

Übung 9 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – {moritz.laupichler, hans-peter.lehmann}@kit.edu
http://algo2.iti.kit.edu/AlgorithmenII_WS22.php

Institut für Theoretische Informatik - Algorithmik II

```
    result = current_weight;
    return true;
}

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

- Geometrische Algorithmen
 - Interval Tree
 - Skyline-Problem
 - Linienschnitt
 - Graham's Scan
- Besprechung ÜB 8

- geometrische **Varianten** bekannter Probleme
 - Spezialfälle oft einfacher
 - allgemeines TSP: nicht approximierbar wenn $P \neq NP$
 - metric TSP: 1.5-Approximation
 - euclidean TSP: ϵ -Approximation
 - Laufzeit in $O(n(\log n)^{O(\frac{1}{\epsilon}\sqrt{d})^{d-1}})$

- geometrisch motivierte Probleme
 - Punktlokalisierung
 - Bewegungsplanung Robotik
 - Sichtbarkeitsgraphen/Prüfung
 - Streckenschnitt
 - ...

Datenstrukturen

■ Baumstrukturen

- Interval Tree
 - Quad Tree
 - k-d-Tree
 - Wavelet-Tree
- 1-dim
2-dim
n-dim
2-dim

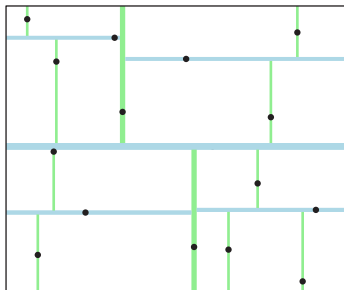
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

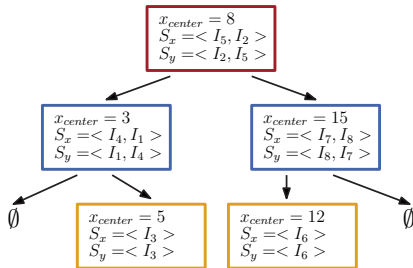
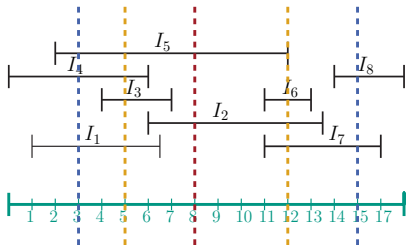
■ Sweepline

- sortiert
- topologisch sortiert



Interval Tree

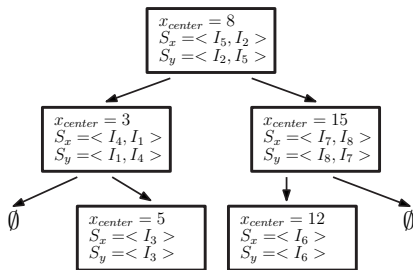
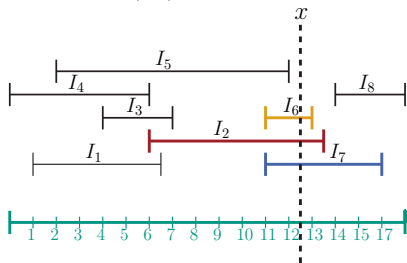
Konstruktion



Interval Tree

Schnitt mit Punkt

$$I \cap x = \langle I_2, I_7, I_6 \rangle$$



Idee

- strukturierte Abarbeitung eines Problems
- nutze Nähe aus
 - geometrisch nahe Objekte beeinflussen sich
 - geometrisch weit entfernte Objekte (nahezu) unabhängig

im Allgemeinen

- reduziere n -dim $\rightarrow (n - 1)$ -dim

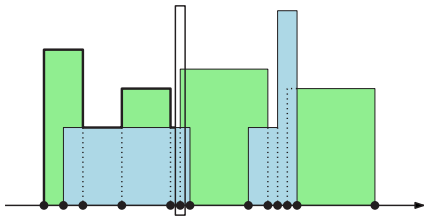
Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem

Maximumsbildung



Problem: effiziente Lösung des 1-dimensionalen Problems

- ineffizient: $O(n^2)$ vergleiche Linienschnitt
- Ziel hier: Algorithmus mit $O(n \log n)$ Zeit

Sortierte Liste

- Array $\rightarrow \mathcal{O}(n)$ für Einfügen/Löschen
- Linked List $\rightarrow \mathcal{O}(n)$ für Positionsbestimmung

Lösung: Priority Queue

- alle Operationen in maximal $\mathcal{O}(\log n)$ möglich

Pseudocode (1/2)

Data: List of buildings (begin, height, end)

Result: Skyline coordinates (x, height)

$i \leftarrow 0; L' \leftarrow \emptyset;$

foreach $(b, h, e) \in L$ **do**

$L' \leftarrow L' \cup (b, h, "b", i) \cup (e, h, "e", i);$
 $i \leftarrow i + 1;$

end

sort L' in lexicographical order;

max priority queue $q \leftarrow \emptyset;$

...

Pseudocode (2/2)

