5.3. Priority - Search - Tree:

5.3.1. Def: Sei $S \subset \mathbb{R}^2$

Ein Priority - Search - Tree (PST) T ist primärorientierter Suchbaum nach den x-Koord. der Pkte aus S.

5.3.2. Speichern der Pkte:

Ann: paarw. verschiedene x-Koordinaten.

$p \in S$ wird in einem Knoten auf Suchpfad nach p_x gespeichert und zwar so, dass T bzgl. der y-Koord. einen Maximumsheap bildet.

⇒ Auf jedem Pfad werden y-Koord. kleiner

⇒ Wurzel enthält Knoten mit max. y-Koord.

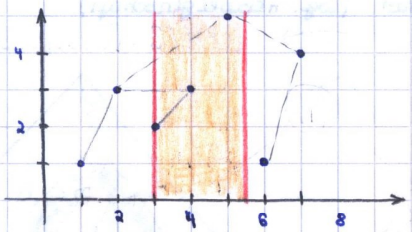
PST hat zwei Eigenschaften:
- Suchbaum nach x-Koord. der Pkte
- Heap nach y-Koord. der Pkte.

Aufbau (rekursiv):

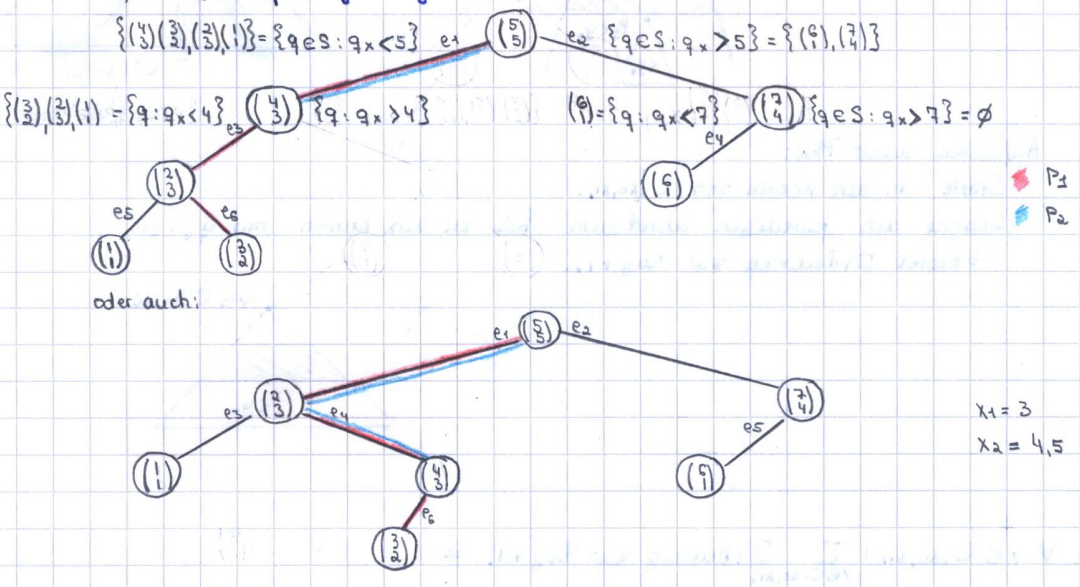
Wurzel w speichert $p \in S$ mit max. y-Koord.
- linker Unterbaum T_ℓ von w ist PST für $\{q \in S : q_x < p_x\}$
- rechter Unterbaum T_r von w ist PST für $\{q \in S : q_x > p_x\}$.

5.3.3. Beispiel:

$S = \{(1),(\frac{3}{2}),(\frac{2}{3}),(\frac{4}{3}),(\frac{6}{1}),(\frac{5}{5}),(\frac{7}{4})\}$.



Im Streifen: $(\frac{3}{2}),(\frac{4}{3}),(\frac{5}{5})$



oder auch:

$x_1 = 3$

$x_2 = 4{,}5$

5.3.4. Problem1 und Lösung:

Wo liegen alle Pkte aus einem vertikalen Streifen?

