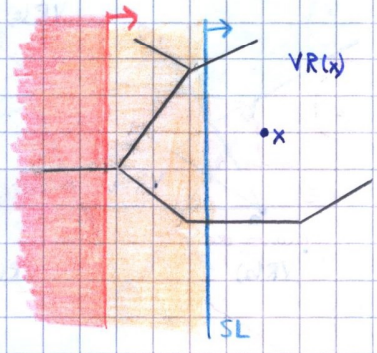
3.4.3. Konstruktion von Voronoi-Diagrammen durch einen "Plane Sweep Algorithmus".

Zunächst: allg. Lage der Orte
- Keine 4 Orte liegen auf einem gemeinsamen Kreis
- paarweise verschiedene x-Koordinaten aller Orte und Events.

→ 3.4.3.1. Ziel: Bewege Senkrechte SL von links nach rechts über die Menge der Orte und gib Knoten und Kanten des VD der bereits besuchten Orte aus.

→ 3.4.3.2 Problem: Die Voronoi-Region VR(x) beginnt links von x, dh. bevor SL x erreicht.



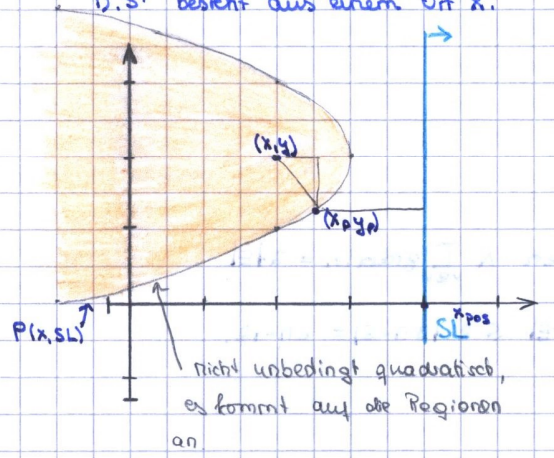
→ 3.4.3.3. Idee:
Definieren ein Gebiet weiter links von SL, in dem das Voronoi-Diagramm schon bekannt ist, dh. nicht abhängig von x.

→ 3.4.3.4. Beobachtung:
Voronoi-Regionen von Orten rechts von SL enthalten keine Pkte, die näher zu schon besuchten Orten liegen als zu SL
Denn sonst würde ein solcher Pkt (unabh. von SL) näher zu einem anderen Ort liegen.
Abstand zu SL ist nicht größer als Abstand zum nächsten Ort.

→ 3.4.3.5 Frage:
Betrachte das Gebiet $L := \{p \in \mathbb{R}^2 : dist(p,x) \leq dist(p,SL)$ für einen schon besuchten Ort $x\}$
⇒ Voronoi-Diagramm von S im Gebiet L hängt nicht von Orten rechts von SL ab.
Welche Form hat L?

→ 3.4.3.6. Beispiele: S' ← Menge der besuchten Orte.

1). S' besteht aus einem Ort x.

