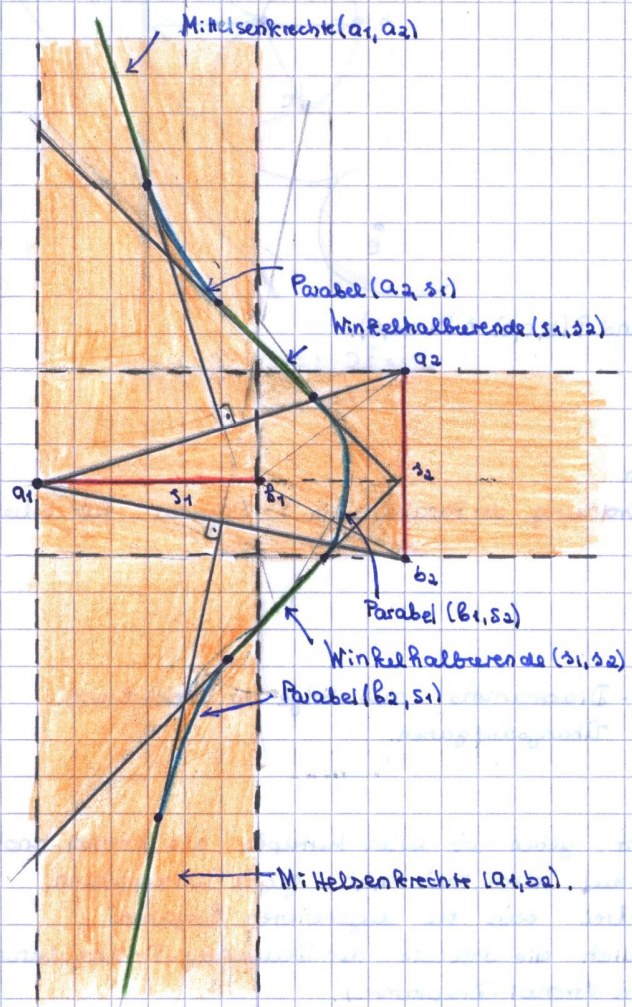
⟶ 3.4.3.16 Bem: Diese Alg. kann verallgemeinert werden für andere Formen von Orten. Dazu muß man kompliziertere Bisektoren betrachten.

⟶ 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.

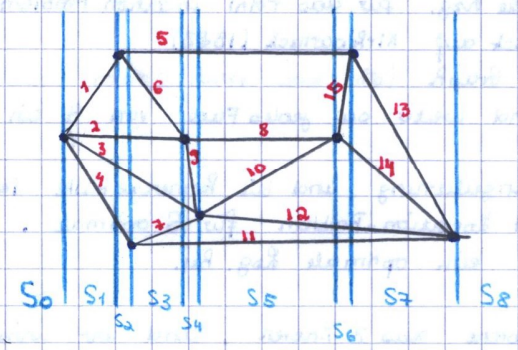
→ 3.4.5.2 Streifenmethode: Idee:

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]:

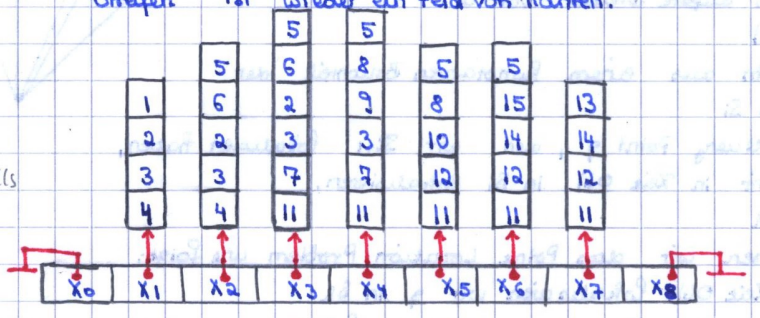
Eine Folge der Kanten von G, die S_i überqueren von unten nach oben (gemäß ihrer Lage in S_i) sortiert.

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.

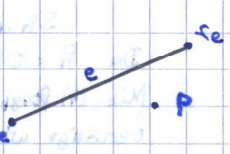


Schritt 2: Binärsuche auf Streifen S[i] selbst.

genauer: sortiere p in Folge der Kanten ein.

