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.

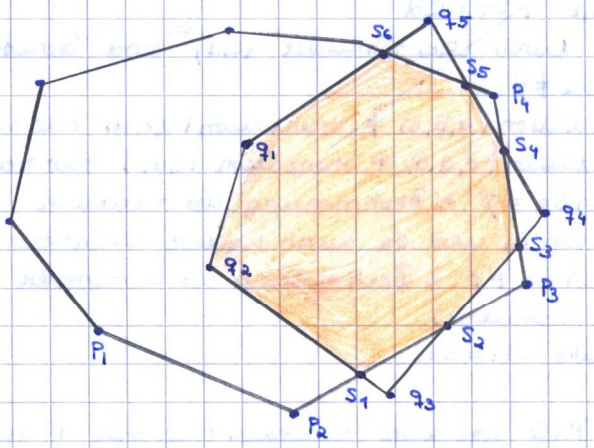
2) Der Schnitt von n Halbebenen kann auch direkt berechnet werden.
   ~> Divide & Conquer.

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:
$q_1 q_2 s_1 s_2 s_3 s_4 s_5 s_6$

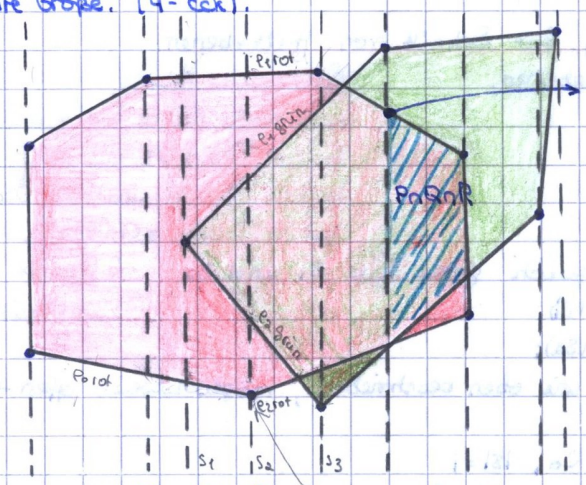


→ 1.5.11.2. Idee:

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.

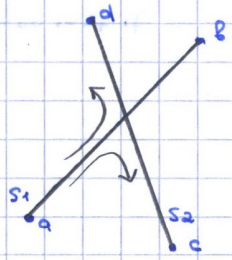
3) Berechne für jeden Streifen R $P \cap Q \cap R := (P \cap R) \cap (Q \cap R)$
Zeit $O(1)$ pro Streifen.

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:

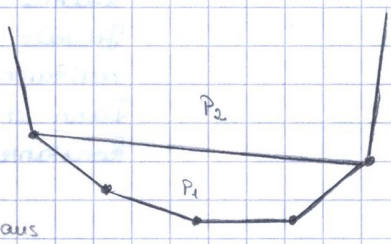
- Test ob Segmente s_1 ein anderes schneidet.
  $s_1 = \overline{a,b}$, $s_2 = \overline{c,d}$
  Gerade durch (a,b) schneidet (c,d) (1) und Gerade durch (c,d) schneidet (a,b) (2)
  $\Leftrightarrow s_1 \cap s_2 \neq \emptyset$
  (1) orientation(a,b,c) $\neq$ orientation(a,b,d) oder beide 0
  (2) orientation(c,d,a) $\neq$ 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: $s_1 = s_2$.



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

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.



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: $O(n \log n)$.

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

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

Kapitel II: Konvexe Polygone

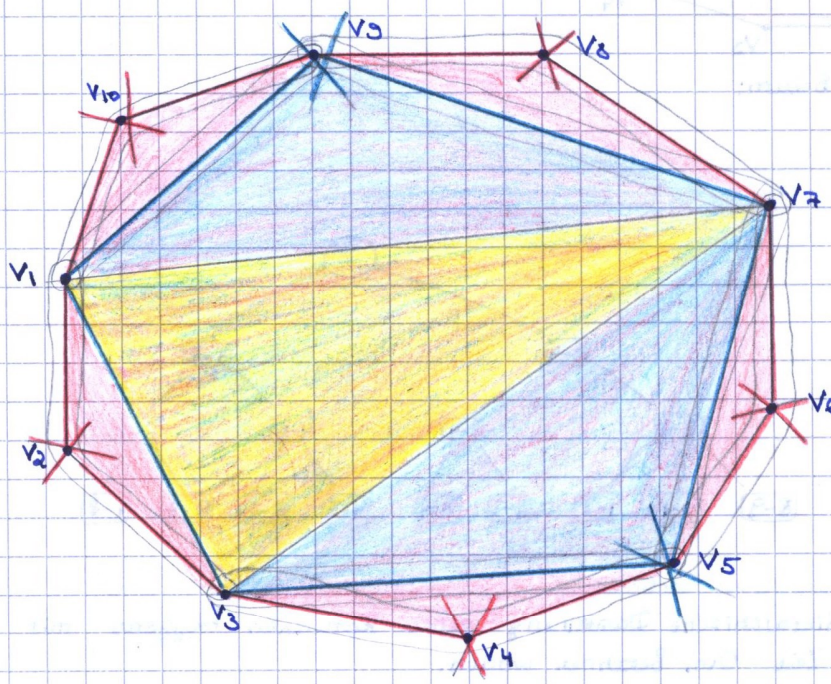
2.1. Einführung:

