

Übung 5 – Algorithmen II

Yaroslav Akhremtsev, Demian Hespe – yaroslav.akhremtsev@kit.edu, hespe@kit.edu

Mit Folien von Michael Axtmann (teilweise)

http://algo2.iti.kit.edu/AlgorithmenII_WS17.php

Institut für Theoretische Informatik - Algorithmik II

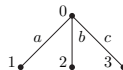
```
    result = current_weight;
    return true;
}

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 );
            }
        }
    }
}
```

- Lempel-Ziv 78 decompression
- Geometrie
(was und warum?)
- Sweepline
 - allgemeines Prinzip
 - Beispiel: Berechnung einer Skyline
- Interval Search Trees
- Rectangle Intersection Problem
- Rechts oder Links?
(Orientierung eines Punktes zu einer Linie)

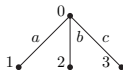
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String													
New node													



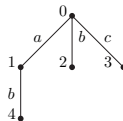
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a												
New node													



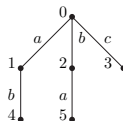
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a	b											
New node	4												



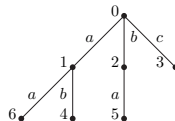
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>										
New node	4	5											



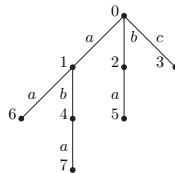
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a	b	a	ab									
New node	4	5	6										



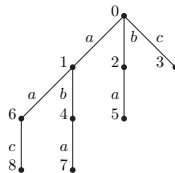
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>								
New node	4	5	6	7									



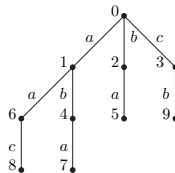
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>							
New node	4	5	6	7	8								



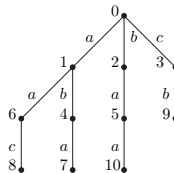
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>						
New node	4	5	6	7	8	9							



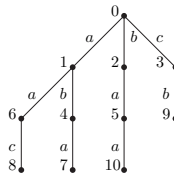
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>					
New node	4	5	6	7	8	9	10						



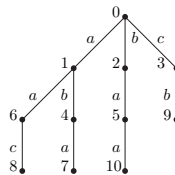
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>					
New node	4	5	6	7	8	9	10	11					



Lempel-Ziv 78 decompression

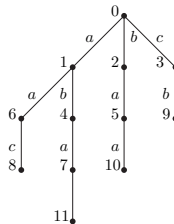
Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>					
New node	4	5	6	7	8	9	10	11					



Node 11 is not in the tree yet

Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>					
New node	4	5	6	7	8	9	10	11					

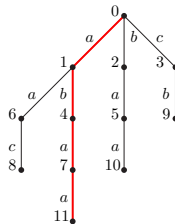


Node 11 is not in the tree yet

This is only possible if node 7 is the parent of node 11

Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a	b	a	ab	aa	c	ba	aba	$abaa$				
New node	4	5	6	7	8	9	10	11					



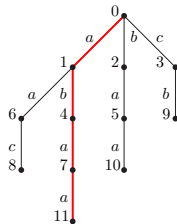
Node 11 is not in the tree yet

This is only possible if node 7 is the parent of node 11

Thus, we descent along arcs $0 - 1, 1 - 4, 4 - 7, 7 - 11$

Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a	b	a	ab	aa	c	ba	aba	$abaa$				
New node	4	5	6	7	8	9	10	11					



Node 11 is not in the tree yet

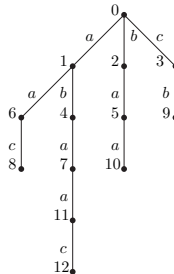
This is only possible if node 7 is the parent of node 11

Thus, we descent along arcs 0 – 1, 1 – 4, 4 – 7, 7 – 11

The symbol on the arc 7 – 11 is a

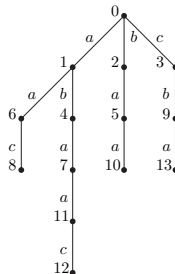
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>	<i>abaa</i>	<i>cb</i>			
New node	4	5	6	7	8	9	10	11	12				



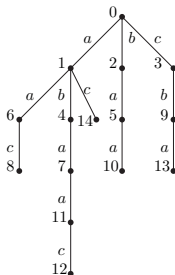
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>	<i>abaa</i>	<i>cb</i>	<i>a</i>		
New node	4	5	6	7	8	9	10	11	12	13			