Liegt oberhalb, auf oder unterhalb einer Kante e

↝ orientation $(l_e, r_e, p) = ?$



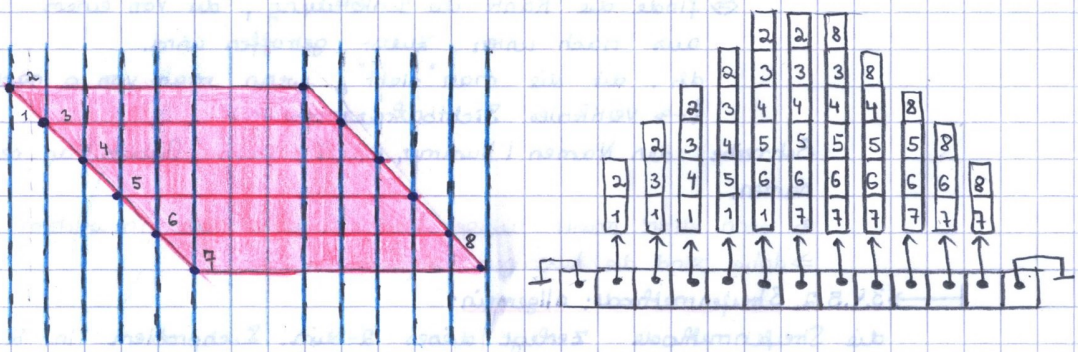
Da insgesamt O(n) Kanten } ⇒ kostet auch Zeit O(log n)
(G ist planarer Graph)

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:



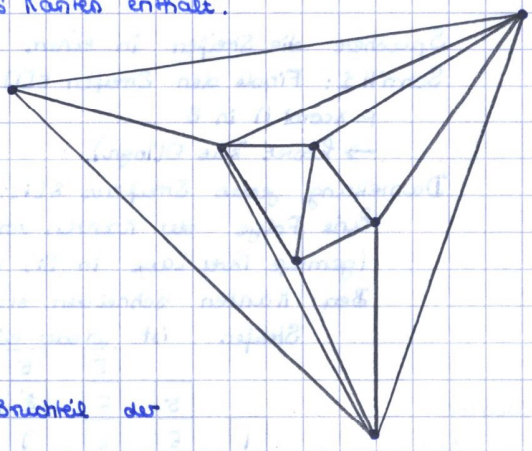
Gesamt: n Knoten (hier n=12)
Jede der $n/2$ Kanten ist in $n/2$ Streifen gespeichert.

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.

⟶ 3.4.5.6 Triangulierungsmethode: Fragen & Antworten.

1) Frage: Wie konstruieren wir die Folge $S_1 .. S_R$?
   → Ber. $S_1 = G_1$.
   Wollen einen konst. Bruchteil der Knoten entfernen.
   Sei v ein non-boundary Knoten von G_1 und sei d sein Grad, dh. in G_1 sind d Kanten inzident zu v.
2) Frage: Was passiert, wenn wir v aus G_1 entfernen?
   → wenn wir v entfernen, entfernen wir auch die d inzidenten Kanten.
   ⇒ wenn wir v entfernen, dann auch die d Dreiecke. Die d Dreiecke werden durch ein einfaches Polygon ersetzt.

Sei q ein beliebiger Pkt im d-gon, das wir durch Entfernen von v erhalten. Dann können wir in Zeit $O(d)$ bestimmen, welches der d Dreiecke von G_1 q enthält.

Um Eig. (5) zu erhalten, sollten wir Knoten vom kleinen Grad entfernen. Für die Eig. (4) sollten wir möglichst viele dieser Knoten (mit kleinem Grad) entfernen.

3) Frage: Ist es immer möglich viele Knoten mit kleinem Grad zu bestimmen?

⟶ 3.4.5.7 Triangulierungsmethode: Def (unabh.):

Eine Teilmenge der Knotenmenge heißt unabhängig, wenn keine zwei Knoten durch eine Kante verbunden sind.

