LMU LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
DEPARTMENT INSTITUTE FOR INFORMATICS
DATABASE SYSTEMS GROUP

Kapitel 6: Algorithmen der Computer-Geometrie

Skript zur Vorlesung

Geo-Informationssysteme

Wintersemester 2014/15

Ludwig-Maximilians-Universität München

6. Algorithmen der Computer-Geometrie

1. Einführung
2. Schnitt von zwei Strecken
3. Punkt-in-Polygon-Test
4. Schnitt orthogonaler Strecken
5. Punkteinschlussproblem

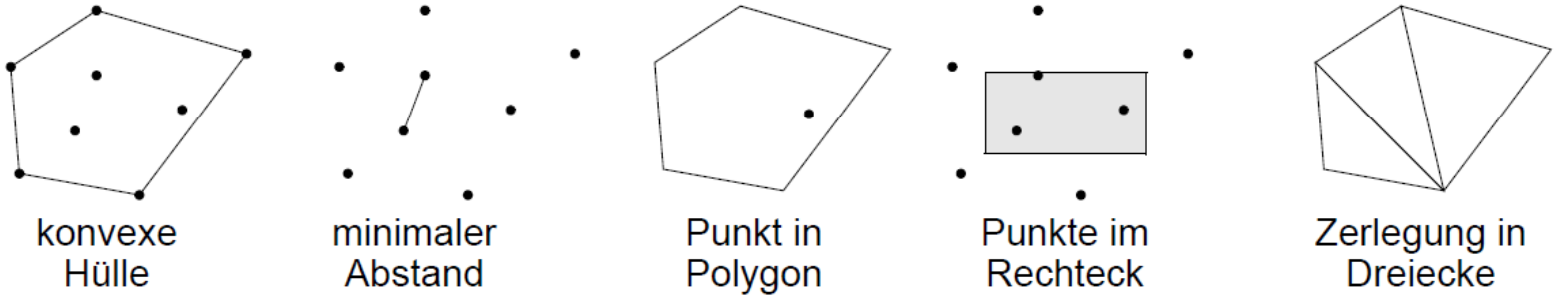
6.1 Computer-Geometrie (I)

Computer Geometrie

- Teilgebiet der Informatik, das sich mit der Lösung geometrischer Fragestellungen durch geeignete Algorithmen und Datenstrukturen beschäftigt
- Die algorithmische Lösung einer einfachen geometrischen Fragestellung kann sehr komplex sein (vom Algorithmus und/oder von der Laufzeit her)
- Die Algorithmische Geometrie legt ihren Schwerpunkt auf die Studie der algorithmischen Komplexität elementarer geometrischer Probleme
- Den Einsatz von geometrischen Algorithmen in einem GIS hat aber u.a. folgende Probleme:
  - Komplexität der Implementierung (viele Sonderfälle)
  - Anfälligkeit für Rundungsfehler
  - Hohe Konstanten bei der Laufzeitkomplexität

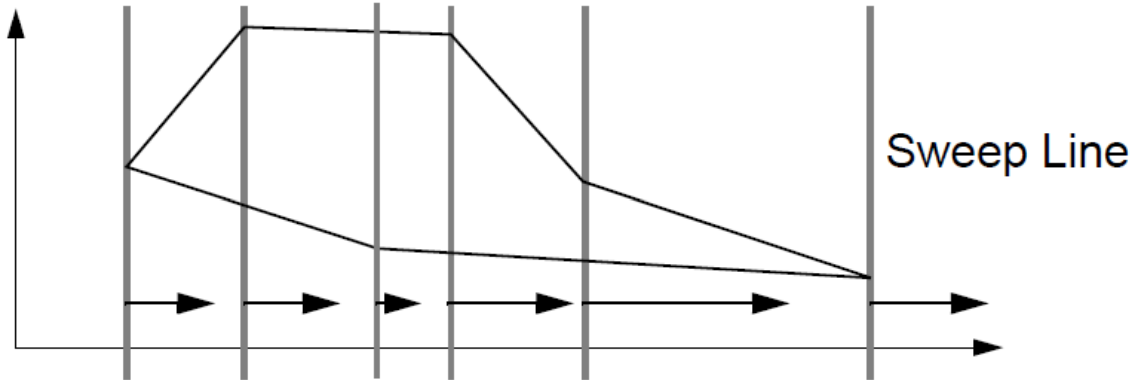
Wichtige Probleme

- Berechnung der konvexen Hülle
- Distanzprobleme:
  - Bestimme das Paar von Punkten mit minimalem Abstand
- Inklusionstest und -berechnungen:
  - Teste, ob ein Punkt im Inneren eines geschlossenen Polygons liegt
  - Bestimme alle Punkte, die in einem Rechteck liegen
- Zerlegungen von Objekten in Primitive:
  - Zerlege ein Polygon in eine minimale Anzahl von Dreiecken



Wichtige Paradigmen

- *Plane-Sweep*
  - Verwendung einer vertikalen Lauflinie (*Sweep Line*)



- *Divide-and-Conquer (Teile und Herrsche)*
  - Aufteilen des Problems in Teilprobleme
  - Lösen der Teilprobleme
  - Kombinieren der Lösungsmengen der Teilprobleme

CG-Probleme beim Spatial Join

- "Welche Paare von MURs besitzen nichtleeren Schnitt?"

  ⇒ Fall1: ein Kantenpaar schneidet sich

  ⇒ Schnitt einer Menge von orthogonalen Strecken (1)

  ⇒ Fall 2: ein MUR liegt vollständig innerhalb des anderen

  ⇒ Punkteinschlussproblem, d.h. Schnitt einer Menge von Punkten und einer Menge von Rechtecken (2)

- "Schneiden sich diese beiden Polygone?"

  ⇒ Schnitt zweier Strecken

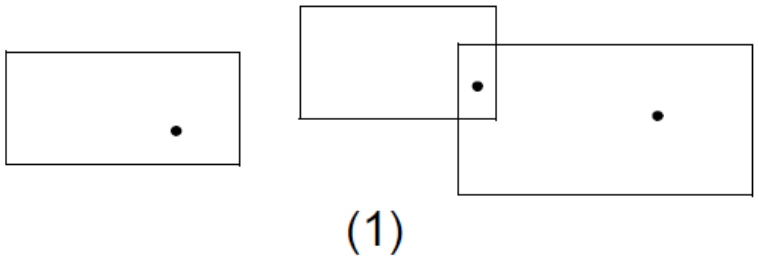
CG-Probleme bei Punktanfragen

- "Welche Punkte liegen in welchen dieser MURs?" (Spatial Join zwischen Punkten und MUR's) mit dem Prädikat "*Inklusion*"

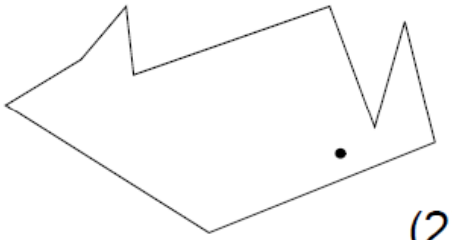
  ⇒ Punkteinschlussproblem (1)

- "Liegt der Anfragepunkt in diesem Polygon?"

  ⇒ Punkt-In-Polygon Test (2)



(1)



(2)

CG-Probleme bei Fensteranfragen

- "Welche dieser MURs schneiden das Anfragefenster?"

  ⇒ Schnitt einer Menge von orthogonalen Strecken
- "Schneidet das Anfragefenster dieses Polygon?"

