

26. Geometrische Algorithmen

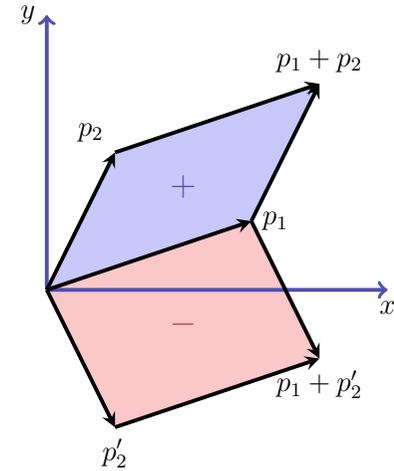
Lage von Strecken, Schnitt vieler Strecken, Konvexe Hülle, Dichtestes Punktepaar [Ottman/Widmayer, Kap. 8.2,8.3,8.8.2, Cormen et al, Kap. 33]

Eigenschaften von Strecken

Kreuzprodukt zweier Vektoren $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ in der Ebene

$$p_1 \times p_2 = \det \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = x_1 y_2 - x_2 y_1$$

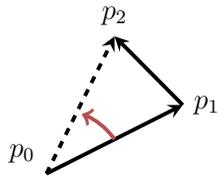
Vorzeichenbehafteter Flächeninhalt des Parallelogramms



746

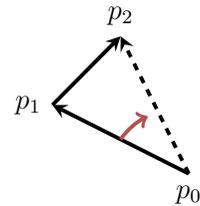
747

Abbiegerichtung



nach links:

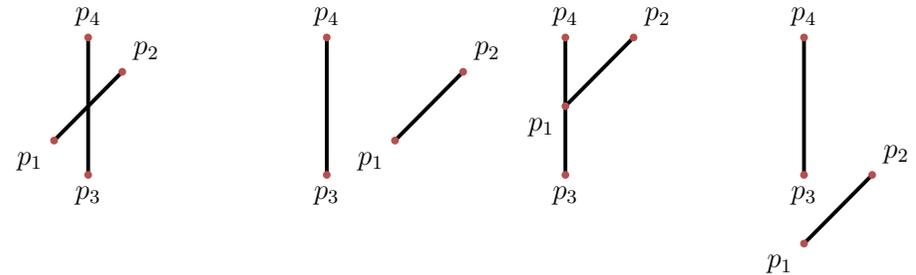
$$(p_1 - p_0) \times (p_2 - p_0) > 0$$



nach rechts:

$$(p_1 - p_0) \times (p_2 - p_0) < 0$$

Schnitt zweier Strecken



Schnitt: p_1 und p_2 gegenüber bzgl. $\overline{p_3 p_4}$ und p_3, p_4 gegenüber bzgl. $\overline{p_1 p_2}$