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:



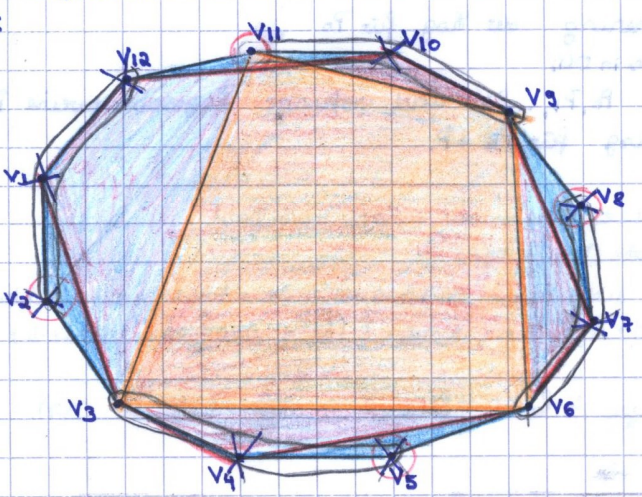
$P = v_1 .. v_{10} = P_2$
$P_1 = v_1 v_8 v_5 v_7 v_9$
$P_0 = v_1 v_8 v_7$
$\Rightarrow \{P_i\}_{i=0}^{2}$ ist die Hierarchische Darstellung von P.

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.

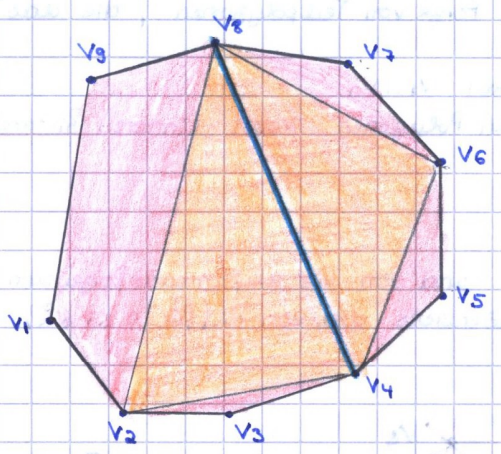
2.1.7. Beispiel:



Blau: P_0, $|P_0| = 12$
rot: P_{k-1}: es wurden möglichst wenig Ecken entfernt.
$|P_{k-1}| = 8$
$P_k \to P_{k-1}$ werden $\frac{1}{2} \cdot |P_k| = \frac{1}{2} \cdot 12 = 4$ 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!

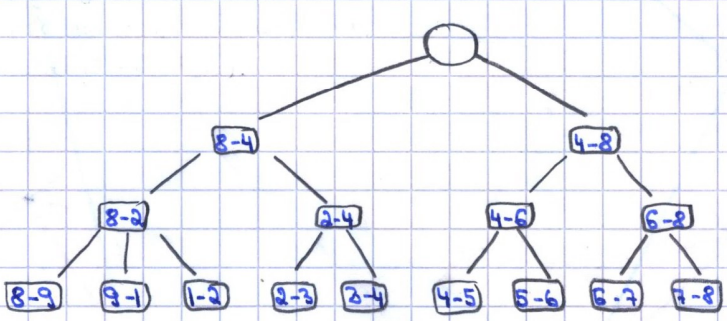
2.1.8 Alternative Darstellung beim Zeichnen:
balancierte Baum über Kanten von P_i $(0 \le i \le n)$.
(muß kein binärer Baum sein).

2.1.9 Beispiel:
1) wie vorhin:



$P_0 = V_4V_8$, $|P_0| = 2$
$P_1 = V_8V_2V_4V_6$, $|P_1| = 4$
$P_2 = P = V_1 \dots V_9$, $|P| = 9$.

