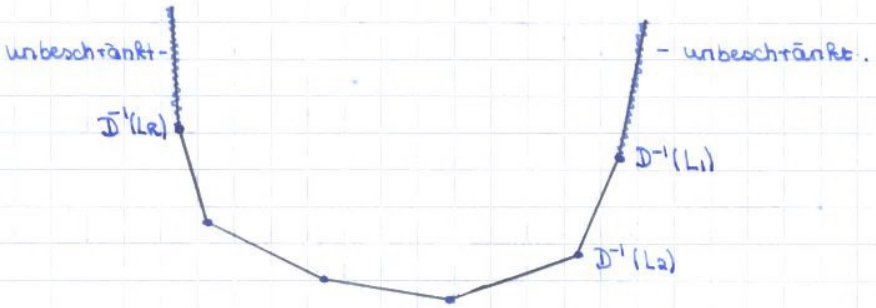
. $e_1 \dots e_R$ Kanten von H von links nach rechts.
⇒ $L_1 \dots L_R$ nach Steigungen absteigend sortiert. (siehe Skizze ⇒ klar!)
⇒ Ecken von S^+ sind nach x-Koordinaten absteigend sortiert, d.h. von rechts nach links.



Frage: Wie finden wir die beiden unbeschränkten Kanten?
Antwort: Linke Gerade mit minimaler Steigung.
Rechte Gerade mit maximaler Steigung.

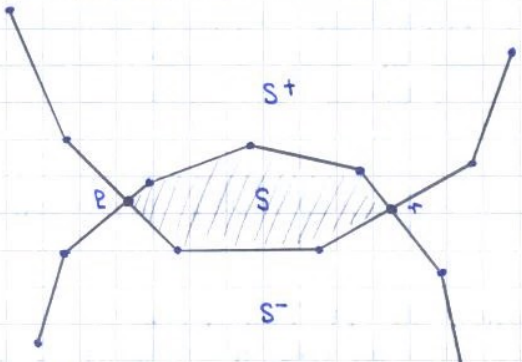
⟶ 1.5.8.6. Zwischenresultat:
$S^+ = \bigcap_{i=1}^{m} l_i^+$ kann in Zeit $O(m \log m)$ berechnet werden.

$\underbrace{O(m)}$ + $\underbrace{O(m \log m)}$
$D, D^{-1} \dots$ Graham's Scan.

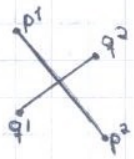
Falls $l_1 \dots l_m$ nach Steigung sortiert gegeben sind, dann Laufzeit $O(m)$ (↝ Graham's Scan)

$S^- = \bigcap_{i=m+1}^{n} l_i^-$ kann genauso (symmetrisch) berechnet werden.

Es bleibt noch $S = S^+ \cap S^-$ auszurechnen.
⟶ 1.5.8.7. Berechne S.

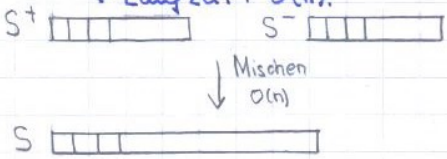


Finde Schnittpkte l und r des Ränder von S^+ und S^-
Achtung: Sonderfälle:
- ex. nicht ($S = \emptyset$)
- $l = r$ ($S = \{l\} = \{r\}$)
- l, r Ecken von S^+, S^-



Allgemein: Innere Schnittpkte von Kanten
Voraussetzung: Die Ränder sind von links nach rechts sortiert.

⇒ Laufzeit: $O(n)$.



und gleichzeitig jeweils das l_i, welches eingefügt wird, in int S^+ und S^- schreiben und vergleichen.
Sobald Änderung ⇒ Kante gefunden. ⇒ Schnittpkt berechnen
⇒ Gesamtlaufzeit: $O(n)$.