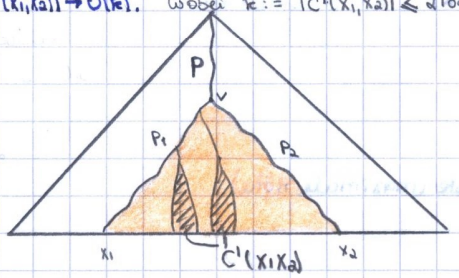
Antwort: auf P_1, P_2, P und allen Knoten zwischen P_1 und P_2!

Achtung: Auf den Pfaden P, P_1, P_2 können auch andere Pkte liegen.

Berechnung der Pkte:
- Filtere die Pkte auf Pfaden nach $[x_1, x_2]$ in Zeit $2 \cdot \log n + k$.
- Gib alle Pkte, die in Knoten zwischen P_1, P_2 gespeichert sind, aus $|C'(x_1,x_2)| \to O(\tilde{k})$. wobei $\tilde{k} := |C'(x_1,x_2)| \leq 2 \log n$

$2(\text{Höhe}(v) + 1 + O(k_1) + 1 + O(k_2) + \dots) =$
$= 2 \cdot (\text{Höhe}(v) + O(k_1) + O(k_2) + \sum 1) =$
$= 2 \cdot (\log n + \tilde{k})$

$\tilde{k}$:= # Knoten in orange.



Im Bsp: 1 Bild ⇒ $\{(\frac{5}{5}),(\frac{4}{3}),(\frac{2}{3}),(\frac{3}{2})\} \supset$ gesuchte Menge

2 Bild ⇒ $\{(\frac{5}{5}),(\frac{2}{3}),(\frac{4}{3}),(\frac{3}{2})\} \supset$ gesuchte Menge

Platz: $O(n)$!

Gesamtlaufzeit: $2\log n + k + O(\log n + \tilde{k}) =$
$= O(\log n + K)$

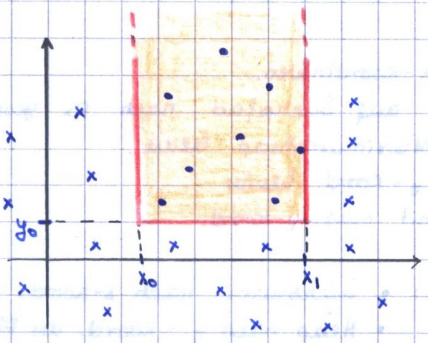
K = # Ausgaben. $(= k + \tilde{k})$

5.3.5 Problem 2 + Lösung:

Wozu Heap-Eigenschaft nach y-Koordinaten?
→ 1½-dimensionale Range Abfragen
(Halb offene oder 3 seitige).

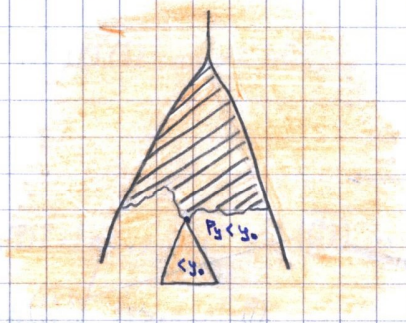
Geg: $x_0, x_1, y_0, \ldots, S \subset \mathbb{R}^2$

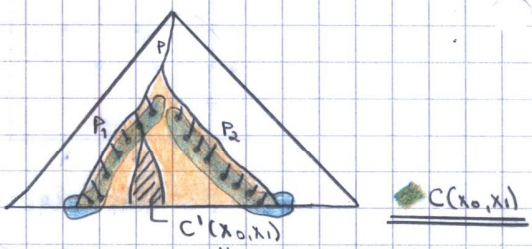
Ges: alle $p \in S$ mit
$x_0 \leq p_x \leq x_1$ und
$p_y \geq y_0$



Lösung:

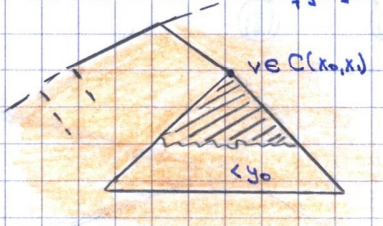
- Filtern der Pfade: $\forall$ Knoten $v \in P_1 \cup P_2 \cup P$ gib enthaltenen Pkt p aus, falls $x_0 \leq p_x \leq x_2 \wedge p_y \geq y_0$
  → Kostet Zeit $O(\log n)$. Mit Ausgabe kostet dies $O(\log n + k)$
- Rest der Ausgabe (normalerweise müssen alle $v \in P_1 \cup P_2 \cup P$ und zwischen P_1 und P_2 betrachtet werden. Oben nur $v \in P_1 \cup P_2 \cup P$, hier also die restlichen, dh. also: v zwischen P_1 und P_2) ist im oberen Bereich von $C'(x_0, x_1)$ gespeichert (wg. Heapeigenschaft)





{"alle kleinen Bäume"}

Auflisten dieser Pkte:
Starte in den Knoten von $C(x_0, x_1)$.
Scanne den jeweiligen Unterbaum bis zu den Knoten mit $p_y < y_0$
→ Kostet O(Beitrag zur Ausg + 1).



Laufzeit: $\forall v \in C(x_0, x_1)$: $\sum_{v \in C(x_0, x_1)} O$(Beitrag zur Ausg + 1) =

$= O(\log n + K')$, wobei K' = Beitrag zur Gesamtlösung von $C'(x_0 x_1)$

$\text{Höhe}(v) + 1 + O(\text{Beitr. zu Ausg}(v_1) + 1) + 1 + O(\ldots) + \ldots + \text{Höhe}(v) + 1 + O(\ldots) + \ldots = 2 \cdot \log n + \underbrace{\sum_{v \in C(x_0, x_1)} O(\text{Beitr. zu Ausg} + 1)}_{=: K'} = O(\log n + K')$

5.3.6. Satz (Zusammenfassung):

Der Priority-Search Tree verwaltet eine Menge von n Pkten im $\mathbb{R}^2$ unter folgenden Operationen und Laufzeiten:

- Insert / Delete: $O(\log n)$
- Drei-seitige Bereichsabfrage: $O(\log n + K)$, K = # Ausgaben.

und benötigt Speicherplatz: $O(n)$.

⇒ Gesamtlaufzeit: $a \log n + k + O(\log n + K') = \underline{O(\log n + \tilde{K})}$
$\tilde{K} = k + K'$, # Ausgaben.

5.3.7 Anwendung:

feste (maximale Höhe).

↑ Range-Abfragen mit PST (→ Übung).

5.4. Das Maßproblem für achsenparallele Rechtecke.

5.4.1. Problem:

Geg: n achsenparallele Rechtecke $R_1 .. R_n$.

Ges: Fläche von $\dot{\bigcup_{i=1}} R_i$

→ Anwendung von Plane Sweep und einer vereinfachten Form von Segmentbäumen.

5.4.2. Idee:

Schnitt von SL mit Vereinigung.

→ Menge von disjunkten vertikalen Segmenten.

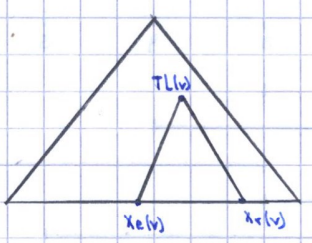
Sei V Menge dieser Segmente.

- an Event-Pkten (linker / rechter Rand von Rechtecken) kann sich V ändern.
- eigentlich brauchen wir nur L = Summe der Längen aller Segmente in V.

Dann können wir die gesuchte Fläche A durch Multiplizieren des aktuellen L-Wertes mit der zw. zwei Events zurückgelegten Distanz und Aufsummieren auf A berechnen.

Zur Verwaltung der Segmente und der Länge L verwenden wir einen Segmentbaum T. Segmente $s_i = R_i \cap SL$.

Sei $TL(v)$ die Länge der Vereinigung aller Segmente geschnitten mit $x_{range}(v)$, wobei $x_{range}(v) := [x_e(v), x_r(v)]$ mit $x_e(v)$ x-Koord. des linken Blattes im UB und $x_r(v)$ die x-Koord. des rechten Blattes im UB.



