EMSB: Nach dem Lemma kann man sich nur auf Delaunay-Kanten beschränken, davon gibt es linear viele, weiter Kruskal $\rightsquigarrow$ insgesamt $O(n \log n)$ Zeit.

Das nächste Problem, das schon in der 1.Vorlesung angesprochen wurde:

Nächstes-Paar-Problem

(NP, aber wir werden es selbstverständlich als **PN** bezeichnen).

Gegeben: $S \subset \mathbb{R}^2$, $|S| = n$
Finde: ein Paar $(x, y) \in S^2$ mit $\|x - y\|$ minimal
Lösung: in der 1.Vorlesung haben wir einen Algorithmus mit der Laufzeit $\Theta(n^2)$ gefunden.
Verallgemeinerung:

Alle-Nächsten-Nachbarn (ANN)

Finde: für jedes $x \in S$ das nächste $y \in S$, $y \neq x$.

Satz: Sei $S \subset \mathbb{R}^2$, $|S| = n$. Dann ANN (und damit PN) lassen sich in Zeit $O(n)$ lösen, falls $VD(S)$ (oder Delaunay-Triangulierung) gegeben ist, d.h. insgesamt (falls nicht gegeben) in $O(n \log n)$ Zeit.

Beweis: Das ist eine einfache Folgerung aus dem Lemma. Für alle x bestimme $\|x - y\|$ für alle y, deren $VR(y)$ an die $VR(x)$ grenzt. Nach dem Lemma: minimale davon ist min insgesamt. Laufzeit: Anzahl der Tests und Berechnungen von $\|x - y\|$ ist gleich 2 mal Anzahl der Voronoi-Kanten, also $O(n)$. □

Bemerkung: PN (und damit ANN) haben eine untere Schranke in der Laufzeit von $\Omega(n \log n)$. Modell: reelle Zahlen, Operationen $\{\leq, +, -, \cdot, \div\}$; Reduktion von *"element uniqueness"* (d.h. gegeben n Zahlen, Frage: Sind alle verschieden?), bekannte untere Schranke $\Omega(n \log n)$.

Kapitel 3. Sweepline - Verfahren

Für *sweepline* gibt es kein gebräuchliches deutsches Wort, man sagt manchmal *scan-line* oder auch *Fegegerade* (G.Rote).

Anwendung:
Gegeben: Menge S von Strecken in der Ebene
Finde: Alle Schnittpunkte von Strecken in S
Frage: In welcher Zeit kann man das machen?

Mit *brute-force*-Ansatz ist Laufzeit $\binom{n}{2} = \Theta(n^2)$. Es kann auch $\binom{n}{2}$ Schnittpunkte geben, diese Zeit ist also **nicht** zu verbessern!
Andere Fragestellung: wenn es "wenig" Schnittpunkte gibt, $k \in \mathbb{N}$, geht es dann besser? Laufzeitanalyse in Abhängigkeit von n und k (öfter $k \ll n^2$). Solche Algorithmen werden *output-sensitiv* genannt.

Verallgemeinerung: Berechnung des *Arrangements* von S, d.h. des geometrischen Graphen (Facetten, Kanten, Ecken), der durch die Einbettung von S entsteht (zusätzliche Ecken sind Schnittpunkte).

Idee von sweepline-Verfahren: Streiche mit vertikaler Geraden (*sweepline*, SL) von links nach rechts über die Szene. Aufrechterhalten wird die Information: Welche Strecken von S schneiden SL in welcher Reihenfolge. Wir konstruieren entsprechende Datenstruktur: SLS, *sweepline-status*. Zunächst ("ganz" links) ist SLS leer; beim ersten Endpunkt einer Strecke wird diese in SLS eingefügt. SLS ändert sich an den *Endpunkten* (Einfügung, Streichung) und an den *Schnittpunkten* (Reihenfolge von 2 Strecken ändert sich). Sie heißen *Ereignispunkte* (*event-points*): x-Koordinaten dieser Punkte werden in einer zusätzlichen Datenstruktur EPS (*event-point-schedule*) gespeichert.

Datenstruktur für SLS: Wörterbuch $\rightsquigarrow$ balancierte Suchbäume: Operationen in $O(\log n)$ Zeit (einfügen, streichen, vertauschen als Kombination aus 2 mal streichen und 2 mal

einfügen).

Datenstruktur für EPS: PrioritätsWS $\rightsquigarrow$ Heap: Operationen in $O(\log n)$ Zeit.

Insgesamt: $O((n+k)\log n)$ Zeit.

