Segment intersections

Input $S = \{ s_1, s_2, \dots s_n \}$ segments in the plane

Output All intersections of segments, every intersection with all segments on which it lies

We do not consider

Sweep line method

Sweep line moves from the top to the bottom.

It crosses so called events

- endpoints of segment
- intersections

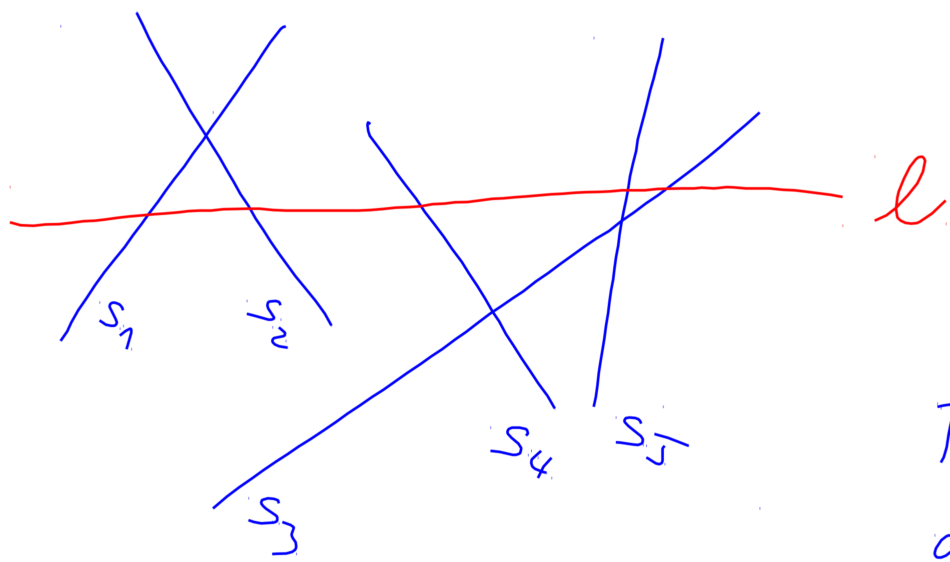
Events are ordered in so called <u>queue</u>.

We will use lexicographic ordering

$$p > q \iff (p_y > q_y) \text{ or } (p_y = q_y \wedge p_x < q_x)$$

At the ~~beg~~ beginning only endpoints of segment are in the queue. In the course of algorithm we add computed intersections into the queue.

Second structure connected with sweep line method is Balanced binary tree — which saves the order of segments in which they are intersected by the sweep line



$s_1 > s_2 > s_4 > s_5 > s_3$

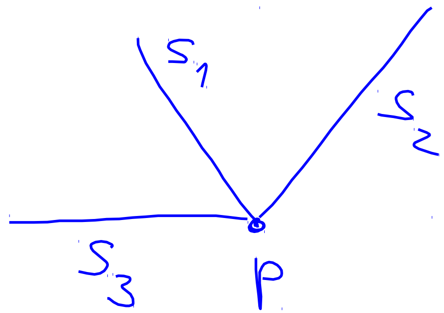
These segments are leaves of a balanced binary tree.

The tree changes when the sweep line crosses an event.

Algorithm

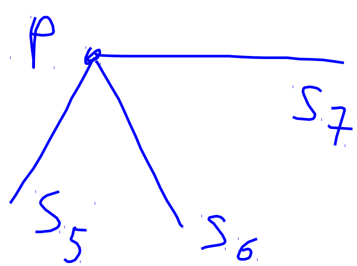
Let p be an events. We describe actions which the algorithm does when the sweep passes p.

$L(p)$ ..... segments with p as lower endpoint



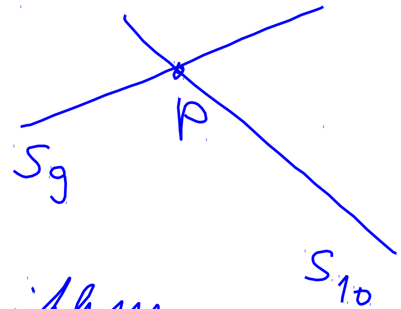
$L(p) = \{ s_3, s_1, s_2 \}$

$U(p)$ ..... segment with p as upper endpoint



$U(p) = \{ s_5, s_6, s_7 \}$

$C(p)$ ... segments with p as an internal points



$C(p) = \{s_9, s_{10}\}$

Description of the algorithm

What happens when the sweep line crossed an event p.

