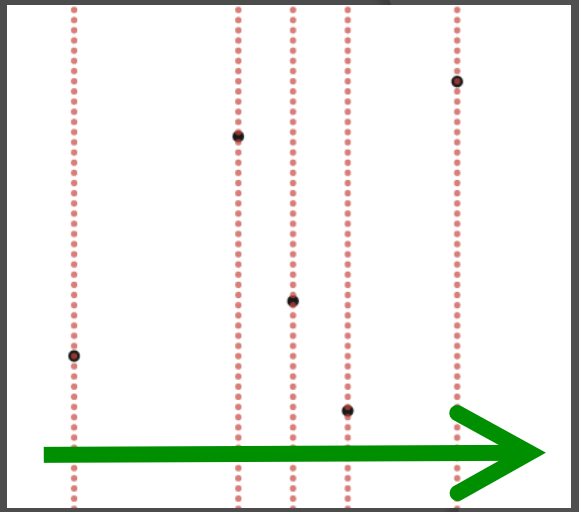
Advanced Computer Contest Preparation

Lecture 21

LINE SWEEP

What is Line Sweep?

- A fundamental concept in computational geometry
- Based around the concept of a *sweep line*
  - A line, normally vertical, that is "swept" across a plane
- Plane has infinite number of points, so only certain points are considered
  - Points are normally visited in order of non-decreasing x coordinate

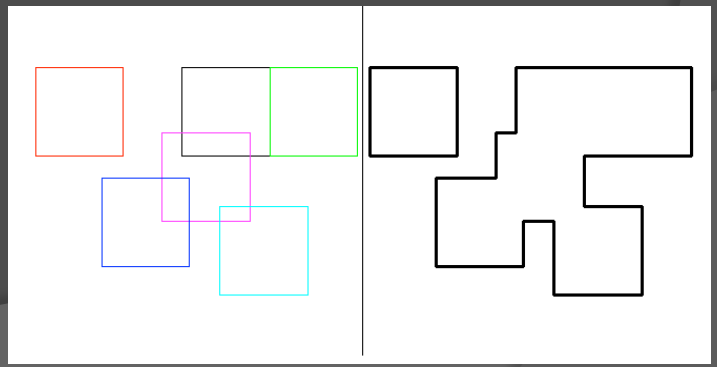


Common Line Sweep Steps

- Identify significant points
- When a point is visited, an *event* occurs
  - Determine what should be done when a point is visited
- Sort points by non-decreasing x coordinate
  - Additional sorting parameters might be required when ties occur
    - e.g. a certain event should go before another when a tie occurs
- Iterate through points and handle events

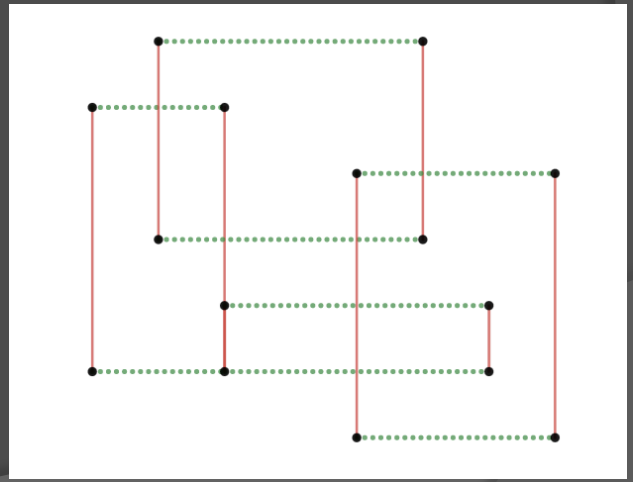
Area of Union of Rectangles

- You are given the top left and bottom right coordinates of N rectangles on a plane
- What is the area enclosed by all of the rectangles?
  - Note: if more than one rectangle covers the same point, it is only counted once