1.5.9. Satz (Zusammenfassung)

1). Der Schnitt von n Halbebenen kann in Zeit $O(n \log n)$ berechnet werden.

2). Falls die definierenden Geraden nach Steigung sortiert gegeben sind, dann ist die Laufzeit $O(n)$.

1.5.10 Bemerkungen:

1) Die geometrische Transformation der Dualität wird auch noch für andere Probleme dieser Vorlesung wichtig sein.

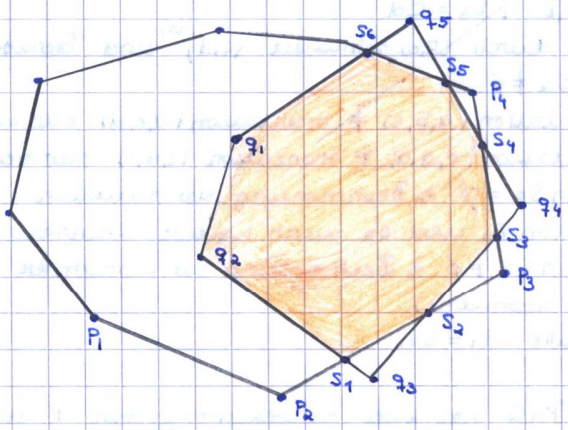
2) Der Schnitt von n Halbebenen kann auch direkt berechnet werden.
   ⇝ Divide & Conquer.

1.5.11. Schnitt von Halbebenen durch Divide and Conquer

Schnitt von zwei konvexen Polygonen.
Hier abgeschlossen (offen → Übung, voriger Abschnitt)
Geg: Seien $P = (p_1 .. p_m)$, $Q = (q_1 .. q_e)$ konvexe abgeschlossene Polygone, geg. durch Eckenfolge gg. den Uhrzeigersinn.
Ausgabe: Eckenfolge von $P \cap Q$ gegen Uhrzeigersinn.

↳ **1.5.11.1. Beispiel.**

Mögliche Ausgabe:
$q_1 q_2 s_1 s_2 s_3 s_4 s_5 s_6$

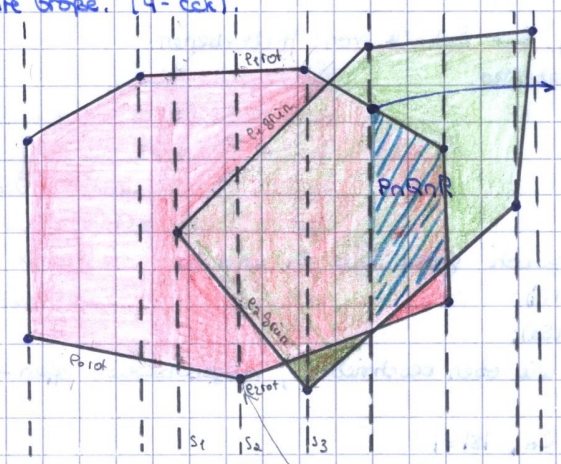


↳ **1.5.11.2. Idee:**

Zerlege Ebene in Regionen so, dass für jede Region R $(P \cap Q) \cap R$ einfach zu berechnen ist.

↳ **1.5.11.3 Regionen:** Vertikale Streifen.

Zeichne durch jede Ecke von P und Q eine senkrechte Gerade.
Jeweils zwei benachbarte Senkrechten definieren eine Region, nämlich den vertikalen Streifen dazwischen.
Dann ist für jeden Streifen R $R \cap P$ und $R \cap Q$ ein Trapezoid, d.h. haben konstante Größe. (4-Eck).



Es gilt $P \cap Q \cap R = (P \cap R) \cap (Q \cap R)$
wegen konst. Größe braucht Zeit $O(1)$

1.5.11.4. Algorithmus:

1). Berechne Streifen von links nach rechts sortiert in Zeit $O(n)$, wobei $n = m+p$. Gesamtzahl der Ecken, durch Mischen der Eckenfolgen von P und Q. (siehe D&C für CH.)

