Segment Tree

[J. L. Bentley; *Solutions to Klee's rectangle problem*, Technical Report, Carnegie-Mellon University, Pittsburgh, 1977]
[Section 10.3 in de Berg, Cheong, van Kreveld, Overmars; *Computational Geometry: Algorithms and Applications*, 3rd edition, 2008]
[Section 2.2 in de Langetepe Zachmann; *Geometric Data Structures for Computer Graphics*, 2006]

A *segment tree* is a static data structure for storing a set of intervals

$$I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$$

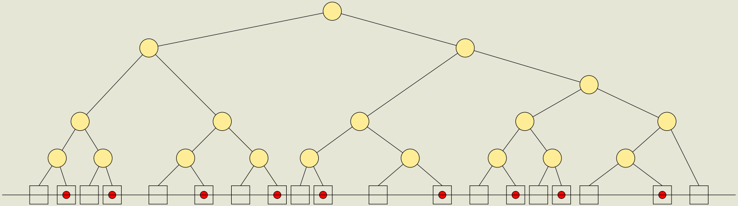
and can be used for solving problems e.g. concerning line segments.

Let $p_1, \ldots, p_m$, $m \leq 2n$, be the ordered list of distinct endpoints of the intervals in I. The ordered sequence of endpoints $p_1, \ldots, p_m$ partitions the real line into a set of *atomic intervals*:

$$(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, (p_{n-1}, p_n), [p_n, p_n], (p_n, \infty)$$

Let $p_1, \ldots, p_m$, $m \leq 2n$, be the ordered list of distinct endpoints of the intervals in I. The ordered sequence of endpoints $p_1, \ldots, p_m$ partitions the real line into a set of *atomic intervals*:

$$(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, (p_{n-1}, p_n), [p_n, p_n], (p_n, \infty)$$



A segment tree is a leaf-oriented balanced binary tree on the atomic intervals according to left to right order.

An internal node v corresponds to the interval which is the union of the atomic intervals of the leaves of the subtree rooted at v.
Let $int(v)$ denote this interval.

An internal node v corresponds to the interval which is the union of the atomic intervals of the leaves of the subtree rooted at v.
Let $int(v)$ denote this interval.

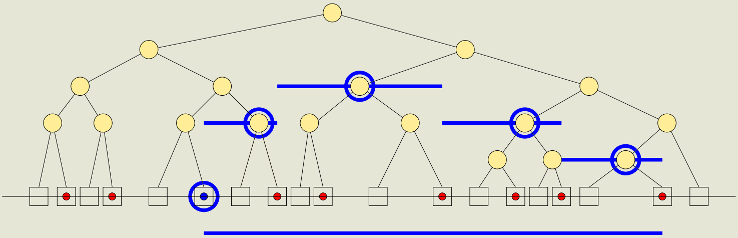
With each node v we store a set $I(v) \subseteq I$:
Interval $[x, x']$ is stored in $I(v)$ if and only if

$$int(v) \subseteq [x, x'] \quad \text{and} \quad int(parent(v)) \not\subseteq [x, x']$$

An internal node v corresponds to the interval which is the union of the atomic intervals of the leaves of the subtree rooted at v.
Let $int(v)$ denote this interval.

With each node v we store a set $I(v) \subseteq I$:
Interval $[x, x']$ is stored in $I(v)$ if and only if

$$int(v) \subseteq [x, x'] \quad \text{and} \quad int(parent(v)) \not\subseteq [x, x']$$



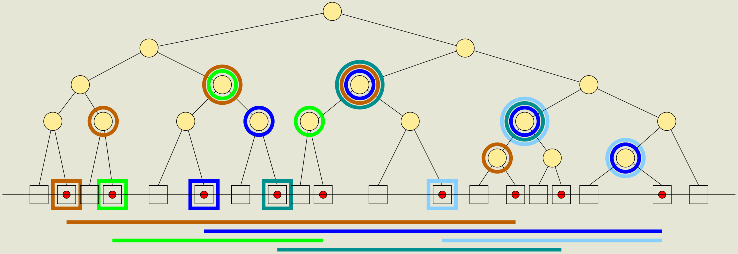
Lemma:

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

Lemma:

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

An interval is stored with at most two nodes at the same depth of the tree.



Lemma:
A segment tree for a set of n intervals can be constructed in $O(n \log n)$ time.

Lemma:
A segment tree for a set of n intervals can be constructed in $O(n \log n)$ time.

