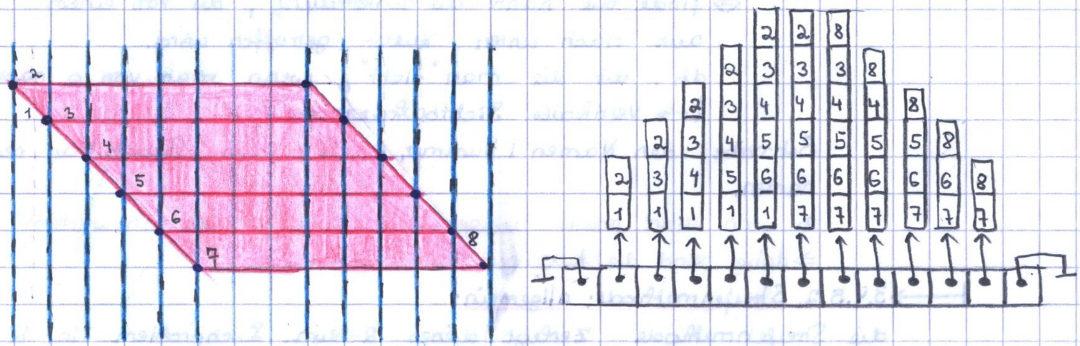
⟼ Streifenmethode: ergebnis:

Laufzeit: $O(\log n)$ (optimal)

Platzbedarf: $O(n^2)$ (unpraktisch für grosse 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:

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

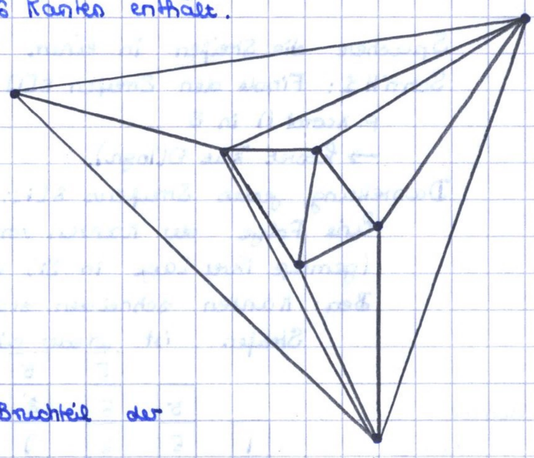
Sei G ein planarer Graph (dh. zwei versch. Kanten überschneiden sich nicht u. ungerichtet) auf n Knoten und weiter sei jedes Face von G ein Dreieck (auch das äußere Face)

⇒ 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 $S_1 .. S_k$ von Triangulierungen, so dass gilt:

(1) $S_1 = G$

(2) S_k ist das äußere Dreieck von G

(3) $k = O(\log n)$

(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: $S_1 .. S_k$

dann lösen wir das Point Location Problem wie folgt:

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

Da $k = O(\log n)$ 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 $S_1 .. S_R$?

→ Bem. $S_1 = G$.

Wollen einen konst. Bruchteil der Knoten entfernen.

Sei v ein non-boundary Knoten von G, und sei d sein Grad, d.h. in G sind d Kanten inzident zu v.

2) Frage: Was passiert, wenn wir v aus G 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 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.

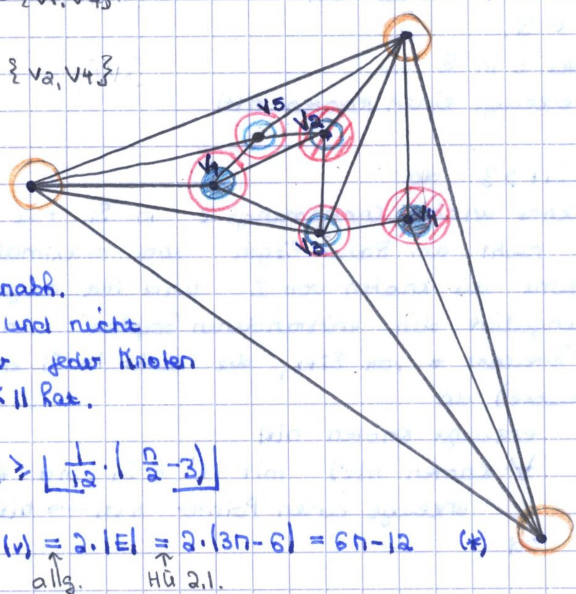
So eine Menge I kann in Zeit $O(n)$ gefunden werden.

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



klar:

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

→

$\Rightarrow$ 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.

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

$\Rightarrow$ ↯ 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.

⟼ 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.

⟼ 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 $v = n(t)$.