2). Berechne für jeden Streifen R Trapezoide $P \cap R$ und $Q \cap R$. Das geht auch in $O(n)$, da wir Eckenfolgen (siehe 1) von links nach rechts sortiert haben.

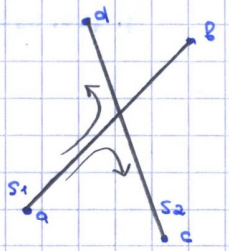
3) Berechne für jeden Streifen R $P \cap Q \cap R := (P \cap R) \cap (Q \cap R)$
Zeit $O(1)$ pro Streifen.

4). Zusammen kleben der Teile aus 3)
Insbesondere Elimination von Ecken, die nicht zur Ausgabe gehören.
Laufzeit: $O(n)$ (Durchlaufen der Senkrechten von links nach rechts).
→ Eckenfolge von $P \cap Q$ gg. Uhrzeigersinn.

1.5.11.5. Implementierungsdetails:

Teilprobleme:

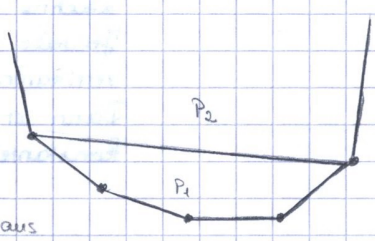
- Test ob Segmente s_1 ein anderes schneidet.
  $s_1 = \overline{a,b}$, $s_2 = \overline{c,d}$
  Gerade durch (a,b) schneidet (c,d) (1) und Gerade durch (c,d) schneidet (a,b) (2)
  $\Leftrightarrow s_1 \cap s_2 \neq \emptyset$
  (1) orientation(a,b,c) $\neq$ orientation(a,b,d) oder beide 0
  (2) orientation(c,d,a) $\neq$ orientation(c,d,b) oder beide 0.
- falls $s_1 \cap s_2 = \emptyset$ ⇒ Bestimmung der relativen Lage von s_1 und s_2 durch weitere Orientierungstests.
- falls $s_1 \cap s_2 \neq \emptyset$ ⇒ Bestimmung des Schnittpkte (anal. Geometrie)
- Sonderfälle: $s_1 = s_2$.



1.5.12. Satz: Der Schnitt $P \cap Q$ von zwei konvexen Polygonen P und Q kann in Zeit $O(n)$ berechnet werden, wobei n = Summe der Ecken von P und Q.
Mischschritt

1.5.13. Fragen:

1) Geht das schneller, wenn alle Polygone im Voraus geg. sind, für die man evtl. Schnittoperationen ausführt.
2). Existiert Algorithmus, der output-sensitiv ist?
Dh. Laufzeit hängt von Größe der Ausgabe ab.
→ siehe nächster Abschnitt über konvexe Polygone.



1.5.14. Divide & Conquer - Alg. für Schnitt von Halbebenen.

Sei S Menge von Halbebenen.

```
INTERSECT(S)
if |S| = 1 then
    return S;
else
    teile S in zwei gleich große Teile S1 und S2
    P ← INTERSECT(S1);
    Q ← INTERSECT(S2);
    return P ∩ Q   // wie oben beschrieben, möglicherweise offen → Übung.
```

Ein unbeschr. Polygon besteht aus

- einem beschr. Pol. P_1 und unbeschr. Pol. P_2 ⇒ $P = P_1 \cup P_2$
  Berechne $P_2 \cap Q_2$ mit D&C und Mischritt von oben ⇒ $O(n \log n)$
  Berechne $P_2 \cap Q_1$ und $P_1 \cap Q_2$ als Schnitt von Polygon mit drei Geraden in Zeit $3 \cdot O(n \log n) = O(n \log n)$. Anschließend berechne $P_2 \cap Q_2$ als Schnitt von 6 Geraden in $O(9) = O(1)$
  ⇒ Gesamtlaufzeit: $O(n \log n)$.