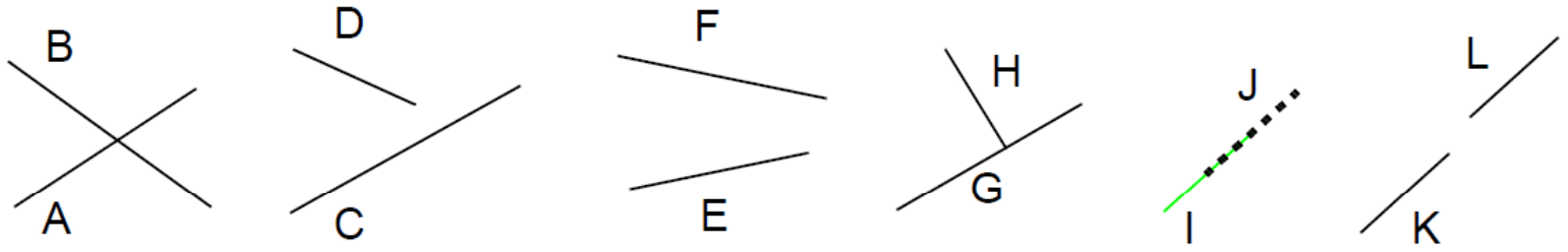
  ⇒ Schnitt zweier Strecken

6.2 Schnitt von zwei Strecken (I)

Gegeben: zwei Strecken S1 und S2

Gesucht: Schneiden sich S1 und S2? Wenn ja: ihr Schnittpunkt.

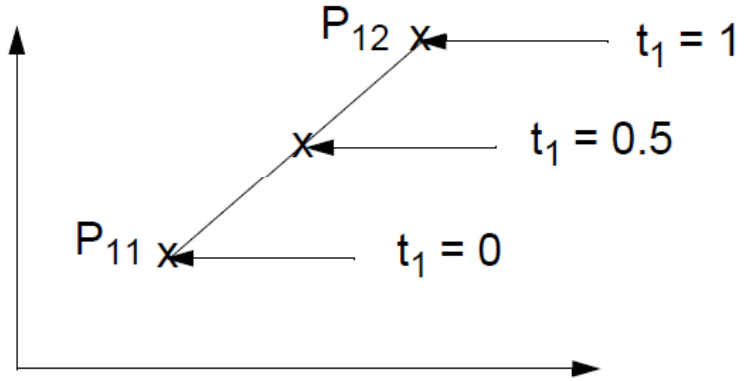
Fälle:



- Strecken A und B schneiden sich
- Strecken C und D und Strecken E und F schneiden sich nicht, jedoch schneiden sich ihre zugehörigen Geraden
- Strecken G und H berühren sich
- Strecken I und J liegen parallel und überlappen sich
- Strecken L und K liegen parallel und schneiden sich nicht

6.2 Schnitt von zwei Strecken (II)

- Gegeben seien die beiden Strecken s_1 und s_2 durch die Eckpunkte P_{i1} und P_{i2}
- *Parameterdarstellung* mit $t_1, t_2 \in [0,1]$:
  - s_1: $(1-t_1) \cdot P_{11} + t_1 \cdot P_{12}$
  - s_2: $(1-t_2) \cdot P_{21} + t_2 \cdot P_{22}$



- 3 Fälle:
  1. s1 und s2 liegen auf der gleichen Geraden
  2. s1 und s2 liegen auf unterschiedlichen parallelen Geraden
  3. Die Geraden von s1 und s2 haben genau einen Schnittpunkt

6.2 Schnitt von zwei Strecken (III)

Schnittpunktberechnung (Annahme: s1 und s2 weder horizontal noch vertikal)

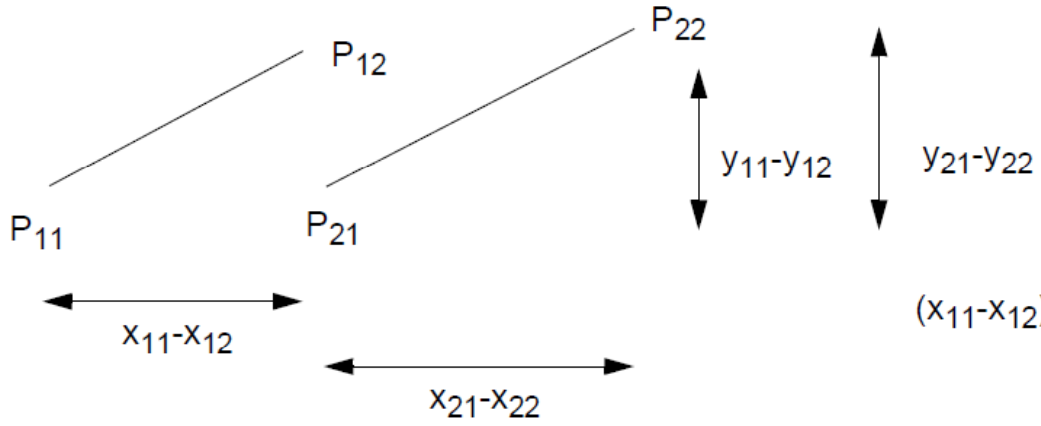
Notation: $x_{ij} := P_{ij}.x$ $y_{ij} := P_{ij}.y$

$(1-t_1) \cdot x_{11} + t_1 \cdot x_{12} = (1-t_2) \cdot x_{21} + t_2 \cdot x_{22}$ // der Schnittpunkt erfüllt

$(1-t_1) \cdot y_{11} + t_1 \cdot y_{12} = (1-t_2) \cdot y_{21} + t_2 \cdot y_{22}$ // beide Streckengleichungen

$$\Rightarrow t_1 = \frac{(x_{11}-x_{21})\cdot(y_{21}-y_{22})-(y_{11}-y_{21})\cdot(x_{21}-x_{22})}{(x_{11}-x_{12})\cdot(y_{21}-y_{22})-(y_{11}-y_{12})\cdot(x_{21}-x_{22})}$$, t_2 analog

Nenner gleich 0 ⇒ Strecken parallel (s_1 und s_2 haben die selbe Steigung)



$(x_{11}-x_{12})\,(y_{21}-y_{22}) = (y_{11}-y_{12})\,(x_{21}-x_{22})$

6.2 Schnitt von zwei Strecken (IV)

Fallunterscheidung

- *Nenner gleich 0*

  Teste, ob P_{11} oder P_{12} auf s_2 liegen, d.h. die Streckengleichung erfüllen.

  a) P_{11} oder P_{12} liegen auf s_2

  Die Strecken s_1 und s_2 überlappen sich. Alle Punkte im Überlappungsbereich sind Schnittpunkte.

  b) weder P_{11} noch P_{12} liegen auf s_2

  Die Strecken s_1 und s_2 liegen nebeneinander, sie besitzen keinen Schnittpunkt.

- *Nenner ungleich 0*

  a) $0 \leq t_1 \leq 1$ und $0 \leq t_2 \leq 1$

  Die Strecken s_1 und s_2 schneiden sich. Berechnung des Schnittpunkts durch Einsetzen von t_i in Gleichung von s_i

