Chapter 16

Data-structures: Interval trees

By Sariel Har-Peled, March 7, 2023[1]

"See? Genuine-sounding indignation. I programmed that myself. It's the first thing you need in a university environment: the ability to take offense at any slight, real or imagined."

Robert Sawyer, Factoring Humanity,

We cover here Chapter 10 in [BCKO08]

16.1. Interval trees

Consider an interval $I = [\alpha, \beta]$. It contains a point $q \in \mathbb{R}$ if $\alpha \leq q \leq \beta$. Given a set of n interval $\mathcal{I}$, consider the problem of building a data-structure such that given a query point, one can report all the intervals that contain it in $O(\log n + k)$ time.

Theorem 16.1.1. *Given a set $\mathcal{I}$ of n intervals, one can build a data-structure in $O(n \log n)$ time, such that given a query point one can report all the interval containing q in $O(\log n + k)$ time. The data-structure, called* **interval tree**, *uses $O(n)$ space.*

A natural extension is to consider horizontal segments. We would like to report all the segments that intersect a vertical query segment q. The idea it to build a top-level interval-tree as above on the x-axis. For an internal node, we have all the horizontal segments intersecting a vertical line. We now preprocess the end points of these segments to orthogonal range searching. Using fractional cascading and the same recursive approach, we get the following.

Theorem 16.1.2. *Let S be a set of n horizontal segments in the plane. One can preprocess them in $O(n \log n)$ time, such that given a vertical segment q, one can report all the segments of S intersecting q in $O(\log^2 n + k)$ time. The data-structure uses $O(n \log n)$ space.*

16.2. Priority search trees

The above required us to answer three-sided queries on a set of points. We solve this problem directly.

Lemma 16.2.1. *Given a set P of n points in the plane, one can preprocess it in $O(n \log n)$ time, such that given a query region $R = [-\infty, x_0] \times [y_1, y_2]$, one can report all the points in $P \cap R$ in $O(\log n + k)$ time.*

Proof: The idea is to build a binary search tree. At the root, we put the point $p_{\min}$ with minimum x coordinate in P. We then split $P - p_{\min}$ "equally" according to the y-axis order median – say

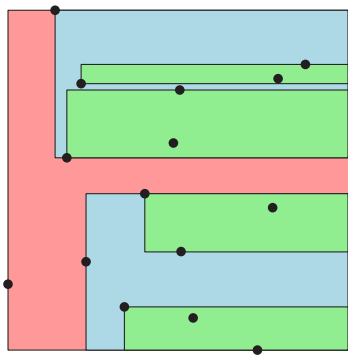

Figure 16.2.1

this value is β. We recursively construct a tree for $P_{>\beta} = \{(x,y) \in P - p_{\min} \mid y > \beta\}$ and $P_{\leq\beta} = \{(x,y) \in P - p_{\min} \mid y \leq \beta\}$ and hang it from the root. This can be interpreted as building a binary hierarchy of three sided rectangles.

The query process is now quite natural – given the three sided rectangle R, you do a recursive traversal of the tree recursing into a node only if its associated region intersects R. It is not hard to argue that the query time is $O(\log n + k)$. ■

16.3. Segment trees

For a set $P \subseteq \mathbb{R}$, an ***atomic interval*** is a maximal closed continuous subset of $\mathbb{R}$ that does not contain any point of P in its interior. Let $\mathcal{I}_A(P)$ be the set of atomic intervals defined by P. By building a balanced binary tree on the atomic intervals of P (here the atomic intervals are sorted from left to right), we get a balanced binary tree T, where each node v correspond to an interval I_v on the real line.

Given a set of intervals $\mathcal{I}$, let P be the set of endpoints of the intervals of $\mathcal{I}$, and let T be the above tree constructed over $\mathcal{I}_A(P)$. We store an interval $I \in \mathcal{I}$ in all the nodes v such that $I_v \subseteq I$, but $I_{\overline{\mathrm{p}}(v)}$ is not contained in I, where $\overline{\mathrm{p}}(v)$ is the parent of v in the tree. It is straightforward to argue that every interval is stored in $O(\log n)$ nodes. Given a query point $q \in \mathbb{R}$, it is straightforward to locate the $m = O(\log n)$ nodes $v_1, \ldots, v_m$ of T that contains q. The set $\mathcal{I}(v_1) \cup \cdots \cup I(v_m)$ then is all the input intervals that contains q. The query time is $O(\log n)$. This tree is known as ***segment tree***

|

Theorem 16.3.1. *Given a set of n intervals on the line, one can build a data-structure using $O(n \log n)$ space, such that given a query point q, one can report all the intervals containing q, by reporting $O(\log n)$ precomputed sets. The disjoint union of the sets is the set of all intervals containing q.*

This readily leads to the following data-structure.

Theorem 16.3.2. *Given a set of n interior disjoint segments S in the plane, one can build a data-structure using $O(n \log n)$ space, such that given a vertical query segment s, one can report all the segments intersecting s in $O(\log^2 n + k)$ time, where $k = |s \cap \mathsf{S}|$. This data-structure can be built in $O(n \log n)$ time.*

The naive construction time in the above algorithm is $O(n \log^2 n)$. However, the construction time can be improved to $O(n \log n)$ by computing partial order on the segments by y order using sweeping. Then, we insert the segments into the tree in bottom to top order. Now, all the segments registered in a node, are registered in their correct y-order.

16.4. Bibliographical notes

Our presentation more or less follows Chapter 10 in [BCKO08]

References

[BCKO08] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications*. 3rd. Santa Clara, CA, USA: Springer, 2008.