1.5.15 Laufzeit:

$$T(n) = \begin{cases} c_0, & |S| = 1 \\ c_1 n + 2 \cdot T(n/2), & |S| > 1 \end{cases}$$

⇒ $T(n) = O(n \log n)$
Alternative zu Dualitätsalgorithmus

1.5.16. Frage: Welcher Algorithmus ist in welchen Fällen schneller? Wahrscheinlich Divide & Conquer

Kapitel II: Konvexe Polygone

2.1. Einführung:

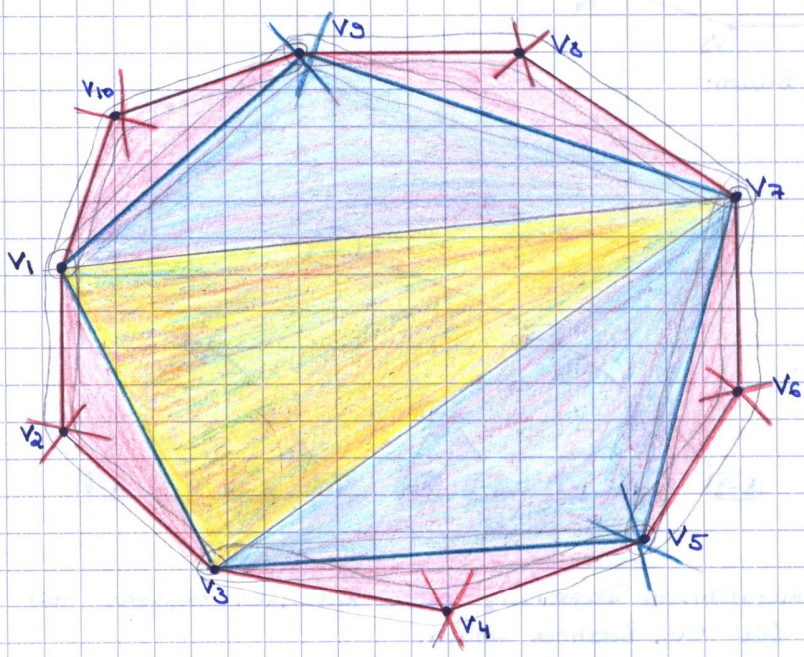
2.1.1. Ziel: Datenstruktur für konvexe Polygone, die verschiedene Operationen effizient unterstützt. → Hierarchische Darstellung.

2.1.2. Idee: Investiere Zeit und Platz $O(n)$ in Aufbau dieser Struktur, um nachfolgende Operationen billig zu machen.
Lohnt sich nur, wenn mit demselben Polygon viele Operationen durchgeführt werden.
Hierarchische Darstellung: Konstruiere Folge von Teilpolygonen, die das Polygon immer genauer beschreiben.

2.1.3. Def: Sei P konvexes Polygon mit n Ecken $v_1 .. v_n$.
Eine Folge $P_0, P_1, ..., P_k$ von konvexen Polygonen heißt (innere) hierarchische Darstellung von P, wenn gilt:

1) P_0 hat ≤ 4 Ecken
2) $P_k = P$, $P_i \neq P_{i+1}$ $\forall i \in \{0..k\}$
3) Jede Ecke von P_i ist Ecke von P_{i+1} und von vier aufeinanderfolgenden Ecken von P_{i+1} ist mindestens eine und höchstens drei von P_i. (für $0 \leq i \leq k$).

2.1.4. Beispiel:



$P = v_1 .. v_{10} = P_2$
$P_1 = v_1 v_3 v_5 v_7 v_9$
$P_0 = v_1 v_3 v_7$
$\Rightarrow \{P_i\}_{i=0}^{2}$ ist die Hierarchische Darstellung von P.