  b) $t_1 < 0$ oder $t_1 > 1$ oder $t_2 < 0$ oder $t_2 > 1$

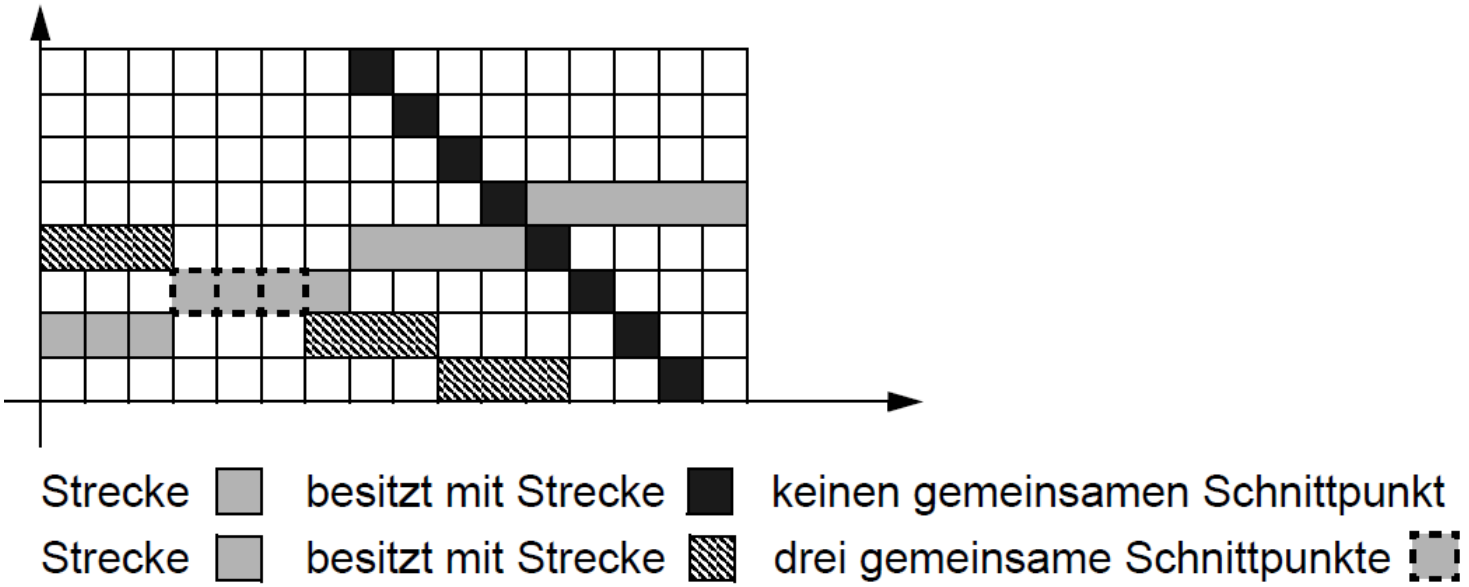
  Die Strecken s_1 und s_2 schneiden sich nicht.

6.2 Schnitt von zwei Strecken (V)

Ungenauigkeiten im Rastermodell

Wieviele Schnittpunkte hat ein Paar von Strecken?

- Euklidische Geometrie: (höchstens) 1 Schnittpunkt
- Rastermodell: ???



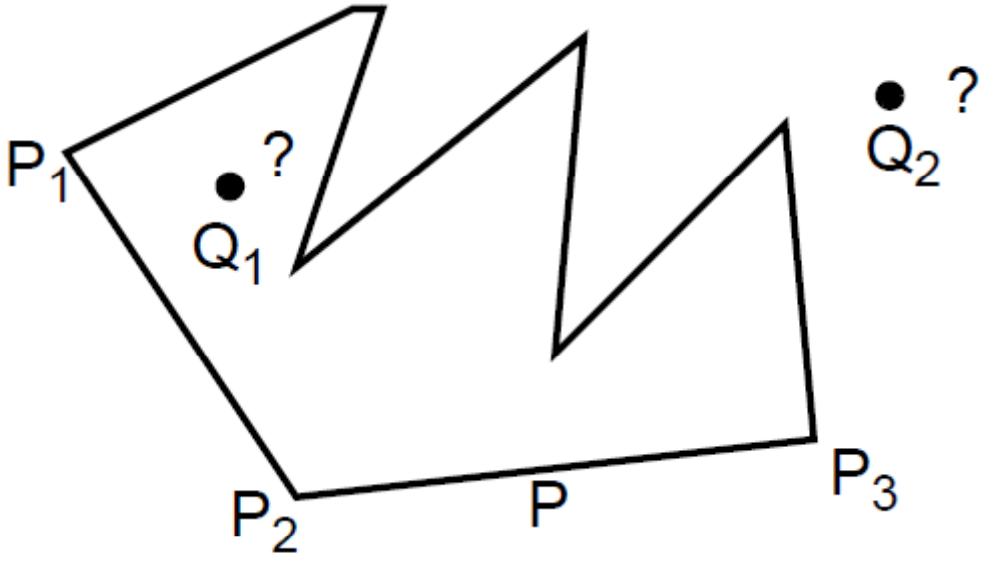
6.3 Punkt-in-Polygon-Test (I)

Gegeben

Ein einfaches Polygon $P = (P_1,\ldots,P_n)$ und ein Anfragepunkt Q

Gesucht

Liegt Punkt Q im Inneren des Polygons P?

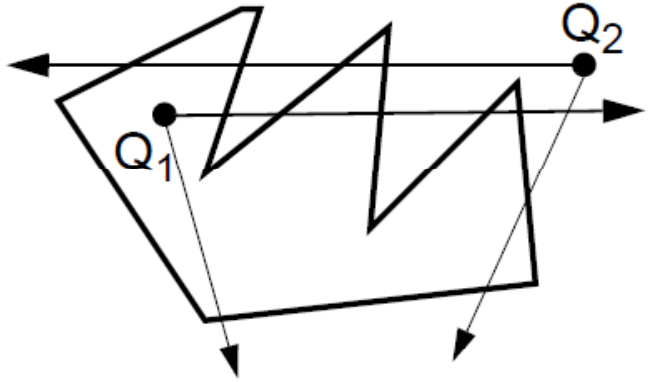


6.3 Punkt-in-Polygon-Test (II)

Lösung 1

- Grundlage: *Jordansches Kurven-Theorem*
  Jedes Polygon zerteilt den Raum in ein Inneres und Äußeres.
- Idee:

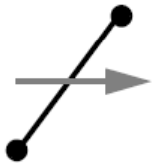
  Schicke einen Strahl S (der Einfachheit halber parallel zur x-Achse) vom Punkt Q und zähle, wie oft er die Kanten des Polygons schneidet:
  - Anzahl ungerade: Q liegt im Inneren
  - Anzahl gerade: Q liegt im Äußeren
- Beispiel:



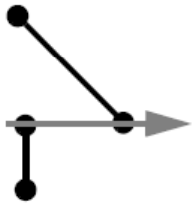
6.3 Punkt-in-Polygon-Test (III)

Algorithmus

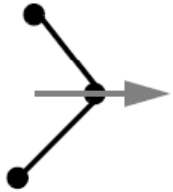
- Für jede der n Kanten des Polygons P:
  Berechne den Schnitt der Kante mit dem Strahl S. Falls Schnittpunkt vorhanden, inkrementiere Zähler.
- Laufzeitkomplexität:
  Schnitt zweier Strecken ist O(1), O(n) Aufrufe davon --> O(*n*)
- Fallunterscheidung (siehe Übung)



1 Schnitt



1, 2 oder 3 Schnitte?



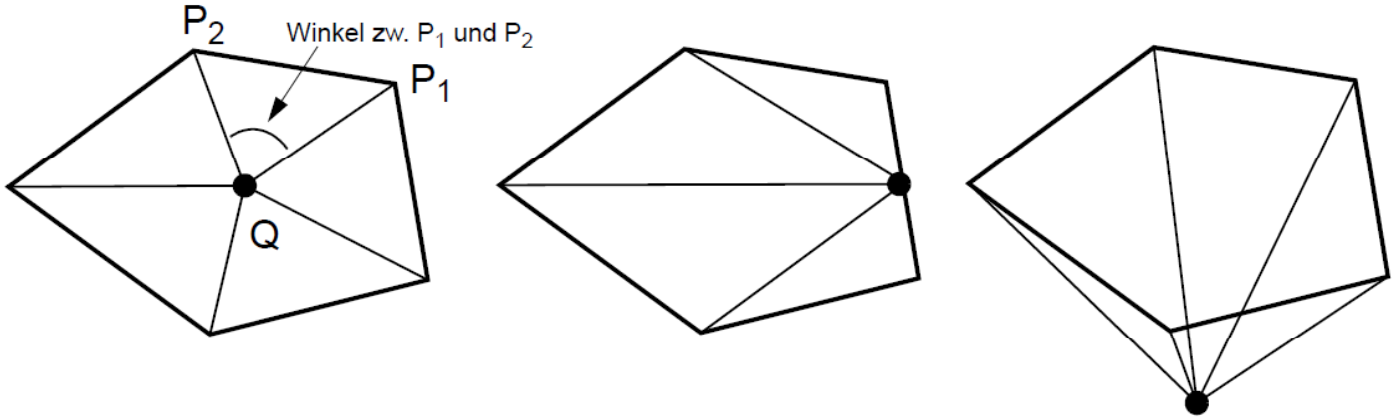
1 oder 2 Schnitte?

wieviel Schnitte?

