26. Geometric Algorithms

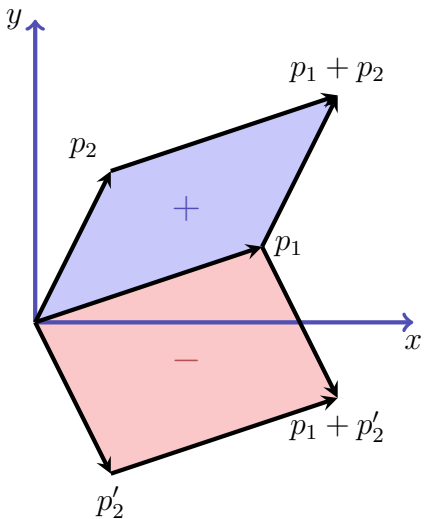
Properties of Line Segments, Intersection of Line Segments, Convex Hull, Closest Point Pair [Ottman/Widmayer, Kap. 8.2,8.3,8.8.2, Cormen et al, Kap. 33]

Properties of line segments.

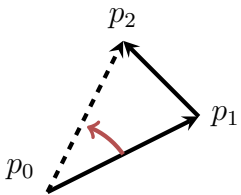
Cross-Product of two vectors $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ in the plane

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

Signed area of the parallelogram

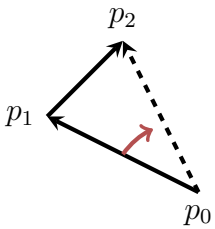


Turning direction



nach links:

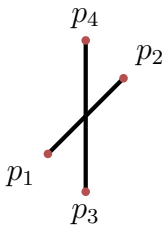
$(p_1 - p_0) \times (p_2 - p_0) > 0$



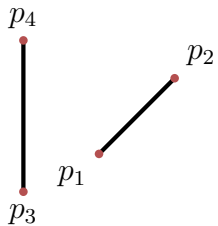
nach rechts:

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

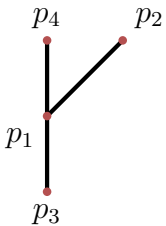
Intersection of two line segments



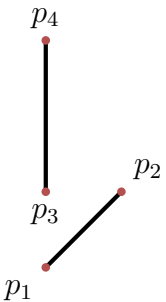
Intersection: p_1 and p_2 opposite w.r.t $\overline{p_3p_4}$ and p_3, p_4 opposite w.r.t. $\overline{p_1p_2}$



