Computational Geometry

October 25, 2019

What is computational geometry?

- Study of algorithms for geometric problem solving.
- Typical problems

  Given a description of a set of geometric objects, e.g., set of points/segments/vertices of a polygon in a certain order.

  Answer a query about this set, e.g.:

  1. do some segments intersect?
  2. what is the convex hull of the set of points?

  ...
- In this lecture, we assume objects represented by a set/sequence of n points $\langle p_0, p_1, \ldots, p_{n-1}\rangle$ where each point p_i is given by its pair of coordinates $(x_i, y_i) \in \mathbb{R}^2$

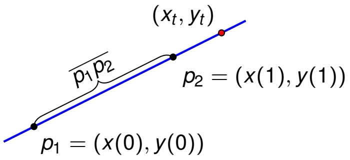
ASSUMPTION: $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ are distinct points.

- The line through p_1 and p_2 is

$$p_1 p_2 = \{(x(t), y(t)) \in \mathbb{R}^2 \mid x(t) = (1-t)\, x_1 + t\, x_2, y(t) = (1-t)\, y_1 + t\, y_2\}$$

- The segment with endpoints p_1 and p_2 is

$$\overline{p_1 p_2} = \{(x(t), y(t)) \in \mathbb{R}^2 \mid \begin{array}{l} x(t) = (1-t)\, x_1 + t\, x_2, \\ y(t) = (1-t)\, y_1 + t\, y_2, 0 \le t \le 1\} \end{array}$$



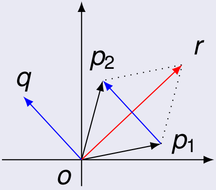
Vectors and their representation

The directed segment (or vector) $\overrightarrow{p_1 p_2}$ imposes an ordering on its endpoints: p_1 is its origin, and p_2 its destination.

Sum of vectors

$o = (0, 0)$ is the origin of the system of coordinates.

- If $p = (x, y)$ is a point, then $\overrightarrow{op}$ is the vector with origin o and destination p.
- If $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ then $p_1 + p_2$ is the point with coordinates $(x_1 + y_1, x_2 + y_2)$, and $p_1 - p_2$ is the point with coordinates $(x_1 - y_1, x_2 - y_2)$



Remarks:

If $r = p_1 + p_2$ and $q = p_2 - p_1$ then

1) op_1rp_2 is a parallelogram

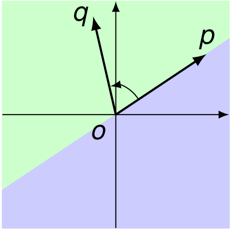
2) the vector $\overrightarrow{or}$ is the sum of vectors $\overrightarrow{op_1}$ and $\overrightarrow{op_2}$

3) the vector $\overrightarrow{p_1 p_2}$ coincides with the vector $\overrightarrow{oq}$

Segments and vectors

Let $p = (x_1, y_1)$, $q = (x_2, y_2)$ be two points.

1. The vector $\overrightarrow{pq}$ and the segment $\overline{pq}$ have the same length, which is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
2. The vector $\overrightarrow{op}$ splits the plane in two parts: the semiplane of points to the left of $\overrightarrow{op}$ (coloured green), and the semiplane of points to the right of $\overrightarrow{op}$ (coloured blue).



Remarks:

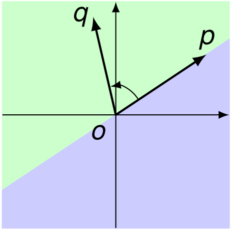
1) q is to the right of $\overrightarrow{op}$ if $\overrightarrow{oq}$ is rotated clockwise w.r.t. $\overrightarrow{op}$

2) q is to the left of $\overrightarrow{op}$ if $\overrightarrow{oq}$ is rotated counterclockwise w.r.t. $\overrightarrow{op}$

Segments and vectors

Let $p = (x_1, y_1)$, $q = (x_2, y_2)$ be two points.

1. The vector $\overrightarrow{pq}$ and the segment $\overline{pq}$ have the same length, which is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
2. The vector $\overrightarrow{op}$ splits the plane in two parts: the semiplane of points to the left of $\overrightarrow{op}$ (coloured green), and the semiplane of points to the right of $\overrightarrow{op}$ (coloured blue).



