segment tree

By Zohre Akbari
January 2014
Arbitrarily oriented segments

Two cases of intersection:

• An endpoint lies inside the query window; solve with range trees

• The segment intersects the window boundary; solve how?
Arbitrarily oriented segments

A simple solution:

- Replace each line segment by

- So we could search in the $4n$ bounding box sides.
Arbitrarily oriented segments

A simple solution:
• Replace each line segment by its bounding box.

In the worst case:
• The solution is quite bad:
  • All bounding boxes may intersect $W$ whereas none of the segments do.

• So we could search in the $4n$ bounding box sides.
Given a set $S$ of line segments with arbitrary orientations in the plane, and we want to find those segments in $S$ that intersect a vertical query segment $q := q_x \times [q_y : q'_y]$. 

Current problem of our intersect:
Why don’t interval trees work?

If the segments have arbitrary orientation, knowing that the right endpoint of a segment is to the right of \( q \) doesn’t help us much.
• Given a set \( S = \{s_1, s_2, \ldots, s_n \} \) of \( n \) segments (Intervals) on the real line, preprocess them into a data structure so that the ones containing a query point (value) can be reported efficiently.

• The new structure is called the **segment tree**.
Locus approach

- The **locus approach** is the idea to partition the parameter space into regions where the answer to a query is the same.
The **locus approach** is the idea to partition the parameter space into regions where the answer to a query is the same.

Our query has only one parameter, $q_x$, so the parameter space is the real line. Let $p_1, p_2, \ldots, p_n$ be the list of distinct interval endpoints, sorted from left to right; $m \leq 2n$
The locus approach is the idea to partition the parameter space into regions where the answer to a query is the same.

Our query has only one parameter, $q_x$, so the parameter space is the real line. Let $p_1, p_2, \ldots, p_n$ be the list of distinct interval endpoints, sorted from left to right; $m \leq 2n$.

The real line is partitioned into
- $(-\infty, p_1)$,
- $[p_1, p_1]$, 
- $(p_1, p_2)$,
- $[p_2, p_2]$, 
- $(p_2, p_3)$, 
- $\ldots$, 
- $(p_m, +\infty)$, these are called the elementary intervals.
Locus approach

- We could make a binary search tree that has a leaf for every elementary interval.

- We denote the elementary interval corresponding to a leaf $\mu$ by $\text{Int}(\mu)$.

- All the segments (intervals) in $S$ containing $\text{Int}(\mu)$ are stored at the leaf $\mu$.

- Each internal node corresponds to an interval that is the union of the elementary intervals of all leaves below it.
We can report the $k$ intervals containing $q_x$ in $O(\log n + k)$ time.

$O(n^2)$ storage in the worst case:
Reduce the amount of storage

- To avoid quadratic storage, we store any segment $s_j$ with $v$ iff

  $Int(v) \subseteq s_j$ but $Int(parent(v)) \not\subseteq s_j$.

- The data structure based on this principle is called a segment tree.
Segment tree

- A **segment tree** on a set $S$ of segments is a balanced binary search tree on the elementary intervals defined by $S$, and each node stores its interval, and its canonical subset of $S$ in a list.

- The **canonical subset** of a node $v$ contains segments $s_j$ such that $\text{Int}(v) \subseteq s_j$ but $\text{Int}(\text{parent}(v)) \not\subseteq s_j$.
Lemma 10.10

A segment tree on a set of $n$ intervals uses $O(n \log n)$ storage.

Proof.

We claim that any segment is stored for at most two nodes at the same depth of the tree.
Query algorithm

Algorithm QUERYSEGMENTTREE($v, q_x$)

*Input.* The root of a (subtree of a) segment tree and a query point $q_x$.
*Output.* All intervals in the tree containing $q_x$.

1. Report all the intervals in $I(v)$.
2. if $v$ is not a leaf
3. then if $q_x \in \text{Int}(lc(v))$
4. then QUERYSEGMENTTREE($lc(v), q_x$)
5. else QUERYSEGMENTTREE($rc(v), q_x$)
Example query
Example query
Using a segment tree, the intervals containing a query point $q_x$ can be reported in $O(\log n + k)$ time, where $k$ is the number of reported intervals.