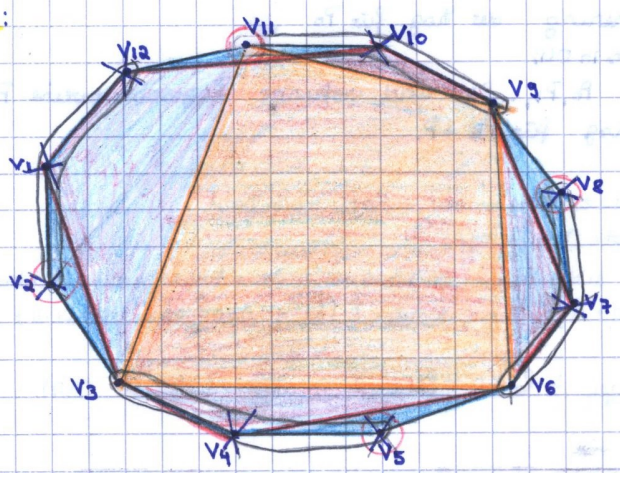
Bea: P_0 kann auch $v_1 v_3$ sein.
Dann wäre die Darstellung bzgl der Hierarchie noch tiefer.
$P_k = P_2$ $|P_2| = 10$
$|P_{k-1}| = |P_1| = 5$

2.1.5 Bemerkung: P_{i-1} entsteht also aus P_i durch entfernen einiger Ecken.
- von jeweils drei aufeinander folgenden Ecken von P_i wird mindestens eine entfernt.
- es werden nie vier aufeinanderfolgende entfernt.

2.1.6. Eigenschaften:

1) Beim Übergang von $P_{i+1} \to P_i$ verlieren wir mindestens einen konstanten Bruchteil der Ecken ($1/3$)
2) P_{i+1} ist ähnlich zu P_i, da wir nie mehr als drei aufeinanderfolgende Ecken entfernen.

2.1.7. Beispiel:



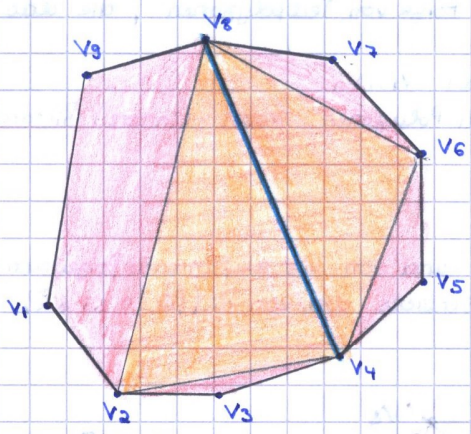
Blau: P_0, $|P_0| = 12$
rot: P_{k-1}: es wurden möglichst wenig Ecken entfernt.
$|P_{k-1}| = 8$
$P_k \to P_{k-1}$ wurden $\frac{1}{3} \cdot |P_k| = \frac{1}{3} \cdot 12 = 4$ entfernt!
orange: P_{k-1}: es wurden möglichst viele Ecken entfernt.
$|P_{k-1}| = 4$
$P_k \to P_{k-1}$ wurden 8 Ecken entfernt!

2.1.8 Alternative Darstellung beim Zeichnen:

balancierte Baum über Kanten von P_i $(0 \le i \le n)$.
(muß kein binärer Baum sein).

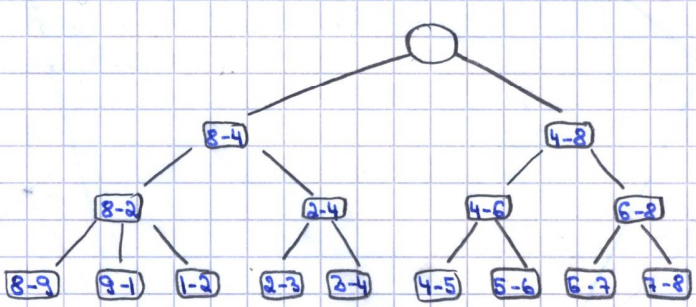
2.1.9 Beispiel:

1) wie vorhin:



$P_0 = v_4v_8$, $|P_0| = 2$
$P_1 = v_8v_9v_1v_2v_3v_4v_5v_6$, $|P_1| = 4$
$P_2 = P = v_1 \dots v_9$, $|P| = 9$.