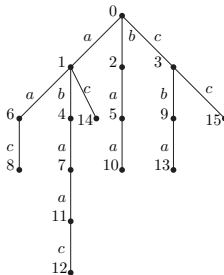
Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	<i>a</i>	<i>b</i>	<i>a</i>	<i>ab</i>	<i>aa</i>	<i>c</i>	<i>ba</i>	<i>aba</i>	<i>abaa</i>	<i>cb</i>	<i>a</i>	<i>c</i>	
New node	4	5	6	7	8	9	10	11	12	13	14		



Lempel-Ziv 78 decompression

Parent	1	2	1	4	6	3	5	7	11	9	1	3	15
String	a	b	a	ab	aa	c	ba	aba	$abaa$	cb	a	c	cc
New node	4	5	6	7	8	9	10	11	12	13	14	15	



- 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 (1-dim)
- Quad Tree (2-dim)
- **B**inary**S**pace**P**artition Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

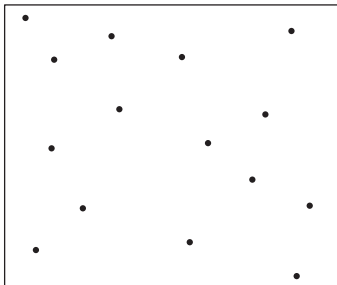
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

■ Sweepline

- sortiert
- topologisch sortiert



Datenstrukturen

■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **BinarySpacePartition Tree** (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

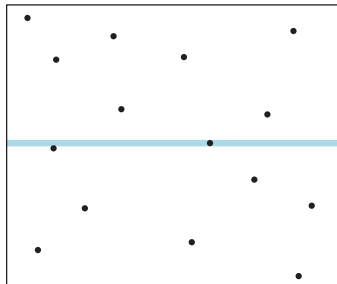
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

■ Sweepline

- sortiert
- topologisch sortiert



Datenstrukturen

■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **BinarySpacePartition Tree** (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

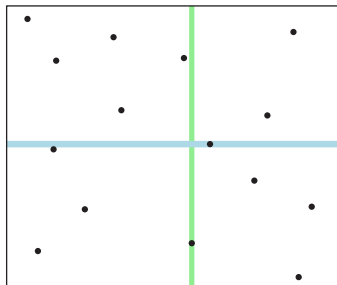
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

■ Sweepline

- sortiert
- topologisch sortiert



Datenstrukturen

■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **BinarySpacePartition** Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

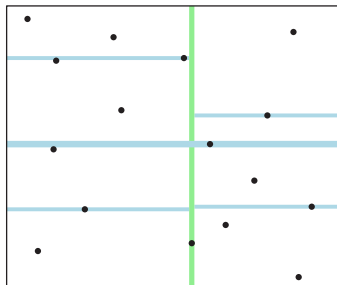
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

■ Sweepline

- sortiert
- topologisch sortiert



Datenstrukturen

■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **B**inary**S**pace**P**artition Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

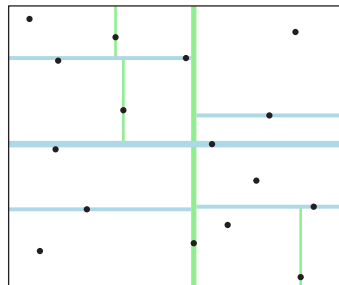
■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

Strukturierter Zugriff

■ Sweepline

- sortiert
- topologisch sortiert



Idee

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

im Allgemeinen

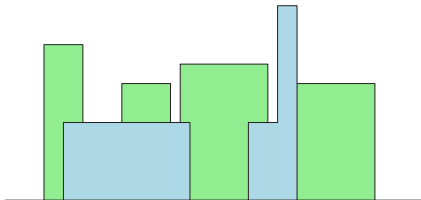
- reduziere n -dim $\rightarrow (n - 1)$ -dim

Sweep-Line

Beispiel: Skyline

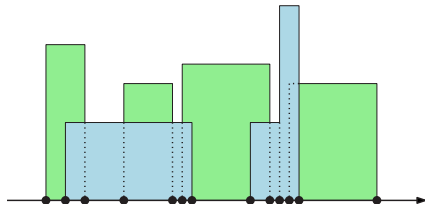
Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)



Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

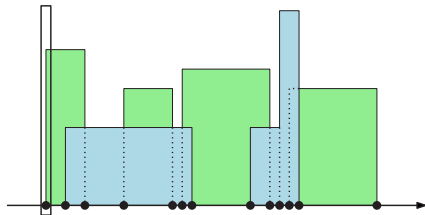


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

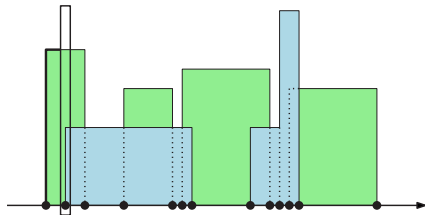


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

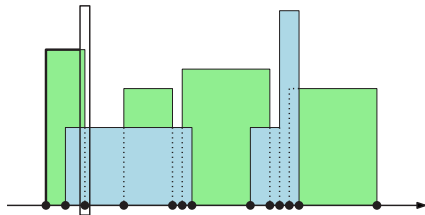


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

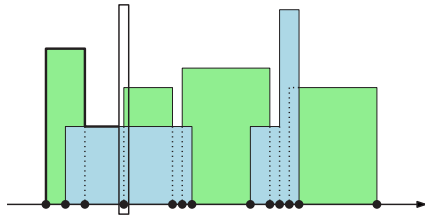


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

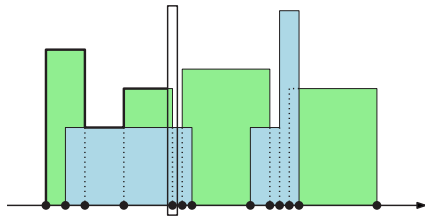


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

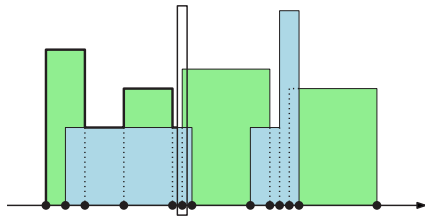


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

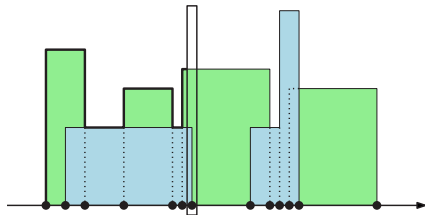


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

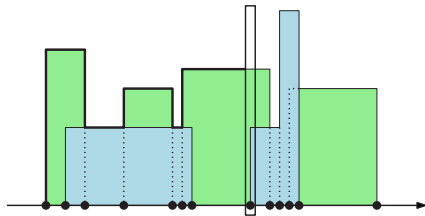


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

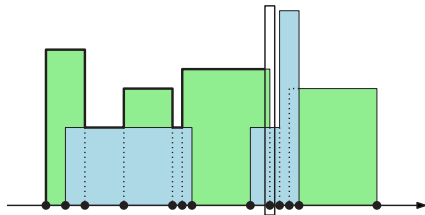


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

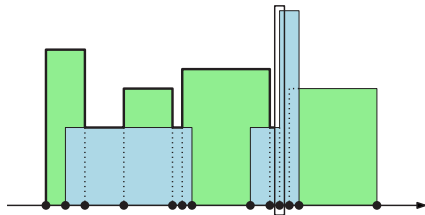


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

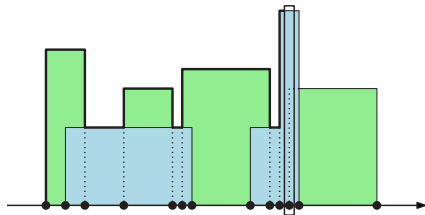


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

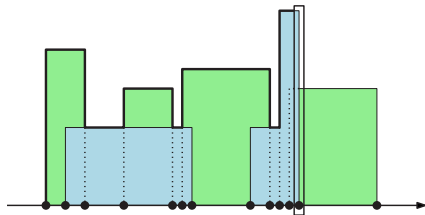


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

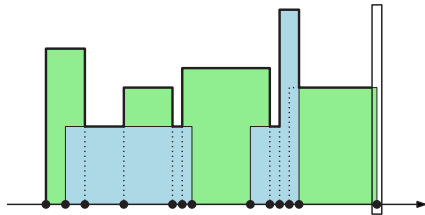


Sweep-Line

Beispiel: Skyline

Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

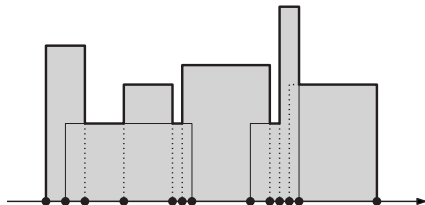


Sweep-Line

Beispiel: Skyline

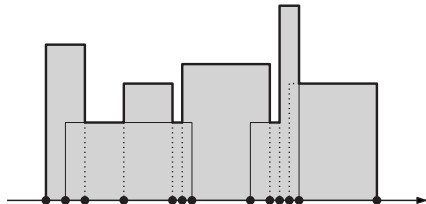
Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)



Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)



Problem: effiziente Lösung des 1-dimensionalen Problems

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

Sortierte Liste

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

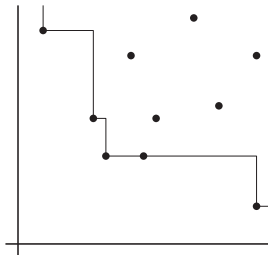
Sortierte Liste

- 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

- $\exists i : a_i < b_i$



Consider query $q = (l, r)$.

$x = \text{root}$

while $x \neq \text{root}$ **do**

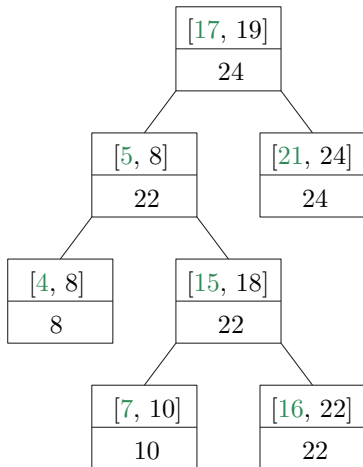
if $x.\text{interval}$ intersects (l, r)

return $x.\text{interval}$

else if $x.\text{left} == \text{null}$ or $x.\text{left.max} < l$

$x = x.\text{right}$

else $x = x.\text{left}$



Interval Search Trees

Consider query $q = (l, r)$.

$x = \text{root}$

while $x \neq \text{root}$ **do**

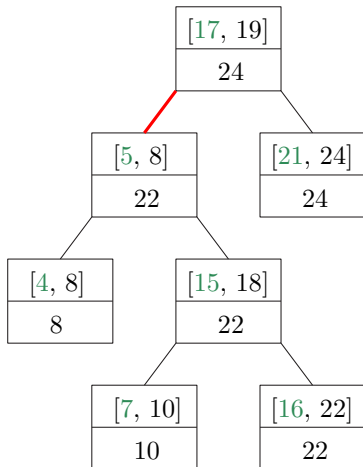
if $x.\text{interval}$ intersects (l, r)

return $x.\text{interval}$

else if $x.\text{left} == \text{null}$ or $x.\text{left}.max < l$

$x = x.\text{right}$

else $x = x.\text{left}$



Interval Search Trees

Consider query $q = (l, r)$.

$x = \text{root}$

while $x \neq \text{root}$ **do**

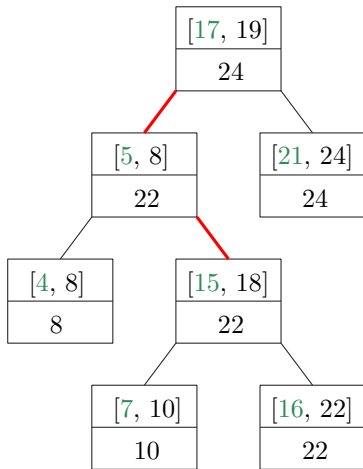
if $x.\text{interval}$ intersects (l, r)

return $x.\text{interval}$

else if $x.\text{left} == \text{null}$ or $x.\text{left.max} < l$

$x = x.\text{right}$

else $x = x.\text{left}$



Interval Search Trees

Consider query $q = (l, r)$.

