Segment Intersection

Piotr Indyk
Segment intersection

- Segment intersection problem:
  - Given: a set of $n$ distinct segments $s_1 \ldots s_n$, represented by coordinates of endpoints
  - Detection: detect if there is any pair $s_i \neq s_j$ that intersects
  - Reporting: report all pairs of intersecting segments
Segment intersection

- Easy to solve in $O(n^2)$ time
- Is it possible to get a better algorithm for the reporting problem?
- NO (in the worst-case)
- However:
  - We will see we can do better for the detection problem
  - Moreover, the number of intersections $P$ is usually small.

Then, we would like an output sensitive algorithm, whose running time is low if $P$ is small.
Result

• We will show:
  – $O(n \log n)$ time for detection
  – $O((n + P) \log n)$ time for reporting

• We will use line sweep approach
  – Many other applications, e.g., Voronoi diagrams, motion planning
Orthogonal segments

- All segments are either horizontal or vertical
- Assumption: all coordinates are distinct
- Therefore, only vertical-horizontal intersections exist
Orthogonal segments

- **Sweep line:**
  - A *vertical line* sweeps the plane from left to right
  - It “stops” at all “important” x-coordinates, i.e., when it hits a V-segment or endpoints of an H-segment
  - Invariant: all intersections on the left side of the sweep line have been already reported
• We maintain sorted y-coordinates of H-segments currently intersected by the sweep line (using a balanced BST $V$)
• When we hit the left point of an H-segment, we add its y-coordinate to $V$
• When we hit the right point of an H-segment, we delete its y-coordinate from $V$
Orthogonal segments ctd.

- Whenever we hit a V-segment having coord. $y_{\text{top}}, y_{\text{bot}}$, we report all H-segments in $V$ with y-coordinates in $[y_{\text{top}}, y_{\text{bot}}]$.
Algorithm

• Sort all V-segments and endpoints of H-segments by their x-coordinates – this gives the “trajectory” of the sweep line
• Scan the elements in the sorted list:
  – Left endpoint: add segment to tree $V$
  – Right endpoint: remove segment from $V$
  – V-segment: report intersections with the H-segments stored in $V$
Analysis

• Sorting: $O(n \log n)$
• Add/delete H-segments to/from vertical data structure $V$:
  – $O(\log n)$ per operation
  – $O(n \log n)$ total
• Processing V-segments:
  – $O(\log n)$ per intersection - SEE NEXT SLIDE
  – $O(P \log n)$ total
• Overall: $O((P + n) \log n)$ time
• Can be improved to $O(P + n \log n)$ - SEE NEXT SLIDE
Analyzing intersections

• Given:
  – A BST $V$ containing $y$-coordinates
  – An interval $I=\left[y_{\text{bot}},y_{\text{top}}\right]$
• Goal: report all $y$’s in $V$ that belong to $I$
• Basic algorithm:
  • $y=\text{Successor}(y_{\text{bot}})$
  • While $y\leq y_{\text{top}}$
    – Report $y$
    – $y:=\text{Successor}(y)$
  • End
  – Time: (number of reported $y$’s)$\times O(\log n) + O(\log n)$
• Faster algorithm:
  • For each $y$ in the tree, store a link to its successor
  • Recompute the successor after each insertion/deletion
  – Time: (number of reported $y$’s) $+ O(\log n)$
The general case

• Assumption: all coordinates of endpoints and intersections distinct

• In particular:
  – No vertical segments
  – No three segments intersect at one point
Sweep line

- Invariant (as before): all intersections on the left of the sweep line have been already reported
- Stops at all “important” x-coordinates, i.e., when it hits endpoints or intersections
- Problem I: Do not know the x-coordinates of intersections in advance!
- The list of intersection coordinates is constructed and maintained dynamically
  (in a “horizontal” data structure H)
Sweep line

- Also need to maintain the information about the segments intersecting the sweep line
- Problem II: Cannot keep the values of y-coordinates of the segments!
- Instead, we will maintain their order. I.e., at any point, we maintain all segments intersecting the sweep line, sorted by the y-coordinates of the intersections (in a “vertical” data structure V)

- Key idea: check only for the intersections of segments that are neighbors in V
Java applet

- Example used in the lecture:
Algorithm

- Initialize the “vertical” BST $V$ (to “empty”)
- Initialize the “horizontal” priority queue $H$ (to contain the segments’ endpoints sorted by x-coordinates)
- Repeat
  - Take the next “event” $p$ from $H$:
    // Update $V$
    - If $p$ is the left endpoint of a segment, add the segment to $V$
    - If $p$ is the right endpoint of a segment, remove the segment from $V$
    - If $p$ is the intersection point of $s$ and $s'$, swap the order of $s$ and $s'$ in $V$, report $p$

(continued on the next slide)
Algorithm ctd.

// Update H
– For each new pair of neighbors $s$ and $s'$ in $V$:
  • Check if $s$ and $s'$ intersect on the right side of the sweep line
  • If so, add their intersection point to $H$
  • Remove the possible duplicates in $H$
– Until $H$ is empty
Analysis

- Initializing $H$: $O(n \log n)$
- Updating $V$:
  - $O(\log n)$ per operation
  - $O((P+n) \log n)$ total
- Updating $H$:
  - $O(\log n)$ per intersection
  - $O(P \log n)$ total
- Overall: $O((P+n) \log n)$ time
Correctness

• All reported intersections are correct
• Assume there is an intersection not reported. Let $p=(x,y)$ be the first such unreported intersection (of $s$ and $s'$)
• Let $x'$ be the last event before $p$. Observe that:
  – At time $x'$ segments $s$ and $s'$ are neighbors on the sweep line
  – Since no intersections were missed till then, $V$ maintained the right order of intersecting segments
  – Thus, $s$ and $s'$ were neighbors in $V$ at time $x'$. Thus, their intersection should have been detected
Optimality

• Is $O((P+n)\log n)$ optimal?
• No, one can get $O(P + n\log n)$: