2D1353, Algorithms, Data structures and Complexity, 1998

Homework 7: Solutions

1. The algorithm works in two stages. The first stage goes on until the convex hull contains three points. At this stage the algorithm keeps track of the points on the convex hull so far (one or two points); an input point that coincides with a point on the convex hull or is on the line segment between two points on a two-point convex hull is immediately discarded. It is easy to see that each input point takes constant time at this stage.

   When the algorithm has a convex hull of three points (this happens after the three first points are processed, if they are not collinear), it picks some point P inside the triangular convex hull. For example, computing the center of gravity of the triangle will accomplish this. From now on, the points in the convex hull are stored in an ordered list, ordered by angle relative to P. The work done when switching to the second stage takes constant time.

   Now, for each of the following input points Q_{x_i}, the angle relative to P is calculated. Then, the algorithm looks up the two points on the convex hull that surrounds the new angle. To see if Q_{x_i} is on the convex hull one can check if P and Q_{x_i} are on different sides of the line connecting the two surrounding points.

   If Q_{x_i} is on the convex hull it is inserted in the ordered list of points. To make sure the polygon is still convex, one can ckeck the angle formed by Q_{x_i} and the next two consecutive points; if it points towards the center of the polygon the middle point is deleted. Repeat this in both directions until both angles point outwards.

   Notice that P will always remain inside the convex hull, so we never need to recompute any angles relative to some other point.

   We notice that if the list of points on the hull is kept in a balanced search tree, a point can be inserted and deleted from the list in time $O(\log n)$. Since at most n insertions and n deleteions are made, we obtain an $O(\log n)$ solution.

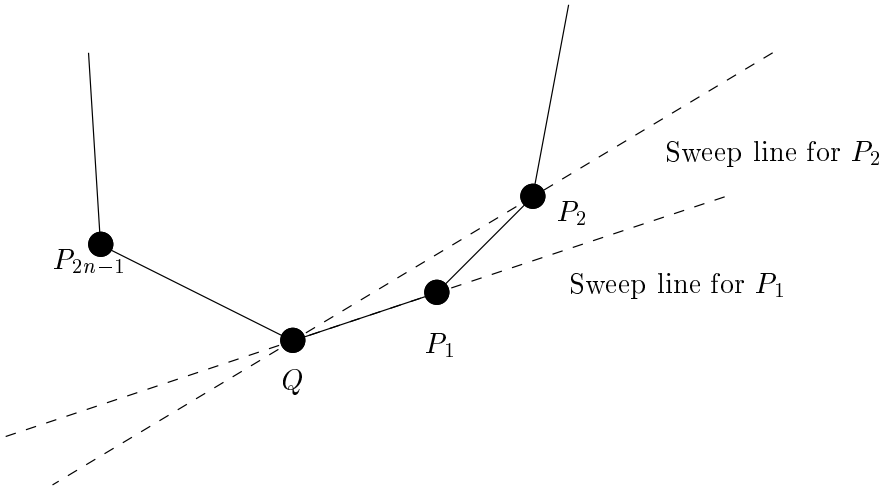
2. (a) We have n ghosts and n Ghostbusters (henceforth abbreviated *busters*) which we will think of as points in the plane.

   A *good* buster—ghost pair is one for which the number of ghosts on any side of the line equals the number of busters on the same side of the line. We want to prove that there always exists a good buster—ghost pair and that it can be found in $O(n \log n)$ time.

   Let Q be the point with the minimum y-coordinate. We assume that this point is a buster (the proof when Q is a ghost is anal-

ogous). For each of the $2n-1$ other points, compute the angle between the vector from Q to the point and the x-axis and sort the points with respect to the angles in ascending order. (As no three points are collinear, we do not have to worry about how to break ties.) This way we generate a sorted list $P_1, P_2, \ldots, P_{2n-1}$ containing all the points except for Q.

Note that Q is on the convex hull and that P_1 and P_{2n-1} are the vertices adjacent to Q on the convex hull. If P_1 is a ghost, the pair QP_1 will be good and if P_{2n-1} is a ghost the pair QP_{2n-1} will be good. From now on, assume that P_1 and P_{2n-1} are busters.



