Interval Trees and Segment Trees

slides by Andy Mirzaian

(a subset of the original slides are used here)
References:

• [M. de Berge et al] chapter 10

Data Structures:

• Interval Trees
• Priority Search Trees
• Segment Trees

Applications:

• Windowing Queries
• Vehicle navigation systems
• Geographic Information Systems
• Flight simulation in computer graphics
• CAD/CAM of printed circuit design
PROBLEM 1: Preprocess a set $S$ of non-crossing line-segments in the plane for efficient query processing of the following type:

**Query:** given an axis-parallel rectangular query window $W$, report all segments in $S$ that intersect $W$. 
PROBLEM 2: Preprocess a set S of horizontal or vertical line-segments in the plane for efficient query processing of the following type:

Query: given an axis-parallel rectangular query window W, report all segments in S that intersect W.
PROBLEM 2: Preprocess a set $S$ of horizontal or vertical line-segments in the plane for efficient query processing of the following type:

**Query:** given an axis-parallel rectangular query window $W$, report all segments in $S$ that intersect $W$. 
**SUB-PROBLEM 1.1 & 2.1:**
Let S be a set of n line-segments in the plane. Given an axis-parallel query window W, the segments of S that have at least one end-point inside W can be reported in $O(K + \log n)$ time with a data structure that uses $O(n \log n)$ space and $O(n \log n)$ preprocessing time, where K is the number of reported segments.

Method:
Use 2D Range Tree on segment end-points and fractional cascading.
INTERVAL TREES

Now consider horizontal (similarly, vertical) segments in $S$ that intersect $W$, but their end-points are outside $W$. They must all cross the left edge of $W$.

**SUB-PROBLEM 2.2:**
Preprocess a set $S_H$ of horizontal line-segments in the plane, so that the subset of $S_H$ that intersects a query vertical line can be reported efficiently.

Method: Use Interval Trees.
INTERVAL TREES

Associated structure for $I_{\text{med}}$:

- $\mathcal{L}_{\text{left}}$ = list of segments in $I_{\text{med}}$ sorted by their left end-points,
- $\mathcal{L}_{\text{right}}$ = list of segments in $I_{\text{med}}$ sorted by their right end-points.

For example:
- $\mathcal{L}_{\text{left}} = 3,4,5$
- $\mathcal{L}_{\text{right}} = 5,3,4$
- $\mathcal{L}_{\text{left}} = 1,2$
- $\mathcal{L}_{\text{right}} = 1,2$
- $\mathcal{L}_{\text{left}} = 6,7$
- $\mathcal{L}_{\text{right}} = 7,6$
**THEOREM:** Interval Tree for a set of $n$ horizontal intervals:
- $O(n)$ storage space
- $O(n \log n)$ construction time
- $O(K + \log n)$ query time

[report all $K$ data intervals that contain a query **x-coordinate**.]
SUB-PROBLEM 2.3:

Now instead of the query being on a vertical line, suppose it is on a vertical **line-segment**.

The primary structure of Interval Trees is still valid. Modify the associated secondary structure.

**SOLUTION:**

\[ \mathcal{L}_{\text{left}} = \text{Range Tree on left end-points of } I_{\text{med}}, \]
\[ \mathcal{L}_{\text{right}} = \text{Range Tree on right end-points of } I_{\text{med}}. \]
**THEOREM:** Interval Tree for a set of n horizontal intervals:
- $O(n \log n)$ storage space
- $O(n \log n)$ construction time
- $O(K + \log^2 n)$ query time

[report all K data intervals that intersect a query **vertical line-segment**.]

**COROLLARY:** Let S be a set of n horizontal or vertical line-segments in the plane. We can preprocess S for axis-parallel rectangular query window intersection with the following complexities:
- $O(n \log n)$ storage space
- $O(n \log n)$ construction time
- $O(K + \log^2 n)$ query time

[report all K data intervals that intersect the query **window**.]
Improving the previous solution: the associated structure can be implemented by Priority Search Trees, instead of Range Trees.

\[ P = \{ p_1, p_2, \ldots, p_n \} \subseteq \mathbb{R}^2. \]

A Priority Search Tree (PST) \( T \) on \( P \) is:

- a binary tree, one point per node,
- heap-ordered by \( x \)-coordinates,
- (almost) symmetrically ordered by \( y \)-coordinates.
PRIORITY SEARCH TREES

\( p_{\text{min}} \leftarrow \text{point in } P \text{ with minimum } x\text{-coordinate.} \)

\( y_{\text{min}} \leftarrow \text{min } y\text{-coordinate of points in } P \)

\( y_{\text{max}} \leftarrow \text{max } y\text{-coordinate of points in } P \)

\( P' \leftarrow P - \{ p_{\text{min}} \} \)

\( y_{\text{med}} \leftarrow \text{y-median of points in } P' \)

\( P_{\text{below}} \leftarrow \{ p \in P' \mid p_y \leq y_{\text{med}} \} \)

\( P_{\text{above}} \leftarrow \{ p \in P' \mid p_y > y_{\text{med}} \} \)

\[ p_{\text{min}} \]

\[ y_{\text{min}}, y_{\text{max}} \]

\[ P_{\text{below}} \]

\[ P_{\text{above}} \]

\[ p_1 \]

\[ y_6 \quad y_3 \]

\[ p_2 \]

\[ y_4 \quad y_2 \]

\[ p_3 \]

\[ y_7 \quad y_7 \]

\[ p_4 \]

\[ y_4 \quad y_4 \]

\[ p_5 \]

\[ y_5 \quad y_5 \]

\[ p_6 \]

\[ y_6 \quad y_6 \]

\[ p_7 \]

\[ y_7 \quad y_7 \]

\[ p_{\text{min}} \]

\[ y_{\text{med}} \]

\[ p_1 \]

\[ p_2 \]

\[ p_3 \]

\[ p_4 \]

\[ p_5 \]

\[ p_6 \]

\[ p_7 \]
Priority Search Tree $\mathcal{T}$ on $n$ points in the plane requires:

- $O(n)$ storage space
- $O(n \log n)$ construction time:
  
  > either recursively, or
  
  > pre-sort $P$ on $y$-axis, then construct $\mathcal{T}$ in $O(n)$ time bottom-up. (How?)