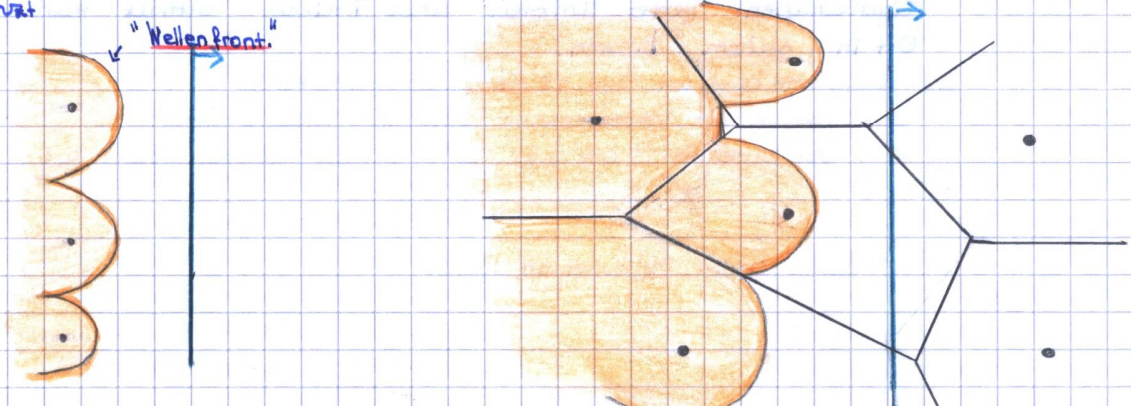


nicht unbedingt quadratisch, es kommt auf die Regionen an

(x,y) und SL bekannt
Wie sieht L aus?
→ Es muß gelten $\forall (x_p y_p) \in L$:
$(x_p - x_{pos})^2 \overset{!}{=} (x_p - x)^2 + (y_p - y)^2$
weil es muß eigentlich gelten:
$x_p - x_{pos} = \sqrt{(x_p - x)^2 + (y_p - y)^2}$.
→ (x,y) = (2,2), SL ≡ 4 bekannt
$\Rightarrow (x_p - 4)^2 = (x_p - 2)^2 + (y_p - 2)^2$
$\Leftrightarrow x_p^2 - 8x_p + 16 = x_p^2 - 4x_p + 4 + y_p^2 - 4y_p + 4$
$\Leftrightarrow -4x_p + 8 = y_p^2 - 4y_p$
$\Leftrightarrow 4x_p = -y_p^2 + 4y_p + 8$
$\Leftrightarrow x_p = -\frac{1}{4}y_p^2 + y_p + 2$
→ dies ist eine Parabelgleichung mit der Variablen y

→ Inneres der Parabel P(x,SL) ist die Menge der Pkte mit gleicher Distanz zu x und SL

2). Im Allgemeinen wird L durch eine Folge von Parabelbögen nach rechts begrenzt



⟶ 3.4.3.7 Beobachtung: Schnittpkte von benachbarten Parabeln in der Wellenfront überstreichen die Kanten von VD(S).

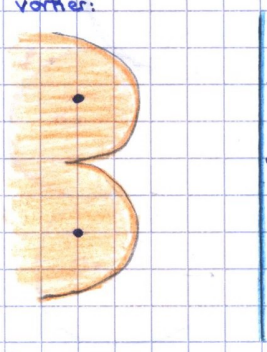
⟶ 3.4.3.8 Idee: Verwalte die Wellenfront der Parabelbögen in einer Y-Struktur.

⟶ 3.4.3.9 Fragen: An welchen Positionen (Transaktionspkten / Events) der SL ändert sich die Y-Struktur strukturell?
→ Welche Events gibt es?

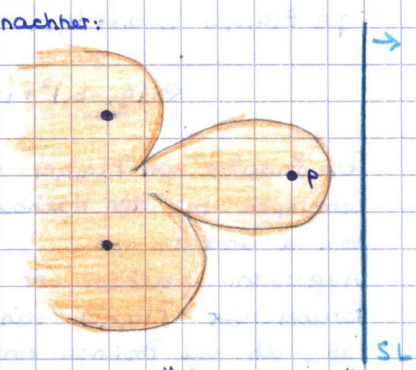
⟶ 3.4.3.10 Zwei Arten von Events:

1). SL überstreicht neuen Ort p "Site Event"

vorher:

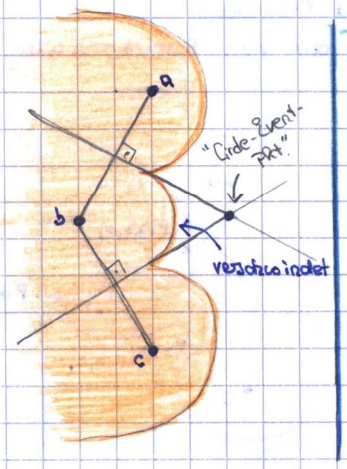


nachher:



2). Ein Parabelbogen verschwindet aus der Wellenfront. "Circle Event"

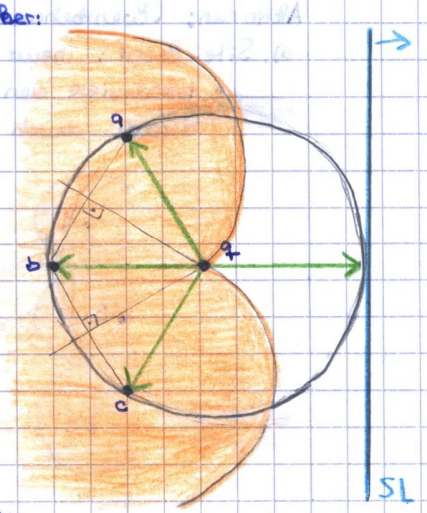
vorher:



? drei Parabeln schneiden sich in einem Pkt

bea: q gehört nicht zu S(1,a) Es ist nur ein Schnpkt von VR'en und zwar Circle-Eventpkt.

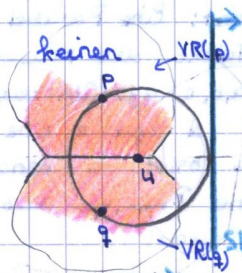
nachher:



↑ Hier entsteht ein neuer Voronoi-Knoten q = Mittelpkt des Kreises durch a,b,c
SL ist zu diesem Zeitpkt Tangente an den Kreis (durch a,b,c).

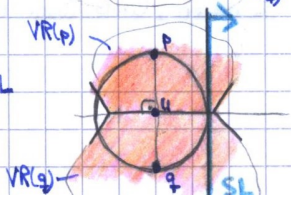
⟶ 3.4.3.11 Lemma: Jeder Pkt auf der Kante von VD(S) kommt während des Sweep als Schnittpkt zweier in der Y-Struktur benachbarter Parabelbögen vor.
(Bei kontinuierlicher Sweep)

Beweis:
Sei e beliebige Voronoi-Kante von VD(S) und u ein beliebiger Pkt auf e. e trennt zwei Voronoi-Regionen VR(p) und VR(q)
u liegt näher zu p und q (gleiche Distanz d) als zu allen anderen Orten.
Kreismittelpkt u enthält im Inneren keinen Ort.
Nun betrachte den Zeitpkt des Sweep, in dem SL rechte Tangente an diesem Kreis K ist



a) u hat den gleichen Abstand zu p und SL
⇒ u liegt auf Parabel von p

b) u hat gl. Abstand zu q und SL
⇒ u liegt auf Parabel von q



⇒ Beh.

3.4.3.12. Implementierungsdetails:

1). Darstellung der Parabelbögen:

Ein Parabelstück wird durch drei Orte a,b,c und die aktuelle Position x_{pos} der Sweepline definiert.

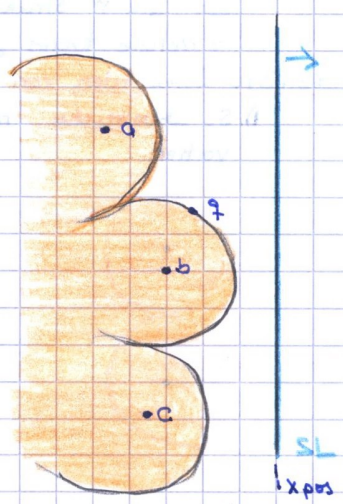
$P(a,b,c)$ bezeichnet Parabelstück von b, das an Parabeln von Ort a und von Ort c grenzt.

Gleichung der Parabel:

$q \in P(a,b,c) \Leftrightarrow dist(q,b) = dist(q,SL)$

auf Parabelstück, gemeint auf dem Rand $\Rightarrow$ "$=$"

$\Leftrightarrow (q_x - b_x)^2 + (q_y - b_y)^2 = (q_x - x_{pos})^2$



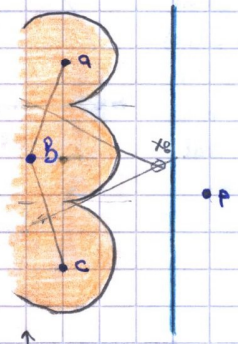
Die Endpkte von $P(a,b,c)$ ergeben sich als Schnittpkte mit den Nachbarparabeln, dh. von a und c.