$x = \text{root}$

while $x \neq \text{root}$ **do**

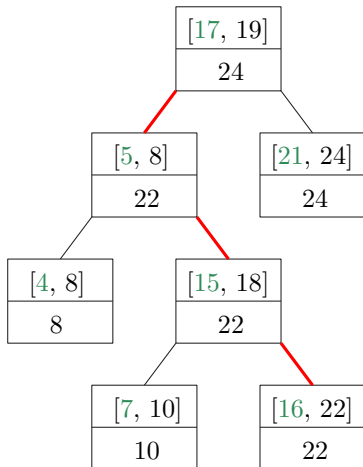
if $x.\text{interval}$ intersects (l, r)

return $x.\text{interval}$

else if $x.\text{left} == \text{null}$ or $x.\text{left.max} < l$

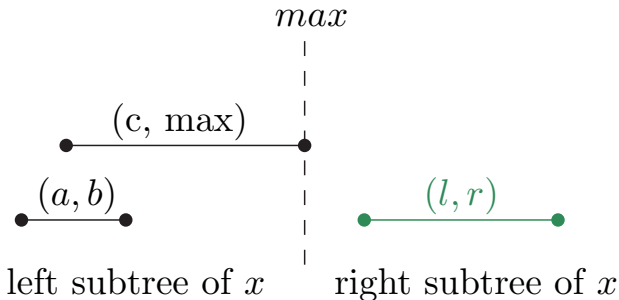
$x = x.\text{right}$

else $x = x.\text{left}$



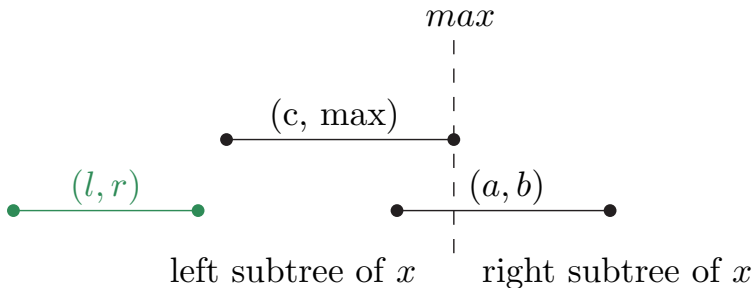
Interval Search Trees

Correctness for choice of a right child



Interval Search Trees

Correctness for choice of a left child



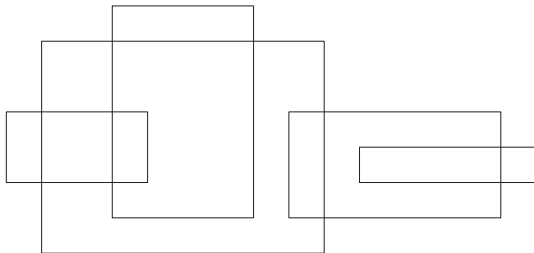
Interval Search Trees

Complexity

- Find an interval that intersects a query interval: $O(\log n)$
- Find all intervals that intersect a query interval: $O(t \log n)$, where t is the number of such intervals