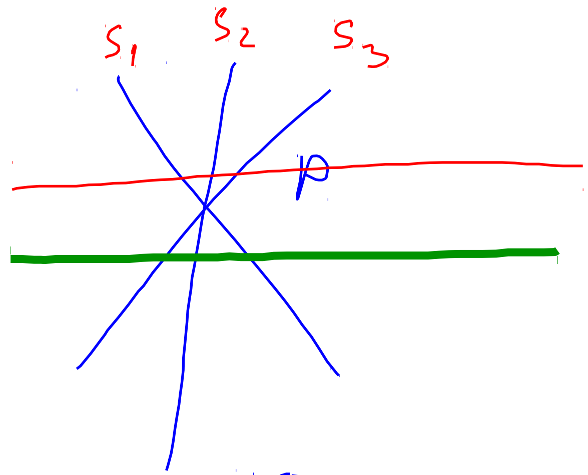
If $|L(p) \cup C(p) \cup U(p)| \geq 2$, algorithm refer p as an intersection.

p is removed from the queue.

The segments from $L(p)$ do not intersect sweep any more, they dissapear from the tree.

The segments from $U(p)$ will appear ~~[illegible]~~ in the tree.

The segments from $C(p)$ change order in the tree.

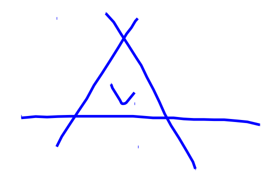


$s_1 > s_2 > s_3$

$s_3 > s_2 > s_1$

Finally: ① Segments from $L(p) \cup C(p)$ are removed from the tree. After every removal we have to rebalance the tree.

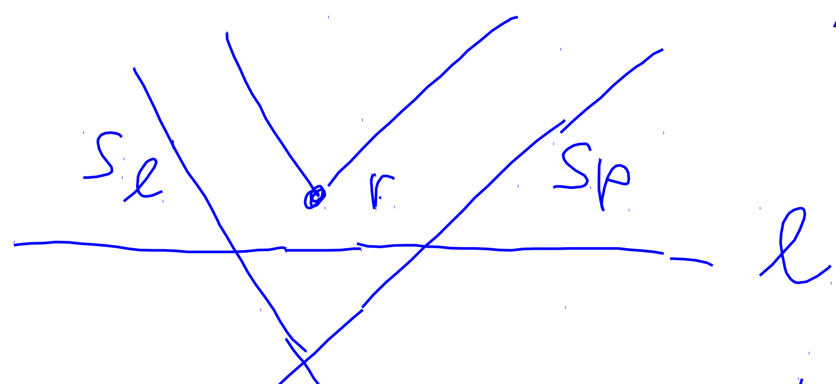
② Segments from $U(p) \cup C(p)$ are added to the tree and the tree is rebalanced again.



Computing intersections

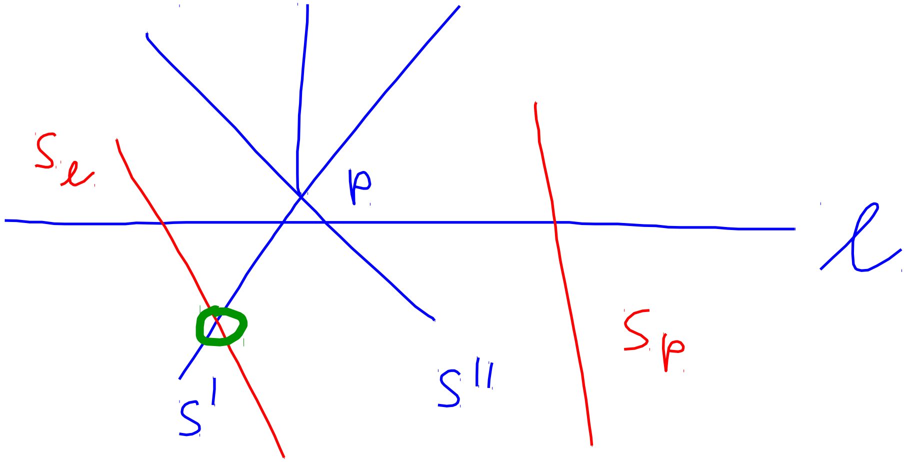
① $U(p) \cup C(p) = \emptyset$

We compute $s_l \cap s_p$



If … intersection lies below the …, we insert it into the queue.