Line Sweep Solution

- What are the significant points?
  - Every x coordinate of the vertices of every rectangle
  - All locations of vertical lines
- What are the events?
  - Left end of a rectangle
  - Right end of a rectangle



Line Sweep Solution

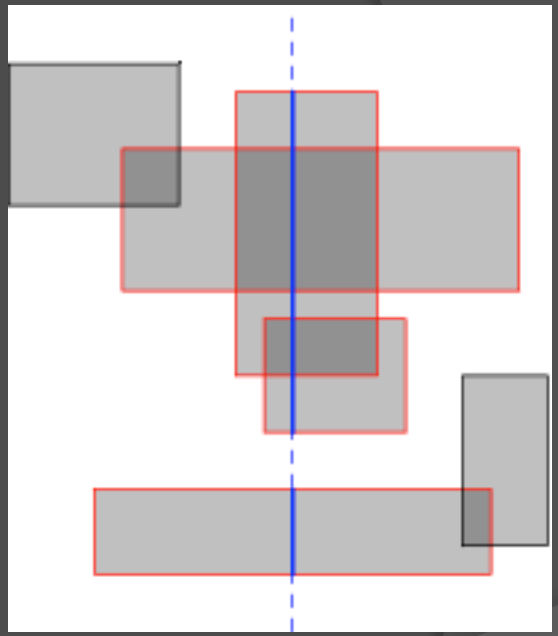
- Compress the y coordinates
  - Index to y coordinate: `y[]`
  - y coordinate to index: `fromY[]`
- Let `ar[i]` be the number of rectangles in between [`y[i]`,`y[i+1]`]
- Left end event
  - Add 1 to each `ar[i]` from i = `fromY[bottom]` until `fromY[top]-1`
- Right end event
  - Subtract 1 to each `ar[i]` from i = `fromY[bottom]` until `fromY[top]-1`

Line Sweep Solution

- Before processing events on a new x coordinate, query area and add to answer
  - If `ar[i]` > 0
    - Add `(newX-prevX)*(y[i+1]-y[i])` to total answer
    - `ar[i]` > 0 means there is at least 1 rectangle
- Order of processing events in case of a tie does not matter

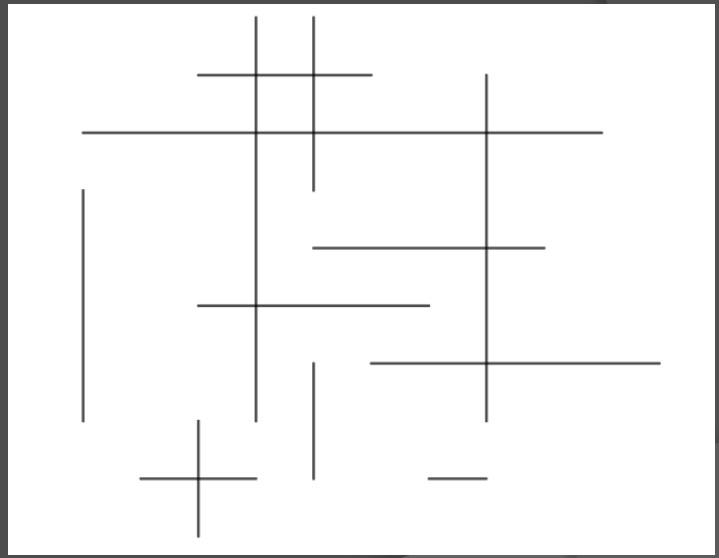
Line Sweep Solution - Analysis

- Sorting points: $O(N \log N)$
- Compressing y coordinates: $O(N \log N)$
- Sweeping: $O(N)$
- Updating array:
  - Regular array: $O(N)$
  - Difference array: $O(1)$
- Querying area: $O(N)$
- Total time complexity: $O(N^2)$
- Using more efficient data structures, we can reduce total time complexity to $O(N \log N)$