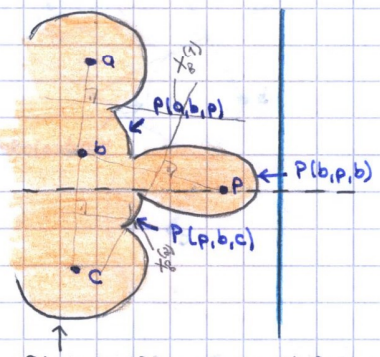
Außerdem speichern wir mit jedem Parabelstück $P(a,b,c)$ sein Circle-Event ab, dh. die Position von SL, bei der $P(a,b,c)$ aus Wellenfront verschwindet.

Aktionen: (Eventbehandlung).

a). Site-Event: neuer Ort p erreicht. i.d. Y-Struktur

- lokalisiere den Parabelbogen $P(a,b,c)$, der von horizontaler Geraden durch p geschnitten wird.





Y-Str.: In der Y-Str. $= \{P(-,a,b), P(a,b,c), P(b,c,+)\}$

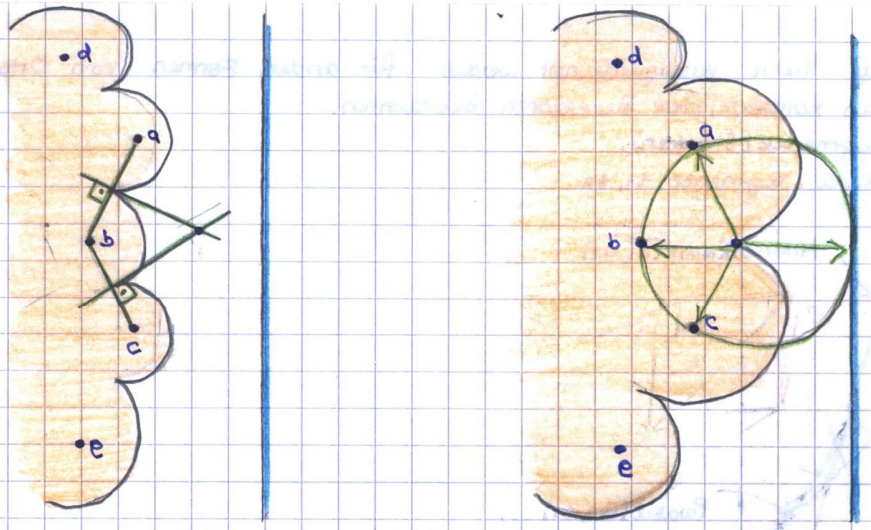
$P(a,b,c)$ löschen aus Y-Str. und drei neue Bögen in die Y-Str. einfügen

$\Rightarrow$ Y-Str. $= \{P(-,a,b), P(a,b,p), P(b,p,b), P(p,b,c), P(b,c,+)\}$

- lösche $P(a,b,c)$ aus Y-Str.
- lösche Circle-Event von $P(a,b,c)$ aus X-Str. (PRkt)
- füge 3 neue Bögen in die Y-Str. ein
- berechne die Circle-Events der neuen Parabelstücke (PRkt) und füge sie in die X-Str. ein.

b). Circle-Event: Eine Parabel $P(a,b,c)$ verschwindet.

- Gib Mittelpkte des Kreises durch a,b,c als Voronoi-Knoten aus mit a,b,c gg. Uhrzeigersinn. (berechne zugleich Mittelsenkrechte von ab und bc und gebe sie aus)
- Seien d und e Orte mit Parabelbögen, die mit Parabel von a bzw. c benachbart sind.
- Entferne $P(a,b,c)$ aus Y-Struktur. und lösche den bearbeiteten Circle-Event-Pkt aus der X-Struktur
- berechne die neuen Circle-Event-Pkte



→ $P(d,a,b) \rightsquigarrow P(d,a,c)$ und $P(b,c,e) \rightsquigarrow P(a,c,e)$.