② $U(p) \cup C(p) \neq \emptyset$



We compute

$s_\ell \cap s'$

$s_p \cap s''$

If intersections exist under ℓ, we insert them into the queue.

Running time is $O((n+k)\log n)$

n .... number of segments

k .... number of intersections

We need the Euler Formula for planar graphs

For planar graph we have

$$n_v - n_e + n_f \geq 2$$

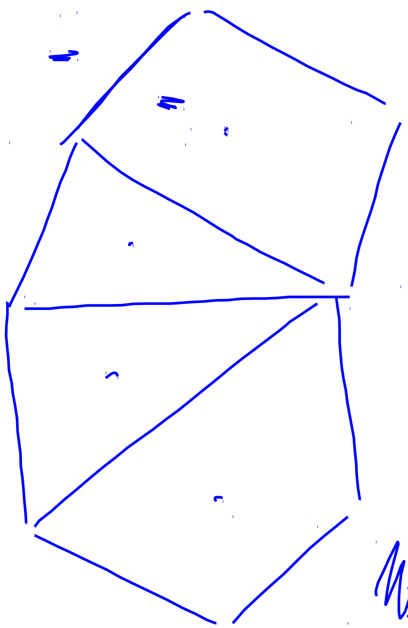
If it is connected $n_v - n_e + n_f = 2$.

Corollary: For any planar graph

$$n_e \leq 3(n_v - 1)$$

Proof

bounded face is restricted at least by 3 edges



Every edge is adjacent to at most two faces

$$n_f \le \frac{2n_e}{3} + 1$$

We substitute this into the Euler formula

$$n_v - n_e + n_f \ge 2$$

$$n_v - n_e + \frac{2n_e}{3} + 1 \ge 2$$

$$n_v - 1 \ge \frac{1}{3} n_e \Rightarrow n_e \le 3(n_v - 1)$$

We compute the running time of the algorithm.

1) At the beginning we order $2n$ endpoints into the queue.
It lasts $O(n \log n)$.

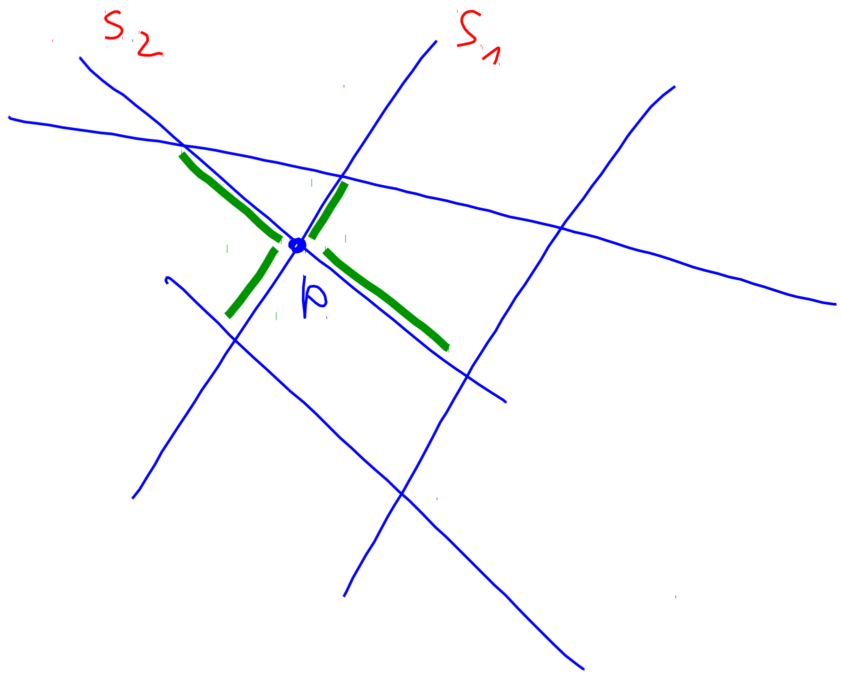
Let $m(p)$ be the number of segments in $L(p) \cup C(p) \cup U(p)$.

