Algorithms and Data Structures: Computational Geometry I and II

Computational Geometry

In general, we will be considering 2-dimensional geometric problems (problems in the real plane).

Notation and basic definitions

- *Points* are pairs (x, y) with $x, y \in \mathbb{R}$.
- A *convex combination* of two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is a point $p = (x, y)$ such that

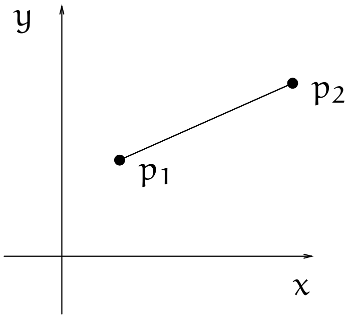
  $$\begin{aligned} x &= \alpha x_1 + (1 - \alpha) x_2 \\ y &= \alpha y_1 + (1 - \alpha) y_2 \end{aligned}$$

  for some $0 \leq \alpha \leq 1$.

  Abbreviate to $p = \alpha p_1 + (1 - \alpha) p_2$.
  Intuitively, a point p is a convex combination of p_1 and p_2 if it is on the line segment from p_1 to p_2

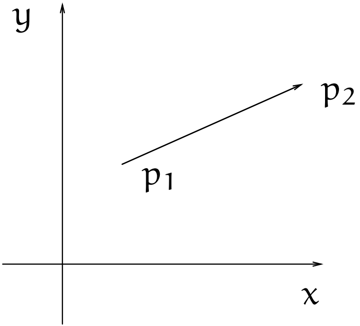
Line Segments

Undirected line segment $\overline{p_1 p_2}$ (set of all convex combinations of p_1 and p_2)



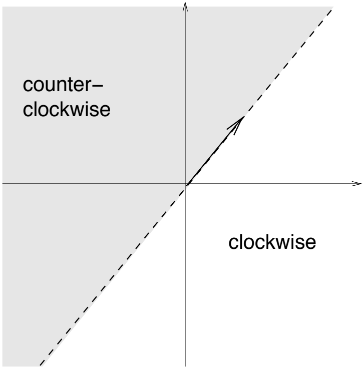
Directed Line Segments

Directed line segment $\overrightarrow{p_1 p_2}$:



When $p_1 = (0, 0)$, the *origin*, *treat* $\overrightarrow{p_1 p_2}$ *as the* vector p_2.

Clockwise and Counterclockwise from a Vector



Basic Problems

1. Given $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_0 p_2}$ is $\overrightarrow{p_0 p_1}$ collinear with, clockwise or counterclockwise from $\overrightarrow{p_0 p_2}$ w.r.t. p_0?
2. Given $\overline{p_1 p_2}$ and $\overline{p_2 p_3}$, if we traverse $\overline{p_1 p_2}$ and then $\overline{p_2 p_3}$ do we make a left, a right, or no turn at p_2?
3. Do $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect?

Design aim: use only $+$, $-$, $\times$ and comparisons.
Avoid division and trigonometric functions.

Straightforward Solutions

Use division and/or trigonometric functions. **Not our approach**

- For Problem (1) (special case with $p_0 = (0,0)$, $p_2 = (x_2, 0)$):

$$\text{vector } p_1 \text{ is clockwise from vector } p_2$$
$$\iff 0 < \measuredangle(p_1, p_2) < \pi \quad \iff \quad \sin(\measuredangle(p_1, p_2)) > 0.$$

  (It turns out, however, that we can compute the *sign* of $\sin(\measuredangle(p_1, p_2))$ *precisely* without using either division or trigonometric functions.)

  In measuring the angle from vector $p1$ *round to vector* p_2*, we measure anti-clockwise from* p_1*. It is a convention.*

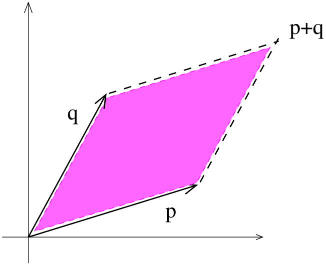
- For Problem (3):
  - Compute intersection point p of lines through p_1, p_2 and through p_3, p_4 (if no such point exists, then the segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ do not intersect).
  - Then check if p is on both segments.