To prove the existence of a good buster—ghost pair we use an angular sweep-line technique. Let $g(i)$ and $b(i)$ denote the number of ghosts and busters below the line QP_i respectively. Let $d(i) = g(i) - b(i)$ be the difference between the number of ghosts and the number of busters below the line.

Consider the following table of function values:

i	$g(i)$	$b(i)$	$d(i)$
1	0	0	0
2	0	1	-1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2n-1$	n	$n-2$	2

The first row corresponds to the fact that there are no points below the line QP_i while the last row says that all the ghosts and all the busters except Q and P_{2n-1} are below the line QP_{2n-1}. The second row illustrates that the point P_1 (which is on the hull, and therefore a buster) is below the line QP_2.

Let us now compare the function values for i and $i+1$. The only

point which is below QP_{i+1} which is not also below QP_i is P_i itself, so we conclude that $g(i+1) = g(i)+1$ and $b(i+1) = b(i)$ if P_i is a ghost. If P_i is a buster, then $g(i+1) = g(i)$ and $b(i+1) = b(i)+1$.

The key observation is now the following: As $d(2) = -1 < 0$, $d(2n-1) = 2 > 0$ and $|d(i+1) - d(i)| = 1$ there must exist a j such that $d(j) = 0$.

Let j be the largest value of i such that $d(i) = 0$. Then the point P_j must be a ghost. Assume not: Then $g(j+1) = g(j)$ and $b(j+1) = b(j)+1$ and thus $d(j+1) = -1$. By the same argument as above, there will exist a $j' > j$ such that $d(j') = 0$ contradicting the assumption that j was the largest index for which $d(j) = 0$. The line QP_j is good as it connects a ghost and a buster with $b(i) = g(i)$ ghosts and busters below it.

The proof of existence above can easily be converted into an algorithm for finding the actual buster—ghost pair. Sorting the $2n-1$ points takes $O(n \log n)$ time. All the rest takes $O(n)$ time as a constant amount of work is done for each point. We conclude that a good buster—ghost pair can be found in $O(n \log n)$ time.

(b) We can use the algorithm from the first part of this problem to form buster-ghost pairs. Assume we have the function PAIR-AND-DIVIDE which takes as input a set of ghosts and busters and returns three sets. The first set contains the selected buster-ghost pair and the other two sets contain the points on either side of the line. These two sets are guaranteed to contain an even number of points, half of them being ghosts and half of them being busters.

Pseudocode for solving the problem:

FORM-PAIRS(P) // P is the set of ghosts and busters.

- If $|P| = 0$ do nothing and return.
- If $|P| = 2$ let the two points form a buster-ghost pair and return.
- (p,L,R) $\leftarrow$ PAIR-AND-DIVIDE(P)
- Form a buster-ghost pair from the pair p
- Call FORM-PAIRS(L) and FORM-PAIRS(R) recursively

The algorithm will terminate since each recursive call is done with a smaller P. Eventually each path in the recursion tree will reach a leaf where $|P| \leq 2$.

The PAIR-AND-DIVIDE function guarantees that there is no interaction between the two subsets L and R. Therefore, the line formed by the pair p will never be crossed. This argument also holds for pairs formed by calling FORM-PAIRS with $|P| = 2$. Hence, the resulting pairing will be good in the sense that there

are no crossed lines. (We could implement a special case for defeating Zuul...)

In each recursion, only a constant amount of work is done outside the PAIR-AND-DIVIDE function. There are $\leq 2n$ calls to FORM-PAIRS since each call either returns directly (when $|P| \leq 2$) or calls FORM-PAIRS twice with $|L| + |R| = |P| - 2$. However, PAIR-AND-DIVIDE is called at most $n - 1$ times since each call reduces the number of unpaired points by two. The resulting running time is thus $O(n) + (n - 1)O(n \log n)$ which is $O(n^2 \log n)$.

— *Gunnar Andersson, Henrik Ståhl, Staffan Ulfberg, March 19*