Dann gilt:

1. $L = TL(v)$
2. $$TL(v) = \begin{cases} x_r(v) - x_e(v), & NL(v) \neq \emptyset, \ v \text{ kein Blatt} \\ 0, & v \text{ Blatt} \\ TL(v_e) + TL(v_r), & NL(v) = \emptyset, \ v \text{ kein Blatt}, \ NL(u_r) \cap NL(u_e) = \emptyset \\ TL(v_e) + TL(v_r) + 1, & NL(v) = \emptyset, \ v \text{ kein Blatt}, \ NL(u_r) \cap NL(u_e) \neq \emptyset \end{cases}$$

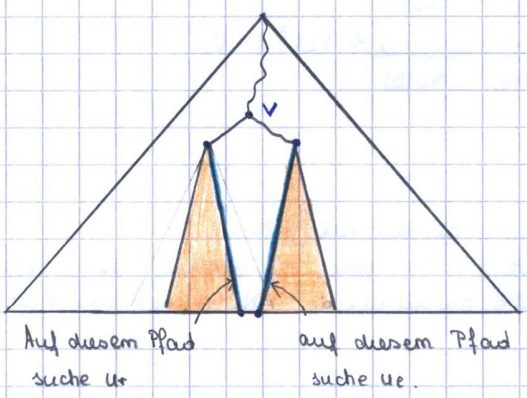
wobei $NL(u_r)$ wie folgt definiert ist:

Betrachte Pfad von v nach dem rechtesten Blatt von dem UB von dem linken Sohn von v. $\forall u \in$ diesem Pfad kontrolliere jeweils $NL(u)$ angefangen mit dem linken Sohn von v.

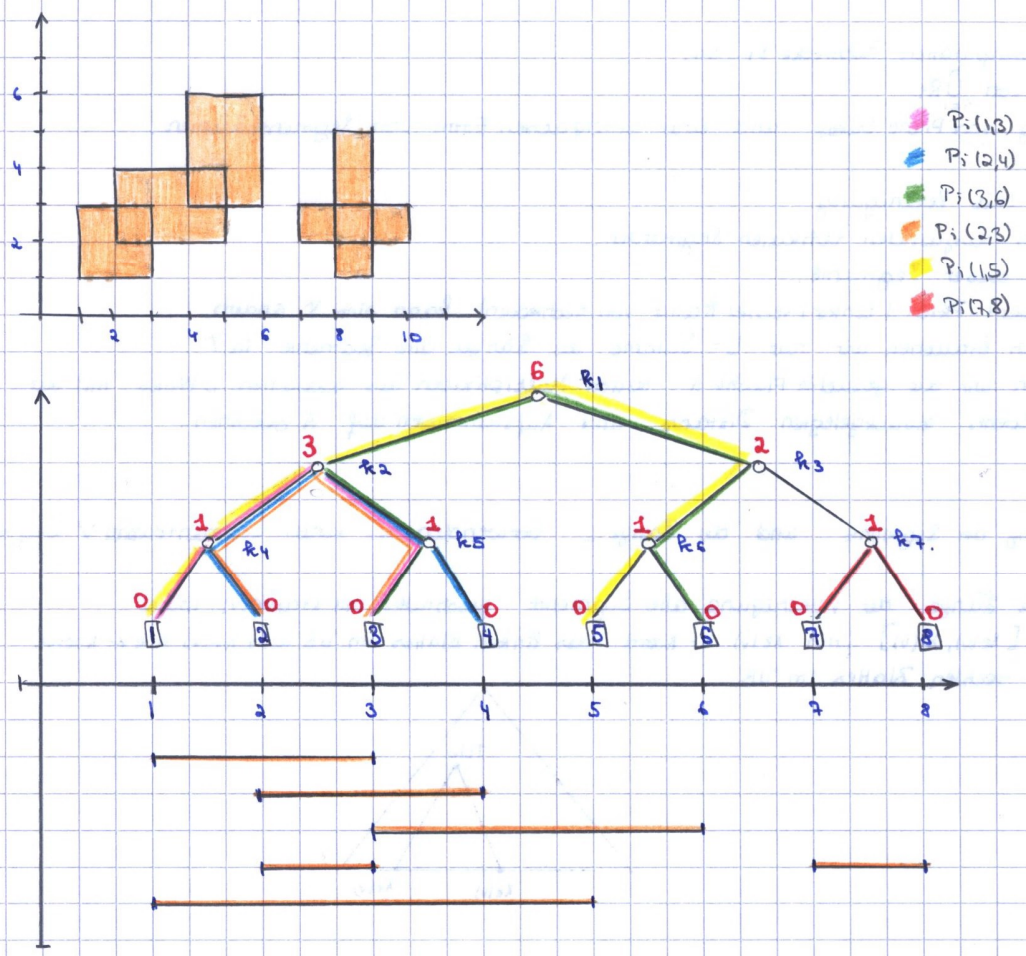
$NL(u_r) := NL(u)$, wobei u der erste Knoten auf dem Pfad, welches erfüllt: $NL(u) \neq \emptyset$