Lemma 10.11
Segment Tree Construction

- Build tree:
  - Sort the endpoints of the segments take $O(n \log n)$ time. This gives us the elementary intervals.
  - Construct a balanced binary tree on the elementary intervals, this can be done bottom-up in $O(n)$ time.
Segment Tree Construction

- Build tree:
  - Sort the endpoints of the segments take $O(n \log n)$ time. This gives us the elementary intervals.
  - Construct a balanced binary tree on the elementary intervals, this can be done bottom-up in $O(n)$ time.
  - Compute the canonical subset for the nodes. To this end we insert the intervals one by one into the segment tree by calling:
Segment Tree Construction

- Build tree :

  - Sort the endpoints of the segments take $O(n \log n)$ time. This gives us the elementary intervals.
  - Construct a balanced binary tree on the elementary intervals, this can be done bottom-up in $O(n)$ time.
  - Compute the canonical subset for the nodes. To this end we insert the intervals one by one into the segment tree by calling:

```
Algorithm INSERTSEGMENTTREE(v, [x : x'])
Input. The root of a (subtree of a) segment tree and an interval.
Output. The interval will be stored in the subtree.
1. if Int(v) ⊆ [x : x']
2. then store [x : x'] at v
3. else if Int(lc(v)) ∩ [x : x'] ≠ ∅
4. then INSERTSEGMENTTREE(lc(v), [x : x'])
5. if Int(rc(v)) ∩ [x : x'] ≠ ∅
6. then INSERTSEGMENTTREE(rc(v), [x : x'])
```
How much time does it take to insert an interval $[x : x']$ into the segment tree?

- an interval is stored at most twice at each level of $\mathcal{T}$
- There is also at most one node at every level whose corresponding interval contains $x$ and one node whose interval contains $x'$.
- So we visit at most 4 nodes per level.
- Hence, the time to insert a single interval is $O(\log n)$, and the total time to construct the segment tree is $O(n \log n)$.
Theorem 10.12

- A segment tree for a set $I$ of $n$ intervals uses $O(n \log n)$ storage and can be built in $O(n \log n)$ time. Using the segment tree we can report all intervals that contain a query point in $O(\log n + k)$ time, where $k$ is the number of reported intervals.
Let S be a set of arbitrarily oriented, disjoint segments in the plane. We want to report the segments intersecting a vertical query segment $q := q_x \times [q_y : q'_y]$. 

Back to windowing problem
• Build a segment tree $\tau$ on the $x$-intervals of the segments in $S$. 
• Build a segment tree $\tau$ on the $x$-intervals of the segments in $S$.
• A node $v$ in $\tau$ can now be considered to correspond to the vertical slab $\text{Int}(v) \times (-\infty: +\infty)$. 
• Build a segment tree $\mathcal{T}$ on the $x$-intervals of the segments in $S$.

• A node $v$ in $\mathcal{T}$ can now be considered to correspond to the vertical slab $Int(v) \times (−\infty: +\infty)$

• A segment $S_i$ is in the canonical subset of $v$, if it crosses the slab of $v$ completely, but not the slab of the parent of $v$. 
• Build a segment tree $T$ on the $x$-intervals of the segments in $S$.

• A node $v$ in $T$ can now be considered to correspond to the vertical slab $Int(v) \times (-\infty: +\infty)$

• A segment $S_i$ is in the canonical subset of $v$, if it crosses the slab of $v$ completely, but not the slab of the parent of $v$.

• We denote canonical subset of $v$ with $S(v)$. 
Querying

• When we search with $q_x$ in $\tau$ we find $O(\log n)$ canonical subsets that collectively contain all the segments whose $x$-interval contains $q_x$.

• A segment $s$ in such a canonical subset is intersected by $q$ if and only if the lower endpoint of $q$ is below $S$ and the upper endpoint of $q$ is above $S$. 
Querying

• segments in the canonical subset $S(ν)$ do not intersect each other.

  This implies that the segments can be ordered vertically.

• we can store $S(ν)$ in a search tree $τ(ν)$ according to the vertical order.
Query time

- A query with $q_x$ follows one path down the main tree (segment tree).

- And at every node $v$ on the search path we search with endpoints of $\tau(v)$ to report the segments in $S(v)$ intersected by $q$ (a 1-dimensional range query).

- The search in $\tau(v)$ takes $O(\log n + k_v)$ time, where $k_v$ is the number of reported segments at $(v)$.

- Hence, the total query time is $O(\log^2 n + k)$. 
Storage

- Because the associated structure of any node $v$ uses storage linear in the size of $S(v)$, the total amount of storage remains $O(n \log n)$.

- Data structure can be build in $O(n \log n)$ time.
Theorem 10.13

Let $S$ be a set of $n$ disjoint segments in the plane. The segments intersecting a vertical query segment can be reported in $O(\log^2 n + k)$ time with a data structure that uses $O(n \log n)$ storage, where $k$ is the number of reported segments. The structure can be built in $O(n \log n)$ time.
Let $S$ be a set of $n$ segments in the plane with disjoint interiors. The segments intersecting an axis-parallel rectangular query window can be reported in $O(\log^2 n + k)$ time with a data structure that uses $O(n \log n)$ storage, where $k$ is the number of reported segments. The structure can be built in $O(n \log n)$ time.