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 $O(\log n)$

Bew: 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

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

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

2. Anwendungen der hierarchischen Darstellung.

verwenden immer dieselbe Strategie:

2.2.1 Strategie:
1). Wir berechnen eine Annäherung der Log (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): 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; $c_2 :=$ Arbeit, um einen Knoten zu streichen

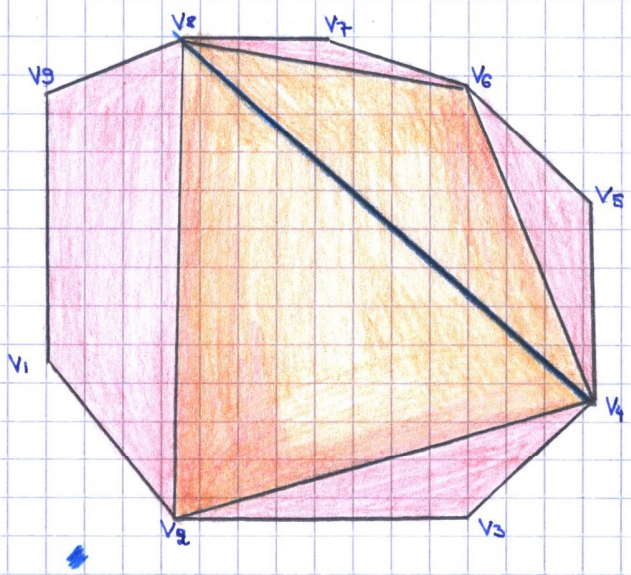
$$\Rightarrow \text{Arbeit} \le c_1 \cdot n + c_2 \cdot \frac{1}{3}n + c_1 \cdot \frac{2}{3}n + c_2 \cdot \left(\frac{1}{3}\right)^2 n + c_1 \cdot \left(\frac{2}{3}\right)^2 n + \dots \le n c_1 \cdot \sum_{0}^{\infty}\left(\frac{2}{3}\right)^{\nu} + c_2 n \sum_{0}^{\infty}\left(\frac{1}{3}\right)^{\nu}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

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

2.1.8. Alternative Darstellung:

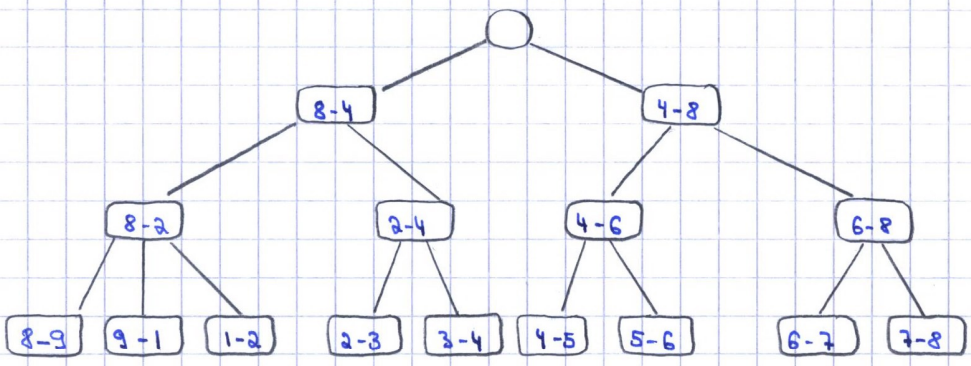
→ balanciertes Baum über Kanten von P_i $(0 \le i \le r)$
(muß kein binärer Baum sein.)

2.1.9. Beispiel:



$P_0 = \{v_4, v_8\}$, $|P_0| = 2$
$P_1 = \{v_2, v_4, v_6, v_8\}$, $|P_1| = 4$
$P_2 = \{v_1 .. v_3\}$, $|P_2| = 9$

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

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

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

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

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:

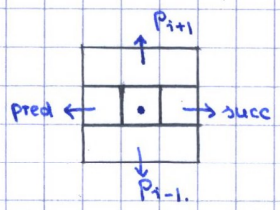
1). Wir berechnen eine Annäherung der Lösung für P_0.
Da P_0 maximal vier Ecken ⇒ in $O(1)$.

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:

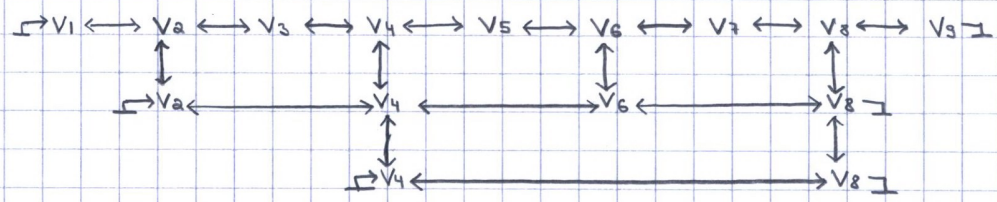
- P_i, $0 \leq i \leq k$ als doppelt verkettete Liste der Ecken und jede Ecke von P_i zeigt zusätzlich auf ihre Kopie in P_{i+1} und umgekehrt.