Remarks:

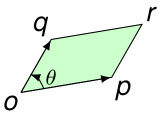
1) q is to the right of $\overrightarrow{op}$ if $\overrightarrow{oq}$ is rotated clockwise w.r.t. $\overrightarrow{op}$

2) q is to the left of $\overrightarrow{op}$ if $\overrightarrow{oq}$ is rotated counterclockwise w.r.t. $\overrightarrow{op}$

We can detect if q is to the left or right of $\overrightarrow{op}$ by computing the sign of a cross product (see next slide).

Let $p = (x_1, y_1)$, $q = (x_2, y_2)$, and $r = p + q$. The cross product $\overrightarrow{op} \times \overrightarrow{oq}$ is

$$\overrightarrow{op} \times \overrightarrow{oq} = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 \cdot y_2 - x_2 \cdot y_1 = -\overrightarrow{oq} \times \overrightarrow{op}$$



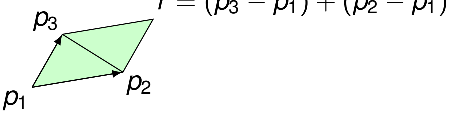
$$\sin(\theta) = \frac{\overrightarrow{op} \times \overrightarrow{oq}}{|\overrightarrow{op}| \cdot |\overrightarrow{oq}|}$$

Geometric interpretation:

- $|\overrightarrow{op} \times \overrightarrow{oq}|$ is the area of the parallelogram *oprq*
- q is to the left of $\overrightarrow{op}$ if $\overrightarrow{op} \times \overrightarrow{oq} > 0$
- q is to the right of $\overrightarrow{op}$ if $\overrightarrow{op} \times \overrightarrow{oq} < 0$
- q is on line *op* if $\overrightarrow{op} \times \overrightarrow{oq} = 0$

Let $p_1 = (x_1, y_1), p_2 = (x_2, y_2), p_3 = (x_3, y_3)$.

1. The area of triangle $p_1 p_2 p_3$ is half of the area of the parallelogram spanned between vectors $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_1 p_3}$:



$$\mathrm{area}(p_1 p_2 r p_3) = |\overrightarrow{p_1 p_2} \times \overrightarrow{p_1 p_3}| = \mathrm{abs}\left(\begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} \right),$$
$$\mathrm{area}(p_1 p_2 p_3) = \mathrm{area}(p_1 p_2 r p_3)/2 = \mathrm{abs}\left(\begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} \right)/2$$

2. p_3 is to the left of $\overrightarrow{p_1 p_2} \Leftrightarrow \overrightarrow{p_1 p_3}$ is rotated counterclockwise w.r.t. $\overrightarrow{p_1 p_2} \Leftrightarrow \overrightarrow{p_1 p_2} \times \overrightarrow{p_1 p_3} > 0$.

ASSUMPTION: $p_i = (x_i, y_i)$ are four distinct points, $1 \leq i \leq 4$.
Question: Do segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect or not?

REMARK: $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect if either (or both) of the following conditions hold:

1. p_1 and p_2 are on different sides of the line $p_3 p_4$; and p_3 and p_4 are on different sides of the line $p_1 p_2$,
2. an endpoint of one segment lies on the other segment (this condition comes from the boundary case).

```
/* check if $\overline{p_1 p_2} \cap \overline{p_3 p_4} \neq \emptyset$ */
SegmentsIntersect($p_1, p_2, p_3, p_4$)
$d_1 = \text{SignedArea}(p_3, p_4, p_1)$
$d_2 = \text{SignedArea}(p_3, p_4, p_2)$
$d_3 = \text{SignedArea}(p_1, p_2, p_3)$
$d_4 = \text{SignedArea}(p_1, p_2, p_4)$
if $((d_1 < 0 \wedge d_2 > 0) \vee (d_1 > 0 \wedge d_2 < 0)) \vee$
   $((d_3 < 0 \wedge d_4 > 0) \vee (d_3 > 0 \wedge d_4 < 0))$
   return TRUE
return FALSE

SignedArea($p_i, p_j, p_k$)
return $((p_k - p_i) \times (p_j - p_i))/2$
```

The segment intersection test problem

Given a set $S = \{s_1, \ldots, s_n\}$ of line segments

Determine if $s_i \cap s_j \neq \emptyset$ for some $1 \leq i \neq j \leq n$.