Cross product

Given $p = (x_p, y_p)$, $q = (x_q, y_q)$. Define *cross product* by:

$$p \times q = \det \begin{pmatrix} x_p & x_q \\ y_p & y_q \end{pmatrix} = x_p y_q - x_q y_p.$$

Intuitively: Signed area of parallelogram spanned by vectors p, q:



Properties of the Cross Product

Lemma 1

$p = (x_p, y_p), q = (x_q, y_q)$ points in the plane. Then

1. $p \times q = -q \times p$
2.
   - *If $p \times q > 0$, then vector p is* clockwise *from q.*
   - *If $p \times q = 0$, then vectors p and q are* collinear*.*
   - *If $p \times q < 0$, then vector p is* counterclockwise *from q.*

Proof: (1) is immediate from the definition. (2) is elementary analytical geometry. For homework, first compute the line through $(0, 0)$ and q. Then check where p should lie in relation to this line - there are 2 cases, $x_q \geq 0$ and $x_q < 0$).

Solution to Problem (1)

Problem

Given $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_0 p_2}$, is $\overrightarrow{p_0 p_1}$ collinear with, clockwise or anti-clockwise from $\overrightarrow{p_0 p_2}$ w.r.t. p_0?

Solution

Use Lemma 1 after moving origin to $(0, 0)$. Just examine sign of:

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0).$$

Tip

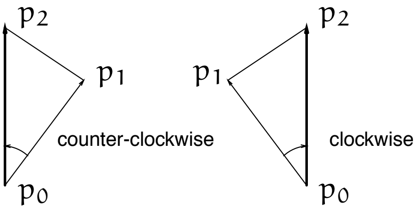
Can do a test: e.g., check vector $\overrightarrow{(0, 0)(2, 0)}$ against the point $(1, 1)$ (which is anti-clockwise of the vector).

Solution to Problem (2)

Problem

Given $\overline{p_0 p_1}$ and $\overline{p_1 p_2}$, if we traverse $\overline{p_0 p_1}$ and then $\overline{p_1 p_2}$ do we make a left, a right, or no turn at p_1?

Solution



$(p_1 - p_0) \times (p_2 - p_0) = 0$: collinear segments — no turn.
$(p_1 - p_0) \times (p_2 - p_0) < 0$: right turn at p_1.
$(p_1 - p_0) \times (p_2 - p_0) > 0$: left turn at p_1.

Solution to Problem (3)

Problem

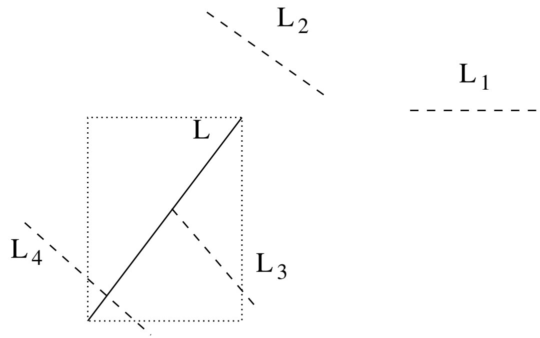
$\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect?

Solution

$\overline{p_1 p_2}$ straddles $\overline{p_3 p_4}$ if p_1 and p_2 lie on different sides of the line through p_3, p_4.
Then $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect if, and only if, one of the following conditions holds:

- *$\overline{p_1 p_2}$ straddles $\overline{p_3 p_4}$ and $\overline{p_3 p_4}$ straddles $\overline{p_1 p_2}$.*
- *An endpoint of one segment lies on the other.*

4 cases for the Intersection Question



L intersects with L_3 (one point of L_3 lies on L) and with L_4 (both "straddle tests" succeed).

L does not intersect L_2 (only one of the "straddle tests" succeeds) or L_1.