6.3 Punkt-in-Polygon-Test (IV)

Lösung 2

- Die zu einem Polygon R und Punkt Q gehörenden Sektorenstrahlen sind durch die Kanten von Q nach P_i gegeben.
- Berechne $\Theta(R,Q)$, die Summe aller Winkel zwischen benachbarten Sektorenstrahlen.
- Es gilt:
  - $\Theta(R,Q) = 2\pi$, so liegt Q im Inneren von R
  - $\Theta(R,Q) = \pi$, so liegt Q auf dem Rand von R
  - $\Theta(R,Q) = 0$, so liegt Q im Äußeren von R



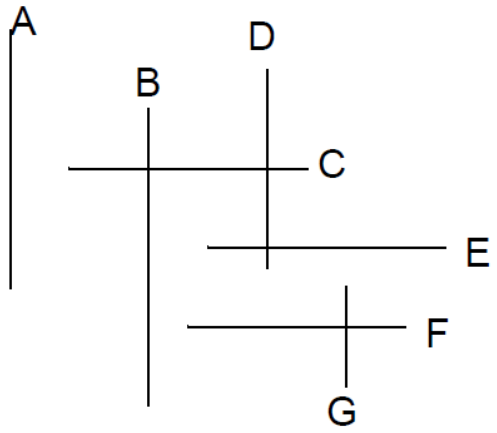
6.4 Schnitt orthogonaler Strecken (I)

Gegeben

Menge von horizontalen und vertikalen *(orthogonalen)* Strecken

Gesucht

Alle Paare sich schneidender Strecken



Lösung:
(B,C)
(C,D)
(D,E)
(F,G)

Annahme

- Die Endpunkte verschiedener Strecken besitzen jeweils unterschiedliche x- und y-Koordinaten
- damit sind Schnitte nur zwischen horizontalen und vertikalen Strecken möglich

6.4 Schnitt orthogonaler Strecken (II)

Naiver Algorithmus

- benötigt $O(n^2)$ *Vergleiche*
- da jede Strecke mit allen anderen Strecken verglichen wird

Bessere Algorithmen

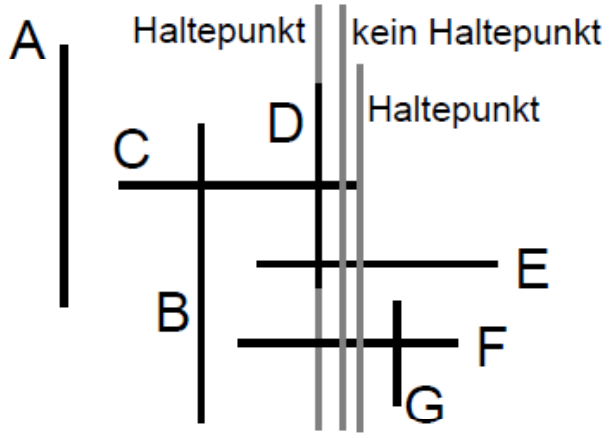
- Vorstellung von zwei Lösungen, die auf den Paradigmen
  - Plane-Sweep
  - Divide-and-Conquer

  basieren
- Beide Algorithmen besitzen eine Laufzeit von $O(n \log n + k)$
- k ist die Zahl der Antworten

6.4 Plane-Sweep-Lösung (I)

Idee

- Schiebe eine vertikale Gerade (*Sweep Line), von links nach rechts durch den 2*-dim. Datenraum
- Halte dabei nur an "interessanten" Stellen *(Event Points)*
- Beobachte dann alle Objekte, die die Sweep Line schneiden



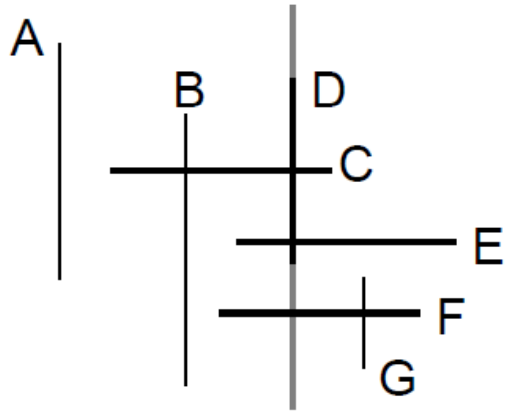
Datenstrukturen

- In dem *Event Point Schedule* werden alle Haltepunkte der Sweep Line in x-Richtung sortiert organisiert.
- In dem *Sweep Line Status* werden die aktuell die Sweep Line schneidenden Objekte verwaltet; insbesondere muß hier das Löschen und Einfügen der Objekte effizient gelöst sein

6.4 Plane-Sweep-Lösung (II)

Beobachtungen für Schnitt orthogonaler Strecken

- Horizontale Strecken schneiden die Sweep Line in einer festen y-Koordinate
- Vertikale Strecken schneiden die Sweep Line in einem y-Intervall. Horizontale
- Strecken, die diese vertikale Strecke schneiden, erfüllen folgende Eigenschaften:
  - Sie schneiden auch die Sweepline
  - Ihre y-Koordinate liegt im y-Intervall



6.4 Plane-Sweep-Lösung (III)

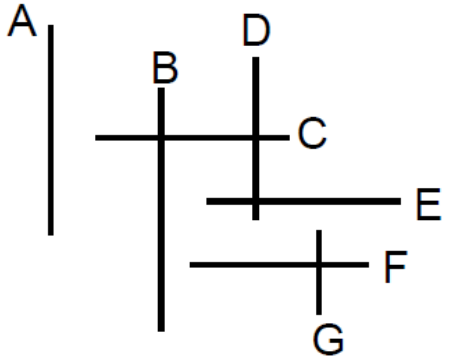
Datenstrukturen

- *Event Point Schedule:* enthält die x-Koord. jeder vertikalen Strecke und die min. und max. x-Koord. jeder horizontalen Strecke (sortiert)
- *Sweep Line Status:* enthält alle horizontalen Strecken, die die Sweep Line momentan schneiden (sortiert)
- Algorithmus

```
FOR each s.x IN Event-Point-Schedule WITH ASCENDING ORDER OF x DO
    IF s is horizontal segment THEN
        IF x is minimum point of segment s THEN
                insert s into Sweep-Line-Status
        ELSIF x is maximum point of segment s THEN
                remove s from Sweep-Line-Status
    ELSIF s is vertical segment THEN
        intersections := Sweep-Line-Status.range-query (s.y1 ≤ t.y ≤ s.y2);
        FOR each t IN intersections DO
                report “s and t intersect”;
        END FOR each t IN intersections;
END FOR each s.x IN Event-Point-Schedule;
```

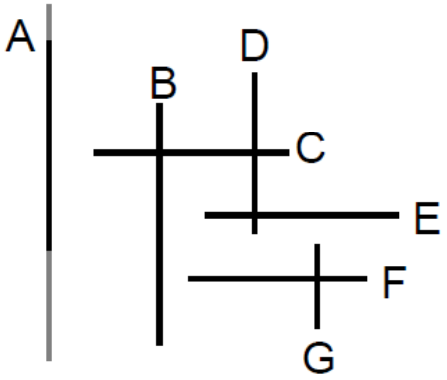
6.4 Plane-Sweep-Lösung (IV)