Falls $\forall u \in$ Pfad gilt: $NL(u) = \emptyset \Rightarrow NL(u_r) :=$ NL(rechtestes Blatt von dem linken UB von v) (darf dann auch leer sein).

$NL(u_e)$ analog



5.4.3. Beispiel:



$C(1,3) = \{1,2,3\}$
$C(2,4) = \{2,3,4\}$
$C(3,6) = \{3,4,5,6\}$
$C(2,3) = \{2,3\}$
$C(1,5) = \{1,2,k_5,5\}$
$C(7,8) = \{7,8\}$

$NL(1) = \{(1,3),(1,5)\}$
$NL(2) = \{(1,3),(2,4),(2,3),(1,5)\}$
$NL(3) = \{(1,3),(2,4),(3,6),(2,3)\}$
$NL(4) = \{(2,4),(3,6)\}$
$NL(5) = \{(1,5),(3,6)\}$
$NL(6) = \{(3,6)\}$

$NL(7) = \{(7,8)\}$
$NL(8) = \{(7,8)\}$
$NL(k_5) = \{(1,5)\}$
$NL(k_1 .. k_4) = \emptyset$

k_4: $(1,3) \in NL(1) \cap NL(2) \Rightarrow TL(k_4) = 0+0+1 = 1$
k_5: $(2,4) \in NL(3) \cap NL(4) \Rightarrow TL(k_5) = 1$
k_6: $(3,6) \in NL(5) \cap NL(6) \Rightarrow TL(k_6) = 1$
k_7: $(7,8) \in NL(7) \cap NL(8) \Rightarrow TL(k_7) = 1$
k_2: $(1,5) \in NL(k_5) \cap \underbrace{NL(u_1)}_{NL(2)} \Rightarrow TL(k_2) = 1+1+1 = 3$
k_3: $\emptyset = \underbrace{NL(u_1)}_{NL(6)} \cap \underbrace{NL(u_2)}_{NL(7)} \Rightarrow TL(k_3) = 2$
k_1: $(1,5) \in \underbrace{NL(u_1)}_{NL(k_6)} \cap \underbrace{NL(u_2)}_{NL(5)} \Rightarrow TL(k_1) = 6.$

5.4.4. Beobachtung:

$NL(v)$ zu speichern nimmt viel Platz
⇒ Idee: definiere $TL(v)$ ohne die genaue Menge von $NL(v)$ zu kennen und speichere nur die Kardinalität $|NL(v)| =: count(v)$.

5.4.5. Genauere Betrachtung der Aktionen:

a) linker Rand von R_i:

1) Füge $s_i = R_i$ in SL ein.
   statt einfügen von NL(v): count(v)++;
   $\forall$ Knoten $\in C(s_i)$, die schon vorher einen Zähler count(v) > 0 hatten, brauchen wir nichts zu tun.
2) Falls count(v) von 0 auf 1 erhöht wird, dann:
   - $TL(v) \leftarrow x_r(v) - x_e(v)$
   - Laufe von v nach oben und korrigiere die TL-Werte.

→ Laufzeit: $O(\log n)$, da wir Suchpfade von $C(s)$ zweimal durchlaufen.
(Laufzeit f. Insert)

b) rechter Rand von R_i:

1) entferne $s_i = R_i$ in SL aus Baum: count(v) − −; $\forall v \in C(s_i)$
2) Falls nun count(v) = 0, dann müssen TL-Werte von v und seinen Vorfahren geändert werden.