⟶ 3.4.5.8 Lemma:

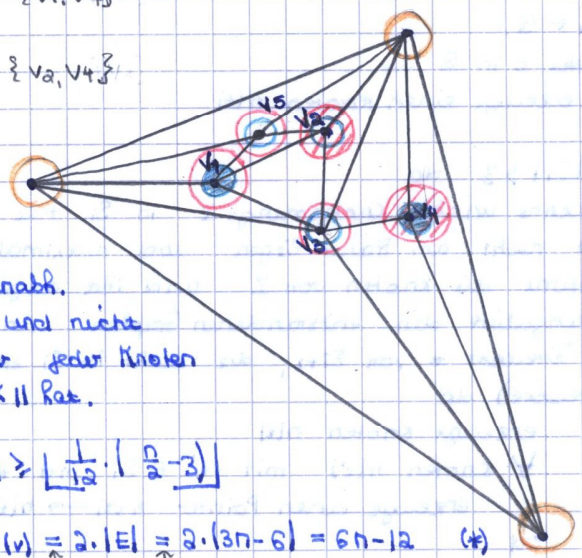
Der Graph G enthält eine unabhängige Knotenmenge I mit der Größe von mindestens $\left\lceil \frac{1}{12}\left(\frac{n}{2}-3\right)\right\rceil$, so dass jeder Knoten aus I höchstens Grad 11 hat und jeder Knoten ein nicht Hüll-Knoten ist. So eine Menge I kann in Zeit $O(n)$ gefunden werden.

Beweis: Betrachte folgenden Grady-Alg:

1. markiere alle drei Hüllknoten.
2. $I \leftarrow \emptyset$
3. repeat
4. wähle einen Knoten v mit Grad ≤ 11, der nicht markiert ist
5. $I \leftarrow I \cup \{v\}$
6. markiere v und alle seine Nachbarn.
7. until es gibt keine unmarkierten Knoten mit Grad ≤ 11.

a) $I \leftarrow \{v_1\} \cup I$, $I \leftarrow \{v_4\} \cup I$ $\Rightarrow I = \{v_1, v_4\}$

b) $I \leftarrow \{v_3\} \cup I$, $I \leftarrow \{v_8\} \cup I$ $\Rightarrow I = \{v_2, v_4\}$



Also:
- Alg. benötigt Zeit $O(n)$
- Er findet eine unabh. Knotenmenge I und nicht Hüllknoten, in der jeder Knoten höchstens Grad ≤ 11 hat.

Es bleibt z.z.: $|I| \geq \left\lfloor \frac{1}{12} \cdot \left(\frac{n}{2} - 3 \right) \right\rfloor$

Wissen: $\sum_{v \in V} \text{degree}(v) \overset{\text{allg.}}{=} 2 \cdot |E| \overset{\text{HÜ 2.1.}}{=} 2 \cdot (3n-6) = 6n-12$ (*)

→

$\Rightarrow$ G enthält mind. $\frac{n}{2}$ Knoten vom Grad ≤ 11.

Denn: Ann: nein.

$\Rightarrow$ mind. $\frac{n}{2}$ Knoten sind vom Grad > 11 bzw. dann ≥ 12.

$\Rightarrow \sum_{v \in V} \text{degree}(v) \geq \frac{n}{2} \cdot 12 = 6n$

$\Rightarrow \lightning$ zu (*)

// (*) sagt $\sum_{v \in V} \text{degree}(v) < 6n$!

Betrachte Algorithmus:

Alg. startet mit Markierung der drei begrenzten Knoten.

In diesem Moment gibt es mind. $\frac{n}{2} - 3$ unmarkierte Knoten mit Grad ≤ 11.

Dann wählt der Alg. einen dieser Knoten und markiert ihn zusammen mit seinen höchstens 11 Nachbarn.

Das wird solange wiederholt bis es keine unmarkierten Knoten mit Grad ≤ 11 mehr gibt.

Deshalb werden während jeder Iteration höchstens 12 Knoten markiert.

Es gibt also mindestens $\left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$

Während jeder Iteration wird ein Knoten zu I hinzugefügt.

Ergebnis: $\forall$ Triang. G gilt: G enthält unabh. Knotenmenge I, so dass

1) $|I| \geq \left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$
2) $\forall v \in I$: degree(v) ≤ 11
3) I enthält keinen der drei Randknoten.
4) I kann in Zeit $O(n)$ berechnet werden.

$\longmapsto$ 3.4.5.9 Lemma: Sei I eine unabh. Knotenmenge von G gemäß 3.4.5.8.
Sei G' der Graph, der durch Entfernen von I aus G entsteht

$\Rightarrow$ 1). Jede Fläche von G' ist ein 1-faches Polygon. mit maximal 11 Knoten

2). Die äußeren Flächen (Dreiecke) von G und G' sind gleich.