Triangulierung S: Pkte, Kanten, Flächen

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

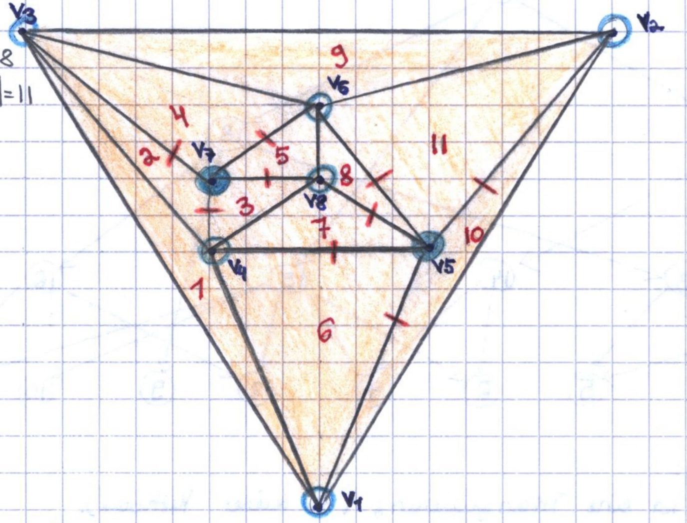
1. $i \leftarrow 1$, $S_1 \leftarrow G$.
2. $\forall$ Dreiecke t in G do
3. erzeuge einen Knoten n(t)
4. od.
5. while $|S_i| > 3$ do
6. berechne unabh. Knotenmenge I in S_i mit mind. $\left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$ Knoten, die nicht am Rand liegen und maximal Grad 11 haben.
7 Entferne die Knoten von I und ihre angegrenzten Kanten aus S_i
8 Trianguliere 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 (dh. 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:

S_1

$n_1 = 8$

$|n(t)| = 11$



$i = 1$

$I = \{ v_5, v_7 \}$

$n(1) \dots n(11)$ $n(16)$

neu Dreiecke: $n(12), n(13), n(14), n(15)$

$n(12) \longrightarrow n(2), n(4), n(3), n(5)$

$n(13) \longrightarrow n(4), n(5)$

$n(14) \longrightarrow n(6), n(7)$

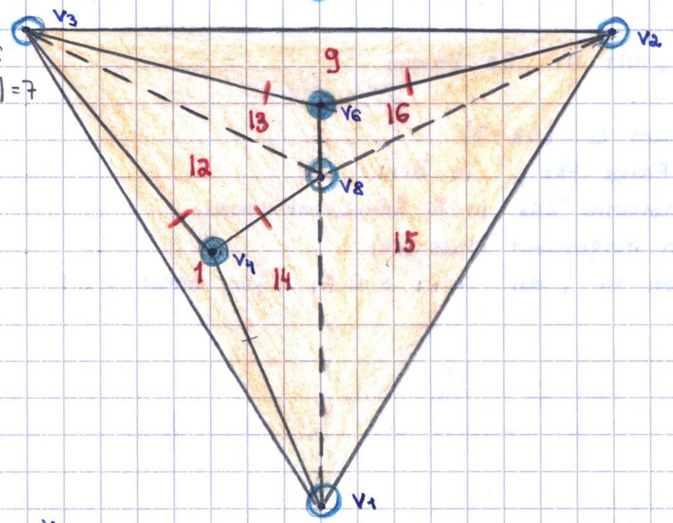
$n(15) \rightarrow n(10), n(6), n(7), n(8), n(11)$

