

1D: Anfrage	
■ suche benachbarte Grenzen	geht in $O(1)$
■ laufe in entsprechenden Teilbaum	passiert $\leq \log_b(n) + w/h$ oft

Erweiterung auf Point Location in einem Streifen

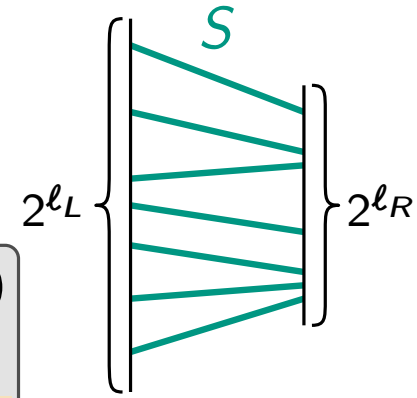


- wähle Grenzstrecken mit ähnlichen Eigenschaften
- Anfrage eines Punktes (x, y)
  - 1D Suche bzgl. y an der Stelle x
  - runde alle Koordinaten → h Bits pro Koordinate
- rekursiver Aufruf zwischen den Grenzen
- **Problem:** Was wenn Grenzen Strecken schneiden?

Schöne Zerlegung

Gegeben

- zwei vertikale Strecken der Länge 2^{ℓ_L} und 2^{ℓ_R}
- Streckenmenge S von n kreuzungsfreie Strecken zwischen den beiden



Lemma (schöne (h, b)-Zerlegung)
Wir können $O(b)$ Grenzen $s_0, s_1, \cdots \in S$ wählen, sodass

- zwischen s_i und s_{i+1} liegen maximal n/b Strecken oder $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ oder $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ für „gerundete“ Grenzen $\tilde{s}_i$

Beweis:
Übung

Anmerkung zur Anfrage eines Punktes $p = (x, y)$

- linke/rechte Endpunkte aller $\tilde{s}_i$ können jeweils in ein Wort gepackt werden
- alle (gerundeten) Schnittpunkte der $\tilde{s}_i$ mit einer vertikalen Gerade bei x können gleichzeitig mit konstant vielen Operationen berechnet werden
- suche auf den gerundeten Schnittpunkten nach y wie im 1D Fall
- Runden ist nicht schlimm: nach der Suche weiß man ausreichend genau, zwischen welchen s_i p liegt $\rightarrow$ nur konstant viele Vergleiche

Zusammenfassung

Heute gesehen

- Berechnungsmodelle: word RAM und real RAM
- die real RAM ist oft ein gutes Modell für die algorithmische Geometrie
  (die beliebig hohe Genauigkeit kann aber auch unrealistisch sein)
- die beschränkte Genauigkeit der word RAM kann auch nützlich sein
  - Suche in $O(\log n / \log \log n)$
  - Point Location in vertikalem Streifen: $O(\log n / \log \log n)$

Was gibt es sonst noch?

- allgemeines Point Location mit $o(\log n)$ Anfragezeit in der word RAM
- Probleme mit $o(n \log n)$ Lösungen in der word RAM:
  - 3D konvexe Hülle
  - Voronoi Diagramm
  - Euklidischer MST
  - Triangulierung eines Polygons (mit Löchern)
  - Linienschnitt

Literaturhinweise

- **Transdichotomous Results in Computational Geometry, I: Point Location in Sublogarithmic Time** (2009)
  Timothy Chan, Mihai Pătraşcu
  https://doi.org/10.1137/07068669X

- **Transdichotomous Results in Computational Geometry, II: Offline Search** (2010)
  Timothy Chan, Mihai Pătraşcu
  https://arxiv.org/abs/1010.1948

- **On the bit complexity of minimum link paths: Superquadratic algorithms for problem solvable in linear time** (1999)
  Simon Kahana, Jack Snoeyink
  https://doi.org/10.1016/S0925-7721(98)00041-8