→ Laufzeit: $O(\log n)$ (s. Insert)

c) Bei jedem Event-Pkt x: $A \leftarrow A + TL(\text{wurzel}) \cdot (x - x_{old})$.

5.4.6. Satz: Das Maß-problem von n achsenparallelen Rechtecken kann in Zeit $O(n \log n)$ und Platz $O(n)$ gelöst werden.

Beweis: 1). Platzbedarf ist $O(n)$, da keine Knotenlisten gespeichert werden, sondern in jedem Knoten zwei Zahlen count(v) und TL(v).

2). Rechtecke sortieren (nach linkem / rechten Rand) nach x-Koord. also
→ x-Struktur: $O(n \log n)$
Pro Event Zeit $O(\log n)$ für Operationen im Segmentbaum.

In unserem Bsp:

Kapitel VI: Drei-dimensionale konvexe Hüllen.

6.1. Einführung:

6.1.1. Problem:

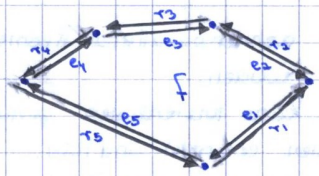
Geg: Menge S von n Pkten im $\mathbb{R}^3$, $p \in S$, $p = (x,y,z)$ (kart. Koord.)
Ges: CH(S) (= kleinste konvexe Menge, die S enthält).
Analog zum Gummiband-Modell: Gummi-Fläche
$\Rightarrow$ CH(S) ist ein konvexes Polyeder.
Ausgabe: Oberfläche des Polyeders.

- besteht aus Ecken, Kanten und Flächen (Euler-Formel)
- kann beschrieben werden durch einen planaren Graphen.

6.1.2. Darstellung des (planaren) Oberflächengraphen:

als zweifach gerichteten (bidirected) Graphen.

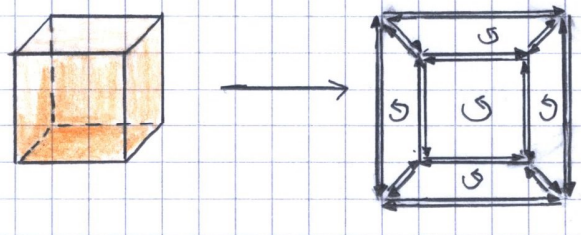
1. Für jeden Knoten (Ecke) v ordnen wir die ausgehenden Kanten gg. den Uhrzeigersinn (bei Ansicht von außen)
2. Jede Kante $e = (v,w)$ hat einen Verweis auf ihre Gegenkante $r = (w,v) =: rev(e)$
3. Die Flächen sind dann implizit definiert
   e_2 = Vorgänger (rev(e)) (in der Adjazenzliste)
   Das definiert einen Kantenzyklus für jede Fläche f.



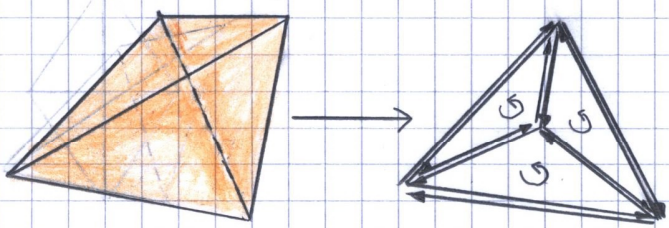
e_2 = G.face_cycle_source(ei)
G.cyclic_pred(G.reversal(ei))

4. In den CH-Algorithmen werden zunächst alle Flächen Dreiecke sein.
   Bei allg. eingebetteten planaren Graphen ist u.U. die äußere Fläche kein Dreieck.

6.1.3. Beispiel:



← nach außen gegen den Uhrzeigersinn bzw. !!



6.1.4. Geometrische Prädikate:

Für $a,b,c,d \in \mathbb{R}^3$:

$$\text{orientation}(a,b,c,d) := \text{sign det} \begin{pmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{pmatrix}$$

Diese Determinante heißt Spatprodukt.
= Vielfaches des Volumens (mit Vorzeichen) des Simplex (a,b,c,d)
d.h. orientation(a,b,c,d) sagt, auf welcher Seite der Ebene durch a,b,c (a,b,c nicht collinear) der Pkt d liegt.

→