No intersection: p_1 and p_2 on the same side of $\overline{p_3p_4}$

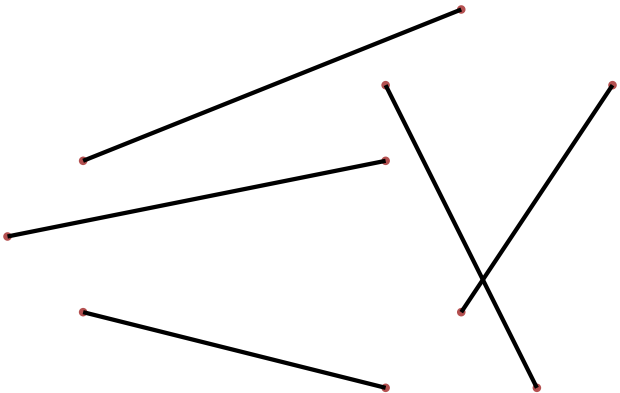


Intersection: p_1 on $\overline{p_3p_4}$

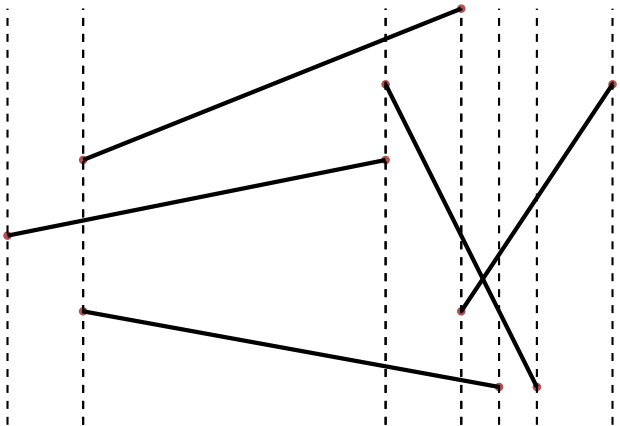


No intersection: p_3 and p_4 on the same side of $\overline{p_1p_2}$

Cutting many line segments



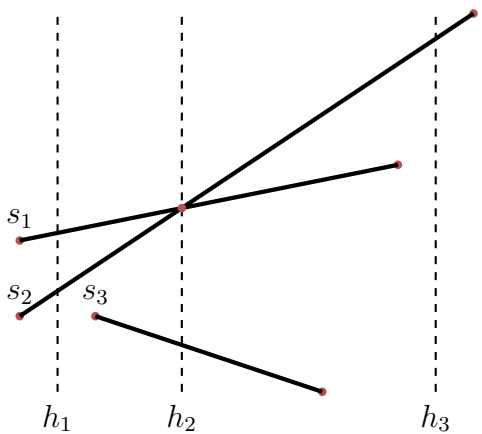
Sweepline Principle



Simplifying Assumptions

- No vertical line segments
- Each intersection is formed by at most two line segments.

Ordering line segments



Preorder (partial order without anti-symmetry)

$$s_2 \preccurlyeq_{h_1} s_1$$
$$s_1 \preccurlyeq_{h_2} s_2$$
$$s_2 \preccurlyeq_{h_2} s_1$$
$$s_3 \preccurlyeq_{h_2} s_2$$

W.r.t. h_3 the line segments are uncomparable.

Moving the sweepline

- *Sweep-Line Status* : Relationship of all objects intersected by sweep-line
- *Event List* : Series of event positions, sorted by x-coordinate. Sweep-line travels from left to right and stops at each event position.

Sweep-Line Status

Preorder T of the intersected line segments Required operations:

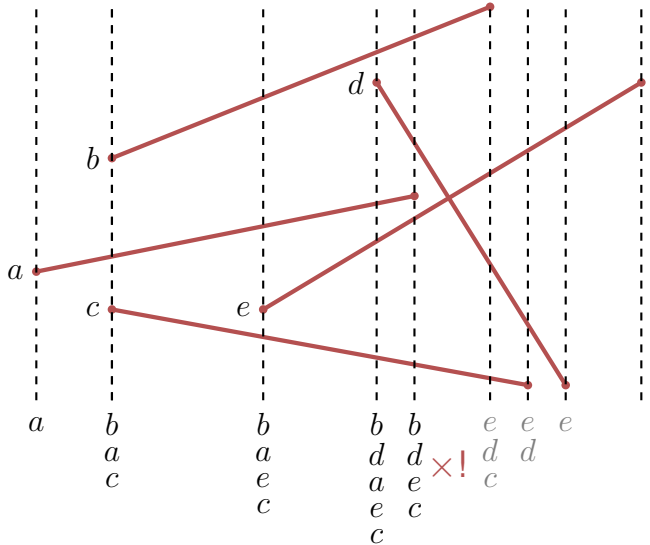
- *Insert*(T, s) Insert line segment s in T
- *Delete*(T, s) Remove s from T
- *Above*(T, s) Return line segment immediately above of s in T
- *Below*(T, s) Return line segment immediately below of s in T

Possible Implementation: Blanced tree (AVL-Tree, Red-Black Tree etc.)