$\longmapsto$ 3.4.5.10 Alg. zu Konstruktion der Datenstruktur D zur Darstellung der Folge $S_1 .. S_k$:

Datenstruktur D: Knoten + Pointer.

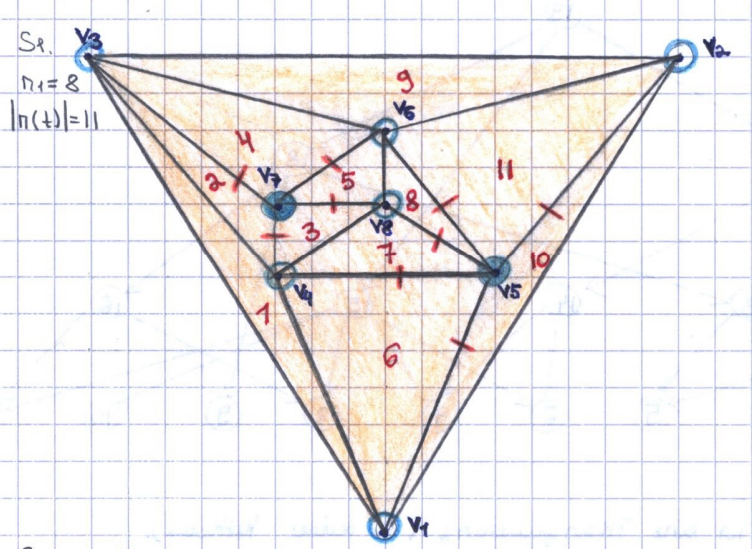
Knoten v speichert ein Dreieck t $\quad v = n(t)$.

Triangulierung S: Pkte, Kanten, Flächen

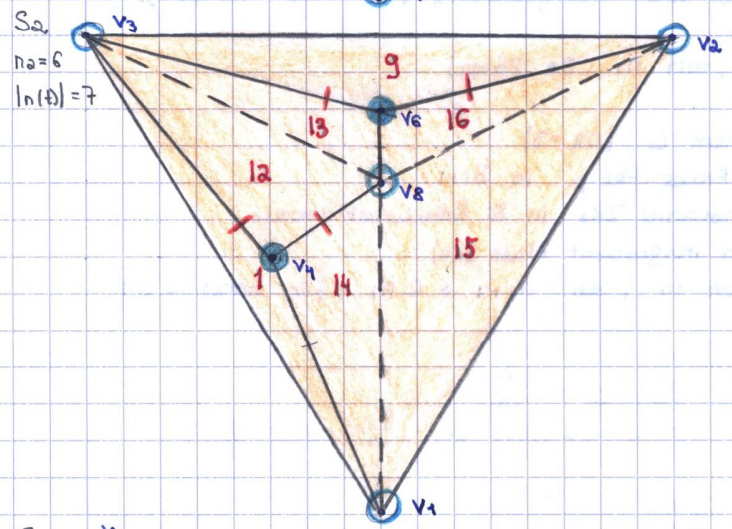
$n_i = |S_i|$ = # der Pkte.

1. $i \leftarrow 1$, $S_1 \leftarrow G$.
2. $\forall$ Dreiecke t in G do
3. erzeuge einen Knoten n(t)
4. od.
5. while $|S_i| > 3$ do
6. berechne unabh. Knotenmenge I in S_i mit mind. $\left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$ Knoten, die nicht am Rand liegen und maximal Grad 11 haben.
7. Entferne die Knoten von I und ihre angegrenzten Kanten aus S_i
8. Triangulire den entstandenen Graphen und nenne das Resultat S_{i+1}.
9. $\forall$ Dreiecke t von S_{i+1}, die nicht in S_i enthalten sind (d.h. neue Dreiecke) do
10. erzeuge Knoten n(t)
11. $\forall$ Knoten n(t') mit $t' \in S_i$ und t' schneidet t do
12. erzeuge einen Pointer $n(t) \longrightarrow n(t')$
13. od
14. od
15. $i \leftarrow i+1$
16. od