$X = \{C(d,a,b), C(a,b,c), C(b,c,e)\}$ $X = \{C(d,a,c), C(a,c,e)\}$.

2) Initialisierung

3). Geometrische Primitive

a) Schnitt von Parabeln

b) Vergleichsoptt für Ordnung der Parabeln in Y-Struktur für beliebigen Suchbaum

Y. locate (x)

Y. insert (P(a,b,c))

Y. delete (...)

4) Konstruktion des Voronoi-Diagramms aus Folge der Circle-Events.

→ 2), 3), 4) das alles in Übungsaufgaben.

⊢→ 3.4.3.13. Ausgabe:

Für jedes Circle-Event geben wir einen Mittelpkt als Voronoi-Knoten v aus und die Orte auf dem Kreis a,b,c gegen Uhrzeigersinn.
(Können mehr als drei sein bei degenerierten Eingaben)
d.h. wir geben eigentlich die Dreiecke der Delaunay-Triangulierung aus (dualer Graph zum Voronoi-Diagramm).

⊢→ 3.4.3.14. Übung:

1). Berechne aus dieser Ausgabe eine explizite Darstellung des Voronoi-Diagramms (dh. Voronoi-Kanten)

2) Modifiziere sie den Alg. so, dass die Voronoi-Kanten direkt ausgegeben werden.

3) Degenerierte Eingaben

Circle-Event: mehr als drei Orte auf einem Kreis ⇒ mehr als ein Bogen verschwindet.

4). Alle geometrischen Primitiven:

- Schnitt von Parabeln
- Vgl. von Parabeln in Y-Struktur.

→ können alle in O(1) Berechnet werden.

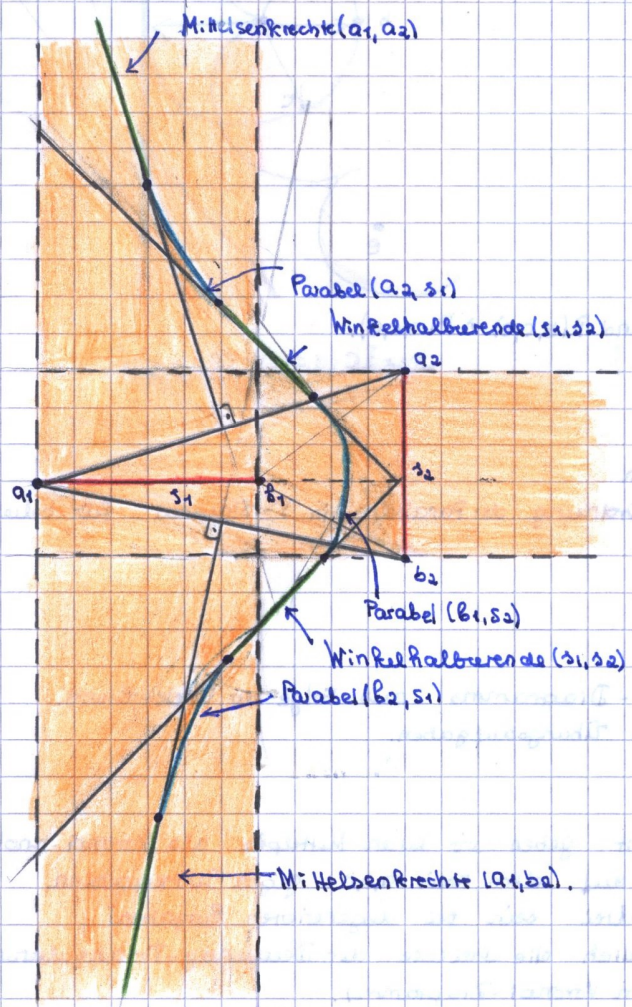
⊢→ 3.4.3.15 Satz: Das Voronoi-Diagramm von n punktförmigen Orten kann durch plane-Sweep in Zeit O(n log n) berechnet werden.