Algorithm Any-Segments-Intersect(S)

Input : List of line segments S
Output : Returns if S contains intersecting segments
$T \leftarrow \emptyset$
Sort endpoints of line segments in S from left to right (left before right and lower before upper)
for Sorted end points p **do**
    if p left end point of a segment 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 right end point of a segment s **then**
        if Above$(T, s) \cap$ Below$(T, s) \neq \emptyset$ **then** **return** true
        Delete(T, s)
return false;

Illustration



Analysis

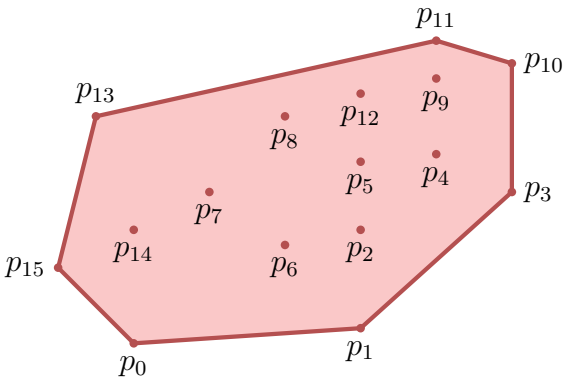
Runtime of the algorithm Any-Segments-Intersect

- Sorting $\mathcal{O}(n \log n)$
- n iterations of the for loop. Each operation on the balanced tree $\mathcal{O}(\log n)$

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

Convex Hull

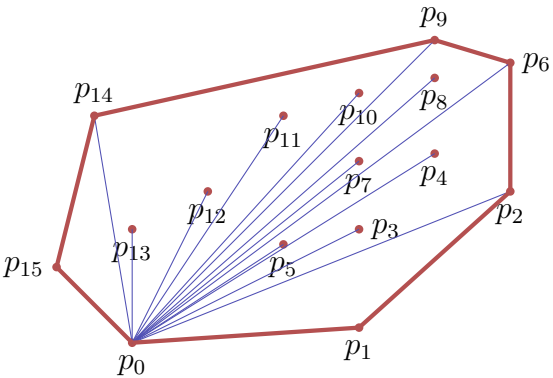
Konvexe Hülle $CH(Q)$ einer Menge Q von Punkten: kleinstes konvexes Polygon P, so dass jeder Punkt entweder auf dem Rand oder im Inneren liegt.



Algorithm Graham-Scan

Input : Set of points Q
Output : Stack S of points of the convex hull of Q
p_0: point with minimal y coordinate (if required, additionally minimal x-) coordinate
$(p_1, \ldots, p_m)$ remaining points sorted by polar angle counter-clockwise in relation to p_0; if points with same polar angle available, discard all except the one with maximal distance from 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

Analysis

Runtime of the algorithm Graham-Scan

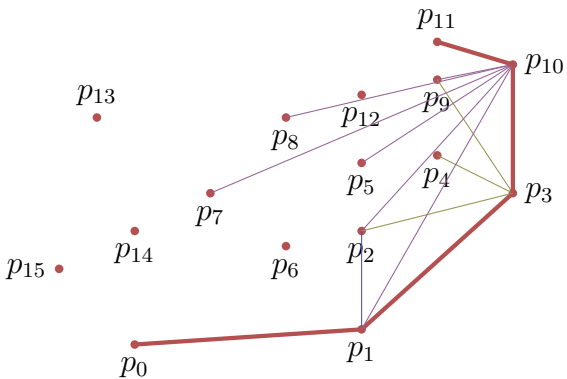
- Sorting $\mathcal{O}(n \log n)$
- n Iterations of the for-loop
- Amortized analysis of the multipop on a stack: amortized constant runtime of mulitpop, same here: amortized constant runtime of the While-loop.

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

Jarvis Marsch / Gift Wrapping algorithm

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 $p = p_0$.

Illustration Jarvis



Analysis Gift-Wrapping

- Let h be the number of corner points of the convex hull.
- Runtime of the algorithm $\mathcal{O}(h \cdot n)$.