The segment intersection test problem

Given a set $S = \{s_1, \ldots, s_n\}$ of line segments

Determine if $s_i \cap s_j \neq \emptyset$ for some $1 \leq i \neq j \leq n$.

We can do this in $O(n \log n)$ time with the sweeping technique:

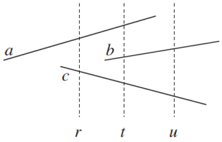
The segment intersection test problem

Given a set $S = \{s_1, \ldots, s_n\}$ of line segments

Determine if $s_i \cap s_j \neq \emptyset$ for some $1 \leq i \neq j \leq n$.

We can do this in $O(n \log n)$ time with the sweeping technique:

- An imaginary vertical sweep line passes through the given set of geometric objects, usually from left to right.
  - We will assume that the sweeping line moves across the x-dimension



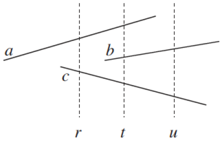
The segment intersection test problem

Given a set $S = \{s_1, \ldots, s_n\}$ of line segments

Determine if $s_i \cap s_j \neq \emptyset$ for some $1 \leq i \neq j \leq n$.

We can do this in $O(n \log n)$ time with the sweeping technique:

- An imaginary vertical sweep line passes through the given set of geometric objects, usually from left to right.
  - We will assume that the sweeping line moves across the x-dimension



Simplifying assumptions

1. No input segment is vertical
2. No three input segments intersect at a single point

Auxiliary notions

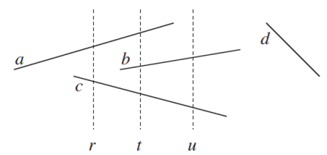
Ordering segments

ASSUMPTIONS: $s_1, s_2 \in S$ are two line segments; sw_x is the vertical sweep line with x-coordinate x

- s_1, s_2 are comparable at x if sw_x intersects both s_1 and s_2
- $s_1 \succeq_x s_2$ if s_1, s_2 are x-comparable, and the intersection point $s_1 \cap sw_x$ is higher than $s_2 \cap sw_x$

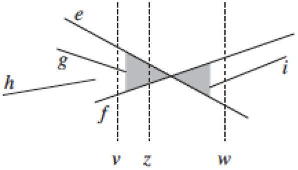
Example

In the figure below, we have $a \succeq_r c$, $a \succeq_t b$, $b \succeq_t c$, and $b \succeq_u c$. Segment d is not comparable with any other segment.



Remark: $\succeq_x$ is a total preorder relation: reflexive, transitive, but neither symmetric nor antisymmetric.

Detecting segment intersections



When line segments e and f intersect, they reverse their orders: we have $e \succeq_v f$ and $f \succeq_w e$.

- Simplifying assumption 2 implies $\exists$vertical sweep line sw_x for which the intersections with segments e and f are consecutive w.r.t. total preorder $\succeq_x$.
  - $\Rightarrow$ Any sweep line that passes through the shaded region in figure above (such as z) has e and f consecutive in its total preorder.

Moving the sweep line

- The sweep line moves from left to right, through the sequence of endpoints sorted in increasing order of the x-coordinate.
- The sweeping algorithm maintains two data structures:

  Sweep line status: the relationships among the objects that the sweep line intersects.

  Event-point schedule: a sequence of points (the *event points*) ordered from left to right according to their x-coordinates.

Moving the sweep line

- The sweep line moves from left to right, through the sequence of endpoints sorted in increasing order of the x-coordinate.
- The sweeping algorithm maintains two data structures:

  Sweep line status: the relationships among the objects that the sweep line intersects.

  Event-point schedule: a sequence of points (the *event points*) ordered from left to right according to their x-coordinates.

Whenever the sweep line reaches the x-coordinate of an event point: the sweep halts, processes the event point, and then resumes

- Changes to the sweep-line status occur only at event points.

THE SWEEP LINE STATUS: container for a total preorder $T = \succeq_x$ between line segments from S

Requirements: to perform efficiently the following operations:

1. $\texttt{insert}(T, s)$: insert segment s into T
2. $\texttt{delete}(T, s)$: delete segment s from T
3. $\texttt{above}(T, s)$: return the segment immediately above segment s in T.
4. $\texttt{below}(T, s)$: return the segment immediately below segment s in T.

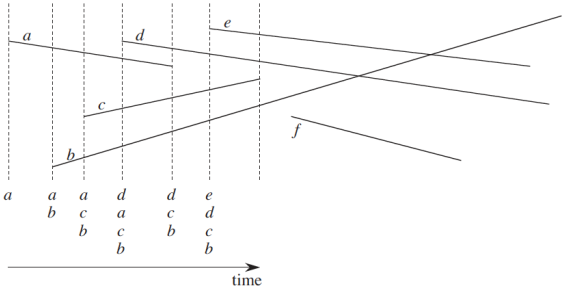
REMARK: all these operations can be performed in $O(\log n)$ time using red-black trees.

The sweeping algorithm for segment intersections

Pseudocode

```
AnySegmentsIntersect(S)
 1. T = ∅
 2. sort the endpoints of the segments in S from left to right,
    breaking ties by putting left endpoints before right endpoints
    and breaking further ties by putting points with lower y-coordinates first
 3. for each point p in the sorted list of endpoints
 4.       if p is the left endpoint of a segment s
 5.           insert(T, s)
 6.           if (above(T, s) exists and intersects s)
                  or (below(T, s) exists and intersects s)
 7.               return TRUE
 8.       if p is the right endpoint of a segment s
 9.           if both above(T, s) and below(T, s) exist
                  and above(T, s) intersects below(T, s)
10.               return TRUE
11.           delete(T,s)
12. return FALSE
```

The sweeping algorithm for segment intersection

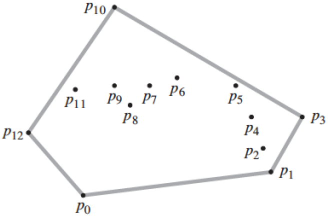


- Every dashed line is the sweep line at an event point.
- The ordering of segment names below each sweep line corresponds to the total preorder *T* at the end of the `for` loop processing the corresponding event point.
- The rightmost sweep line occurs when processing the right endpoint of segment *c*.

ASSUMPTION: Q is a finite set of n points.
The convex hull $CH(Q)$ of Q is the smallest convex polygon P with vertices in Q, such that each point in Q is either on the boundary of P or in its interior.

Intuition: each point of Q is a nail stuck in a board $\Rightarrow$ convex hull = the shape formed by a tight rubber band that surrounds all the nails.

EXAMPLE:



The Graham's scan method

Computes $CH(P)$ in $O(n \log n)$, where $n = |Q|$ with a technique named rotational sweep:

- vertices are processed in the order of the polar angles they form with a reference vertex.

MAIN IDEA: Maintain a stack S of candidate points for the vertices of P in counterclockwise order.

- each point of Q is pushed onto S one time.
- the points in already S, which are not in $CH(Q)$, are popped from S.
- Related operations: $\texttt{push}(p, S)$, $\texttt{pop}(S)$, and
  - $\texttt{top}(S)$ return, but do not pop, the point on top of S
  - $\texttt{nextToTop}(S)$: return the point one entry below the top of S without changing S

Convex hull

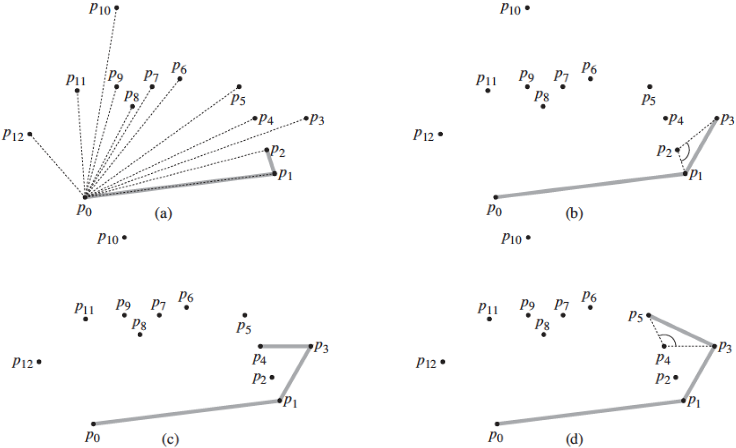
Graham's scan algorithm: pseudocode

GraHamScan(Q)

let p_0 be the point in Q with the minimum y-coordinate,
    or the leftmost such point in case of a tie
let $\langle p_1, p_2, \ldots, p_m \rangle$ be the remaining points in Q, sorted by polar angle
    in counterclockwise order around p_0 (if more than one point has the same angle,
    remove all but the one that is farthest from p_0)
let S be an empty stack
`push`(p_0, S)
`push`(p_1, S)
`push`(p_2, S)
`for` $i = 3$ `to` m
        `while` the angle formed by `nextToTop`(S), `top`(S), and p_i
                makes a nonleft turn
                `pop`(S)
`push`(p_i, S)
`return` S

Graham's scan algorithm: pseudocode

Snapshots of algorithm execution



Given a set Q of $n \geq 2$ points $P_i(x_i, y_i)$, $1 \leq i \leq n$

Find a closest pair of points in Q.

Remarks

- "closest" refers to the usual euclidean distance between two points $P(x_1, y_1)$ and $Q(x_2, y_2)$, which is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

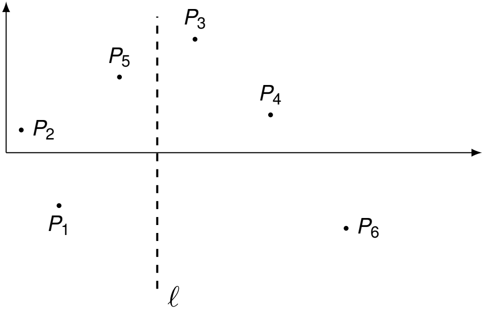
- A simple, brute-force approach is to compute the distances between all $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of points
$\Rightarrow$ alg. with time complexity $O(n^2)$
- We will indicate an algorithm that solves this problem in time $O(n \log n)$

- Each recursive call of the algorithm takes as input a subset $P \subseteq Q$ with $|P| > 3$, and arrays X and Y, each of which contains all the points of the input set P:
  - X contains the elements of P sorted in increasing order of the x-coordinate
  - Y contains the elements of P sorted in increasing order of the y-coordinate
- The base case of the algorithm is when $|P| \leq 3$: in this case we try all the $\binom{|P|}{2}$ pairs and return the closest pair.

Consists of three substeps:

- Divide
- Conquer
- Combine

1. Find a vertical line ℓ that bisects the point set P into two sets P_L and P_R such that $|P_L| = \lceil |P|/2 \rceil$, $Q_L = \lfloor |P|/2 \rfloor$, all points in P_L are <u>on or to the left</u> of line l, and all points in P_R are <u>on or to the right</u> of l.
2. Divide the array X into arrays X_L and X_R, which contain the points of P_L and P_R respectively, sorted by monotonically increasing x-coordinate.
3. Similarly, divide the array Y into arrays Y_L and Y_R, which contain the points of P_L and P_R respectively, sorted by monotonically increasing y-coordinate.



$X = [P_2, P_1, P_5, P_3, P_4, P_6]$
$Y = [P_6, P_1, P_2, P_4, P_5, P_3]$
$X_L = [P_2, P_1, P_5]$
$X_R = [P_3, P_4, P_6]$
$Y_L = [P_1, P_2, P_5]$
$Y_R = [P_6, P_4, P_3]$

Make two recursive calls, one to find the closest pair of points in P_L and the other to find the closest pair of points in P_R.

- The inputs to the first call are the subset P_L and arrays X_L and Y_L
- the second call receives the inputs P_R, X_R, and Y_R.

Let the closest-pair distances returned for P_L and P_R be δ_L and δ_R, respectively, and let $\delta = \min(\delta_L, \delta_R)$.

The closest pair is either

- the pair with distance δ found by one of the recursive calls, or
- a pair of points with one point in p_L and the other in p_R.

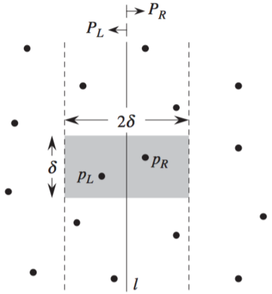
The algorithm determines whether there is a pair with one point in p_L and the other point in p_R and whose distance is less than δ.