Bew: Alle Operationen auf X- bzw. Y- Str. können in Zeit O(log n) ausgeführt werden, wenn sie durch binäre Suchbäume realisiert werden.
Außerdem hat VD lineare Größe in Zahl der Orte.

⟶ 3.4.3.16 Bem: Diese Alg. kann verallgemeinert werden für andere Formen von Orten. Dazu muß man kompliziertere Bisektoren betrachten.

⟶ 3.4.3.17 Bsp: Kurvensegmente (Strecken).

Bisektor für zwei Segmente s_1, s_2:



Bisektor von zwei Strecken ist eine Kurve, die sich aus Parabelstücken, Mittelsenkrechten und Winkelhalbierenden zusammensetzt.
Je nach Lage und Länge der Strecken können Kurvenarten fehlen, z.B. Winkelhalbierende oder so nicht vorhanden.

⟶ 3.4.3.18: Bemerkung:
Ann. Segmente schneiden sich nicht (sonst Zerlegung an Schnittpkten)
Prinzipiell funktioniert der Sweep-Alg. zur Berechnung des VD von Segmenten genauso wie für Pkte.
Lediglich geometrische Primitiven sind komplizierter: Schnitt & Ordnung von Bisektoren.

3.4.4 Planar point location:

⟶ 3.4.4.1 Aufgabe: Finde für beliebigen Pkt $p \in \mathbb{R}^2$ die Voronoi-Region, die p enthält. Dann ist der Ort dieser Region, der p am nächsten liegende Ort.

⟶ 3.4.4.2 Idee der Vorverarbeitung:
Verbrauche $O(n \log n)$ für Konstruktion von VD, um danach (viele) Anfragen effizient beantworten zu können.

⟶ 3.4.4.3 Ziel: Laufzeit $O(\log n)$ pro Frage.

Frage: ab wieviel Abfragen lohnt es sich VD zu konstruieren?
Abfragen: $= M < n \Rightarrow$ Konstruktion v. VD + anschließend Streifenmethode kostet für M Abfragen:
$O(n \log n) + M \cdot O(\log n) = O(n \log n) + O(M \cdot \log n) = O(n \log n)$ } ⇒ Falls $M < n$, besser kein VD
Suche nach Region durch Lineare Suche (ohne VD) kostet für M Abfragen: $O(M \cdot n) = O(n)$ } aufbauen!
Für $M > n \Rightarrow$ 1 Mögl. kostet: $O(n \log n) + O(M \log n) = O(M \log n)$ } ⇒ Besser VD aufbauen! [weil $\log n < M$].
2 Mögl. kostet: $O(Mn) = O(M^2)$

3.4.5. Point-Location allgemein (unabh. v. Voronoi-Diagramm).

→ 3.4.5.1. Problem:

Geg: Beliebige planare Unterteilung der Ebene (Subdivision)

Ges: point location.

dh. finde für beliebigen Pkt p das Gebiet in dem p liegt.

⇔ finde die Kante der Unterteilung, die von einem vertikalen Strahl von p aus nach unten zuerst getroffen wird.

dh. die die man "sieht", wenn man von p nach unten schaut.

(→ vertikales Sichtbarkeitsproblem).

Schreibe den Namen (Nummer, ID, ...) jedes Gebiets an seine untere Folge von Kanten.

⇒ wenn untere Kante ausgeg. wird, so weiß man in welchem Gebiet p liegt.

⇒ daher sind die Aussagen äquivalent

→ 3.4.5.2. Streifenmethode: allgemein:

die Streifenmethode zerlegt dieses 2-dim. Suchproblem in 2 1-dimensionale.

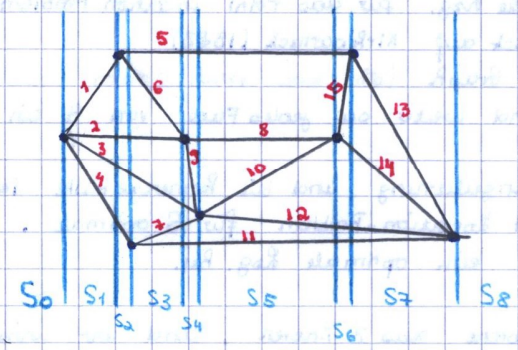
→ 3.4.5.2 Streifenmethode: Idee:

Zerlege die Ebene in vertikale Streifen $S_0, S_1, .., S_n$ durch n vertikale Geraden jeweils durch einen Knoten von G.

G = planare Unterteilung

n = # Knoten von G.

Preprocessing: $\Theta(n^2)$ im schlechtesten Fall.



Speichere die Streifen in einem Feld S[0..n].

Schritt 1: Finde den Streifen S[i], der p enthält durch binäre Suche nach p.xcoord() in S

→ kostet Zeit O(log n).

Darstellung jedes Streifens S[i]:

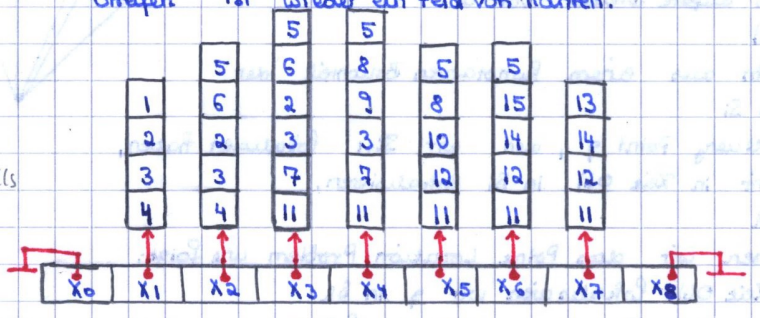
Eine Folge der Kanten von G, die S_i überqueren von unten nach oben (gemäß ihrer Lage in S_i) sortiert.

Bem: Kanten schneiden sich nicht innerhalb eines Streifens, dh. jeder Streifen ist wieder ein Feld von Kanten.

Ein Feld von 2-dim Pkten bzw. Intervallen $x_0 := [-\infty, x_0], [x_0, x_1], [x_1, x_2] \dots, [x_8, \infty]$

Zweites Feld von Zeigern:

Zeiger zeigen auf Felder von ebenfalls 2-dim Pkten.

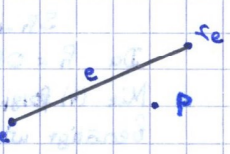


Schritt 2: Binärsuche auf Streifen S[i] selbst.

genauer: sortiere p in Folge der Kanten ein.

Liegt oberhalb, auf oder unterhalb einer Kante e

↝ orientation $(l_e, r_e, p) = ?$



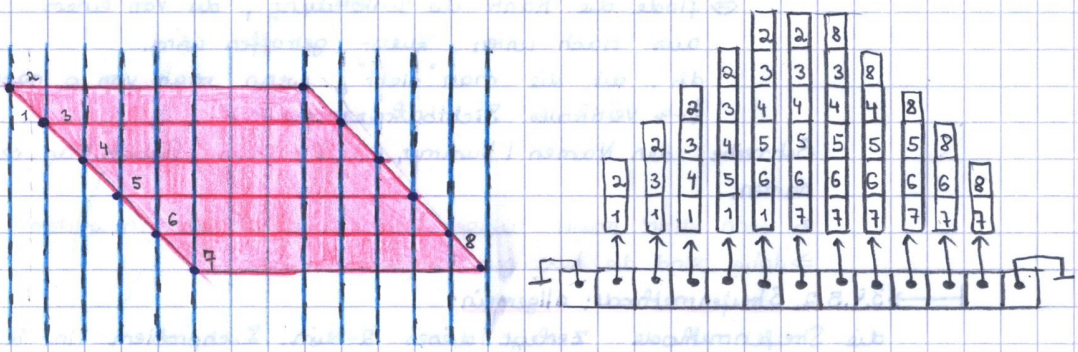
Da insgesamt O(n) Kanten } ⇒ kostet auch Zeit O(log n)
(G ist planarer Graph)