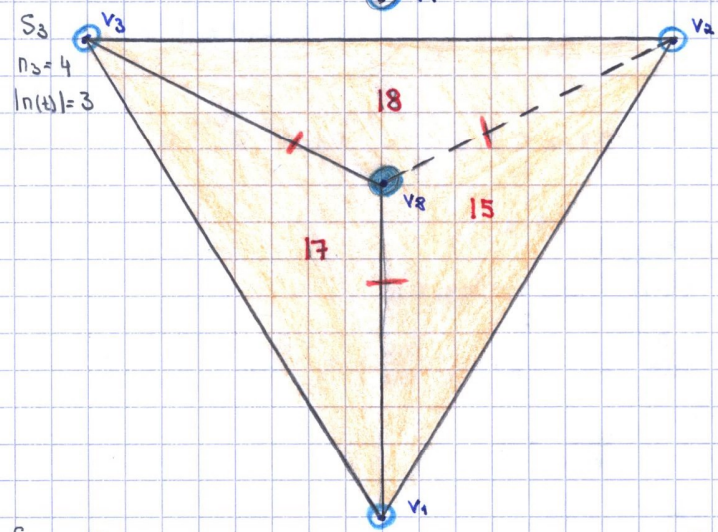
⟶ 3.4.5.11. Beispiel:



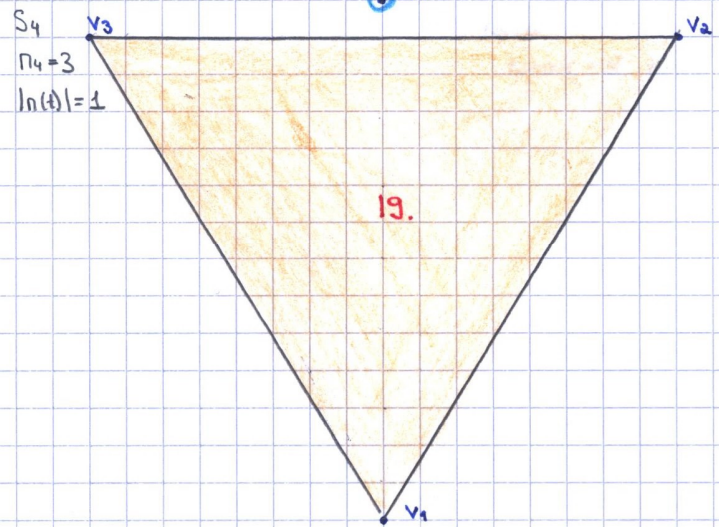
i = 1
I = { v5, v7 }
n(1) ... n(11) n(16)
neu Dreiecke: n(12), n(13), n(14), n(15)
n(12) ⟶ n(2), n(4), n(3), n(5)
n(13) ⟶ n(4), n(5)
n(14) ⟶ n(6), n(7)
n(15) ⟶ n(10), n(6), n(7), n(8), n(11)
n(16) ⟶ n(11), n(8)



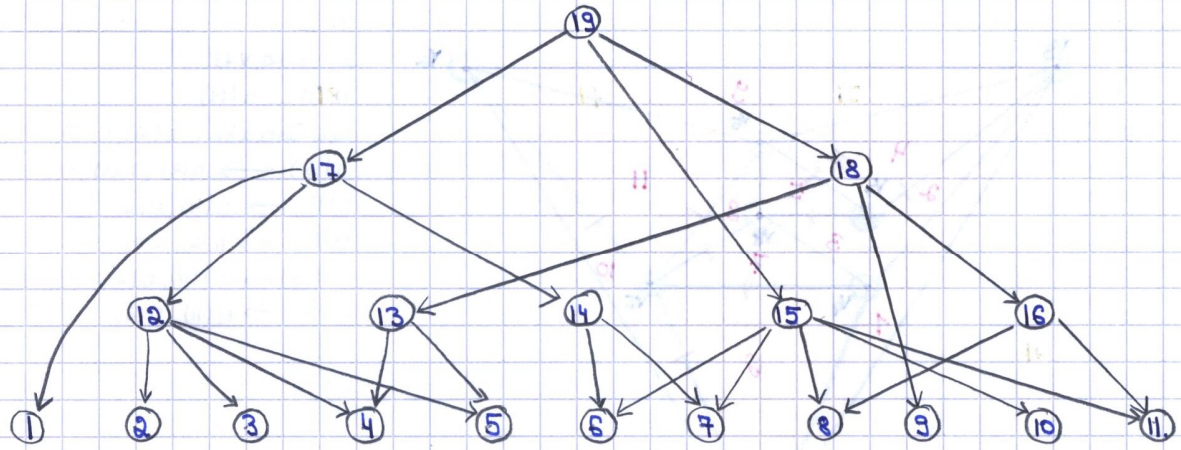
i = 2
I = { v6, v4 }
neue Dreiecke: n(16), n(17), n(18)
n(17) ⟶ n(1), n(14), n(12)
n(18) ⟶ n(13), n(9), n(16)



i = 3
I = { v8 }
neues Dreieck: n(19)
n(19) → n(15), n(17), n(18)



Datenstruktur D:



3.4.5.12. Zusammenfassung:

Planare Unterteilung G ist eine Triangulierung (bei dieser Methode).
(auch Rand ist Δ).