Closest Point Pair

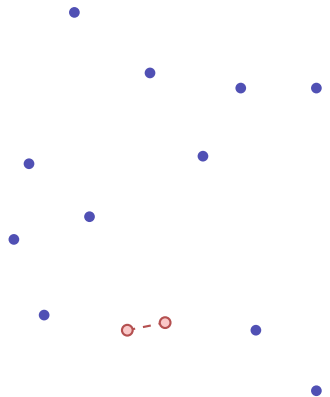
Euclidean Distance $d(s,t)$ of two points s and t:

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

Problem: Find points p and q from Q for which

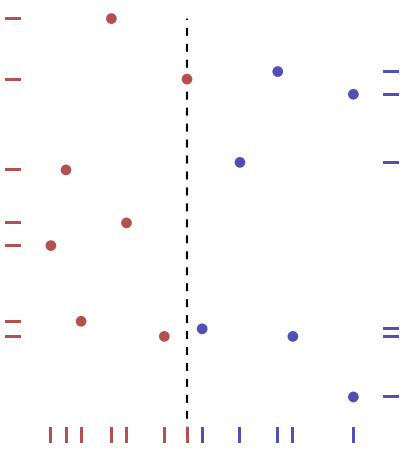
$$d(p,q) \leq d(s,t) \; \forall \; s,t \in Q, s \neq t.$$

Naive: all $\binom{n}{2} = \Theta(n^2)$ point pairs.



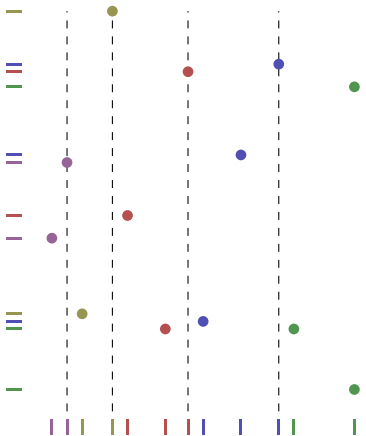
Divide And Conquer

- Set of points P, starting with $P \leftarrow Q$
- Arrays X and Y, containing the elements of P , sorted by x- and y-coordinate, respectively.
- Partition point set into two (approximately) equally sized sets P_L and P_R, separated by a vertical line through a point of P.
- Split arrays X and Y accrodingly in X_L, X_R. Y_L and Y_R.



Divide And *Conquer*

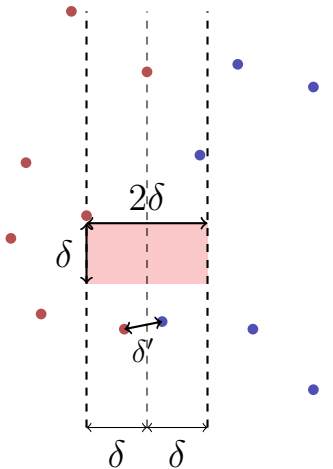
- Recursive call with P_L, X_L, Y_L and P_R, X_R, Y_R. Yields minimal distances δ_L, δ_R.
- (If only $k \leq 3$ points: compute the minimal distance directly)
- After recursive call $\delta = \min(\delta_L, \delta_R)$. Combine (next slides) and return best result.



Combine

- Generate an array Y' holding y-sorted points from Y, that are located within a 2δ band around the dividing line
- Consider for each point $p \in Y'$ the seven! (!) points after p . Compute minimal distance δ'.
- If $\delta' < \delta$, then a closer pair in P than in P_L and P_R found. Return minimal distance.

*It can be shown that maximally eight points from P can be located in the shown rectangle. Here without proof.



Implementation

- Goal: recursion equation (runtime) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$.
- Consequence: forbidden to sort in each steps of the recursion.
- Non-trivial: only arrays Y and Y'
- Idea: merge reversed: run through Y that is presorted by y-coordinate. For each element follow the selection criterion of the x-coordinate and append the element either to Y_L or Y_R. Same procedure for Y'. Runtime $\mathcal{O}(|Y|)$.

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