```
...  
while  $L' \neq \emptyset$  do  
  actpos  $\leftarrow L'.first().pos$ ;  
  while  $L' \neq \emptyset$  and  $actpos = L'.first().pos$  do  
    (pos,height,label,index)  $\leftarrow L'.popfirst()$ ;  
    if label = "b" then  
      |  $q.insert(height,index)$ ;  
    else  
      |  $q.remove(index)$ ;  
    end  
  end  
  if  $q \neq \emptyset$  then  
    | print (actpos,  $q.max().height$ );  
  else  
    | print actpos,0;  
  end  
end
```

Events

1. *Start-Event* $e := (y, \text{start}, s = \overline{(x, y)(x', y')})$
2. *End-Event* $e := (y', \text{end}, s = \overline{(x, y)(x', y')})$
3. *Intersection-Event* $e := (y, \text{intersection}, (s_j, s_j))$
 - s_i und s_j sind Linien die sich im Intersection-Event schneiden

Initialisierung

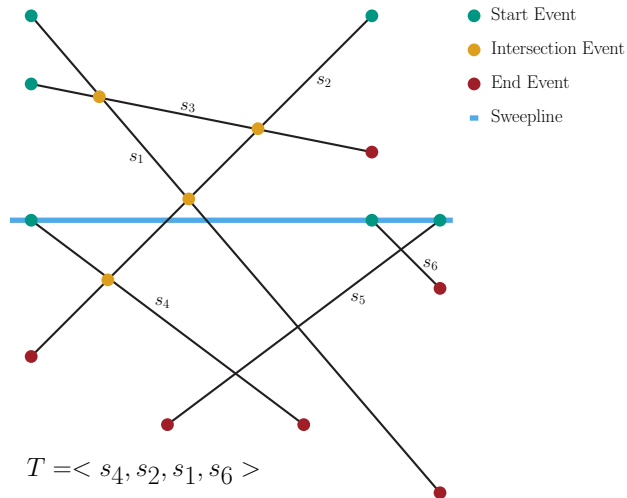
- Speichere *Start-* und *Stop-*Events in einer Priority Queue Q (nach y -Wert sortiert) $O(n \log n)$

Linearer Scan

1. Entferne nächstes Event aus Priority-Queue Q $O(\log n)$
 - 1.1 *Start-Event*: Füge s in eine sortierte Liste T hinzu (nach x -Koordinate) und teste mit *Nachbarn* in T auf Linienchnitt $O(\log n)$
 - 1.2 *End-Event*: Lösche s aus sortierter Liste T und teste beide vorherigen *Nachbarn* in T auf Linienchnitt $O(\log n)$
 - 1.3 *Intersection-Event*: Vertausche s_i und s_j in sortierter Liste T und teste s_i mit seinem neuen rechten und s_j mit seinem neuen linken *Nachbarn* in T auf Linienchnitt $O(\log n)$
 - 1.4 Falls ein Schnittpunkt zweier Linien einen Schnittpunkt ergibt, füge *Intersection-Event* zu Q hinzu $O(\log n)$
- ⇒ Laufzeit $O((n + k) \log n)$ wobei k Anzahl an Schnittpunkten

Linienchnitt

Illustration Sweepline-Algorithmus



- Gegeben drei Punkte $P_0, P_1, P_2 \Rightarrow$ bestimme Orientierung

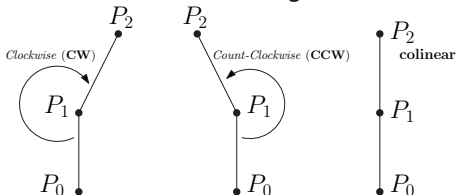
- Sei $\vec{a} = \overrightarrow{P_0P_1}$ und $\vec{b} = \overrightarrow{P_1P_2}$

- $CCW(P_0, P_1, P_2) = a_x \cdot b_y - a_y \cdot b_x$

- $CCW(P_0, P_1, P_2) > 0 \Rightarrow$ **CCW**

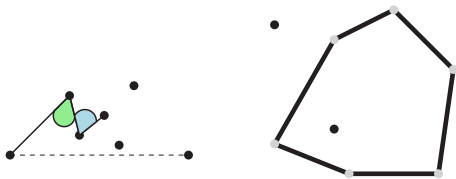
- $CCW(P_0, P_1, P_2) = 0 \Rightarrow$ **Colinear**

- $CCW(P_0, P_1, P_2) < 0 \Rightarrow$ **CW**



- Unterproblem von Algorithmen

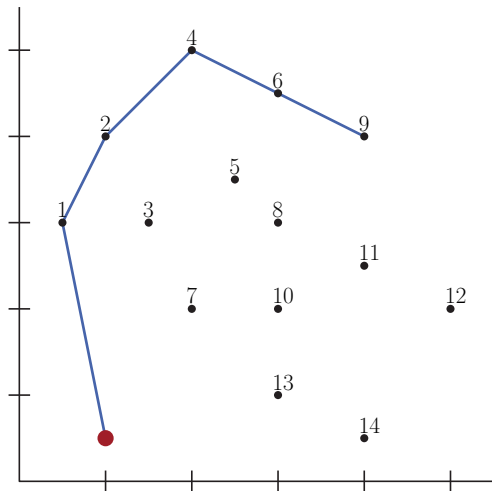
- Graham Scan
- Test auf Enthaltensein
Punkt in konvexem Polygon



Konvexe Hülle

Graham-Scan

1. Finde Punkt P_0 mit kleinster y -Koordinate
 - Gibt es mehrere Punkte mit gleicher y -Koordinate, dann nehme Punkt mit kleinster x -Koordinate
2. Sortiere alle Punkte in absteigendem Winkel relativ zu P_0
3. Iteriere über alle Punkte P_i ($i > 2$) und betrachte Dreieck $H_{k-1}H_kP_i$ (wobei H_i i -ter Punkt in der aktuellen konvexen Hülle)
 - *Rechtsknick*: Füge P_i zur konvexen Hülle H hinzu
 - *Linksknick*: Lösche H_k aus bisheriger konvexen Hülle H



Ende!



Feierabend!