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

2.2.3 Beispiel:

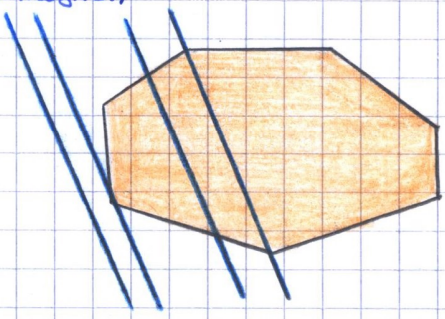
Sei P gegeben wie in 2.1.9.
⇒



2.2.4. 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! $P \cap g = \emptyset$.

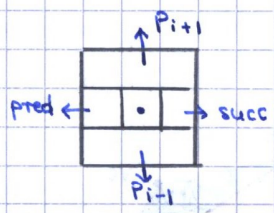


→ 2.2.4.2. Idee für Algorithmus:

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 $q_0 \in P_0$ mit minimalen Abstand zu g.

2.2.2. Darstellung im Rechner:

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

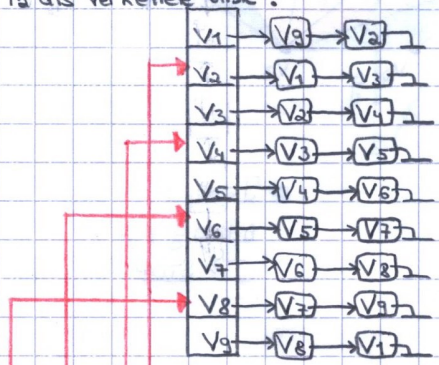


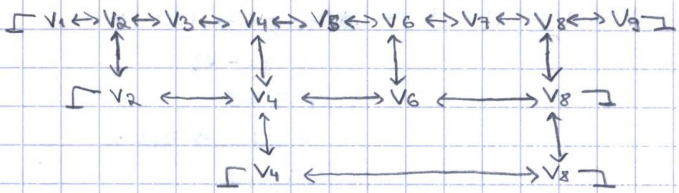
- explizite Speicherung des Kantenbaumes
  → Dynamisierung (Ecken einfügen / löschen)
  (wie bei Suchbäumen)

2.2.3 Beispiel:

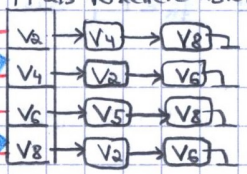
Sei P gegeben wie in 2.1.9.

P_0 als verkettete Liste:

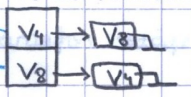




P_1 als verkettete Liste:



P_0 als verkettete Liste:



2.2.4. Anwendung: Schnitt mit einer Geraden:

→ 2.2.4.1. Ziel:

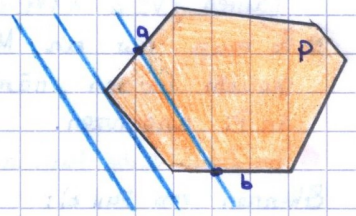
Geg: Hierarchische Darstellung $P_0 .. P_k$ eines konvexen Polygons P und eine Gerade g.

Ausgabe: $P \cap g$

Konvexität → $P \cap g$ = Strecke

→ 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!, Schnitt = $\emptyset$



2.2.4.2. Idee für Algorithmus:

1) Schnittest mit P_0:
   2 Fälle: a) $P_0 \cap g = \emptyset$ (konst Operationen, weil $|P_0| \le 4$)
   b) $P_0 \cap g \neq \emptyset$
   Ann: wir haben Fall a)
   → sei $g_0 \in P_0$ mit minimalem Abstand zu g
   Im Fall b → STOP

d) Durchlaufe die hierarchische Darstellung $P_1, P_2 \ldots$ und finde entweder:

a) $q_i \in P_i$ mit minimalen Abstand, falls $P_i \cap g = \emptyset$

oder

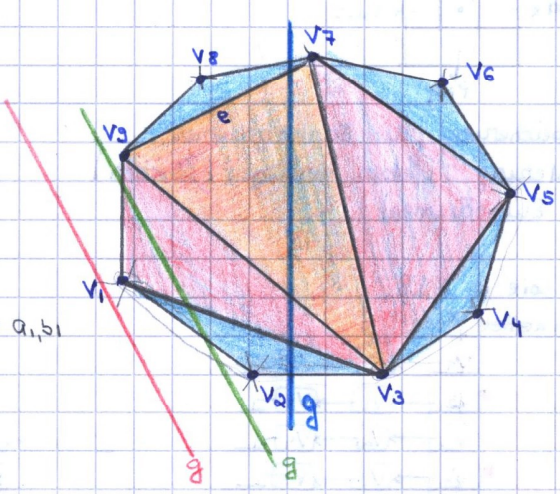
b) Schnittpkte a_i, b_i von P_i mit g.