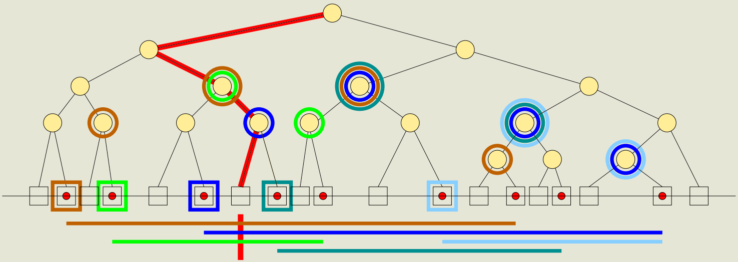
- build leaf-oriented balanced binary search tree on atomic intervals
- insert intervals one by one using InsertSegmentTree:

InsertSegmentTree($v, [x, x']$)

if $int(v) \subseteq [x, x']$
  then add $[x, x']$ to $I(v)$
  else **if** $int(lc(v)) \cap [x, x'] \neq \emptyset$
      then InsertSegmentTree($lc(v), [x, x']$)
    if $int(rc(v)) \cap [x, x'] \neq \emptyset$
      then InsertSegmentTree($rc(v), [x, x']$)

QUERY

QUERY(q_x) reports all segments containing query point q_x.



$\textsc{QuerySegmentTree}(v, q_x)$

Report all the intervals in $I(v)$
if v is not a leaf
  then if $q_x \in int(lc(v))$
    then $\textsc{QuerySegmentTree}(lc(v), q_x)$
    else $\textsc{QuerySegmentTree}(rc(v), q_x)$

QUERYSEGMENTTREE(v, q_x)

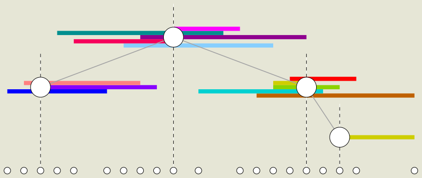
Report all the intervals in $I(v)$
if v is not a leaf
  then if $q_x \in int(lc(v))$
    then QUERYSEGMENTTREE$(lc(v), q_x)$
    else QUERYSEGMENTTREE$(rc(v), q_x)$

Lemma:
Using a segment tree, we can report all k intervals that contain a query point q_x, in time $O(k + \log n)$.

Interval Tree

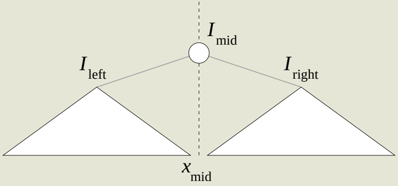
[H. Edelsbrunner; *Dynamic Data Strcutures for Orthogonal Intersection Queries*, Tech. Report. TU Graz, 1980]
[E. M. McCreight; *Efficient Algorithms for Enumerating Intersection Intervals and Rectangles*, Tech. Report, Xerox Paolo Alto Reserach Center, 1980]
[Section 10.1 in de Berg, Cheong, van Kreveld, Overmars; *Computational Geometry: Algorithms and Applications*, 3rd edition, 2008]

An *interval tree* stores a set of intervals on a real line. Let $I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$ be a set of n closed intervals.



Let x_{mid} be the median of the interval endpoints of the intervals in I.

Let x_{mid} be the median of the interval endpoints of the intervals in I.



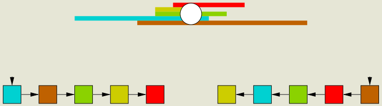
$$
\begin{aligned}
I_{\text{left}} &= \{[x_j, x'_j] \in I : x'_j < x_{\text{mid}}\} \\
I_{\text{mid}} &= \{[x_j, x'_j] \in I : x_j \leq x_{\text{mid}} \leq x'_j\} \\
I_{\text{right}} &= \{[x_j, x'_j] \in I : x_{\text{mid}} < x_j\}
\end{aligned}
$$

I_{mid} is stored at the root of the interval tree. The left subtree is an interval tree of I_{left} and the right subtree is an interval tree of I_{right}.

The interval tree of an empty set of intervals is an external node.

The interval tree of an empty set of intervals is an external node.

Associated data structures at internal node v:



- $L_{\text{left}}(v)$: list of the left endpoints of the intervals stored at v in increasing order.
- $L_{\text{right}}(v)$: list of right endpoints in decreasing order.

Theorem:
An interval tree for a set of n intervals uses $O(n)$ storage and has height $O(\log n)$. It can be built in $O(n \log n)$ time.

