Sweeping Algorithms

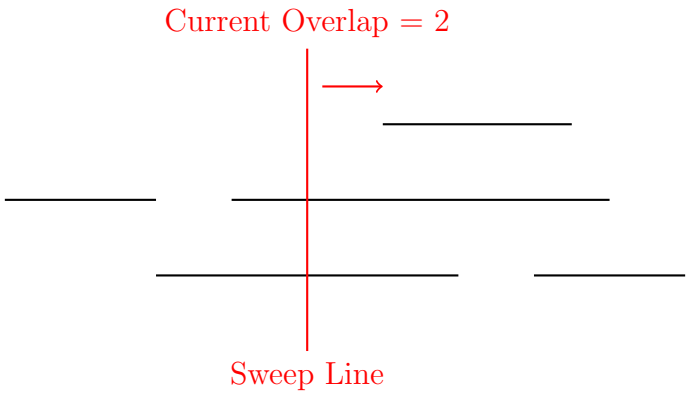
Sweeping algorithms are a common technique used in computational geometry that relies on the fact that certain physical properties can be ordered to drastically save time.

Line Sweep

In this technique, we are usually given a list of segments and asked to prove something about them. Let us look at an example problem of finding the location of the maximum overlap between a bunch of line segments:

> Given a list of segments all on the same line in the form of l_i, r_i where l_i represents the left endpoint of the i^{th} segment and r_i represents its right endpoint, find the maximum number of lines overlapped with each other.

Sweeping algorithms relies on 2 things: a sweep line and a bunch of events. To solve this problem we imagine a vertical sweep line moving from left to right. When it encounters an event (either enters a segment on a l_i or leaves a segment on a r_i), we will update the counter of how many lines the vertical sweep line is crossing at any given point (i.e. how many lines overlap at a given point), and in that process we can find the maximum number of lines that overlap with each other.



On an implementation note, the way one would actually go about doing this would be to turn the line segments into a list of **events**, which can either have a role of activate or deactivate (corresponds to l or r index respectively). We can then sort this list by the coordinates and go trough the sorted list, which is equivalent to moving the sweep line from left to right. Every time we encounter an activate event we increment our counter by one, and every time we encounter a deactivate event we decrement our counter by one. This way by the end of the list we will know the maximum overlaps. This procedure takes $O(n \log n + n)$ time, first to sort the list then to access it, so the overall time is $O(n \log n)$.

A more complex example involves given a group of lines and finding all lines that intersect. Formally, given a list of (x_1, y_1, x_2, y_2) coordinates marking the endpoints of several segments,

and assuming that no three intersect at the same point, determine if any of the two lines intersect. For this problem we will once again use a sweep line form left to right and use left and right endpoints of segments as events. However, what we do at each event is different. We keep a list of active segments, which are segments that the sweep line intersects for any given x value, whenever the sweep line encounters an event, we first update the set of active segments and sort them by their y-values. In order for two segments to intersect. The order of their x values must be the opposite of the order of their y values, and for two to intersect, at some event they must be adjacent to each other on the ordering by y coordinates. Therefore our algorithm only needs to check for intersection of the segments right above and below the segment of the event. This reduces the amount of checking needed to $O(n \log n)$ instead of $O(n^2)$.

Radial Sweep

Radial sweeps use a similar concept except instead of having a vertical sweep line move from left to right we have an rotating sweep line to sweep either clockwise or counterclockwise. Again the line is imaginary, what we actually do is have a list of events then sort them based on the angle. This technique can be used to find the maximum number of collinear points in a set of points. We can do this based off of the fact that if three points are collinear then from one of the points, the angle to the other two would be the same. Therefore, to find the maximum number of collinear points, we go to each point, compute its angle to every other point, then sort that list (effectively performing a radial sweep), then we can go through that list and count how many elements are collinear (we can do this in n time). Therefore we can do this in $O(n^2 \log n)$ time, which is pretty good.