Kein Schnitt: p_1 und p_2 auf der gleichen Seite von $\overline{p_3 p_4}$

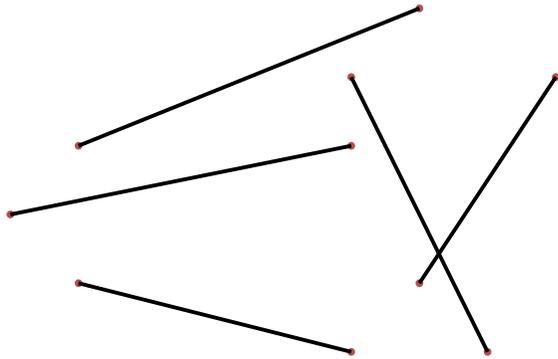
Schnitt: p_1 auf $\overline{p_3 p_4}$

Kein Schnitt: p_3 und p_4 auf der gleichen Seite von $\overline{p_1 p_2}$

748

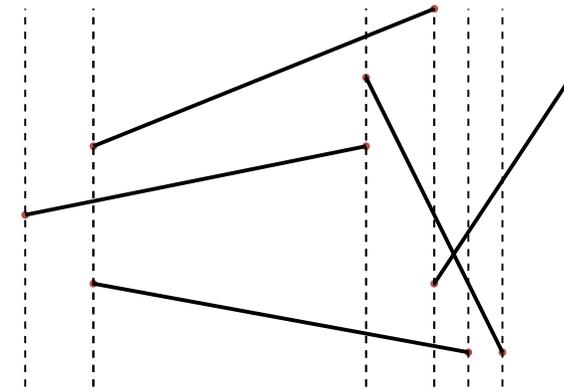
749

Schnittpunkt vieler Strecken



750

Sweepline Prinzip

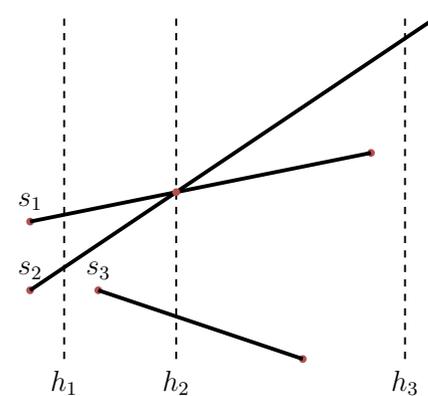


751

Vereinfachende Annahmen

- Keine Strecke verläuft senkrecht
- Jeder Schnittpunkt wird von maximal zwei Strecken gebildet.

Anordnen von Strecken



Quasiordnung (Halbordnung ohne Antisymmetrie)

$$s_2 \preceq_{h_1} s_1$$

$$s_1 \preceq_{h_2} s_2$$

$$s_2 \preceq_{h_2} s_1$$

$$s_3 \preceq_{h_2} s_2$$

Bzgl. h_3 sind die Strecken unvergleichbar.

752

753

Sweep-Line bewegen

- **Sweep-Line Status** : Beziehung der durch Sweep-Line geschnittenen Objekte
- **Ereignisliste** : Folge von Ereignispunkten, nach x -Koordinate geordnet. Sweepline wandert von links nach rechts und hält an jedem Ereignispunkt.

Sweep-Line Status

Vorordnung T der geschnittenen Strecken Benötigte Operationen:

- **Insert(T, s)** Füge Strecke s in T ein
- **Delete(T, s)** Entferne s von T
- **Above(T, s)** Rückgabe Strecke unmittelbar oberhalb von s in T
- **Below(T, s)** Rückgabe Strecke unmittelbar unterhalb von s in T

Mögliche Implementation: Balancierter Baum (AVL-Baum, Rot-Schwarz Baum etc.)

754

755

Algorithmus Any-Segments-Intersect(S)

Input : Liste von Strecken S

Output : Rückgabe ob S schneidende Strecken enthält

$T \leftarrow \emptyset$

Sortiere Endpunkte der Strecken in S von links nach rechts (links vor rechts und unten vor oben)

for Sortierte Endpunkte p **do**

if p linker Endpunkt einer Strecke s **then**

 Insert(T, s)

if Above(T, s) $\cap s \neq \emptyset \vee$ Below(T, s) $\cap s \neq \emptyset$ **then return true**

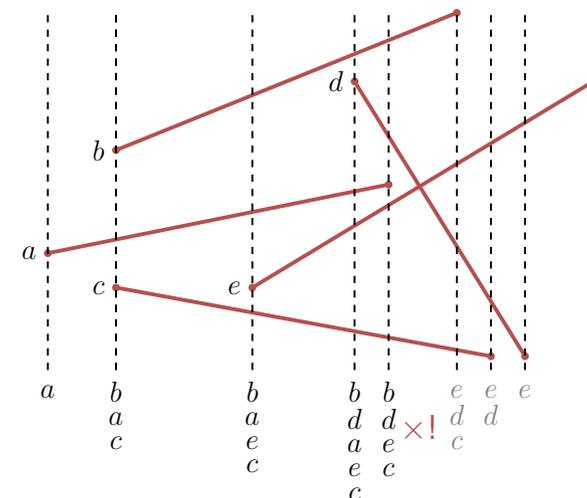
if p rechter Endpunkt einer Strecke s **then**

if Above(T, s) \cap Below(T, s) $\neq \emptyset$ **then return true**

 Delete(T, s)

return false;

Illustration



756

757

Analyse

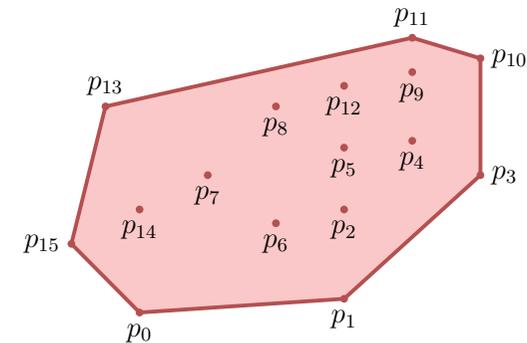
Laufzeit des Algorithmus Any-Segments-Intersect