Stabbing Query

Stabbing Query: Given point x, report all intervals that contain x.

Query(v,x)

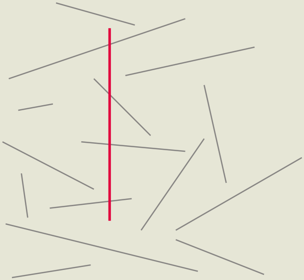
if v is not an external node
    then if $x < x_{\mathrm{mid}}(v)$
        then walk along $L_{\mathrm{left}}(v)$ until x is not contained anymore
            in the current interval; report containing intervals
            Query$(lc(v),x)$
        else walk along $L_{\mathrm{right}}(v)$ until x is not contained anymore
            in the current interval; report containing intervals
            Query$(rc(v),x)$

Theorem:
Using an interval tree, we can report all k intervals that contain a query point x, in time $O(k + \log n)$.

Vertical Stabbing Queries for Disjoint Line Segments

Let S be a set of pairwise disjoint line segments in the plane. We want to maintain S in a data structure that allows us to quickly find the segments intersected by a vertical query segment

$$q_x \times [q_y, q_y']$$

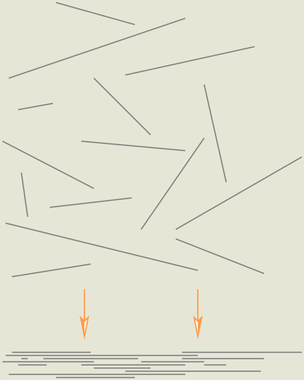


Augmented Segment Tree

for Vertical Stabbing Queries for Disjoint Line Segments

Underlying data structure:

Basically, the underlying data structure is a segment tree for the projection intervals of the segments in S onto the x-axis. However, we don't store the intervals in $I(v)$ at a node v explicitly.

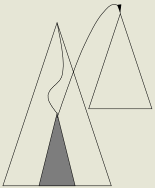


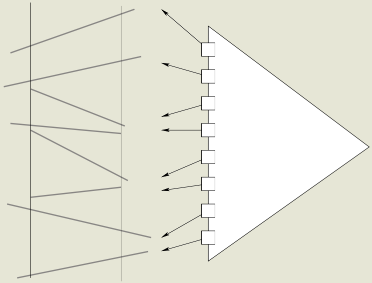
“Additional” information stored at each node v:

For a node v, let $S(v)$ be the set of segments corresponding to the intervals in $I(v)$.

"Additional" information stored at each node v:

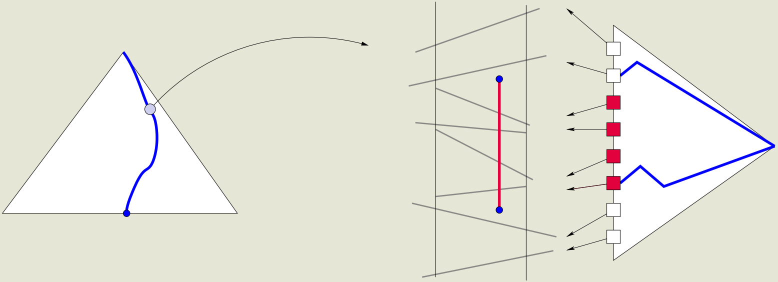
For a node v, let $S(v)$ be the set of segments corresponding to the intervals in $I(v)$. Store the segments in $S(v)$ in a leaf-oriented balanced binary search tree based on the order of the elements in the slab $int(v) \times (-\infty, \infty)$.





QUERY

$$\text{QUERY}(q_x \times [q_y, q'_y])$$



Search for q_x in the underlying segment tree. For each visited node v, search for q_y and q'_y in the balanced binary search tree for $S(v)$. Report segments which are between q_y and q'_y at q_x.

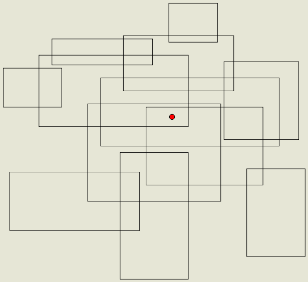
Lemma:

Let S be a set of n disjoint segments in the plane. S can be stored in a data structure such that the segments in S intersected by a vertical query segment can be reported in time $O(k+(\log n)^2)$, where k is the number of reported segments. The data structure uses $O(n \log n)$ storage space and can be built in $O(n(\log n)^2)$ time.

