Line Sweep and Convex Hull

Kevin Geng*

16 December 2016

1 Intersecting line segments

Suppose we have N horizontal and vertical line segments. How can we find all points at which two of these intersect? One solution is to look at every pair of horizontal and vertical line segments, and determine whether they intersect by checking that the x coordinate of the vertical line falls within the horizontal line, and vice versa. This solution is $O(N^2)$.

However, we can do better by using a line sweep algorithm. Conceptually, imagine a line sweeping from left to right across the xy-plane. We will examine the line at certain x-coordinates we are interested in. We can think of these as *events* which we sort in order of increasing x-coordinate. For this problem, we are interested in:

1. A vertical segment
2. The start of a horizontal segment
3. The end of a horizontal segment

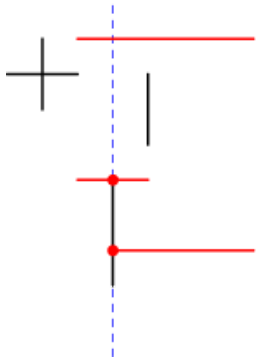

Figure 1: An example of a sweep line. *Credit: TopCoder.*

The key to line sweep is to have the sweep line hold information. We will keep an *active set* describing what horizontal segment are currently intersected by the sweep line. Whenever we encounter the start or end of a horizontal segment, we will add or remove it from the set. To implement this active set, we can use a binary search tree, and sort the horizontal segments by their y-coordinate.

Now, whenever we encounter a vertical line, we simply examine our active set. In the range of y-coordinates that the vertical line spans, is the sweep line currently intersecting any horizontal lines? If so, we have an intersection! Each insertion and deletion takes $O(logN)$. If we need to keep track of insertions, the total complexity is $O(NlogN + I)$.

1.1 More problems

- Solve the intersecting line segments problem, but for lines that are not necessarily horizontal or vertical.
- Given N points in two-dimensional space, find the two points that are closest together.
- Area of union of rectangles
- USACO 2013 November, Silver Problem 2: Crowded Cows

  Farmer John's N cows ($1 \leq N \leq 50,000$) are grazing along a one-dimensional fence. Cow i is standing at location $x(i)$ and has height $h(i)$ ($1 \leq x(i), h(i) \leq 1,000,000,000$).

  A cow feels "crowded" if there is another cow at least twice her height within distance D on her left, and also another cow at least twice her height within distance D on her right ($1 \leq D \leq 1,000,000,000$). Since crowded cows produce less milk, Farmer John would like to count the number of such cows. Please help him.

*Thanks to information on TopCoder and lectures created by Alex Chen and Hariank Muthakana.

- USACO 2013 February, Silver Problem 2: Square Overlap
- USACO 2012 February, Silver Problem 1: Overplanting

2 Convex hull

The convex hull of a set of points is the smallest convex polygon that can enclose all of them. One way to visualize this is to think of the points as pins, then imagine wrapping a rubber band around them.

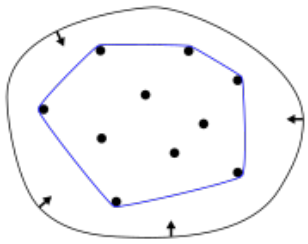

Figure 2: Analogy between a convex hull and an elastic band. *Credit: Wikipedia.*

2.1 Gift wrapping

This is the simplest way to find the convex hull. We start at the leftmost point in the set, p_0. At each step i, we find the angle from p_i to every other point, and pick p_{i+1} so that all points lie to the right of the line segment between p_{i-1} and p_i. We continue until we reach the original point. Since we examine n points for each of h elements in the convex hull, the complexity is $O(nh)$.

2.2 Graham scan

We can solve the convex hull problem using the *Graham scan* algorithm, which functions similar to a line sweep. First, we pick an "origin" point to start from, perhaps the one with the lowest y-coordinate. Then, we find the angles from that point to all other points, and sort the points in counterclockwise order.

Then we iterate through the points in that order. For each point p_i, we examine the angle it forms with the previous two points, $\angle p_i p_{i-1} p_{i-2}$. If the angle would result in a concave polygon, we delete p_{i-1} and check again. We can check this quickly by performing a cross product and examining its sign.

The initial sort has a complexity of $O(NlogN)$. The actual loop only has a complexity of $O(N)$, because each point is checked at most twice. So the overall complexity is $O(NlogN)$.

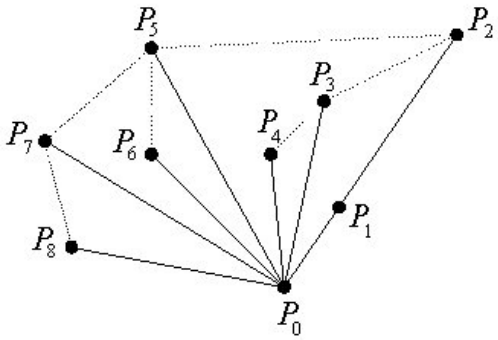

Figure 3: Representation of Graham scan algorithm. *Credit: geomalgorithms.com*