2) durch balancierten Baum:



2.1.10 Lemma: Könnte man zB mit ähnlichem Alg wie bei Triangulierungsmethode machen.

1) Eine balancierte hierarchische Darstellung eines konvexen Polygons mit n Ecken kann in Zeit $O(n)$ berechnet werden.
2) benötigt Speicherplatz $O(n)$
3) die Tiefe k dieser Darstellung ist $O(\log n)$

Beweis: Alle Teile folgen unmittelbar aus 2.1.6. i). bzw. Baumdarstellung.

zu 2) von $P_i \to P_{i-1}$ verlieren mind $\frac{1}{3}$ der Knoten

$$\Rightarrow \#\text{Knoten} \le n + \tfrac{2}{3}n + \tfrac{2}{3}\left(\tfrac{2}{3}n\right) + \dots = n\left(1 + \tfrac{2}{3} + \tfrac{4}{9} + \dots\right) = n \cdot \sum_{\nu=0}^{\infty} \left(\tfrac{2}{3}\right)^\nu \le n \cdot \sum_{\nu=0}^{n} \left(\tfrac{2}{3}\right)^\nu =$$

$$= n \cdot \frac{\left(\frac{2}{3}\right)^{n+1} - 1}{\frac{2}{3} - 1} = n \cdot \frac{\left(\frac{2}{3}\right)^{n+1} - 1}{-\frac{1}{3}} = n \cdot \frac{1 - \left(\frac{2}{3}\right)^{n+1}}{\frac{1}{3}} = n \cdot 3 \cdot \underbrace{\left(1 - \left(\tfrac{2}{3}\right)^{n+1}\right)}_{\le 1} \le 3n = O(n)$$

2.2. Anwendungen der hierarchischen Darstellung.

verwenden immer dieselbe Strategie:

2.2.1. Strategie:

1). Wir berechnen eine Annäherung der Lsg (des Problems) für P_0
Da P_0 maximal 4 Ecken $\Rightarrow$ in $O(1)$
2) Dann durchlaufe die Folge $P_0, P_1, \dots P_k$ und verfeinere Lösungsschrittweise $P_i \to P_{i+1}$.
3) Am Ende haben wir die Lsg für $P_k = P$.

zu 2.1.10 3): $\text{Höhe}(T) \le \underbrace{\text{Höhe}(\tilde{T})}_{=O(\log n)}$, wobei $\tilde{T}$ binärer Baum mit entspr. Hierarch. Darst.
$\Rightarrow$ Höhe $(T) = O(\log n)$.

zu 2.1.10. 1) Sei $c_1 :=$ Arbeit, um einen Knoten in den Baum zu schreiben (4 Stk); $c_2 :=$ Arbeit, um einen Knoten zu streichen