*Priority Search Trees* can replace the secondary structures (range trees) in *Interval Trees*.

- simpler (no fractional cascading)
- linear space for secondary structure.
How to use PST to search for a query range \( R = (-\infty: q_x] \times [q_y : q'_y] \)?

**ALGORITHM** \( \text{QueryPST} (v, R) \)

1. **if** \( v = \text{nil} \) \text{ or } \( p_{\min_x}(v) > q_x \) \text{ or } \( y_{\min}(v) > q'_y \) \text{ or } \( y_{\max}(v) < q_y \)
   **then** return
2. **if** \( p_{\min_x}(v) \leq q_x \) \text{ and } \( q_y \leq y_{\min}(v) \leq y_{\max}(v) \leq q'_y \)
   **then** \( \text{Report.In.Subtree} (v, q_x) \)
3. **else do**
   **if** \( p_{\min_x}(v) \in R \) \text{ then } report \( p_{\min_x}(v) \)
   \( \text{QueryPST} (l_c(v), R) \)
   \( \text{QueryPST} (r_c(v), R) \)
4. **end else**
5. **end**

**PROCEDURE** \( \text{Report.In.Subtree} (v, q_x) \)

1. **if** \( v = \text{nil} \) \text{ then return}
2. **if** \( p_{\min_x}(v) \leq q_x \) \text{ then do}
   **report** \( p_{\min_x}(v) \)
   \( \text{Report.In.Subtree} (l_c(v), q_x) \)
   \( \text{Report.In.Subtree} (r_c(v), q_x) \)
3. **end if**
4. **end**

Truncated Pre-Order on the Heap: \( O(1 + K_v) \) time.
**LEMMA:** Report.In.Subtree(v, q_x) takes $O(1 + K_v)$ time to report all points in the subtree rooted at $v$ whose $x$-coordinate is $\leq q_x$, where $K_v$ is the number of reported points.

**THEOREM:** Priority Search Tree for a set $P$ of $n$ points in the plane has complexities:

- $O(n)$ Storage space
- $O(n \log n)$ Construction time
- $O(K + \log n)$ Query time

[report all $K$ points of $P$ in a query range $R = (-\infty: q_x] \times [q_y : q'_y]$ .]
Back to Problem 1: Arbitrarily oriented line segments.

Solution 2: Use Segment Trees.
   a) Segments with end-points in W can be reported using range trees (as before).
   b) Segments that intersect the boundary of W can be reported by Segment Trees.

SUB-PROBLEM 1.1: Preprocess a set S of n non-crossing line-segments in the plane into a data structure to report those segments in S that intersect a given *vertical* query segment $q = q_x \times [q_y : q'_y]$ efficiently.
### Elementary x-intervals of S

\[ (-\infty : p_1), [p_1 : p_1], (p_1 : p_2), [p_2 : p_2], \ldots, (p_{m-1} : p_m), [p_m : p_m], (p_m : +\infty). \]

Build a balanced search tree with each **leaf** corresponding (left-to-right) to an elementary interval (in increasing x-order).

**Leaf v:**

**Int(v) =** set of intervals (in S) that contain the elementary interval corresponding to v.

### IDEA 1: Store Int(v) with each leaf v.

**Storage O(n^2),** because intervals in S that span many elementary intervals will be stored in many leaves.
IDEA 2:
∀ internal node v:
Int(v) = union of elementary intervals corresponding to the leaf-descendents of v.

Store an interval [x:x'] of S at a node v iff Int(v) ⊆ [x:x'] but Int(parent(v)) ⊈ [x:x'].
Each interval of S is stored in at most 2 nodes per level (i.e., O(log n) nodes).
Thus, storage space reduces to O(n log n).

What should the associated structure be?
SEGMENT TREES

\[ S(v_1) = \{s_3\} \]
\[ S(v_2) = \{s_1, s_2\} \]
\[ S(v_3) = \{s_5, s_7\} \]

\[
S(v_1) = \{s_3\}
\]
\[
S(v_2) = \{s_1, s_2\}
\]
\[
S(v_3) = \{s_5, s_7\}
\]

Diagram:

- \( v_1 \)
- \( v_2 \)
- \( v_3 \)

Lines:

- \( s_1 \)
- \( s_2 \)
- \( s_3 \)
- \( s_4 \)
- \( s_5 \)
- \( s_6 \)
- \( s_7 \)
**SEGMENT TREES**

**Associated structure**

is a balanced search tree based on the vertical ordering of segments $S(v)$ that cross the slab $\text{Int}(v) \times (-\infty : +\infty)$. 

![Diagram of segment tree and vertical segments](image)
THEOREM:
Segment Tree for a set $S$ of $n$ non-crossing line-segments in the plane:

- $O(n \log n)$ Storage space
- $O(n \log n)$ Construction time
- $O(K + \log^2 n)$ Query time

[report all $K$ segments of $S$ that intersect a vertical query line-segment.]

COROLLARY:
Segment Trees can be used to solve Problem 1 with the above complexities. That is, the above complexities applies if the query is with respect to an axis-parallel rectangular window.