Straddle Test

$\overline{p_1 p_2}$ straddles $\overline{p_3 p_4}$ if, and only if,

$$\big((p_1 - p_3) \times (p_4 - p_3)\big)\big((p_2 - p_3) \times (p_4 - p_3)\big) < 0.$$

Point on Segment

p_3 is on segment $\overline{p_1 p_2}$ if

$$(p_3 - p_1) \times (p_2 - p_1) = 0$$

and

$$\min(x_1, x_2) \leq x_3 \leq \max(x_1, x_2)$$

and

$$\min(y_1, y_2) \leq y_3 \leq \max(y_1, y_2)$$

The last two conditions simply say that p is in the rectangle with (diagonally opposite) corner points p_1, p_2

Solution of Problem (3) Completed

Algorithm SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)

1. $d_{12,3} \leftarrow (p_3 - p_1) \times (p_2 - p_1)$
2. $d_{12,4} \leftarrow (p_4 - p_1) \times (p_2 - p_1)$
3. $d_{34,1} \leftarrow (p_1 - p_3) \times (p_4 - p_3)$
4. $d_{34,2} \leftarrow (p_2 - p_3) \times (p_4 - p_3)$
5. **if** $d_{12,3} d_{12,4} < 0$ **and** $d_{34,1} d_{34,2} < 0$ **then return** TRUE
6. **else if** $d_{12,3} = 0$ **and** IN-BOX(p_1, p_2, p_3) **then return** TRUE
7. **else if** $d_{12,4} = 0$ **and** IN-BOX(p_1, p_2, p_4) **then return** TRUE
8. **else if** $d_{34,1} = 0$ **and** IN-BOX(p_3, p_4, p_1) **then return** TRUE
9. **else if** $d_{34,2} = 0$ **and** IN-BOX(p_3, p_4, p_2) **then return** TRUE
10. **else return** FALSE

Algorithm IN-BOX(p_1, p_2, p_3)

1. **return** $\min(x_1, x_2) \leq x_3 \leq \max(x_1, x_2)$
   and $\min(y_1, y_2) \leq y_3 \leq \max(y_1, y_2)$

The Convex Hull

Definition 2

1. A set C of points is *convex* if for all $p, q \in C$ the whole line segment $\overline{pq}$ is contained in C.
2. The *convex hull* of a set Q of points is the smallest convex set C that contains Q.

Observation 3

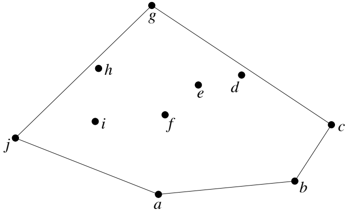
The convex hull of a finite set Q of points is a convex polygon whose vertices (corner points) are elements of Q.

The Convex Hull Problem

Input: *A finite set Q of points in the plane*

Output: *The vertices of the convex hull of Q in counterclockwise order.*

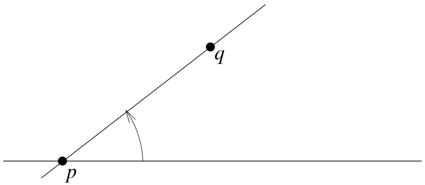
Example:



Output of a convex-hull algorithm: a, b, c, g, j

Polar Angles

The *polar angle* of a point q with respect to a point p is the (as usual anti-clockwise) angle between a horizontal line and the line through p and q.



Lemma 4

There is an algorithm that, given points $p_0, p_1, \ldots, p_n$, sorts $p_1, \ldots, p_n$ by non-decreasing polar angle with respect to p_0 in $O(n \lg n)$ time.

Graham's Scan

IDEA

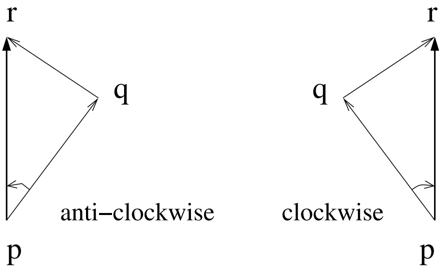
- Let p_0 be a "bottom-most" point in the set. Start walking around the points in the order of increasing polar angles.
- As long as you turn left, keep on walking.
- If you have to turn right to reach the next point, discard the current point and step back to the previous point. Repeat this until you can turn left to the next point.
- The points that remain are the vertices of the convex hull.

Turning Left (reminder)

Problem

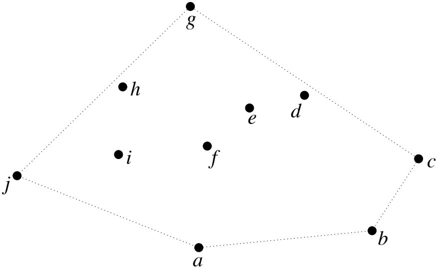
Given p, q, r in the plane, if we walk from $p \rightarrow q \rightarrow r$, do we make a left, a right, or no turn at q?