- Sortieren $\mathcal{O}(n \log n)$
- n Iterationen der For-Schleife. Jede Operation auf dem balancierten Baum $\mathcal{O}(\log n)$

Insgesamt $\mathcal{O}(n \log n)$

Konvexe Hülle

Konvexe Hülle $CH(Q)$ einer Menge Q von Punkten: kleinstes konvexes Polygon P , so dass jeder Punkt entweder auf dem Rand oder im Inneren liegt.



758

759

Algorithmus Graham-Scan

Input : Menge von Punkten Q

Output : Stack S von Punkten der konvexen Hülle von Q

p_0 : Punkt mit minimaler y - (gegebenenfalls zusätzlich minimaler x -) Koordinate
 (p_1, \dots, p_m) restlichen Punkte sortiert nach Polarwinkel gegen Uhrzeigersinn relativ zu p_0 ; Wenn Punkte mit gleichem Polarwinkel vorhanden, verwirfe alle ausser dem mit maximalen Abstand von p_0

$S \leftarrow \emptyset$

if $m < 2$ **then return** S

Push(S, p_0); Push(S, p_1); Push(S, p_2)

for $i \leftarrow 3$ **to** m **do**

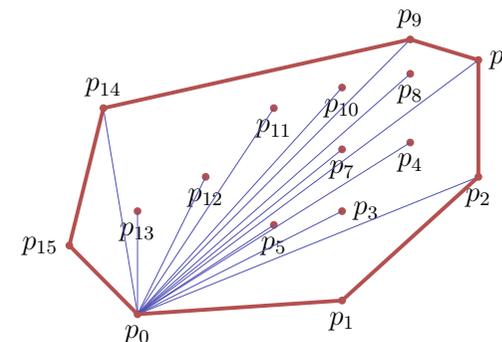
while Winkel (NextToTop(S), Top(S), p_i) nicht nach links gerichtet **do**

 Pop(S);

 Push(S, p_i)

return S

Illustration Graham-Scan



Stack:

p_{15}

p_{14}

p_9

p_6

p_2

p_1

p_0

760

761

Analyse

Laufzeit des Algorithmus Graham-Scan

- Sortieren $\mathcal{O}(n \log n)$
- n Iterationen der For-Schleife
- Amortisierte Analyse des Multipop beim Stapel: amortisiert konstante Laufzeit des Multipop, ebenso hier: amortisiert konstante Laufzeit der While-Schleife.

Insgesamt $\mathcal{O}(n \log n)$

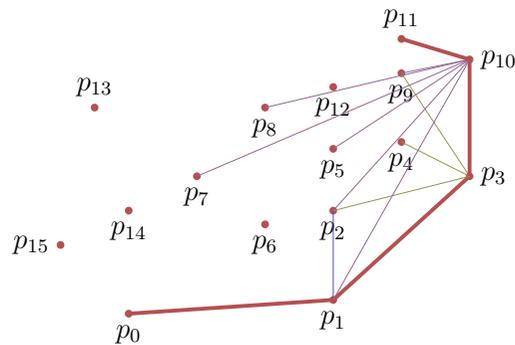
Jarvis Marsch / Gift Wrapping Algorithmus

- 1 Starte mit Extrempunkt (z.B. unterster Punkt) $p = p_0$
- 2 Suche Punkt q , so dass \overline{pq} am weitesten rechts liegende Gerade, d.h. jeder andere Punkt liegt links von der Geraden \overline{pq} (oder auf der Geraden näher bei p).
- 3 Fahre mit $p \leftarrow q$ bei (2) weiter, bis $p = p_0$.

762

763

Illustration Jarvis



Analyse Gift-Wrapping

- Sei h die Anzahl Eckpunkte der konvexen Hülle.
- Laufzeit des Algorithmus $\mathcal{O}(h \cdot n)$.