Wir schauen nun genauer hin:

Zur Vereinfachung: Angenommen, S in *allgemeiner* Lage: x-Koordinaten der Endpunkte sind paarweise verschieden, es gibt keine mehrfachen Schnittpunkte, Koordinaten von Schnittpunkten stimmen nicht mit denen von Endpunkten überein usw.

Algorithmus

$SLS := \{-\infty, \infty\}$;

$EPS :=$ Menge aller Punkte von Strecken in S;

while $EPS \neq 0$ **do**

$p :=$ deleteMin(EPS); /* Min wird ausgegeben und gestrichen

if p rechter Endpunkt einer Strecke $s \in S$ **then**

bestimme obere und untere Nachbarn von s in SLS: $s_1,\ s_2$;

streiche s aus SLS;

teste, ob Schnittpunkt $s_1 \bigcap s_2$ existiert und füge ihn ggf. in EPS ein;

if p linker Endpunkt einer Strecke $s \in S$ **then**

füge s in SLS ein;

bestimme obere und untere Nachbarn von s in SLS: $s_1,\ s_2$;

teste, ob $s \bigcap s_1$, $s \bigcap s_2$ existieren und füge sie ggf. in EPS ein;

if p Schnittpunkt von $s_1, s_2 \in S$ **then**

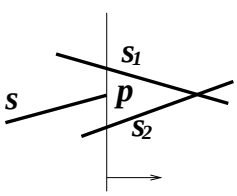
gib p aus;

vertausche s_1, s_2 in SLS ; /*streichen und umgekehrt wieder einfügen

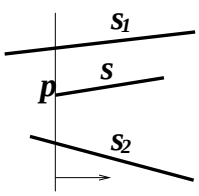
bestimme die Nachbarn: s_3 (oberhalb von s_1 bisher) und

s_4 (unterhalb von s_2 bisher);

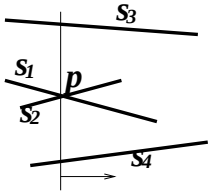
teste, ob $s_1 \bigcap s_4$ oder $s_2 \bigcap s_3$ existieren und füge sie ggf. in EPS ein;



p rechter Endpunkt



p linker Endpunkt



p Schnittpunkt

Analyse: insgesamt $2n + k$ Ereignispunkte, für jeden $O(\log n)$ Zeit für Operationen $\rightsquigarrow$ $O((n + k) \log n)$ Zeit (es geht sogar in $O(n \log n + k)$ Zeit).

Satz: Die Menge aller Schnittpunkte einer Menge von n Strecken kann in $O((n + k) \log n)$ Zeit gefunden werden, wobei k= Anzahl der Schnittpunkte.

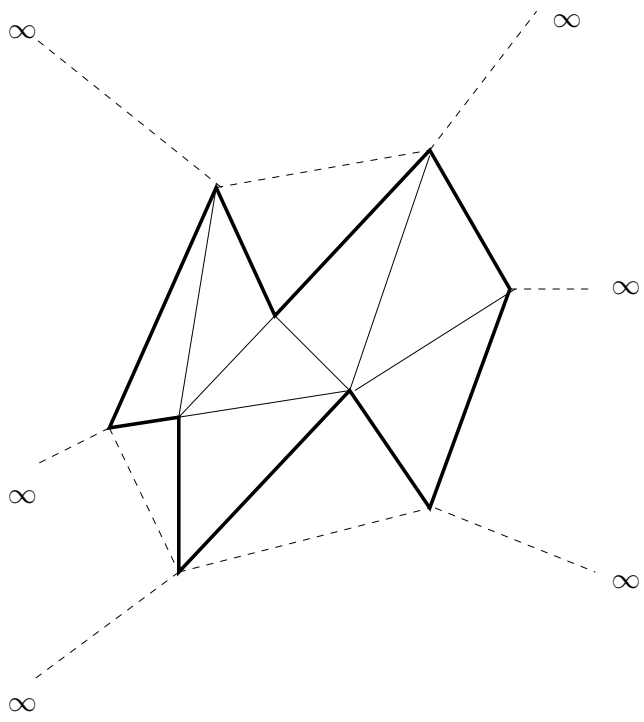
(gilt auch für nicht allgemeine Lage $\rightarrow$ Übung).

Triangulierung eines einfachen Polygons

Def: *Triangulierung* einer Menge S von Punkten heißt ein triangulierter geometrischer Graph $G = (V, E)$ mit $V = S$.

Triangulierung eines einfachen Polygons P:

Suchen Triangulierung der Eckenmenge $\bigcup \{\infty\}$, die alle Kanten von P enthält.



Die äußere Facette muss auch Dreieck sein $\rightsquigarrow$ (- - -) Kanten. *Innere* Triangulierung: Teil der Triangulierung, der im Inneren von P liegt.