

• Algorithmus:

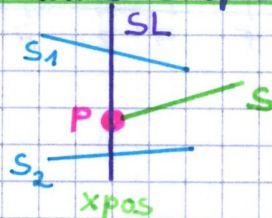
SWEEP(S)

- X-Struktur $X \leftarrow \emptyset$ $O(1)$
- Y-Struktur $Y \leftarrow \emptyset$ $O(1)$
- double $xpos \leftarrow -\infty$ // $xpos$ ist die Position der SL $O(1)$
- for all $s \in S$ do
 - X.insert(s.left)
 - X.insert(s.right)

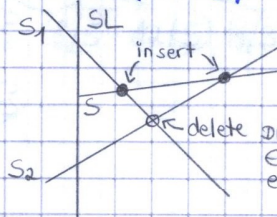


- od
 - while ($X \neq \emptyset$) {
 - $p \leftarrow X.findmin()$
 - $xpos \leftarrow p.xcoord()$ // schiebt SL zum Pkt p
- Fallunterscheidungen gemäß der verschiedenen Events:

• case linker Endpkt von s: { 6

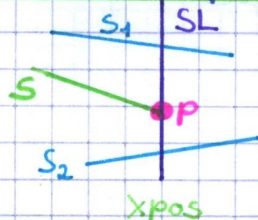


- Y.insert(s)
- $S_1 \leftarrow Y.succ(s)$ S_1 exist nicht immer
- $S_2 \leftarrow Y.pred(s)$ S_2 " "
- X.delete($S_1 \cap S_2$) $S_1 \cap S_2$ " "
- X.insert($S_1 \cap s$) $S_1 \cap s$ " "
- X.insert($s \cap S_2$) $s \cap S_2$ " "



Dieser Schnittpkt wird jetzt gelöscht um Platz zu sparen. Er wird aber später (wenn die SL nahe genug ist) wieder entdeckt.

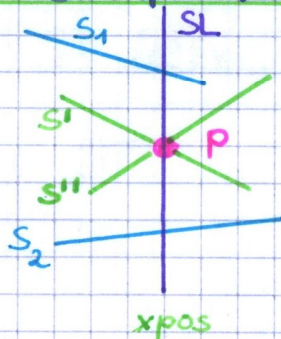
• case rechter Endpkt von s: { 4



- Y.delete(s)
- $S_1 \leftarrow Y.succ(s_2)$
- $S_2 \leftarrow Y.pred(s_1)$
- Y.delete(s)
- X.insert($S_1 \cap S_2$) $S_1 \cap S_2$ liegt recht von SL !!

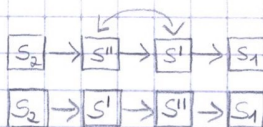


• case schnittpkt von s' und s'': { 8



Geht nicht !!
Weil man in der Y-Strukt. nur Vorgänger & Nachf. kennt. Und wenn s weg ist, weiß man nicht mehr dass es zw. s_1 und s_2 war, dass man diese beiden jetzt neu "verbinden" muss.

$S_1 \leftarrow Y.\text{succ}(s')$
 $S_2 \leftarrow Y.\text{pred}(s'')$
 $Y.\text{swap}(s', s'')$
 $X.\text{delete}(S_1, s')$
 $X.\text{delete}(S_2, s'')$
 $X.\text{insert}(S_1, s'')$
 $X.\text{insert}(S_2, s')$
 Ausgabe: "p = s' n s"



} end switch } X.delete(p)
 end while }

• Annahmen:

- Alle x-Koordinaten von Segment-Anf-, -End- und -Schnittpktn sind paarweise verschieden dh in einem Pkt schneiden sich höchstens 2 Segmente
- ~~A~~ vertikalen Segmente

• Laufzeit:

- Alle Operationen auf X und Y benötigen höchstens Zeit $O(\log n)$ (siehe: Operationen auf X bzw Y-Struktur, 2 Seiten vorher)
 - Die Initialisierung vor der while-Schleife benötigt Zeit $O(n \log n)$ Einfügen von 2n Anfangs- bzw Endpktn
 - Die while-Schleife wird $2n + s$ $s \hat{=}$ # Schnittpkte mal ausgeführt. Jeder Schleifendurchlauf kostet $O(\log n)$ da in jedem Fall konstant viele Operationen auf X und Y und konstant viele Schnitttests ausgeführt werden
- \Rightarrow insgesamt: $O((n+s) \log n) = O(n \cdot \log n + s \cdot \log n)$
 Dies ist wie gewünscht output-sensitiv!

3.2.1 1. Modifikation

orientation-Tests statt "compare" Fkt

- Zur Definition der linearen Ordnung in der Y-Struktur wurde im obigen Algorithmus die "compare"-Fkt verwendet. Anstelle dieser Fkt werden jetzt nur noch orientation-Tests durchgeführt.



Für jedes Segment s in Y werden der letzte Schnittpkt bzw der linke Endpkt als $v(s)$ gespeichert.

1. Fall: orientation($v(s_1), b_1, v(s_2)$) > 0

$\Rightarrow S_1 < S_2$

2. Fall: orientation($v(s_1), b_1, v(s_2)$) < 0

$\Rightarrow S_1 > S_2$

3. Fall: orientation($v(s_1), b_1, v(s_2)$) = 0

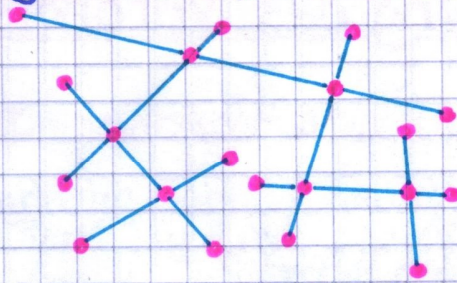
$\Rightarrow v(s_1) = v(s_2)$

$< 0 \Rightarrow S_1 > S_2$
 $= 0 \Rightarrow \text{collinear}$
 $> 0 \Rightarrow S_1 < S_2$

3.2.2 2. Modifikation

Ausgabe des planaren Graphen

- Ausgabe des planaren Graphen:



$$G = (V, E)$$

V : Anf-, End- und Schnittpkte

E : Intervalle durch Zerlegung der Segmente durch V

- Erzeugung von Knoten:

- für jeden Anf- und Endpkt der Segmente
- für jeden Schnittpkt der Segmente

- Erzeugung von Kanten:

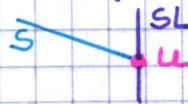
- bei Schnittpkt:



2 neue Kanten entstehen:

$v(s_1) \leftrightarrow u$ und $v(s_2) \leftrightarrow u$ (jeweils 2 fach gerichtet)

- bei Endpkt:



1 neue Kante entsteht:

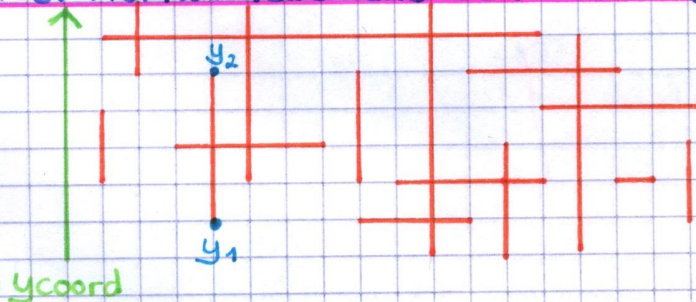
$v(s) \leftrightarrow u$

(2 fach gerichtet)

3.2.3 3. Modifikation

nur horizontale und vertikale Segmente

- Nur horizontale und vertikale Segmente:



$n = 16$ (# Segmente)

$s = 10$ (# Schnittpkte)

$l = 9$ (# vertikale Segmente)

- Events:

- linker Endpkt von Horizontalen
- rechter Endpkt von Horizontalen
- vertikales Segment