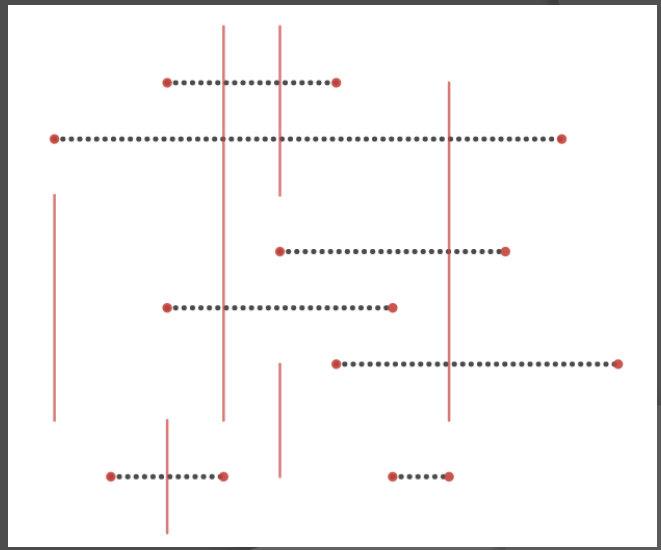
Axis-Aligned Line Intersection

- Given N lines, each either parallel to the x or y axis, how many intersections are there?
- To simplify, no two lines will intersect at either line's end points, parallel lines will not intersect
- Naive algorithm: Count the number of other lines each line intersects with
  - Time Complexity: $O(N^2)$
- Can you determine a line sweep solution?



Line Sweep Solution

- What are significant points?
  - Every end point
- What are the events?
  - Left end point of a horizontal line
  - Right end point of a horizontal line
  - Vertical line



Line Sweep Solution

- Compress y coordinates of horizontal lines
  - Index to y coordinate: `y[]`
- Let `ar[i]` represent how many horizontal lines are currently at y coordinate `y[i]`
  - Since no intersections at endpoints, `y[i]` will only ever be 0 or 1
- Left endpoint event
  - `ar[i]++`, where `y[i]` = point's y coordinate
- Right endpoint event
  - `ar[i]--`, where `y[i]` = point's y coordinate

Line Sweep Solution

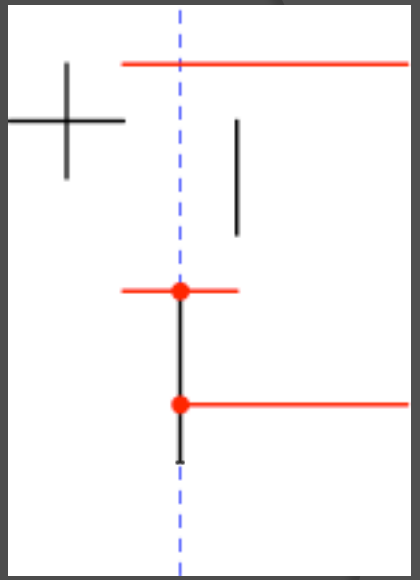
- Vertical line event
  - Query the number of horizontal lines in between the endpoints of the vertical line using `ar`
  - `ans += ar[l] + ar[l+1] +...+ ar[r]`
  - `l` determined by smallest `y[l]` ≥ lower endpoint
  - `r` determined by largest `y[r]` ≤ upper endpoint

Line Sweep Solution

- Order of processing events
  - Since intersections are guaranteed to not occur at a line's endpoints, order does not matter in case of a tie
- If this is not guaranteed, what should the order be?
  - Left end, query, right end