Vorgehen am Beispiel

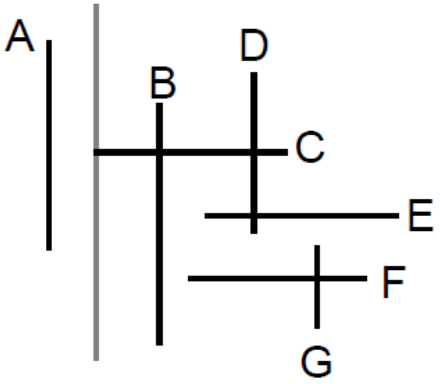


Event Point Schedule mit allen Haltepunkten:
A.x, C.x1, B.x, F.x1, E.x1,
D.x, C.x2, G.x, F.x2, E.x2

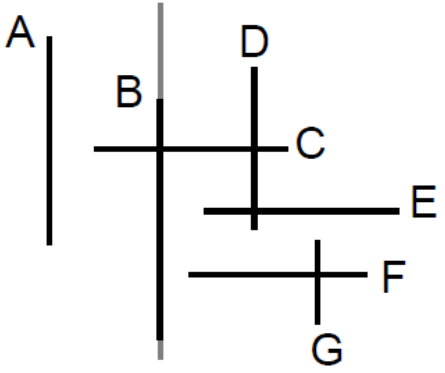
Sweep Line Status:
Stat = {}



Haltepunkt: A.x
Stat = {}



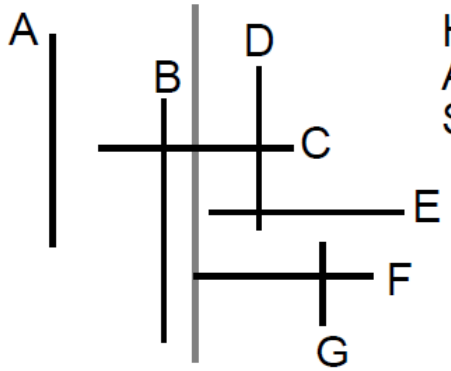
Haltepunkt: C.x1
Aktion: Füge C in Stat ein
Stat = {C}.



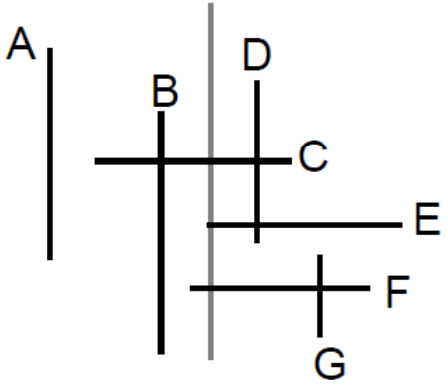
Haltepunkt: B.x
Aktion: Finde alle l ∈ Stat
mit B.y1 ≤ l.y ≤ B.y2
Stat = {C} ⇒ l = C
Antwort: (B,C)

6.4 Plane-Sweep-Lösung (V)

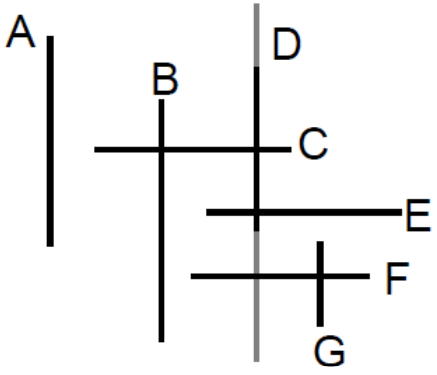
Vorgehen am Beispiel (Forts.)



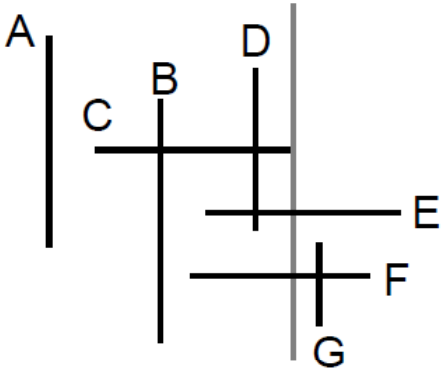
Haltepunkt: F.x1
Aktion: Füge F in Stat ein
Stat = {C, F}.



Haltepunkt: E.x1
Aktion: Füge E in Stat ein
Stat = {C,E,F} (*"sortiert"*!)



Haltepunkt: D.x
Aktion: Finde alle $l \in$ Stat
mit $D.y1 \leq l.y \leq D.y2$
Stat = {C,E,F} $\Rightarrow l \in$ {C,E}
Antworten: (D,C), (D,E)



Haltepunkt: C.x2
Aktion: Lösche C aus Stat
Stat = {E,F}.

6.4 Plane-Sweep-Lösung (VI)

Komplexitätsberechnung

- Zum Abspeichern des Sweep Line Status wird ein höhenbalancierter Baum (z.B. ein 2-3+-Baum) verwendet, wobei die Blätter bzgl. der Sortierung miteinander verkettet sind
- wichtige Eigenschaften (bei n gespeicherten Strecken):
  - Einfügen und Löschen benötigt $O(\log n)$ Zeit
  - Bereichsanfrage (mit r Antworten) benötigt $O(\log n + r)$ Zeit
- Laufzeitkomplexität des Plane-Sweep-Algorithmus
  Aufbau des Event Point Schedule: $O(n \log n)$
  Durchlauf der Sweep Line:
  - Einfügen in Sweep Line Status: $O(n \log n)$
  - Löschen aus Sweep Line Status: $O(n \log n)$
  - Bereichsanfragen: $O(n \log n + k)$

$\Rightarrow$ Gesamtkosten: $O(n \log n + k)$

6.4 Divide-and-Conquer-Lösung (I)

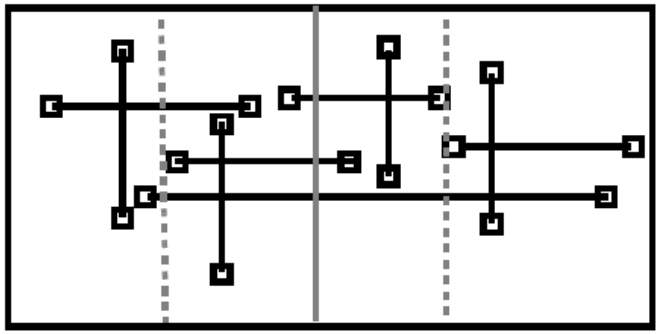
Idee

- Teile die Menge der Objekte in zwei Teilmengen *(Divide)*
- Löse das Problem für die Teilmengen *(Conquer)*
- Verschmelze die Teil-Lösungen zu einer Gesamtlösung *(Merge)*

Anwendung auf Schnittproblem orthogonaler Strecken

- Divide:
  - Betrachte die Menge S aller x-Koordinaten von vertikalen Strecken und aller Endpunkte horizontaler Strecken
  - Teile S in zwei etwa gleichgroße Mengen S_1 und S_2 auf, so daß für alle $x_1 \in S_1$ und $x_2 \in S_2$ gilt: $x_1 < x_2$
- Conquer:

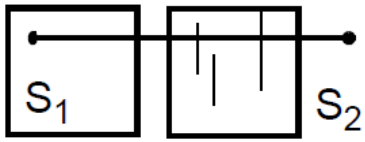
  Löse das Problem für die Mengen S_1 und S_2



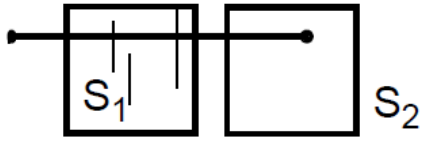
Merge beim Schnittproblem orthogonaler Strecken

Beim Merge werden Schnittpunkte nur für folgende Fälle berechnet:

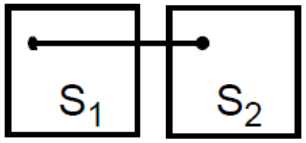
1. Der linke Endpunkt einer horizontalen Strecke liegt in S_1, und der rechte Endpunkt liegt rechts von S_2.
2. Der rechte Endpunkt einer horizontalen Strecke liegt in S_2, und der linke Endpunkt liegt links von S_1.



Strecke schneidet alle vertikalen Strecken in S_2 bzw. S_1, deren y-Intervall die y-Koordinate überdeckt



Insbesondere sind damit alle Schnitte für den Fall, in dem der linke Eckpunkt einer Strecke in S_1 und der rechte Endpunkt in S_2 liegt, immer schon vorher berechnet worden.



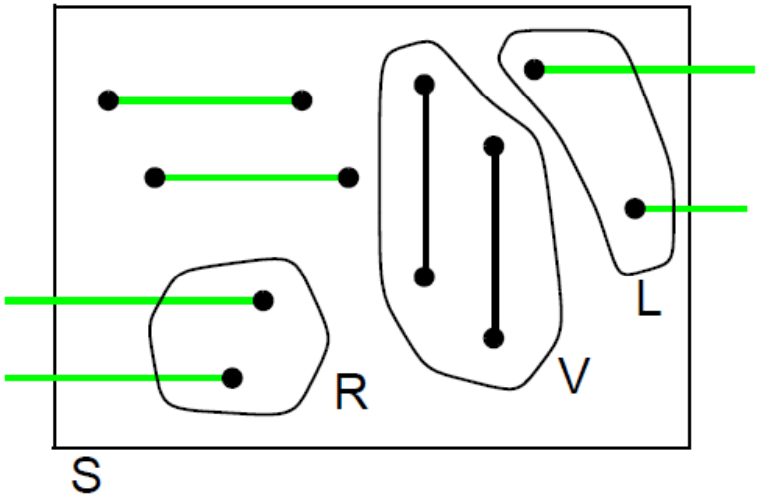
Alle Schnitte mit Strecken in S_1 bzw. S_2 sind schon berechnet.

Definition: $Y \bigstar V := \{ (v.x , y) \mid v \in V, y \in Y, v.y1 \leq y \leq v.y2\}$

6.4 Divide-and-Conquer-Lösung (III)

Datenstrukturen

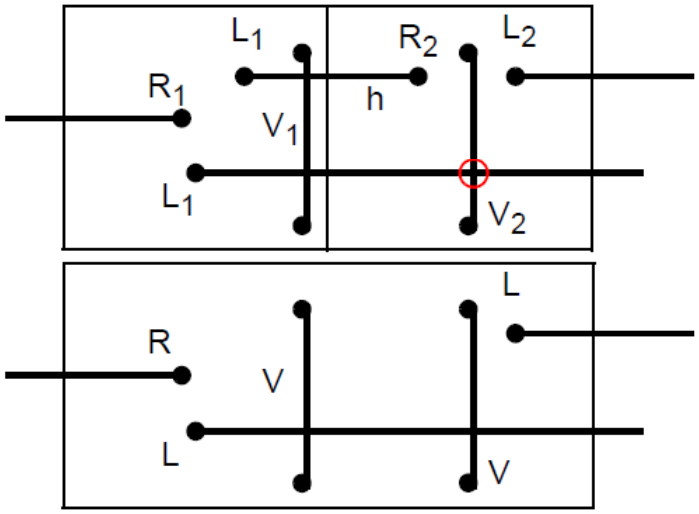
- S ist ein nach den x-Koordinaten sortiertes Array von horizontalen Eckpunkten bzw. vertikalen Strecken
- L ist eine nach y-Koordinaten sortierte Liste aller linken Endpunkte in S, deren rechter Endpunkt nicht in S liegt
- R ist eine nach y-Koordinaten sortierte Liste aller rechten Endpunkte in S, deren linker Endpunkt nicht in S liegt
- V ist eine nach y-Koordinaten des unteren Endpunkts sortierte Liste aller vertikalen Strecken in S



6.4 Divide-and-Conquer-Lösung (IV)

Algorithmus StreckenSchnitt ($_{IN}$ S, $_{OUT}$ L, $_{OUT}$ R, $_{OUT}$ V)

IF |S| = 1 THEN
  L := ∅; R := ∅; V := ∅;
  IF S[1] linker Eckpunkt THEN L := {s}; (* s ∈ S *)
  IF S[1] rechter Eckpunkt THEN R := {s};
  IF S[1] vertikale Strecke THEN V := {s};
ELSE
  Teile S in zwei gleich große Mengen S_1 und S_2 auf, so daß
    für alle $x_1 \in S_1$ und $x_2 \in S_2$ gilt: $x_1 < x_2$;
  StreckenSchnitt (S_1, L_1, R_1, V_1);
  StreckenSchnitt (S_2, L_2, R_2, V_2);
  h := $L_1 \cap R_2$;
  Gebe ($L_1 \setminus h$) ✯ V_2 aus; (* Fall 1 *)
  Gebe ($R_2 \setminus h$) ✯ V_1 aus; (* Fall 2 *)
  L := ($L_1 \setminus h$) ∪ L_2;
  R := $R_1 \cup (R_2 \setminus h)$;
  V := $V_1 \cup V_2$;



// Y ✯ V := { (v.x , y) | v ∈V, y ∈ Y, v.y1 ≤ y ≤ v.y2}

6.4 Divide-and-Conquer-Lösung (V)

Komplexitätsanalyse

- Sortierung von S: O(*n* log *n*)
- Divide (ein Schritt): O(1)
- Merge:
  - Die Mengenoperationen (∩, ∪) auf L, R und V (ein Schritt) können in O(*n*) bearbeitet werden
  - Die Operation M ✯ V kann in O(|M| + |V| + |M ✯ V|) durchgeführt werden (d.h. lineare Laufzeit + Anzahl der gefundenen Paare)
  - Anzahl der Rekursionen: O(log *n*)
- Gesamtaufwand: O(*n* log *n* + *k*)

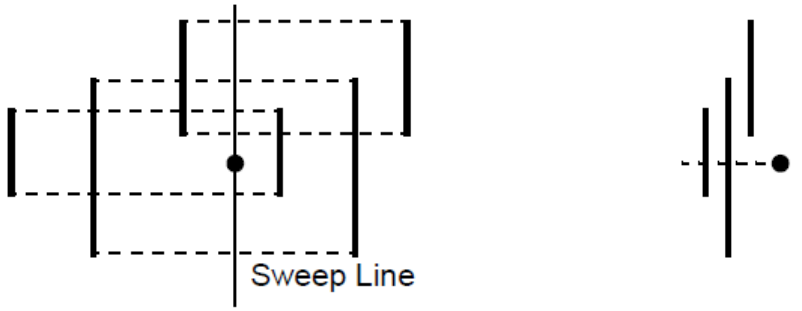
6.5 Punkteinschlußproblem (I)

Gegeben Menge P von Punkten; Menge R von Rechtecken.

Gesucht Alle Paare (p,r) mit p ∈ P, r ∈ R, wobei p in r liegt

Plane-Sweep-Lösung

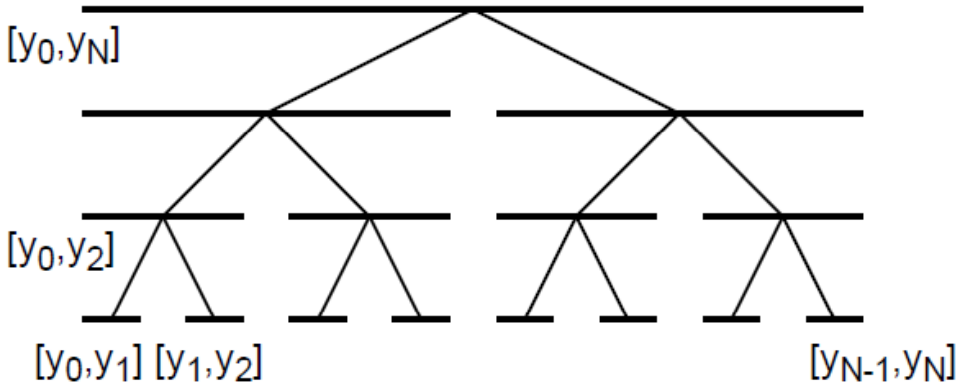
- Betrachte nur die vertikalen Kanten der Rechtecke
- Menge der Haltepunkte (Event Point Schedule) setzt sich zusammen aus allen Punkten p und den x-Koordinaten der vertikalen Kanten (nach x sortiert)
- "Linke" Kanten werden in den Sweep Line Status eingefügt
- "Rechte" Kanten werden aus dem Sweep Line Status entfernt
- Für einen Punkt p des Event Point Schedule müssen alle Kanten im Sweep Line Status bestimmt werden, deren y-Intervall die y-Koordinate von p enthält



6.5 Segmentbaum (I)

Definition

- Datenstruktur zur effizienten Implementierung des Sweep Line Status beim Punkteinschlußproblem
- Sei $y_0,\ldots y_N$ eine Menge von Punkten im 1-dimensionalen Datenraum mit $y_{i-1} < y_i$ für $1 \leq i \leq N$. Der *Segmentbaum* für diese Menge ist ein binärer Baum mit minimaler Höhe, dessen Knoten folgende Segmente zugeordnet sind:
  - jedem Blattknoten ist ein Segment $[y_{i-1},y_i]$ zugeordnet
  - jedem inneren Knoten ist die Vereinigung der Segmente seines Teilbaums zugeordnet

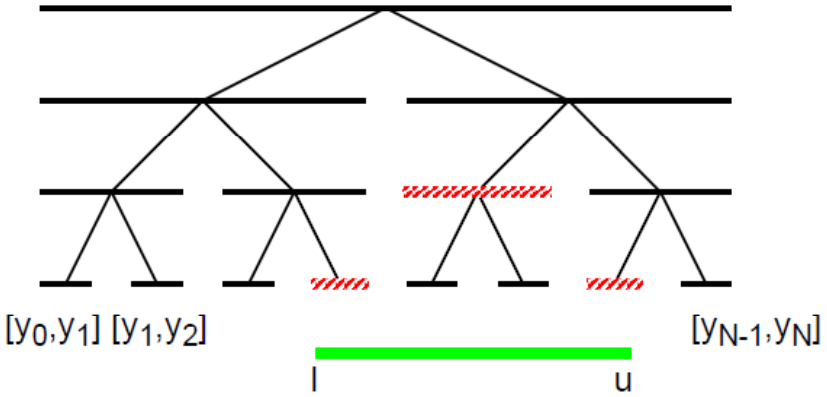


- Jeder Knoten besitzt zusätzlich eine Liste für abzuspeichernde Intervalle

6.5 Segmentbaum (II)

Einfügen eines Intervalls

- Ein Intervall i = [l,u] wird in der Liste eines Knotens abgelegt, falls i das zugehörige Segment des Knotens überdeckt, jedoch das Segment des Vaterknotens **nicht** überdeckt



Punktanfrage

- Finde zu einem gegebenen Punkt $p \in [y_0,y_N]$ alle Intervalle i, die p überdecken.
- Bestimme den Pfad von der Wurzel zu dem Blatt, so daß das zugeordnete Segment des Knotens stets p enthält. Die Antwortmenge der Anfrage ist die Vereinigung der Intervallisten aller Knoten dieses Pfades.

6.5 Segmentbaum (III)

Komplexitätsanalyse

- Aufbau des Segmentbaums mit leeren Intervall-Listen:
  - O(N) Zeit und O(N) Speicherplatz
- Einfügen eines Intervalls in den Baum:
  - Maximal zwei Knoten auf jeder Stufe des Baums sind durch das Einfügen betroffen
  - Einfügen in eine (unsortierte) Intervalliste benötigt O(1) Zeit

    ⇒ O(log N) Zeit
- Löschen aus dem Baum:
  - analog zu Einfügen, d.h. O(log N) Zeit
- Punktanfrage:
  - O(log N + r) Zeit, wenn r die Größe der Antwortmenge ist

6.5 Punkteinschlußproblem (II)

Komplexitätsanalyse

- Bezeichner
  - n_r = Anzahl der Rechtecke, n_p = Anzahl der Punkte, $n = n_r + n_p$
  - k = Anzahl der gefundenen Paare (Punkt,Rechteck)
- Sortierung der Haltepunkte (pro Rechteck 2, pro Punkt 1):
  - O(n log n) Zeit
- Aufbau des leeren Segmentbaums: Berechne die Segmentstruktur aus den y-Koordinaten der Rechtecke, sortiere diese Koordinaten und erzeuge den leeren Segmentbaum:
  - $O(n_r \log n_r) = O(n \log n)$ Zeit
- Einfügen und Löschen im Segmentbaum: $O(n_r \log n_r)$ Zeit
- Punktanfragen: $O(n_p \log n_r + k)$ Zeit
- Gesamtkosten: $O(n \log n + k)$ Zeit und $O(n + n_r \log n_r)$ Speicherplatz

  ⇒ *Nachteil:* hoher Speicherplatzaufwand

6.5 Intervallbaum (I)

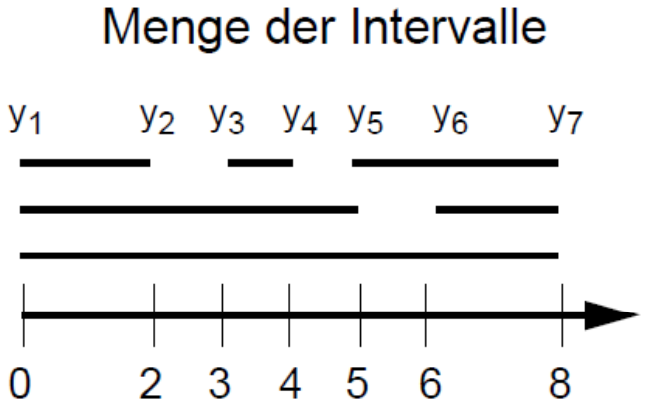
Motivation

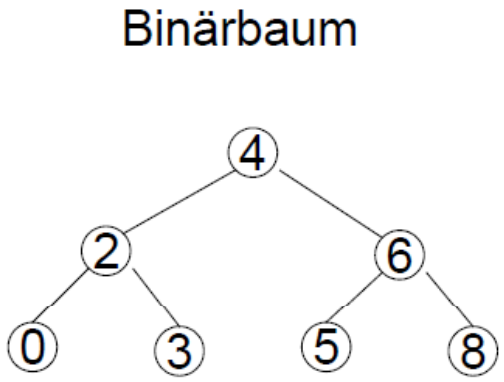
- Verwendung anstelle des Segmentbaums beim Punkteinschlußproblem
- Unterschied zum Segmentbaum: Knoten des Baums repräsentieren nicht ein Segment, sondern einen Punkt
- Vorteil: Speicherplatzbedarf O(n)