Im Fall b) → STOP.

Bsp: $P = V_1 .. V_9 = P_2$

$P_1 = V_1 V_3 V_5 V_7 V_9$

$P_0 = V_2 V_7 V_9$



- Im Alg.:
  1) $P_0 \cap g = \emptyset$
     ⇒ $q_0 = V_9$
  2) $P_1 \cap g \neq \emptyset$
     ⇒ Fall b) ⇝ finde a_1, b_1
     STOP

- Im Alg.:
  1) $P_0 \cap g = \emptyset$
     ⇒ $q_0 = V_9$
  2) $P_1 \cap g \neq \emptyset$
     ⇒ Teil a) ⇒ $q_1 = V_1$
     $P_2 \cap g = \emptyset$ -
     ⇒ Teil a) ⇒ $q_2 = V_1$.

- Im Alg.:
  1) $P_0 \cap g \neq \emptyset$
     ⇒ STOP

Zwei mögliche Ausgänge: (wenn $i = k$)

a) $P_k \cap g = P_n \cap g = \emptyset$ und wir kennen Ecke $q_k \in P$, die g am nächsten

b) $P_i \cap g = (a_i, b_i)$ für $0 \leq i \leq k$

Im Bsp:

- ⇝ Ausgang b) ⇒ $P_1 \cap g = (a_1, b_1)$ für $i = 1$.
- ⇝ Ausgang a) ⇒ $P_2 \cap g = \emptyset$, $q_2 = V_1 \in P$.
- ⇝ Ausgang b) ⇒ $P_0 \cap g = (a_0, b_0)$ für $i = 0$.

Ausgang a) ⇒ fertig, $P \cap g = \emptyset$

Ausgang b) ⇒ Wir müssen Schnittpkte (a_i, b_i) später verfeinern bis (a_k, b_k) } Phase 2.
gefunden.

Nun: Übergang von $q_i \to q_{i+1}$ (Fall a)) bzw. Test, ob Fall b) gilt.

2.2.4.3 Laufzeit:

Phase 1: Kandidaten für nächsten Pkt q_{i+1} sind schon genau die Ecken von P_{i+1} die Verfeinerungen der Nachbarkanten von q_i in P_i

Das folgt unmittelbar aus der Konvexität.

- Im Bsp: $i = 0$, $q_0 = V_9$, Kand. für q_1 ist Ecke von P_1 nämlich V_1
  Dies ist die Ecke der Verfeinerungen der Nachbarkanten von $q_0 = V_9$, nämlich der Kanten $(V_9, V_1) \wedge (V_1, V_3)$

⇒ Es muß nur ein Minimum in einer konstanten Zahl von Kandidaten gesucht werden, nämlich in den Verfeinerungen der Kanten, die an q_i in P_i angrenzen.

- Im Bsp: Suche Min. in (V_9, V_1), (V_1, V_3)

Erkennen von Fall b):

Teste in der Kandidatenmenge, ob ein Pkt auf der anderen Seite von g liegt. (Orientation)

⇒ (a_{i+1}, b_{i+1}) in Konst. Zeit denn Konst viele Kand.

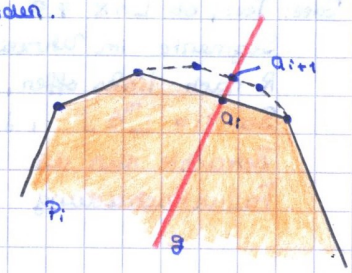
(Schnittberechnung mit Verfeinerungskanten).

- Im Bsp: $i = 0$, $q_0 = V_9$ ⇒ Kandidatenmenge: (V_9, V_1) und (V_1, V_3)
  V_1 liegt auf der anderen Seite von g, denn $\text{orient}(V_9, V_1, g) \geq 0$
  ⇒ Teil b) erkannt
  (bez.: $\text{orient}(V_9, V_3, g) \leq 0$)

Laufzeit: • $O(1)$ pro Hierarchiestufe
• Haben nur $O(\log n)$ Stufen
$\Rightarrow O(\log n)$ für die erste Phase.

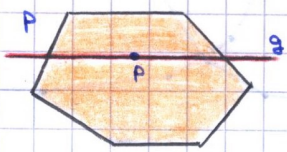
Nach Phase 1 haben wir zwei Fälle:
Fall a) $\Rightarrow$ fertig.
Fall b) $\Rightarrow$ machen weiter zu Phase 2.