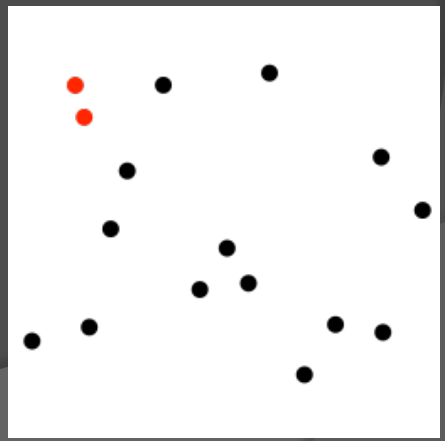
Line Sweep Solution: Analysis

- Sorting x coordinates: $O(N \log N)$
- Compressing y coordinates: $O(N \log N)$
- Sweeping: $O(N)$
- Updating array: $O(1)$
- Querying array: $O(N)$
- Total time complexity: $O(N^2)$
- Using more efficient data structures, we can reduce total time complexity to $O(N \log N)$



Closest Pair of Points

- Given N points on a plane, which 2 points are the closest to each other?
  - Euclidean, or straight-line, distance
- Naive algorithm: check every pair of two points, get Euclidean distance
  - Time complexity: $O(N^2)$
- Can we do better?



Line Sweep Solution

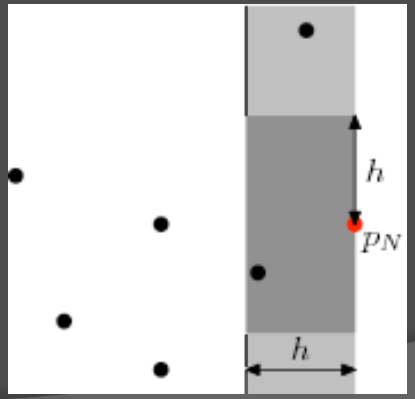
- Sort points by x coordinate
- Use 2 sets of points: one ordered by x coordinate (**`sx`**) and one ordered by y coordinate (**`sy`**)
- We are at point p, and the current shortest distance is h
- Remove points from **`sx`** and **`sy`** that have an x coordinate $< p_x - h$
  - Use **`sx`** to determine which points to remove from **`sy`**

Line Sweep Solution

- Iterate through points in **`sy`** with p_y-$h \leq y$ coordinate $\leq p_y$+h and get distance to p
  - There are on average $O(1)$ points that are iterated over each time
- Update h if smaller distance is found
- Add p to **`sx`** and **`sy`**
- Time complexity: $O(N \log N)$

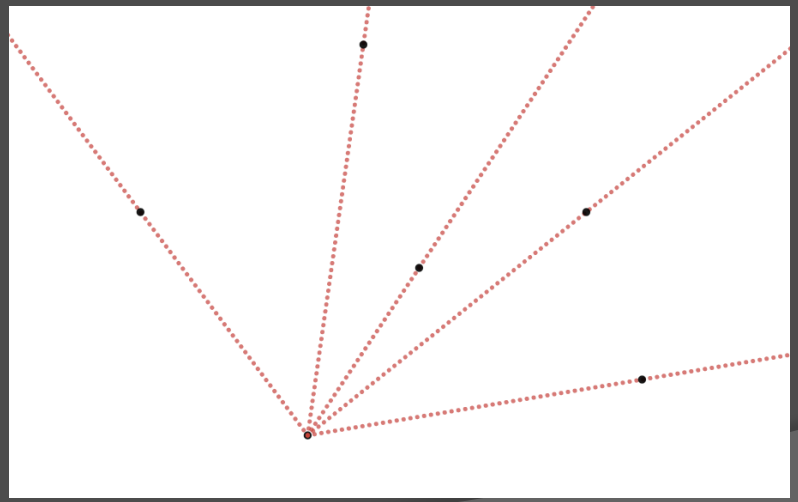
Line Sweep Solution

- As we sweep through, our current answer is h
- All points in the gray region are in **`sx`** and **`sy`**
- All points in the dark gray region will be iterated over



Radial Sweep

- Instead of a vertical line sweeping across a plane in the positive x direction, a ray sweeps around the plane in a clockwise or counterclockwise direction



THANK YOU!