} Einfügen und löschen

} Test ob Schnittpkte exist

- 1-dim Bereichsabfrage: hat immer Zeit $O(\log n + s)$ $s =$ Ausgabe
Welche horizontalen Segmente in Y drinstehen

$$\sum_{i=1}^l \log n + k_i \text{ für } l \text{ vertikale Segmente}$$

$$\text{Hier: } \log n + 0 + \log n + 1 + \log n + 1 + \log n + 2 + \log n + 0 + \log n + 3 + \log n + 1 + \log n + 2 + \log n + 0 = 9 \cdot \log n + 10$$

$$\Rightarrow O(n \cdot \log n + s)$$

hier zwar $q = l \neq n$ aber es könnte ja sein, dass nur vertikale Segmente existieren! (dann wäre $l = n$)

- Laufzeit:
 $O(n \cdot \log n + s)$

3.3 Anwendung II: Post Office Probleme

3.3.1 Definition des Voronoi-Diagramms

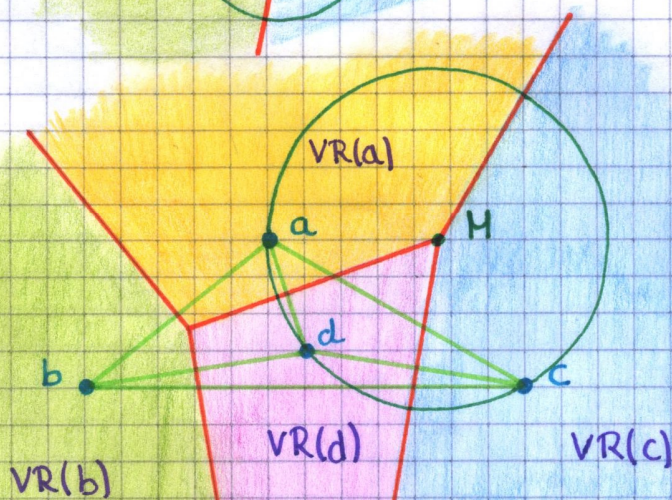
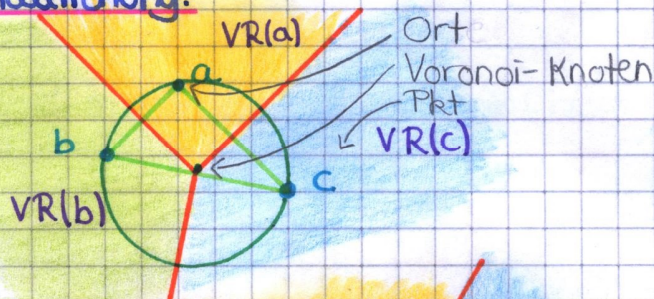
- Das Voronoi-Diagramm ist eine wichtige geometrische Datenstruktur zur Lsg der sog. "Post Office Probleme"
 Geg.: Menge S von n Orten (z.B. Postämter)
 Ges.: Ort $x \in S$, sodass $\text{dist}(p, x)$ minimal für $p \in \mathbb{R}^2$ bel
 ↓
Idee:
 - Berechne für jeden Ort $x \in S$ das Gebiet aller Pkte, deren Abstand zu x kleiner (gleich) ist, als zu allen anderen Orten $\in S$.
 → planare Unterteilung: Voronoi-Diagramm von S
 - Teste für Eingabepkt $p \in \mathbb{R}^2$ in welchem Gebiet des VD(S) er liegt.
 → Teilproblem: Point Location

- Definition:
 Voronoi-Region für Ort $x \in S$:

$$VR(x) := \{ p \in \mathbb{R}^2 : \text{dist}(x, p) \leq \text{dist}(y, p) \text{ für } x \neq y \}_{y \in S, y \text{ Ort}}$$

$$= \bigcap_{y \in S \setminus \{x\}} H(x, y)$$
 ⇒ $VR(x)$ ist konvexes Polygon als Schnitt von $n-1$ Halbebenen

Veranschaulichung:



$M =$ Mittelpunkt des Umkreises durch a, d, c
 = Schnittpkt der Mittelsenkrechten

dualer Graph = hellgrün
 (Kreiseigenschaft muss dabei erfüllt sein !!)

planarer Graph = orange
 aber auch = hellgrün