764

765

Dichtestes Punktepaar

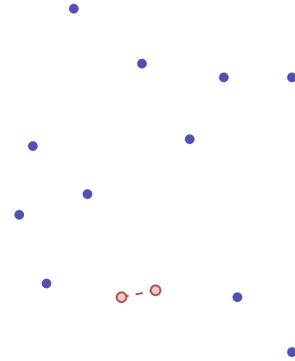
Euklidischer Abstand $d(s, t)$ zweier Punkte s und t :

$$d(s, t) = \|s - t\|_2 = \sqrt{(s_x - t_x)^2 + (s_y - t_y)^2}$$

Problem: Suche Punkte p und q aus Q , für welche gilt

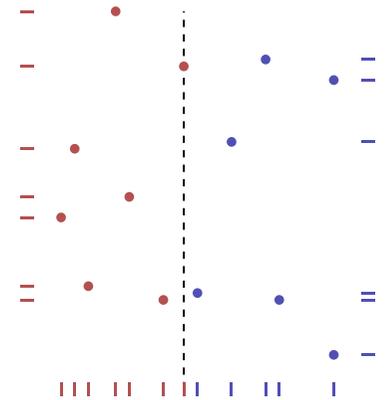
$$d(p, q) \leq d(s, t) \quad \forall s, t \in Q, s \neq t.$$

Naiv: alle $\binom{n}{2} = \Theta(n^2)$ Punktepaare.



Divide And Conquer

- Punktmenge P , zu Beginn $P \leftarrow Q$
- Arrays X und Y , welche die Punkte aus P enthalten, sortiert nach x - bzw. nach y -Koordinate.
- Teile Punktmenge ein in zwei (annähernd) gleich grosse Mengen P_L und P_R , getrennt durch vertikale Gerade durch einen Punkt von P .
- Teile Arrays X und Y entsprechend in X_L, X_R, Y_L und Y_R .

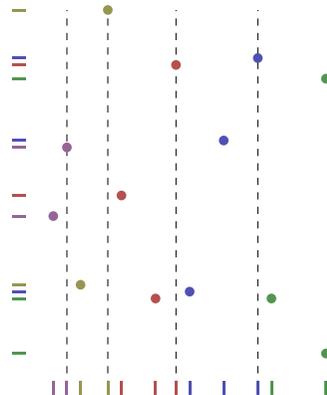


766

767

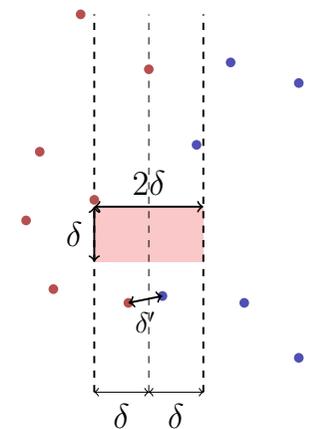
Divide And Conquer

- Rekursiver Aufruf jeweils mit P_L, X_L, Y_L und P_R, X_R, Y_R . Erhalte minimale Abstände δ_L, δ_R .
- (Wenn nur noch $k \leq 3$ Punkte: berechne direkt minimalen Abstand)
- Nach rekursivem Aufruf $\delta = \min(\delta_L, \delta_R)$. Kombiniere (nächste Folie) und gib bestes Resultat zurück.



Kombinieren

- Erzeuge Array Y' mit y -sortierten Punkten aus Y , die innerhalb des 2δ Streifens um die Trennlinie befinden
- Betrachte für jeden Punkt $p \in Y'$ die sieben* (!) auf p folgenden Punkte. Berechne minimale Distanz δ' .
- Wenn $\delta' < \delta$, dann noch dichteres Paar in P als in P_L und P_R gefunden. Rückgabe der minimalen Distanz.



*Man kann zeigen, dass maximal acht Punkte aus P im gezeigten Rechteck liegen können. Hier ohne Beweis.

768

769

Implementation

- Ziel: Rekursionsgleichung (Laufzeit) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$.
- Konsequenz: in den Schritten ist das Sortieren verboten!
- Nichttrivial: nur Arrays Y und Y'
- Idee: Merge umgekehrt: durchlaufe (nach y -Koordinate vorsortiertes) Y und hänge dem Auswahlkriterium der x -Koordinate folgend an Y_L und Y_R an. Genauso für Y' . Laufzeit $\mathcal{O}(|Y|)$.

Gesamtlaufzeit: $\mathcal{O}(n \log n)$.