$n(16) \rightarrow n(11), n(8)$

S_2

$n_2 = 6$

$|n(t)| = 7$



$i = 2$

$I = \{ v_6, v_4 \}$

neue Dreiecke: $n(16), n(17), n(18)$

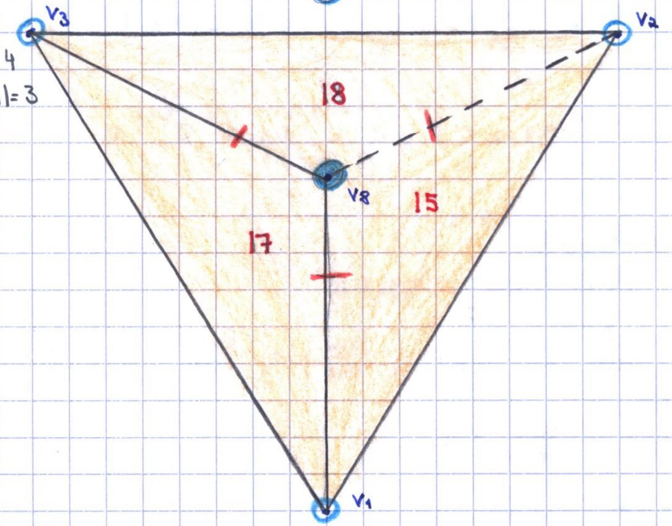
$n(17) \rightarrow n(1), n(14), n(12)$

$n(18) \rightarrow n(13), n(9), n(16)$

S_3

$n_3 = 4$

$|n(t)| = 3$



$i = 3$

$I = \{ v_8 \}$

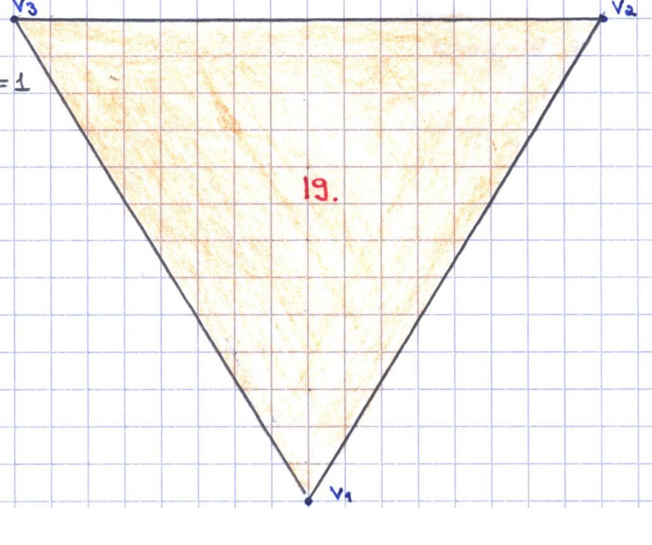
neues Dreieck: $n(19)$

$n(19) \rightarrow n(15), n(17), n(18)$

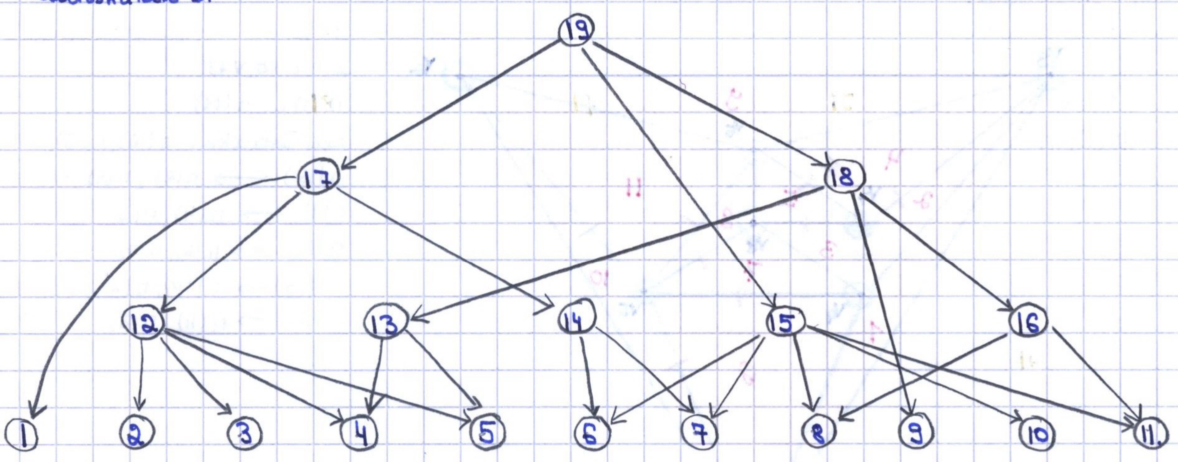
S_4

$n_4 = 3$

$|n(t)| = 1$



Datenstruktur D:



→ 3.4.5.12. Zusammenfassung:

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

Ziel: Folge $S_1 .. S_k$ (mit $n_1 .. n_k$ Pkten)

1) $S_1 = G$
2) S_k besteht aus einem Δ ($n_k = 3$)
3) Lokalisierung von Frage-Pkt p in S_{i+1}
   ⇒ p kann in konstanter Zeit in S_i lokalisiert werden.
4) $S_1 .. S_k$ 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 $c \leq 1$
   ⇒ $k = O(\log n)$.

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

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

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

Gilt $\forall i$ und alle $c_i < 1$

⇒ Wähle $c := c_k \quad [\; n_3 < \underbrace{c_3}_{<1} \cdot n_2 < \underbrace{c_3}_{<1} \cdot \underbrace{c_2}_{<1} \cdot n_1 < c_3 \cdot n_1 \;]$

$\Rightarrow n_{i+1} < c \cdot n_i$

$\Rightarrow 3 = n_k < c \cdot n_{k-1} < c^2 \cdot n_{k-2} < \dots < c^{k-1} \cdot n_1 = c^{k-1} \cdot n$

$\Rightarrow n > \frac{3}{c^{k-1}}$

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

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

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

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

$\Rightarrow k \leq M \cdot \log n$ mit $M = $ konst

$\Rightarrow k = O(\log n)$.

3.4.5.13. Algorithmus für Point-Location:

Eingabe: Pkt q
Datenstruktur D.

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 ← Wurzel
    while v kein Blatt (d.h. D.outdeg(v)>0) do
        forall Knoten u mit ∃ Pointer v → u do
            if q innerhalb Dreieck von u then
                v ← 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:

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