Solution



$(q - p) \times (r - p) = 0$: collinear segments — no turn.
$(q - p) \times (r - p) < 0$: right turn at q.
$(q - p) \times (r - p) > 0$: left turn at q.

Example (BOARD)



Implementation

Algorithm GRAHAM-SCAN(Q)

1. Let p_0 be the point in Q with minimum y coordinate.
   (if there is a tie, take the leftmost such point).
2. Sort $Q \setminus \{p_0\}$ “lexicographically” in terms of (primary key) non-decreasing polar angle with respect to p_0 and (secondary key) distance from p_0.

   For angles with more than one point, delete all corresponding points except the one farthest from p_0.

   Let $\langle p_1, \ldots, p_m \rangle$ be the resulting list..
3. **if** $m \leq 2$ **then return** $\langle p_0, \ldots, p_m \rangle$
4. **else** {
5.   Initialise stack S
6. S.PUSH(p_0)
7. S.PUSH(p_1)
8. S.PUSH(p_2)
9.   **for** $i \leftarrow 3$ **to** m **do**
10.   **while** the angle formed by the topmost two elements of S and p_i
                does not make a left turn **do**
11. S.POP
12. S.PUSH(p_i)
13.   **return** S
14. }

Analysis of Running time

Let $n = |Q|$, then $m \leq n$.

- Lines 3–8, 13 require time $\Theta(1)$.
- Line 1 requires time $\Theta(n)$ in the worst case.
- Line 2 requires time $\Theta(n \lg n)$.
- The outside (**for**) loop in lines 9–12 is iterated $m - 2$ times. Thus, disregarding the time needed by the inner **while** loop, the loop requires time $\Theta(m) = O(n)$.
- The inner loop in lines 10–11 is executed at most once for each element, because *every element enters the stack at most once and thus can only be popped once*. Thus overall the inner loop requires time $O(n)$.

Thus the overall worst-case running time is

$$\Theta(n \lg n).$$

Proof of Correctness

(I) First we consider the effect of executing lines 1 and 2 to get the (possibly smaller) set of points $P = p_0, p_1, \ldots, p_m$.

CLAIM (I): The convex hull of Q is equal to the convex hull of P.

Proof of CLAIM (I): We only discard a point $q \in Q$ if it *has the same polar angle wrt p_0 as some point $p_i \in P$, AND q is closer to p_0 than this p_i*. When q satisfies these 2 conditions, then q lies on $\overline{p_0 p_i}$. The convex hull of P by definition must contain $\overline{p_0 p_i}$ for every p_i, so the convex hull of P must contain q.
Applying this inductively (on the entire set of points removed) we find that the convex hull of P equals that of Q.

(II) Next we must prove that lines 3-14 compute the convex hull of $p_0, p_1, \ldots, p_m$.

If $m \leq 2$ then the alg returns all $m+1$ (1, 2, or 3) points (line 3). Correct.

Else $m > 2$ and the algorithm executes lines 5.-13.

For any $2 \leq i \leq m$, define C_i to be the convex hull of $p_0, \ldots, p_i$.

After executing lines 5.-8., the points on stack S are the vertices of C_2 (clockwise). We now prove that this situation holds for C_i after we execute the **for** loop with i.

Proof of Correctness ($m > 2$) cont'd

CLAIM (II): *Let i be such that $2 \leq i \leq m$. Then after the 'i'-execution of the* **for** *loop (lines 9-12), the points on S are the vertices of C_i in clockwise order.*

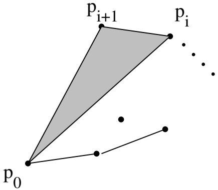
Proof of CLAIM (II): Our proof is by induction.

Base case (i=2): In this case there is no i-iteration of the loop. However, the stack holds p_0, p_1, p_2 (lines 6.-8.), which form the convex hull of $\{p_0, p_1, p_2\}$.

Induction hypothesis (IH): Assume CLAIM (II) holds for some i, $2 \leq i < m$.

Induction step: We will show CLAIM (II) also holds for $i + 1$.