Phase 2: Geg: $\{a_i, b_i\} = P \cap g$ für $i < k$ (sonst fertig) und es sind keine Ecken (sonst ebenfalls fertig).
Betrachte a_i (b_i analog):
a_{i+1} muß Schnittpkt sein mit einer Verfeinerungskante von e, wobei $a_i = e \cap g$
Im Bsp: Betrachte a_0, $i=0<2$
a_1 ist Schnpkt mit Verfeinerungskante von e, wobei $a_0 = e \cap g$ und der Geraden g.
Die Verfeinerungskanten von e sind (v_9, v_8) und (v_8, v_7)
$\Rightarrow a_{i+1}$ kann in Zeit $O(1)$ aus a_i berechnet werden.
(b_{i+1} analog aus b_i)
Weil # Verf.kanten konst. Genauer ≤ 4.
$\Rightarrow$ Laufzeit von Phase 2 auch $O(\log n)$.
$\Rightarrow$ Gesamte Laufzeit also $O(\log n)$!



2.2.5 Satz: Für ein konvexes Polygon P kann:
1) die Hierarchische Darstellung in Zeit $O(n)$ aufgebaut werden.
2) Braucht Platz $O(n)$
3) für Beliebige Gerade g kann $P \cap g$ mit Hilfe der Hierarchischen Darstellung in Zeit $O(\log n)$ berechnet werden.

2.2.6 Bem.
1) Teil 3) $\Rightarrow$ Alternativer $\log n$-Alg. zum Test, ob Pkt $p \in P$ ist.
Idee:



$p \in P \Leftrightarrow p \in P \cap g$
• Betr. Gerade g durch p parallel zu x-Achse
• Berechne hieras. Darst.
• Berechne Schnpkte
• Schaue ob $p_x \in [a_x, b_x]$
$\Rightarrow$ Zeit: $O(1) + O(n) + O(\log n) + O(1) = O(n)$!

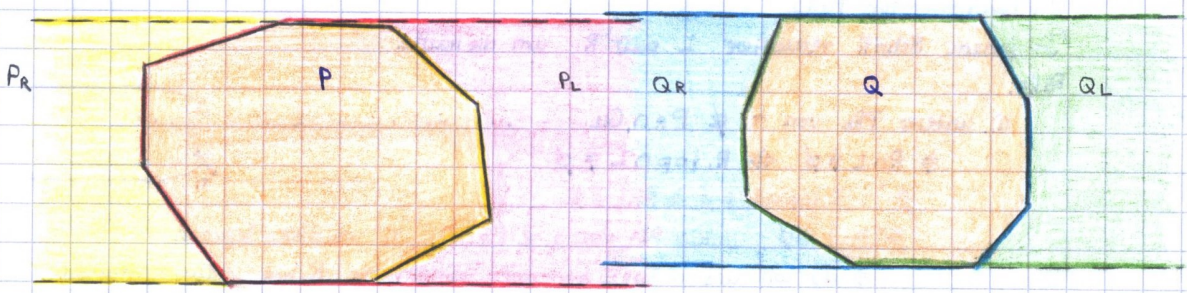
2) Hieras. Darstellung auch im $\mathbb{R}^3$ möglich, d.h. für konvexe Polyeder.
(dazu später mehr...).

2.3 Weiteres Problem auf konvexen Polygonen: Schnitt von zwei konvexen Polygonen.

2.3.1 Ziel: Geg: zwei konvexe Polygone P, Q
Test ob $P \cap Q \neq \emptyset$

Anm. Wir wissen schon, wie man $P \cap Q$ in Zeit $O(n)$ berechnet, wobei n = Gesamtzahl der Ecken.

2.3.2 Idee:
1) Betrachte ein einfacheres Problem: Test, ob sich zwei polygonale Ketten schneiden.
• Zerlege P in linken (P_L) & rechten (P_R) Rand durch Zerschneiden an Extremalpkten gemäß yx-Sortierung der Ecken.
• Erweitere durch entsprechende horizontale Strahlen.



Beobachtung: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset$

Beweis: $P \cap Q \neq \emptyset \Rightarrow P_R \cap Q_L \neq \emptyset \wedge P_L \cap Q_R \neq \emptyset$ klar.