6.5 Intervallbaum (I)

Aufbau eines Binärbaums

- Sei $I_1, I_2, \ldots, I_n$ eine Menge von Intervallen mit $I_i = [u_i, o_i]$, $1 \leq i \leq n$.
- Sei $P = \{ y_1, \ldots, y_s \}$ die Menge der Punkte, die Eckpunkt von zumindest einem Intervall sind, $s \leq 2n$. Sortiere P, so daß $y_i < y_{i+1}$ gilt.
- Erzeuge einen binären Suchbaum mit minimaler Höhe für die Punkte $y_1,\ldots,y_s$





6.5 Intervallbaum (II)

Einfügen eines Intervalls

- Jeder Knoten *Node* besitzt einen *Punkt* Node.y; jedem Knoten wird eine *Menge von Intervallen* $I_1,\ldots,I_m$ zugeordnet, die den Punkt Node.y enthalten.
- Es werden zwei sortierte Listen über der Menge $I_1,\ldots,I_m$ geführt:
  - u-Liste: alle Intervalle sind nach den u-Werten aufsteigend sortiert
  - o-Liste: alle Intervalle sind nach den o-Werten absteigend sortiert
- Ein Intervall I wird in die beiden Listen des ersten Knotens K eingetragen, dessen Punkt K.y in I enthalten ist (Durchlauf von der Wurzel beginnend):

6.5 Intervallbaum (II)

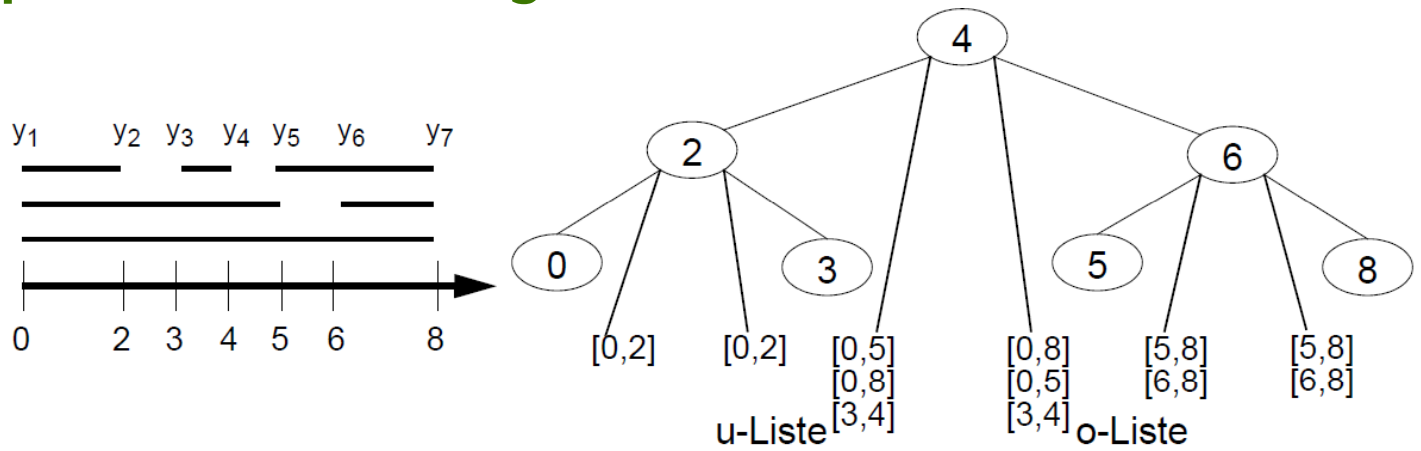
Einfügen eines Intervalls (cont.)

Algorithmus Insert (I, Node) (erster Aufruf mit Node = Wurzel)

```
IF I.u ≤ Node.y ≤ I.o THEN
        InsertIntoList (u_list, I.u);
        InsertIntoList (o_list, I.o);
ELSIF I.o < Node.y THEN
        Insert (I, Node.left)
ELSE
        Insert (I, Node.right)
```

6.5 Intervallbaum (III)

Beispiel für das Einfügen



Komplexitätsanalyse

- Verwende zur Organisation der Listen balancierter Bäume (z.B. 2-3+-Baum) ⇒ Einfügen in die o- und u-Liste benötigt O(log n) Zeit
- Erreichen des Knotens im Intervallbaum: O(log n) Zeit
- Gesamtaufwand für eine Einfügung: O(log n) Zeit
  (bei O(1) Speicherplatz)

  ⇒ Speicherplatzaufwand für n Intervalle: O(n)

6.5 Intervallbaum (IV)

Punktanfrage

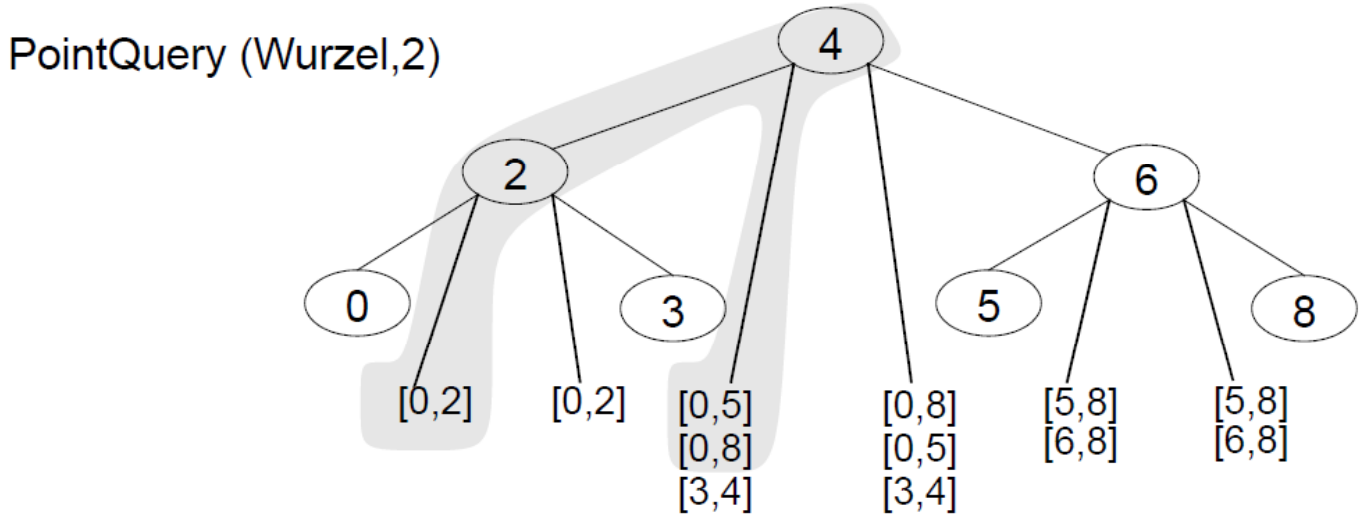
- Gegeben: Anfragepunkt y
- Gesucht: alle Intervalle I, die y enthalten

Algorithmus PointQuery (Node, y)

```
IF y = Node.y THEN
    Gebe alle Intervalle der u-Liste aus;
ELSIF y < Node.y THEN
    Gebe sequentiell den Anfang der u-Liste aus, bis ein u-Wert rechts von y liegt;
    PointQuery (Node.left, y);
ELSE
    Gebe sequentiell den Anfang der o-Liste aus, bis ein o-Wert links von y liegt;
    PointQuery (Node.right, y);
```

6.5 Intervallbaum (V)

Beispiel für die Punktanfrage



Komplexitätsanalyse der Punktanfrage

- Für jeden Knoten des Intervallbaums wird maximal auf ein Intervall zugegriffen, das nicht Antwort ist
- Die Anfrage ist auf einen Pfad des Baums beschränkt

  ⇒ Gesamtaufwand: O(log n + k) (k = Größe der Antwortmenge)