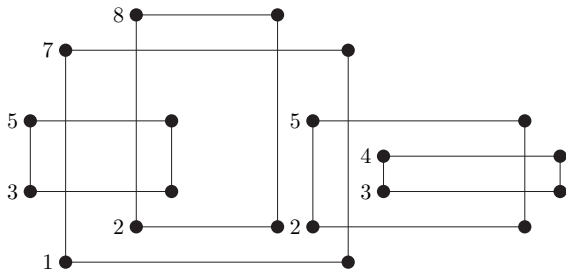
Rectangle Intersection Problem

Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



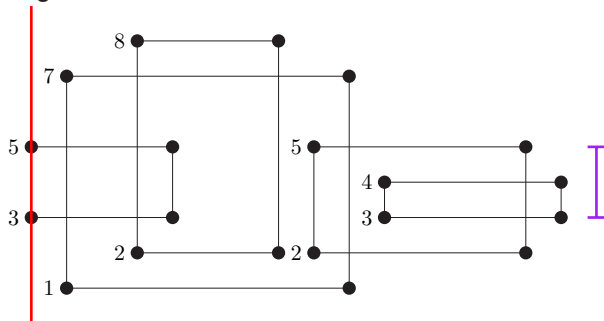
Rectangle Intersection Problem

Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



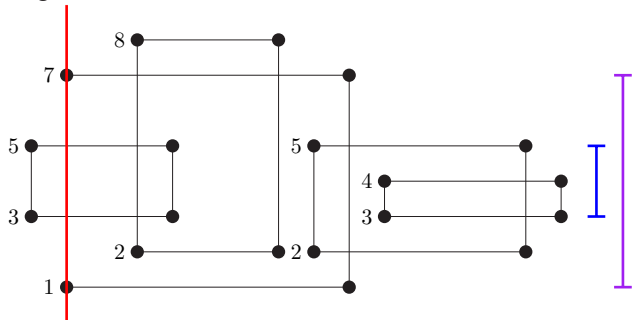
Rectangle Intersection Problem

Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



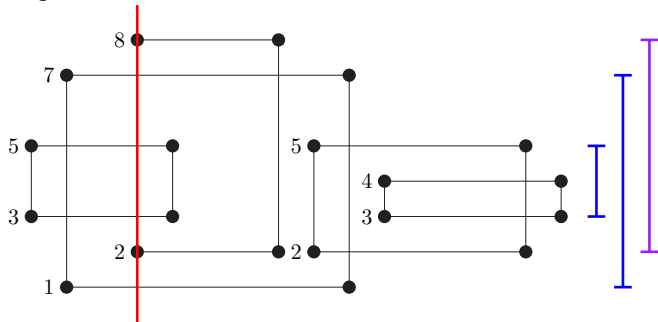
Interval Search Tree: (3, 5)

Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles

Interval Search Tree: $(3, 5)(1, 7)$

Rectangle Intersection Problem

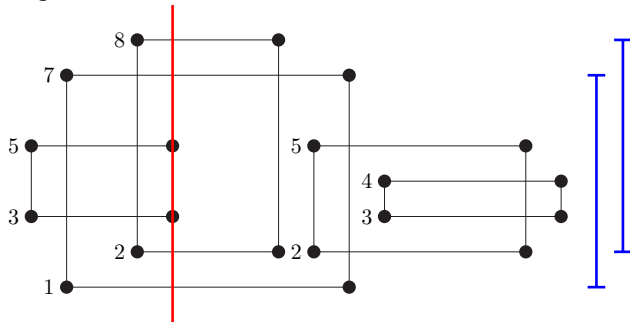
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: $(3, 5)(1, 7)(2, 8)$

Rectangle Intersection Problem

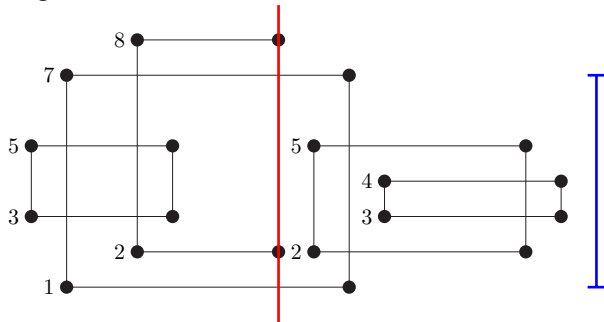
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: $(1, 7)(2, 8)$

Rectangle Intersection Problem

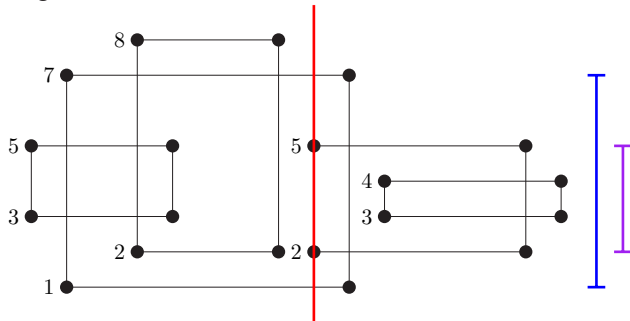
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: (1, 7)

Rectangle Intersection Problem

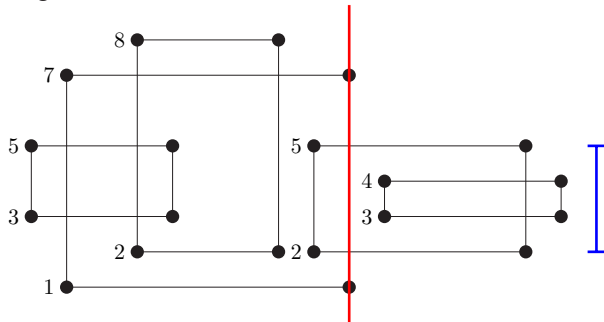
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: $(1, 7)(2, 5)$