Actions in p .... one segment removing or adding + rebalancing
.... $O(\log n)$
.... finding s', s'', s_l, s_r .... $O(\log n)$
.... computing intersections .... $O(\text{const}) = O(1)$
.... inserting the intersection into the queue
$O(\log n)$

Overall time is

$$O(n \log n) + \sum_{p \text{ event}} m(p) \cdot O(\log n)$$

Segments, their endpoints and intersections form a planar graph.



p is a vertex of our graph

$p \in s_1 \cap s_2$

The degree of p in the graph is 4

$s(p)$ ... the degree of p in the graph

$m(p) = 2 \le s(p) = 4$

Generally $m(p) \le s(p)$.

$$\sum m(p) \le \sum_p s(p) = 2n_e \le 6(n_v - 1) \le 6(2n+k-1) \le 12(n+k) = O(n+k)$$

previous lemma

Estimate

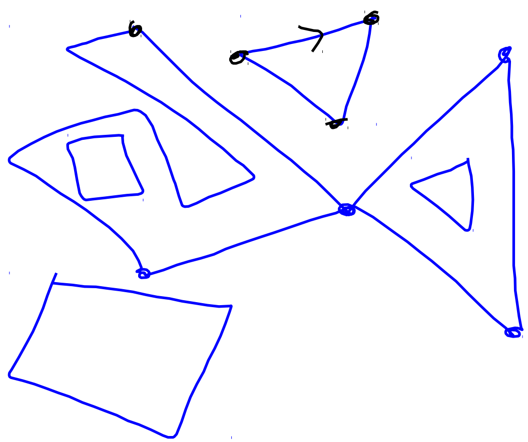
$$O(n \log n) + \sum m(p)\, O(\log n) \le O(n \log n) + 12(n+k)\, O(\log n)$$

$$= O((n+k) \log n)$$

Chapter 3 Maps overlap

Description of planar subdivisions

~ a maps with vertices, segments and faces.



Descriptions – doubly connected edge list

vertices

edges ..... oriented segments

faces .....

DCEL consists of 3 tables

Table for vertices

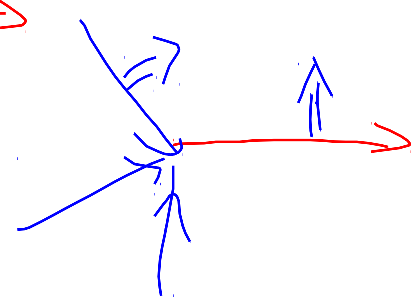
name of vertex | coordinates | one edge coming from the vertex

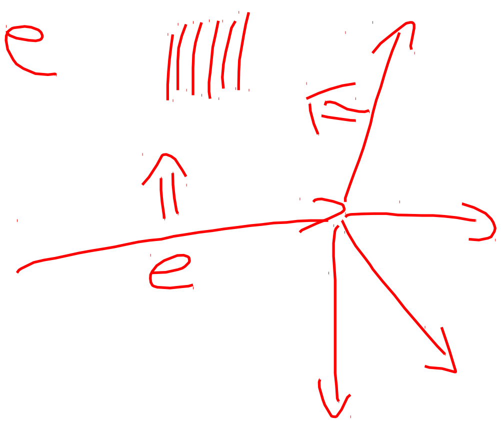
Table for edges

name | origin (vertex from the edge is coming) | twin | next edge

previous edge | adjacent face

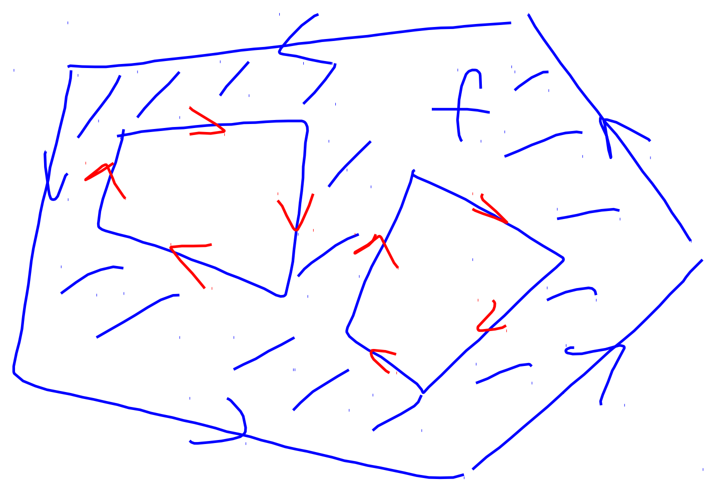
f is adjacent face to e





Third table for faces

face 1 edge from outer cycle 1 edge from every inner ~~cycles~~ cycle



outer cycle

$e_1 e_2, \dots e_n$

such that

$e_2 = next(e_1)$

$e_3 = next(e_2)$

$e_1 = next(e_n)$

f is adjacent to all of them