Für 

mit $P_R \cap Q_L \neq \emptyset \Leftrightarrow P \cap Q \neq \emptyset$
(Hierbei gilt auch $P_L \cap Q_R \neq \emptyset$)

Für 

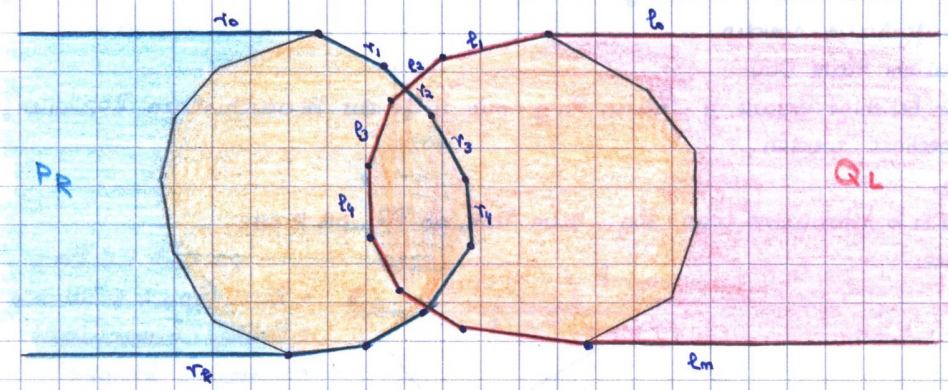
mit $Q_R \cap P_L \neq \emptyset \Leftrightarrow P \cap Q \neq \emptyset$
(Hierbei gilt auch $Q_L \cap P_R \neq \emptyset$)

$\Rightarrow$ falls $P_R \cap Q_L \neq \emptyset \wedge P_L \cap Q_R \neq \emptyset \Rightarrow P \cap Q \neq \emptyset$

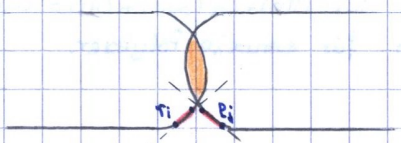
2). Lösung des einfacheren Problems:

Bea: Wir wissen nicht, ob P linkes und Q rechtes Polygon oder umgekehrt.

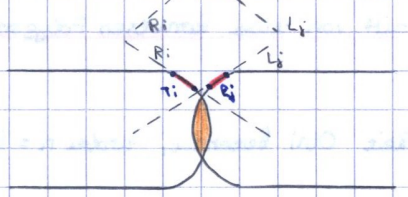
Idee: Test, ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$, wobei $R = r_0 .. r_k$ gegeben durch die Segmente im Uhrzeigersinn (r_0, r_k Strahlen) und $L = \ell_0 .. \ell_m$ analog gg. Uhrzeigersinn.
R nach links offen, L nach rechts offen.
Sei $i := \lfloor m/2 \rfloor$, $j := \lfloor k/2 \rfloor$
Betrachte nun die mittleren Segmente r_i und ℓ_j.
Fallunterscheidung gemäß der Lage von ℓ_j und r_i auf Geraden R_i und L_j.



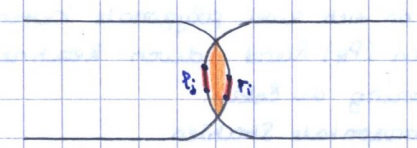
Es gibt nun mehrere Fälle:

1) 

← $P_R \cap Q_L$ oberhalb von den unteren Endpkten von r_i und ℓ_j

2) 

← $P_R \cap Q_L$ unterhalb von den oberen Endpkten von r_i und ℓ_j

3) 

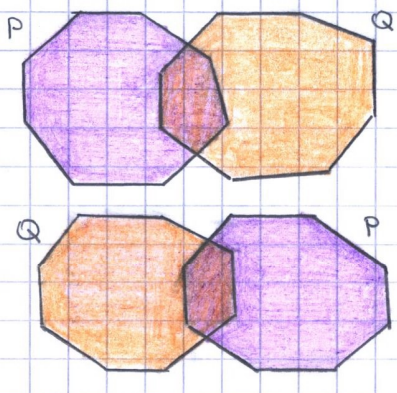
← Endpkte von r_i und $\ell_j \in P_R \cap Q_L$
(Bea: Hierbei müssen sich die Polygone nicht notw. schneiden!)

Wir betrachten hier Fall 1
In jedem Schritt reduziere L oder R um die Hälfte.
Fälle:

a). unterer Pkt von $r_i \notin P_R \cap Q_L$
$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$
„$\Leftarrow$": $R.top \cap L \neq \emptyset \Rightarrow R \cap L \neq \emptyset$
„$\Rightarrow$" $P \cap L \neq \emptyset$, unterer Pkt von $r_i \notin P_R \cap Q_L$, r_i mittleres Segment
$\Rightarrow$ da wir uns im Fall 1 befinden, können wir R.bot rausschmeissen.
$\Rightarrow R.top \cap L \neq \emptyset$.