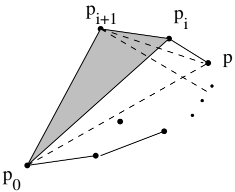
- Since the polar angle of p_{i+1} is *strictly greater* than the polar angle of p_i, therefore $p_0 p_i p_{i+1}$ forms a triangle *that is not contained* in C_i.



- Note p_{i+1} is NOT contained in C_i and thus is *definitely a vertex* of C_{i+1}.

Proof of Correctness ($m > 2$) cont'd

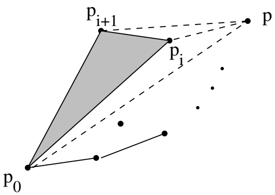
- By (IH) any q "'popped" so far is in the convex hull formed by the points currently on stack S ... $\Rightarrow$ ... the convex hull C_{i+1} is contained in the convex hull of p_{i+1} and the points on S.
- **Left:** First suppose the "next-to-top" point p on S, followed by the "top" point p_i, followed by p_{i+1} creates a "left turn":



  - Then the triangle $p_0 p p_{i+1}$ does *NOT* contain all of triangle $p_0 p_i p_{i+1}$
  - $\Rightarrow$ p_i must be on the Convex Hull C_{i+1}.
  - Using *convexity of the points on S*, $p_0 \longrightarrow \widehat{p} \longrightarrow p_{i+1}$ is a left turn for all points $\widehat{p}$ on S
  - $\Rightarrow$ all such $\widehat{p}$ must be on the Convex Hull C_{i+1}.
  - $\Rightarrow$ hence the decision to "push" p_{i+1} and leave all items of S there, correctly constructs C_{i+1}. $\Rightarrow$ CLAIM (II) **Left** proven.

Proof of Correctness ($m > 2$) cont'd

- **Right:** Otherwise suppose the "next-to-top" point p on S, followed by the "top" point p_i, followed by p_{i+1}, creates a "right turn":



  - Then the triangle $p_0 p p_{i+1}$ *does* contain all of triangle $p_0 p_i p_{i+1}$.
  - $\Rightarrow$ C_{i+1} does not need to include the point p_i.
  - $\Rightarrow$ decision to "pop" p_i (top item on S) on line 11 is correct. After the "pop", it is still true that the vertices of the convex hull C_{i+1} are from the set of points on S, together with p_{i+1}.
  - We can *apply this iteratively* by considering the "turn direction" of the top two items on the stack, $p^*, \widehat{p}$ say (taking the roles of p, p_i), followed by p_{i+1}, "popping" until there is a left turn.
  - *Once we find a left turn* slide 12 applies, and we push p_{i+1} onto S on line 12, to complete C_{i+1}. $\Rightarrow$ CLAIM (II) **right** proven.

Proof of Correctness ($m > 2$) cont'd

Wrapping up ...

- We have proven the inductive step for CLAIM (II).
- Hence CLAIM (II) holds after the consideration of every point $p_3, \ldots, p_m$, and in particular for $i = m$:
- $\Rightarrow$ *after the m-execution (the final execution) of the* **for**, *the points on the stack S are the vertices of C_m in clockwise order.*

  The vertices C_m are the vertices of the original set of points Q (by CLAIM (I)).

Hence Graham's scan computes the Convex Hull of its input correctly.

Optimality

- The best-known algorithm for finding the convex hull has a running time of $O(n \lg h)$, where h is the number of vertices of the convex hull.
- It can be shown (based on fairly natural assumptions) that every algorithm for finding the convex hull has a worst-case running time of

  $$\Omega(n \lg n).$$

  The *proof* of this lower bound is due to the fact that we can implement real-number sorting using Convex Hull.

Reading Assignment

Section 33.3 of [CLRS].

Problems

1. Exercises 33.3-3 and 33.3-5 of [CLRS].
2. Show how to sort a collection of n points by polar angle (wrt some lowest point p_0) in $O(n \lg(n))$ time, without using division or trigonometry.
3. Prove that the problem of finding the Convex Hull of n points has a lower bound of $\Omega(n \lg n)$. For this, think about using a reduction from sorting to Convex Hull (that is, think about how to use a Convex Hull algorithm to sort a list of numbers).