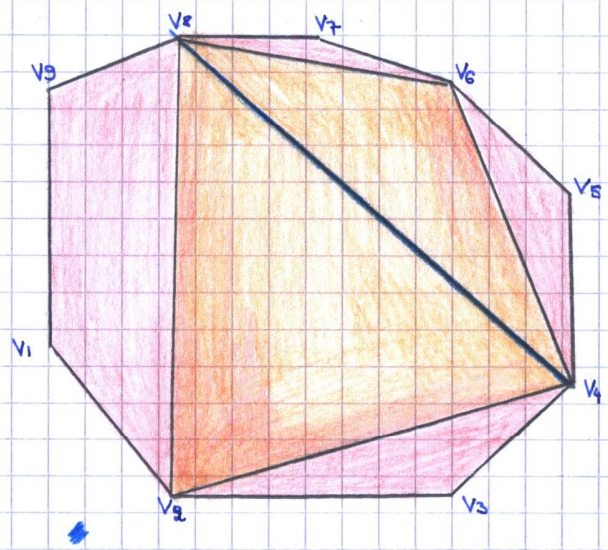
$$\Rightarrow \text{Arbeit} \le c_1 \cdot n + c_2 \cdot \tfrac{1}{3}n + c_1 \cdot \tfrac{2}{3}n + c_2 \cdot \left(\tfrac{1}{3}\right)^2 n + c_1 \left(\tfrac{2}{3}\right)^2 n + \dots \le n c_1 \cdot \sum_{0}^{\infty} \left(\tfrac{2}{3}\right)^\nu + c_2 n \sum_{0}^{n} \left(\tfrac{1}{3}\right)^\nu \le 3c_1 \cdot n + \tfrac{3}{2} \cdot c_2 n = O(n)$$

Aus Listen dann in Baum einfügen $\Rightarrow$ Kostet #Element in allen Listen $\Rightarrow O(n)$.

2.1.8. Alternative Darstellung:

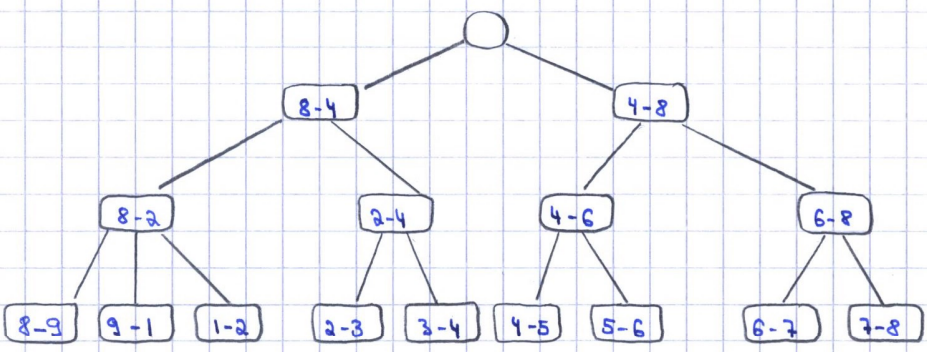
→ balancierter Baum über Kanten von P_i $(0 \le i \le n)$
(muß kein binärer Baum sein.)

2.1.9. Beispiel:



$P_0 = \{v_4, v_8\}$, $|P_0| = 2$
$P_1 = \{v_2, v_4, v_6, v_8\}$, $|P_1| = 4$
$P_2 = \{v_1 .. v_9\}$, $|P_2| = 9$

Darstellung durch einen balancierten Baum:



2.1.10 Lemma:

1. Eine balancierte hierarchische Darstellung eines konvexen Polygons mit n Ecken kann in Zeit $O(n)$ berechnet werden.
2. benötigt Speicherplatz $O(n)$
3. die Tiefe k dieser Darstellung ist $O(\log n)$.

Beweis: Alle Teile folgen unmittelbar aus 2.1.6 i) bzw. Baumdarstellung.

Beachte:

zu 1):

→ könnte man zB. mit ähnlichem Alg. wie bei der Triangulierungsmethode machen

$$c \cdot n + c \cdot \left(\tfrac{2}{3}\right) \cdot n + c \cdot \left(\tfrac{2}{3}\right)^2 \cdot n + \ldots =$$

$$= c \cdot n \cdot \sum_{\nu=0}^{\log n} \left(\tfrac{2}{3}\right)^\nu \le c \cdot n \cdot \sum_{\nu=0}^{\infty} \left(\tfrac{2}{3}\right)^\nu = c \cdot n \cdot \frac{1 - \left(\frac{2}{3}\right)^{n+1}}{1 - \frac{2}{3}} =$$

$$= c \cdot n \cdot 3 \cdot \underbrace{\left(1 - \left(\tfrac{2}{3}\right)^{n+1}\right)}_{\substack{n \to \infty \\ \to 0}} \le \underbrace{M}_{\text{const}} \cdot n = O(n).$$

zu 2):

$\#\text{Knoten} = O(n)$

zu 3): $\text{Höhe}(T) \le \underbrace{\text{Höhe}(\tilde{T})}_{= O(\log n)}$, wobei $\tilde{T}$ binärer Baum.

2.2. Anwendung der Kirkpatrickschen Darstellung:

→ verwenden immer dieselbe Strategie:

2.2.1. Strategie:

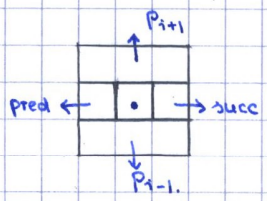
1). Wir berechnen eine Annäherung der Lösung für P_0.
Da P_0 maximal vier Ecken ⇒ in $O(1)$.

2). Dann durchlaufe die Folge $P_0, P_1 .. P_k$ und verfeinere die Lösung schrittweise $P_i \to P_{i+1}$.

3) Am Ende haben wir die Lsg für $P_k = P$.

2.2.2. Darstellung im Rechner:

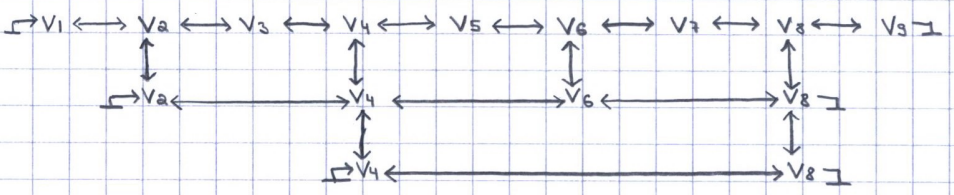
- P_i, $0 \le i \le k$ als doppelt verkettete Liste der Ecken und jede Ecke von P_i zeigt zusätzlich auf ihre Kopie in P_{i+1} und umgekehrt.



- explizite Speicherung des Kantenbaumes
→ Dynamisierung (Ecken einfügen | löschen)
(wie bei Suchbäumen).

2.2.3 Beispiel:

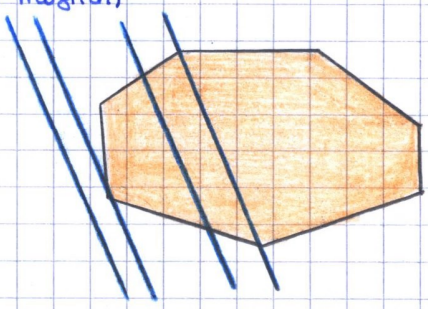
Sei P gegeben wie in 2.1.9.
⇒



2.24. Anwendung: Schnitt mit einer Geraden:

⟶ 2.2.4.1. Ziel:

Geg: Hierarchische Darstellung $P_0 \dots P_k$ eines konvexen Polygons P und eine Gerade g
Ausgabe: $P \cap g$
Konvexität ⇒ $P \cap g$ = Strecke (wenn nicht leer und Ecke)
⇒ Es genügt Schnittpkte a, b mit den Randsegmenten zu berechnen.
Fälle: 1) $a \neq b$ (allg. Fall, Ecken möglich)
2) $a = b$ (eine Ecke)
3) ex. nicht! $P \cap g = \emptyset$.



⟶ 2.24.2. Idee für Algorithmus:

1). Schnittest mit P_0:
2 Fälle: a) $P_0 \cap g = \emptyset$
b) $P_0 \cap g \neq \emptyset$
Im Fall B) → STOP
Im Fall a):
sei $g_0 \in P_0$ mit minimalem Abstand zu g.