Rectangle Intersection Problem

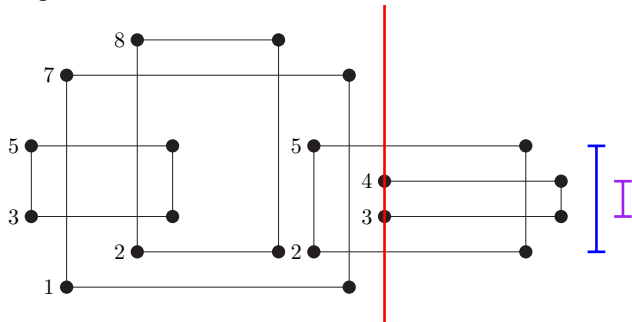
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: (2, 5)

Rectangle Intersection Problem

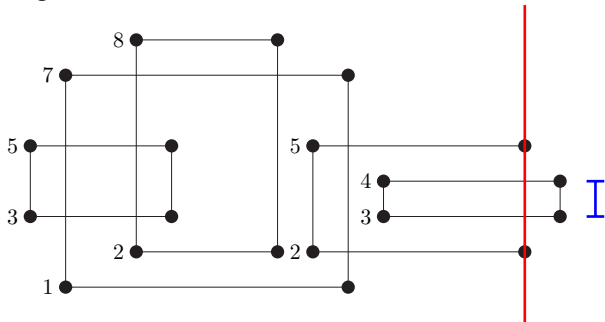
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: $(2, 5)(3, 4)$

Rectangle Intersection Problem

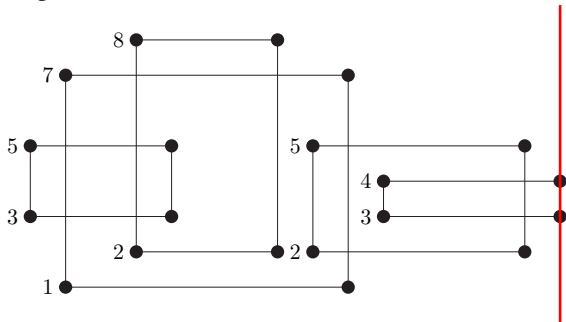
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



Interval Search Tree: (3, 4)

Rectangle Intersection Problem

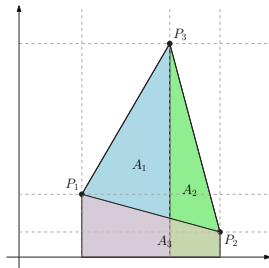
Complexity $n \log n + t \log n$, where t is the number of intersecting rectangles



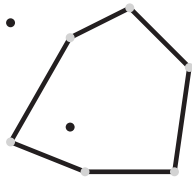
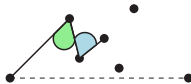
Interval Search Tree:

Vorzeichenbehaftete Fläche

- Dreiecksfläche als Summe dreier Trapezflächen
- Z-Koordinate von Kreuzprodukt der Vektoren $\overrightarrow{P_1P_2}$ und $\overrightarrow{P_1P_3}$:
 $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$
- Verschiebung von P_1 nach $(0, 0)$ ergibt:
$$A = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}$$
- Vorzeichen positiv gdw. P_1, P_2, P_3 CCW



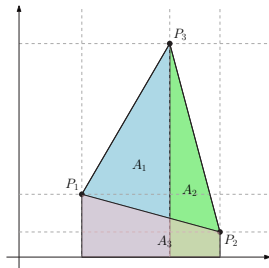
- Unterproblem von Algorithmen
 - Graham Scan
 - Test auf Enthaltensein



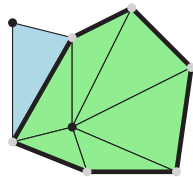
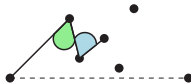
(Punkt in konvexem Polygon)

Vorzeichenbehaftete Fläche

- Dreiecksfläche als Summe dreier Trapezflächen
- Z-Koordinate von Kreuzprodukt der Vektoren $\overrightarrow{P_1P_2}$ und $\overrightarrow{P_1P_3}$:
 $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$
- Verschiebung von P_1 nach $(0, 0)$ ergibt:
$$A = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}$$
- Vorzeichen positiv gdw. P_1, P_2, P_3 CCW



- Unterproblem von Algorithmen
 - Graham Scan
 - Test auf Enthaltensein



(Punkt in konvexem Polygon)

Ende!



Feierabend!