Beobachtung: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset$.

Denn:



Hier: $P \cap Q \neq \emptyset \Leftrightarrow P_R \cap Q_L \neq \emptyset$

Bea: Man weiß in der Regel nicht, welches Polygon rechts und welches links liegt.

Hier: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset$

$\Rightarrow$ Insgesamt gilt also: falls $P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset \Rightarrow P \cap Q \neq \emptyset$

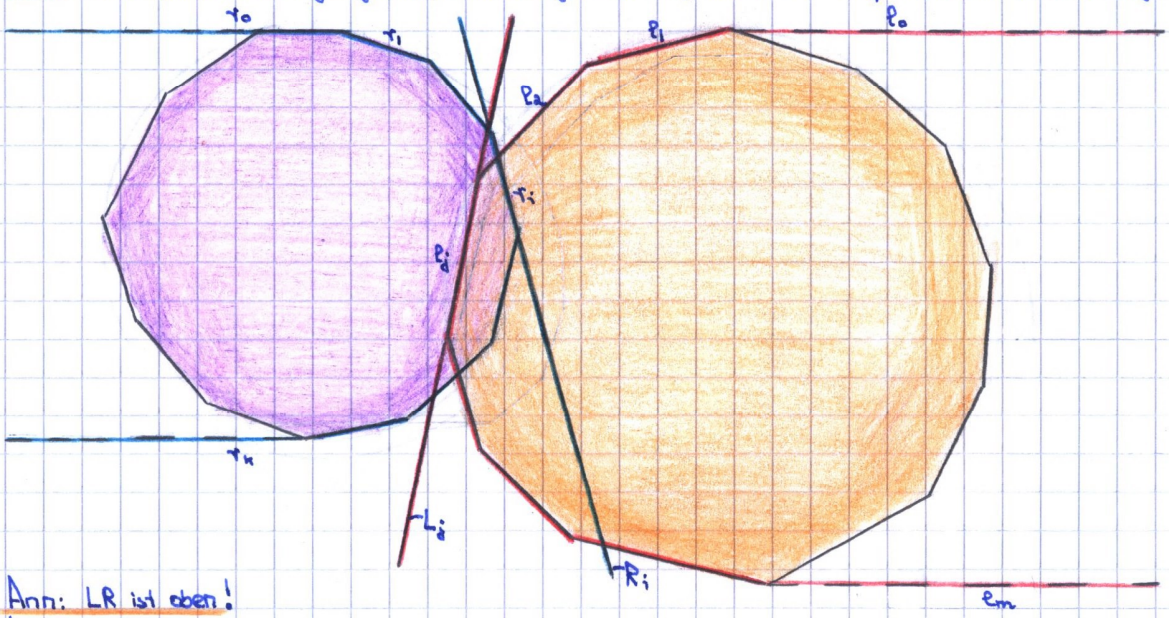
„$\Leftarrow$" klar

$\Rightarrow$ die obige Beh.

2) Lösung des einfacheren Problems:

Idee: Test, ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$, wobei $R = \{r_0 \dots r_k\}$ und $L = \{\ell_0 \dots \ell_m\}$ gegeben durch die Segmente im Uhrzeigersinn (für R) und gegen Uhrzeigersinn (für L).
(r_0 und r_k bzw ℓ_0 und ℓ_m sind Strahlen).
R ist nach links offen und L nach rechts offen.
Sei $i := \lfloor m/2 \rfloor$ und $j := \lfloor k/2 \rfloor$
Betrachte nun die mittleren Segmente r_i und ℓ_j

Nun: Fallunterscheidung gemäß der Lage von ℓ_j und r_i auf Geraden R_i und L_j:

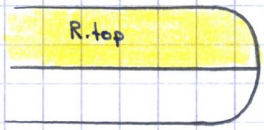


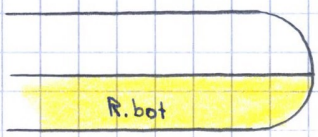
Ann: LR ist oben!

In jedem Schritt reduziere L oder R um die Hälfte.

Fälle:

a) unterer Pkt von r_i ist nicht in LR
$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$
wobei:





ß). unterer Pkt von ℓ_j ist nicht in LR.
$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.L.top \neq \emptyset$

c) beide unteren Pkte sind in LR
(i) falls u_{r_i} unterhalb von u_{ℓ_j}
$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$
(ii) falls u_{ℓ_j} unterhalb von u_{r_i}
$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R \cap L.top \neq \emptyset$

Hierbei: $u_{r_i} \hat{=}$ unterer Endpkt von r_i
$u_{\ell_j} \hat{=}$ unterer Endpkt von ℓ_j.