Computational Geometry · Lecture
Range Searching II: Windowing Queries

Tamara Mchedlidze · Chih-Hung Liu
23.11.2015
Object types in range queries

Setting so far:

- **Input**: set of points $P$
  (here $P \subset \mathbb{R}^2$)
- **Output**: all points in $P \cap [x, x'] \times [y, y']$
- **Data structures**: $kd$-trees or range trees
Object types in range queries

Setting so far:
- Input: set of points $P$ (here $P \subset \mathbb{R}^2$)
- Output: all points in $P \cap [x, x'] \times [y, y']$
- Data structures: $kd$-trees or range trees

Further variant
- Input: set of line segments $S$ (here in $\mathbb{R}^2$)
- Output: all segments in $S \cap [x, x'] \times [y, y']$
- Data structures: ?
Axis-parallel line segments

special case (e.g., in VLSI design): all line segments are axis-parallel
Axis-parallel line segments

Problem:

Given $n$ vertical and horizontal line segments and an axis-parallel rectangle $R = [x, x'] \times [y, y']$, find all line segments that intersect $R$. 

special case (e.g., in VLSI design): all line segments are axis-parallel
Axis-parallel line segments

special case (e.g., in VLSI design): all line segments are axis-parallel

Problem:
Given $n$ vertical and horizontal line segments and an axis-parallel rectangle $R = [x, x'] \times [y, y']$, find all line segments that intersect $R$.

How to approach this case?
Axis-parallel line segments

special case (e.g., in VLSI design): all line segments are axis-parallel

Problem:

Given \( n \) vertical and horizontal line segments and an axis-parallel rectangle \( R = [x, x'] \times [y, y'] \), find all line segments that intersect \( R \).

Case 1: \( \geq 1 \) endpoint in \( R \)
→ use range tree

Case 2: both endpoints \( \notin R \)
→ intersect left or top edge of \( R \)
Case 2 in detail

**Problem:**
Given a set $H$ of $n$ horizontal line segments and a vertical query segment $s$, find all line segments in $H$ that intersect $s$. (Vertical segments and a horizontal query are analogous.)
Case 2 in detail

Problem:
Given a set $H$ of $n$ horizontal line segments and a vertical query segment $s$, find all line segments in $H$ that intersect $s$. (Vertical segments and a horizontal query are analogous.)

One level simpler: vertical line $s := (x = q_x)$

Given $n$ intervals $I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$ and a point $q_x$, find all intervals that contain $q_x$. 

\[ \text{Interval diagram} \]
Case 2 in detail

Problem:
Given a set $H$ of $n$ horizontal line segments and a vertical query segment $s$, find all line segments in $H$ that intersect $s$. (Vertical segments and a horizontal query are analogous.)

One level simpler: vertical line $s := (x = q_x)$

Given $n$ intervals $I = \{[x_1, x'_1], [x_2, x'_2], \ldots, [x_n, x'_n]\}$ and a point $q_x$, find all intervals that contain $q_x$.

What do we need for an appropriate data structure?
Interval Trees

Construction of an interval tree $\mathcal{T}$

- if $I = \emptyset$ then $\mathcal{T}$ is a leaf
- else let $x_{\text{mid}}$ be the median of the endpoints of $I$ and define
  
  $$I_{\text{left}} = \{ [x_j, x'_j] \mid x'_j < x_{\text{mid}} \}$$
  $$I_{\text{mid}} = \{ [x_j, x'_j] \mid x_j \leq x_{\text{mid}} \leq x'_j \}$$
  $$I_{\text{right}} = \{ [x_j, x'_j] \mid x_{\text{mid}} < x_j \}$$

$\mathcal{T}$ consists of a node $v$ for $x_{\text{mid}}$ and
- lists $\mathcal{L}(v)$ and $\mathcal{R}(v)$ for $I_{\text{mid}}$ sorted by left and right interval endpoints, respectively
- left child of $v$ is an interval tree for $I_{\text{left}}$
- right child of $v$ is an interval tree for $I_{\text{right}}$

$$\mathcal{L} = s_3, s_4, s_5 \quad \mathcal{R} = s_5, s_3, s_4$$

$$\mathcal{L} = s_1, s_2 \quad \mathcal{R} = s_1, s_2$$

$$\mathcal{L} = s_6, s_7 \quad \mathcal{R} = s_7, s_6$$

$I_{\text{left}}$ $s_1$ $s_2$ $I_{\text{mid}}$ $s_5$ $s_6$ $I_{\text{right}}$ $s_7$
Properties of interval trees

**Lemma 1:** An interval tree for $n$ intervals needs $O(n)$ space and has depth $O(\log n)$. It can be constructed in time $O(n \log n)$. 
Properties of interval trees

**Lemma 1:** An interval tree for $n$ intervals needs $O(n)$ space and has depth $O(\log n)$. It can be constructed in time $O(n \log n)$.
Properties of interval trees

**Lemma 1:** An interval tree for $n$ intervals needs $O(n)$ space and has depth $O(\log n)$. It can be constructed in time $O(n \log n)$.

\[
\text{QueryIntervalTree}(v, qx) \\
\text{if} \ v \text{ no leaf then} \\
\quad \text{if} \ qx < x_{\text{mid}}(v) \text{ then} \\
\qquad \text{search in } L \text{ from left to right for intervals containing } qx \\
\qquad \text{QueryIntervalTree}(lc(v), qx) \\
\text{else} \\
\quad \text{search in } R \text{ from right to left for intervals containing } qx \\
\quad \text{QueryIntervalTree}(rc(v), qx)
\]
Properties of interval trees

**Lemma 1:** An interval tree for \( n \) intervals needs \( O(n) \) space and has depth \( O(\log n) \). It can be constructed in time \( O(n \log n) \).

\[
\text{QueryIntervalTree}(v, qx)
\]

\[
\begin{align*}
\text{if } v \text{ no leaf then} & \\
\text{if } qx < x_{\text{mid}}(v) \text{ then} & \\
& \text{search in } \mathcal{L} \text{ from left to right for intervals containing } qx \\
& \text{QueryIntervalTree}(lc(v), qx) \\
\text{else} & \\
& \text{search in } \mathcal{R} \text{ from right to left for intervals containing } qx \\
& \text{QueryIntervalTree}(rc(v), qx)
\end{align*}
\]

**Lemma 2:** Using an interval tree we can find all \( k \) intervals containing a query point \( qx \) in \( O(\log n + k) \) time.
From lines to line segments

How can we adapt the idea of an interval tree for query segments $qx \times [qy, q'y]$ instead of a query line $x = qx$?
From lines to line segments

How can we adapt the idea of an interval tree for query segments $q_x \times [q_y, q'_y]$ instead of a query line $x = q_x$?
From lines to line segments

How can we adapt the idea of an interval tree for query segments $q_x \times [q_y, q'_y]$ instead of a query line $x = q_x$?

The correct line segments in $I_{mid}$ can easily be found using a range tree instead of simple lists.
From lines to line segments

How can we adapt the idea of an interval tree for query segments $q_x \times [q_y, q'_y]$ instead of a query line $x = q_x$?

The correct line segments in $I_{\text{mid}}$ can easily be found using a range tree instead of simple lists.

**Theorem 1:** Let $S$ be a set of horizontal (axis-parallel) line segments in the plane. All $k$ line segments that intersect a vertical query segment (an axis-parallel rectangle $R$) can be found in $O(\log^2(n) + k)$ time. The data structure requires $O(n \log n)$ space and construction time.
Arbitrary line segments

Map data often contain arbitrarily oriented line segments.

Problem:

Given \( n \) disjoint line segments and an axis-parallel rectangle \( R = [x, x'] \times [y, y'] \), find all line segments that intersect \( R \).

How to proceed?
Arbitrary line segments
Map data often contain arbitrarily oriented line segments.

**Problem:**
Given $n$ disjoint line segments and an axis-parallel rectangle $R = [x, x'] \times [y, y']$, find all line segments that intersect $R$.

**Case 1:** $\geq 1$ endpoint in $R \rightarrow$ use range tree
**Case 2:** both endpoints $\notin R \rightarrow$ intersect at least one edge of $R$
Decomposition into elementary intervals

Interval trees don’t really help here

\([-\infty, q_x] \times [q_y, q'_y]\)
Decomposition into elementary intervals

Interval trees don’t really help here

Identical 1d base problem:
Given $n$ intervals $I = \{[x_1, x'_1], [x_2, x'_2], \ldots, [x_n, x'_n]\}$ and a point $q_x$, find all intervals that contain $q_x$.

- sort all $x_i$ and $x'_i$ in list $p_1, \ldots, p_{2n}$
- create sorted elementary intervals
  $(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, [p_{2n}, p_{2n}], (p_{2n}, \infty)$
Segment trees

Idea for data structure:
• create binary search tree with elementary intervals in leaves
• for all points $q_x$ in the same elementary interval the answer is the same
• leaf $\mu$ for elementary interval $e(\mu)$ stores interval set $I(\mu)$
• query requires $O(\log n + k)$ time
Segment trees

Idea for data structure:
- create binary search tree with elementary intervals in leaves
- for all points $q_x$ in the same elementary interval the answer is the same
- leaf $\mu$ for elementary interval $e(\mu)$ stores interval set $I(\mu)$
- query requires $O(\log n + k)$ time

Any problem?
**Segment trees**

Idea for data structure:
- create binary search tree with elementary intervals in leaves
- for all points $q_x$ in the same elementary interval the answer is the same
- leaf $\mu$ for elementary interval $e(\mu)$ stores interval set $I(\mu)$
- query requires $O(\log n + k)$ time

**Problem:** Storage space is worst-case quadratic
→ store intervals as high up in the tree as possible
Segment trees

Idea for data structure:
- create binary search tree with elementary intervals in leaves
- for all points \( q_x \) in the same elementary interval the answer is the same
- leaf \( \mu \) for elementary interval \( e(\mu) \) stores interval set \( I(\mu) \)
- query requires \( O(\log n + k) \) time

Problem: Storage space is worst-case quadratic
→ store intervals as high up in the tree as possible

- node \( v \) represents interval \( e(v) = e(lc(v)) \cup e(rc(v)) \)
- input interval \( s_i \in I(v) \Leftrightarrow e(v) \subseteq s_i \) and \( e(\text{parent}(v)) \not\subseteq s_i \)
Segment trees

Idea for data structure:
- create binary search tree with elementary intervals in leaves
- for all points \( q_x \) in the same elementary interval the answer is the same
- leaf \( \mu \) for elementary interval \( e(\mu) \) stores interval set \( I(\mu) \)
- query requires \( O(\log n + k) \) time

**Problem:** Storage space is worst-case quadratic

\[ \rightarrow \text{store intervals as high up in the tree as possible} \]

- node \( v \) represents interval \( e(v) = e(lc(v)) \cup e(rc(v)) \)
- input interval \( s_i \in I(v) \iff e(v) \subseteq s_i \text{ and } e(parent(v)) \not\subseteq s_i \)
Properties of segment trees

**Lemma 3:** A segment tree for $n$ intervals requires $O(n \log n)$ space and can be constructed in $O(n \log n)$ time.
Properties of segment trees

Lemma 3: A segment tree for $n$ intervals requires $O(n \log n)$ space and can be constructed in $O(n \log n)$ time.

Sketch of proof:

\[
\begin{align*}
\text{InsertSegmentTree} & (v, [x, x']) \\
\text{if } e(v) \subseteq [x, x'] \text{ then} & \text{ store } [x, x'] \text{ in } I(v) \\
\text{else} & \text{ if } e(lc(v)) \cap [x, x'] \neq \emptyset \text{ then} \\
& \quad \text{ InsertSegmentTree}(lc(v), [x, x']) \\
& \text{if } e(rc(v)) \cap [x, x'] \neq \emptyset \text{ then} \\
& \quad \text{ InsertSegmentTree}(rc(v), [x, x'])
\end{align*}
\]
Queries in segment trees

\textbf{QuerySegmentTree}(v, q_x)

\begin{algorithmic}
\State return all intervals in $I(v)$
\If{$v$ no leaf}
\If{$q_x \in e(lc(v))$}
\State QuerySegmentTree($lc(v), q_x$)
\Else
\State QuerySegmentTree($rc(v), q_x$)
\EndIf
\EndIf
\end{algorithmic}

\textbf{Lemma 4:} All $k$ intervals that contain a query point $q_x$ can be computed in $O(\log n + k)$ time using a segment tree.
Queries in segment trees

\[ \text{QuerySegmentTree}(v, q_x) \]

\[
\text{return all intervals in } I(v) \\
\text{if } v \text{ no leaf then} \\
\quad \text{if } q_x \in e(lc(v)) \text{ then} \\
\quad \quad \text{QuerySegmentTree}(lc(v), q_x) \\
\quad \text{else} \\
\quad \quad \text{QuerySegmentTree}(rc(v), q_x)
\]

**Lemma 4:** All \( k \) intervals that contain a query point \( q_x \) can be computed in \( O(\log n + k) \) time using a segment tree.

Lemma 4 yields the same result as interval trees. What is different?
Queries in segment trees

QuerySegmentTree\( (v, q_x) \)

\[
\text{return all intervals in } I(v) \\
\text{if } v \text{ no leaf then} \\
\quad \text{if } q_x \in e(lc(v)) \text{ then} \\
\quad \quad \text{QuerySegmentTree}(lc(v), q_x) \\
\quad \text{else} \\
\quad \quad \text{QuerySegmentTree}(rc(v), q_x)
\]

Lemma 4: All \( k \) intervals that contain a query point \( q_x \) can be computed in \( O(\log n + k) \) time using a segment tree.

Lemma 4 yields the same result as interval trees. What is different?

→ all intervals stored in a positive node \( v \) contain \( q_x \) – in an interval tree one would have to continue searching
Back to arbitrary line segments

- create segment tree for the $x$ intervals of the line segments
- each node $v$ corresponds to a vertical strip $e(v) \times \mathbb{R}$
- line segment $s$ is in $I(v)$ iff $s$ crosses the strip of $v$ but not the strip of parent($v$)
- at each node $v$ on the search path for the vertical segment $s' = q_x \times [q_y, q_y']$ all segments in $I(v)$ cover the $x$-coordinate $q_x$
Back to arbitrary line segments

- create segment tree for the $x$ intervals of the line segments
- each node $v$ corresponds to a vertical strip $e(v) \times \mathbb{R}$
- line segment $s$ is in $I(v)$ iff $s$ crosses the strip of $v$ but not the strip of parent($v$)
- at each node $v$ on the search path for the vertical segment $s' = q_x \times [q_y, q'_y]$ all segments in $I(v)$ cover the $x$-coordinate $q_x$
- find segments in the strip that cross $s'$ using a vertically sorted auxiliary binary search tree
Summary

Theorem 2: Let $S$ be a set of interior-disjoint line segments in the plane. All $k$ segments that intersect a vertical query segment (an axis-parallel query rectangle $R$) can be found in time $O(k + \log^2 n)$. The corresponding data structure uses $O(n \log n)$ space and $O(n \log^2 n)$ construction time.
Summary

**Theorem 2:** Let $S$ be a set of interior-disjoint line segments in the plane. All $k$ segments that intersect a vertical query segment (an axis-parallel query rectangle $R$) can be found in time $O(k + \log^2 n)$. The corresponding data structure uses $O(n \log n)$ space and $O(n \log^2 n)$ construction time.

Remark:
The construction time for the data structure can be improved to $O(n \log n)$. 
Discussion

Space requirement of interval trees
Discussion

Space requirement of interval trees

We have used range trees with $O(n \log n)$ space as auxiliary data structure in the interval trees. Using modified heaps this can be reduced to $O(n)$, see Chapter 10.2 in [BCKO'08].
Discussion

Space requirement of interval trees

We have used range trees with $O(n \log n)$ space as auxiliary data structure in the interval trees. Using modified heaps this can be reduced to $O(n)$, see Chapter 10.2 in [BCKO'08].

How can you efficiently count the intersected segments?
Discussion

Space requirement of interval trees

We have used range trees with $O(n \log n)$ space as auxiliary data structure in the interval trees. Using modified heaps this can be reduced to $O(n)$, see Chapter 10.2 in [BCKO’08].

How can you efficiently count the intersected segments?

Segment and interval trees support efficient counting queries (independent of $k$) with minor modifications → see exercises.
Discussion

**Space requirement of interval trees**

We have used range trees with $O(n \log n)$ space as auxiliary data structure in the interval trees. Using modified heaps this can be reduced to $O(n)$, see Chapter 10.2 in [BCKO’08].

**How can you efficiently count the intersected segments?**

Segment and interval trees support efficient counting queries (independent of $k$) with minor modifications → see exercises.

**What to do for non-rectangular query regions?**
Discussion

Space requirement of interval trees

We have used range trees with $O(n \log n)$ space as auxiliary data structure in the interval trees. Using modified heaps this can be reduced to $O(n)$, see Chapter 10.2 in [BCKO'08].

How can you efficiently count the intersected segments?

Segment and interval trees support efficient counting queries (independent of $k$) with minor modifications → see exercises.

What to do for non-rectangular query regions?

By triangulating the query polygon, the problem can be reduced to triangular queries. Suitable data structures can be found, e.g., in chapter 16 of [BCKO'08].