Line Sweep and Convex Hull

Richard Zhan*

November 2019

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

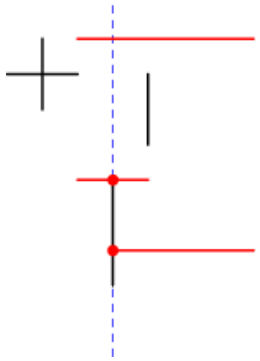

Figure 1: An example of a sweep line. *Credit: TopCoder.*

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.
- 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.

*based on Mihir Patel/Kevin Geng's lecture

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.

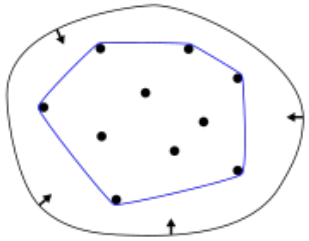

Figure 2: Analogy between a convex hull and an elastic band. *Credit: Wikipedia.*

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.

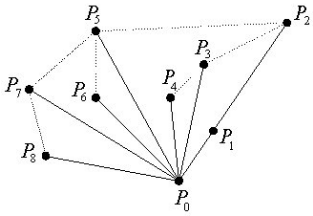

Figure 3: Representation of Graham scan algorithm. *Credit: geomalgorithms.com*

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. A concave polygon is formed if the angle is facing outwards. We can check this quickly by performing a cross product and examining its sign (right-hand rule!). The left-most point is taken as the origin, with ties being broken with preference to higher y values. 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 $O(NlogN)$.

2.3 Andrew's algorithm

Andrew's algorithm is another approach to solving the convex hull problem. This approach is also $O(nlogn)$ and creates the hull by building a top and bottom section. Firstly, all the points are sorted by their x coordinate. Then, a line is drawn to split the point that is the furthest left and the point that is furthest right. In the diagram to the right, this would be a line between 1 and 7. All points above the line are considered to build the upper hull and all points below are considered for the lower hull.

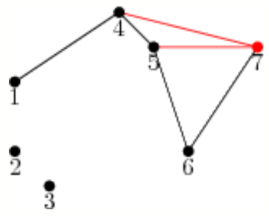

Figure 4: Andrew's algorithm for updating hull.

Let us assume that all points in the image to the right are for the upper hull. In order to construct the hull, we add each point from the left to the right in order. Whenever we add a point, we check to see if the point is strictly right of the line segment previously formed by the last two points. So in the image to the right, if we drew a ray from 5 to 6, we see that 7 is to the left of that ray (if we look from 5). If this happens, we delete the second to last point (6) and repeat.

To check which side of the line the point is on, we do $(x_1 - x_0) * (y_2 - y_0) - (x_2 - x_0) * (y_1 - y_0)$ where p_0 is the second to last point, p_1 is the last point, and p_2 is the current point. If this is positive, the last point is to the left of the line and vice-versa. This is derived by taking the cross product of the vectors from p_0 to p_1 and p_0 to p_2 and is how we calculate the signed area of a triangle, which tells us if the points are clockwise or anti-clockwise.

When we have iterated through all the points, we have constructed the entire top hull. We do the same for the lower region, except now we check to see if the point is left of the line and then merge the hulls.