Zu Laufzeit: binäre Suche ist $T(n) = T(\frac{n}{2}) + c_1$ ⇒ $f(n) = c_1$, $\log_8 a = \log_2 1 = 0$

⇒ $f(n) = \Theta(n^{\log_8 a})$ ⇒ $T(n) = \Theta(n^{\log_8 a} \cdot \log n) = \underline{\Theta(\log n)}$

Streifenmethode: ergebnis:

Laufzeit: $O(\log n)$ (optimal)
Platzbedarf: $O(n^2)$ (unpraktisch für großes n) $O(n)+O(n^2)=O(n^2)$
Ziel: $O(\log n)$ Suchzeit und $O(n)$ Platz.
Beispiel für quadratischen Platz:



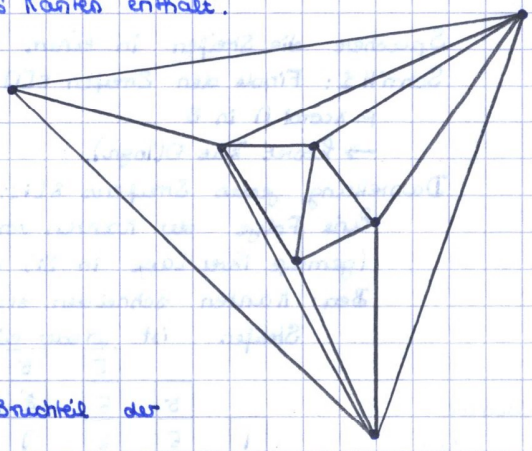
Gesamt: n Knoten (hier n=12)
Jede der $n/2$ Kanten ist in $n/2$ Streifen gespeichert.

3.4.5.4 Triangulierungsmethode: allgemein:

- Ber. eine optimale Lsg. für das Point Location Problem.
- Methode geht zurück auf Kirkpatrick (1983)

Sei G ein planarer Graph (dh. zwei versch. Kanten überschneiden sich nicht u. ungerichtet) auf n Knoten und weiter sei jedes Face von G ein Dreieck (auch das äußere Face)
$\Rightarrow$ G ist eine Triangulierung und die konvexe Hülle ist auch Dreieck. $|CH|=3$.
Wir lösen das Point Location-Problem für G optimal. Später leiten wir für jede planare Unterteilung eine optimale Lsg. her.

Die konv. Hülle besteht aus 3 Knoten. Und wir wissen weiter, dass jede Triangulierung dieser n Knoten $3n-6$ Kanten enthält.
$(3n-k-3,\ k=|CH|)$.



3.4.5.5 Triangulierungsmethode: Idee für Algorithmus:

Konstruiere eine Folge $S_1 .. S_R$ von Triangulierungen, so dass gilt:
(1) $S_1 = G$
(2) S_R ist das äußere Dreieck von G
(3) $R = O(\log n)$
(4) S_{i+1} besteht aus einem konstanten Bruchteil der Knoten von S_i
(5) Für jeden Query Point q, den wir S_{i+1} lokalisiert haben, können wir in Zeit $O(1)$ in S_i lokalisieren.

Geg: $S_1 .. S_R$
dann lösen wir das Point Location Problem wie folgt:

- in Zeit $O(1)$ lokalisieren wir q in S_R
- dann unter Benutzung von (5) lokalisieren wir nacheinander q in $S_{R-1}, S_{R-2}, .., S_1 = G$.

Da $R = O(\log n)$ haben wir eine Suchzeit von $O(\log n)$
Mit (4) folgt, dass für jede Folge der Triangulierung nur Platz $O(n)$ benötigt wird.