- If such a pair exists, both points of the pair must be within δ units of line ℓ. Thus, they both must reside in the $2\,\delta$-wide vertical strip centered at line ℓ. The way to find such a pair, if one exists, is explained next.

The recursive step

3. The combine phase (contd.)

1. Create an array Y', which is the array Y with all points not in the $2\,\delta$-wide vertical strip removed. The array Y' is sorted by y-coordinate, just as Y is.



2. For each point p in Y', find if there is a point q in Y' whose distance to p is δ' smaller than δ. It turns out that it is sufficient to consider only the (max.) 7 points that follow p in Y'.

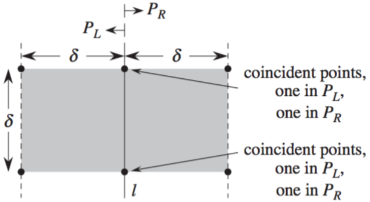
3. If $\delta' < \delta$, then the vertical strip does indeed contain a closer pair than the recursive calls found. Return this pair and its distance δ'. Otherwise, return the closest pair and its distance δ found by the recursive calls.

Suppose that at some level of the recursion, the closest pair of points is $p_L \in P_L$ and $p_R \in P_R$. Let δ' be the distance between p_L and p_R. Note that $\delta' < \delta$ and

- p_L is on or to the left of ℓ, and p_L is on or to the right of ℓ.
- both p_L ane p_R are less than δ units away from ℓ.
- p_L and p_R are within δ units of each other vertically.

$\Rightarrow$ p_L and p_R are within a $\delta \times 2\delta$ rectangle centered t line ℓ

- there may be other points in this rectangle as well, but
- at most 8 points of P can reside in the $\delta \times 2\delta$ rectangle:



We know from the Master theorem that, if we have the recurrence

$$T(n) = 2T(n/2) + O(n)$$

where $T(n)$ is the running time of the alg. for a set of n points, then $T(n) = O(n \log n)$.

- To ensure this runtime complexity, we must ensure that the combine phase gets executed in $O(n)$ time.
- This happens if, after partitioning P into P_L and P_R, we can form arrays Y_L and Y_R in linear time:
  - This is possible, because we can use Y (which is P sorted in increasing order of the y-coordinate) to compute Y_L and Y_R in linear time (see pseudo-code on next slide)

The divide-and-conquer algorithm

Implementation and running time (contd.)

The following algorithm splits Y into Y_L and Y_R

```
let Y_L[1 .. Y.length] and Y_R[1 .. Y.length] be new arrays
Y_L.length = Y_R.length = 0
for i = 1 to Y.length
    if Y[i] ∈ P_L
        Y_L.length = Y_L.length + 1
        Y_L[Y_L.length] = Y[i]
    else Y_R.length = Y_R.length + 1
        Y_R[Y_R.length] = Y[i]
```

Given two segments a and b in the plane, specified by their endpoints, where each enpoint is a pair giving its (x, y) coordinates.

Give an algorithm that determines if the segments a and b intersect, and if so, returns the (x, y) coordinate of a point from their intersection.

SUGGESTION: use the fact that a point $R(x, y)$ is on the segment with endpoints $P(x_1, y_1)$ and $Q(x_2, y_2)$ iff there exists $0 \le t \le 1$ such that

$$\begin{cases} x & = t \cdot x_1 + (1 - t) \cdot x_2 \\ y & = t \cdot y_1 + (1 - t) \cdot y_2 \end{cases}$$

There are $\mathtt{N}$ slides lying on the table. Each of them is transparent and formed as a rectangle.

- Every slide is specified by the (x, y) coordinates of the bottom-left and top-right corner of its rectangular shape.

In a traditional problem, one may have to calculate the intersecting area of these $\mathtt{N}$ slides. The definition of intersection area is such area which belongs to all of the slides.
I want to take out one of the $\mathtt{N}$ slides, so that the intersecting area of the remaining $\mathtt{N} - 1$ slides should be maximal. Tell me the maximum answer.

References

- Chapters 33: Computational Geometry from the book
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, 2000.