Ziel: Folge $S_1 .. S_R$ (mit $n_1 .. n_R$ Pkten)

✓ 1) $S_1 = G$

✓ 2) S_R besteht aus einem Δ ($n_R = 3$)

3) Lokalisierung von Frage-Pkt p in S_{i+1}
⇒ p kann in konstanter Zeit in S_i lokalisiert werden.

4) $S_1 .. S_R$ brauchen insgesamt Platz $O(n)$

5) n_{i+1} ist Bruchteil von n_i, d.h. $n_{i+1} \leq c \cdot n_i$ für konst $c \leq 1$
⇒ $R = O(\log n)$.

Es gilt: $n_{i+1} < n_i \ \forall i$

⇒ $n_{i+1} < c_i \cdot n_i$ für $c_i > \frac{n_{i+1}}{n_i}$

Ferner: $\frac{n_{i+1}}{n_i} < 1$ ⇒ c_i wählbar als $\frac{n_{i+1}}{n_i} < c_i < 1$

Gilt $\forall i$ und alle $c_i < 1$
⇒ wähle $c := c_R$ $\quad [\, n_3 < \underbrace{c_3}_{<1} \cdot n_2 < \underbrace{c_3 \cdot c_2}_{<1} \cdot n_1 < c_3 \cdot n_1 \,]$

⇒ $n_{i+1} < c \cdot n_i$

⇒ $3 = n_R < c \cdot n_{R-1} < c^2 \cdot n_{R-2} < \ldots < c^{R-1} \cdot n_1 = c^{R-1} \cdot n$

⇒ $n > \frac{3}{c^{R-1}}$

⇒ $\log_2 n > \log_2 3 + \log_2 \left(\frac{1}{c}\right)^{R-1}$

⇒ $\log_2 n > \frac{\ln 3}{\ln 2} + (R-1) \cdot (\underbrace{\log_2 1}_{=0} - \log_2 c)$

⇒ $R - 1 < \frac{\log n - \frac{\ln 3}{\ln 2}}{-\frac{\ln c}{\ln 2}} = \log n \cdot \left(-\frac{\ln c}{\ln 2}\right) + \frac{\ln 3}{\ln c}$

⇒ $R < \underbrace{-\frac{\ln c}{\ln 2}}_{\substack{>0 \text{ weil } c<1 \\ \text{const}}} \cdot \log n + \underbrace{\frac{\ln 3}{\ln c} + 1}_{\text{const}}$

⇒ $R \leq M \cdot \log n$ mit $M = \text{konst}$

⇒ $R = O(\log n)$.

3.4.5.13. Algorithmus für Point-Location:

Eingabe: Pkt q
Datenstruktur $\mathcal{D}$.

$\mathcal{D}$.search(q) liefert Dreieck von G, das q enthält.

if q außerhalb Dreieck i.d. Wurzel then
  Aussage: "äußeres Gebiet von G".
else
  $v \leftarrow$ Wurzel
  while v kein Blatt (dh. $\mathcal{D}$.outdeg(v)>0) do
    forall Knoten u mit $\exists$ Pointer $v \to u$ do
      if q innerhalb Dreieck von u then
        $v \leftarrow u$;
      fi
    od
  od
  Ausgabe: Dreieck von v
fi

Schleifeninvariante: Während der Ausführung der while-Schleife gilt stets, dass $q \in$ Dreieck von v.

Bemerkung: Platz:

$$\#\text{Knoten} = n_1 + n_2 + \dots + n_R \leq$$
$$\leq n_1 + c \cdot n_1 + \dots + c^{R-1} \cdot n_1 =$$
$$= n_1 \sum_{\nu=0}^{R-1} c^\nu = n_1 \cdot \frac{c^R - 1}{c-1} = n \cdot \underbrace{\frac{c^R - 1}{c-1}}_{\text{const}} = \mathcal{O}(n)$$