Construction time can be improved to $O(n \log n)$.

Point Stabbing Queries for Rectangles

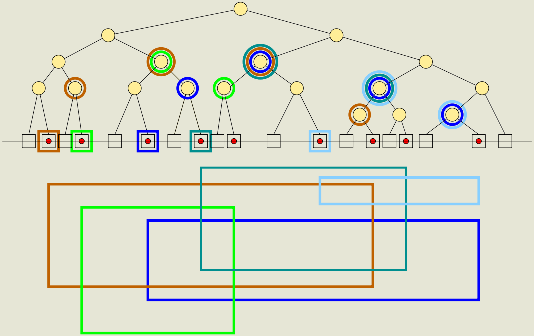
Segment trees can be augemented such that point enclosure problems for axis-aligned rectangles in 2D can be solved efficiently.



Point enclosure problems are also called “inverse range queries”.

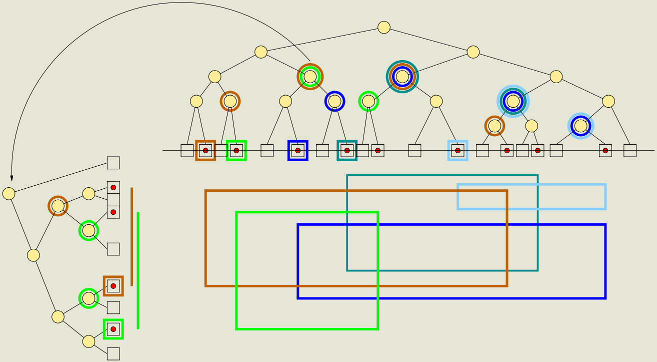
Multi-Level Segment Tree

The underlying data structure of a 2-*dimensional segment tree* is a segment tree for the projection intervals of the rectangles onto the x-axis.



"Additional" information stored at node v:

Let $R(v)$ be the set of rectangles whose x-interval are associated with v. The secondary data structure associated with v is a standard segment tree for the projection intervals of the rectangles in $R(v)$ onto the y-axis.



Theorem:
A 2-dimensional segment tree for solving point enclosure problems for n rectangles in the plane can be built in $O(n(\log n)^2)$ time and takes $O(n(\log n)^2)$ space.

QUERY

$$\text{QUERY}(q_x, q_y)$$

Search for q_x in the underlying segment tree. For each visited node v, search for q_y in the associated segment tree and report the rectangles whose y-intervals contain q_y.

QUERY

$$\text{QUERY}(q_x, q_y)$$

Search for q_x in the underlying segment tree. For each visited node v, search for q_y in the associated segment tree and report the rectangles whose y-intervals contain q_y.

Theorem:
Using a 2-dimensional segment tree, point enclosure queries for n rectangles in the plane can be answered in time $O(k + (\log n)^2)$ time, where k is the number of reported rectangles.

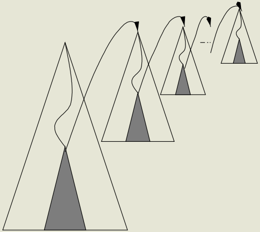
2-dimensional segment trees can be extended to higher dimensions for point stabbing queries for axis-aligned rectangular boxes.

As before the underlying data structure is a segment tree for the projection intervals with respect to the first coordinate.

2-dimensional segment trees can be extended to higher dimensions for point stabbing queries for axis-aligned rectangular boxes.

As before the underlying data structure is a segment tree for the projection intervals with respect to the first coordinate.

The secondary data structure associated with a node v is a $(d-1)$-dimensional segment tree according to the remaining coordinates for the boxes corresponding to the intervals stored at v. More precisely, it is a $(d-1)$-dimensional segment tree for the boxes formed by the remaining $(d-1)$ coordinates.



Theorem:
A d-dimensional segment tree for a set of n axis-aligned rectangular boxes in $\mathbb{R}^d$ can be built in $O(n(\log n)^d)$ time and takes $O(n(\log n)^d)$ space.

Theorem:
A d-dimensional segment tree for a set of n axis-aligned rectangular boxes in $\mathbb{R}^d$ can be built in $O(n(\log n)^d)$ time and takes $O(n(\log n)^d)$ space.

Theorem:
Using a d-dimensional segment tree, point enclosure queries for a set of n axis-aligned rectangular boxes in $\mathbb{R}^d$ can be answered in time $O(k + (\log n)^d)$ time, where k is the number of reported boxes.