"Two intervals are *independent* if neither is contained in the other and their intersection is non-empty. Into how many pieces do we have to cut n intervals to guarantee that no two pieces are independent?"

1 Segment trees

Historically, computational geometry was first practiced by computer scientists working on algorithms and data structures. At the beginning of the 70's, a large body of knowledge on data structures for single-key items was available, and there were a few attempts to extend the known techniques to items determined by more than one key, see [3, chapter 6.5]. Possibly the simplest such item is an interval delimited by two keys. A popular data structure for storing a collection of intervals is the segment tree. It has originally been introduce by Bentley [1], and good description can be found in [4]. The segment tree has a number of variations, and some are described in the exercises and in later sections of these notes.

Definitions. Let S be a set of n intervals on the real line, $\mathbf{R}^1$. For convenience, we treat only half-open intervals, in particular, each interval is closed to the left and open to the right. Consider the $k \leq 2n$ endpoints indexed in increasing order and add $-\infty$ and $+\infty$ to the list: $-\infty = a_0 < a_1 < a_2 < \ldots < a_k < a_{k+1} = +\infty$. The $k+1$ intervals delimited by consecutive endpoints are the *atomic segments* defined by S. The *segment tree* of S is an ordered minimum height binary tree with $k+1$ leaves so that the ith leaf from the left corresponds to the ith atomic segment, $[a_{i-1}, a_i)$, see figure 1.1. For each node κ in the segment tree, the *segment* of κ, $\mathrm{s}(\kappa)$, is the corresponding

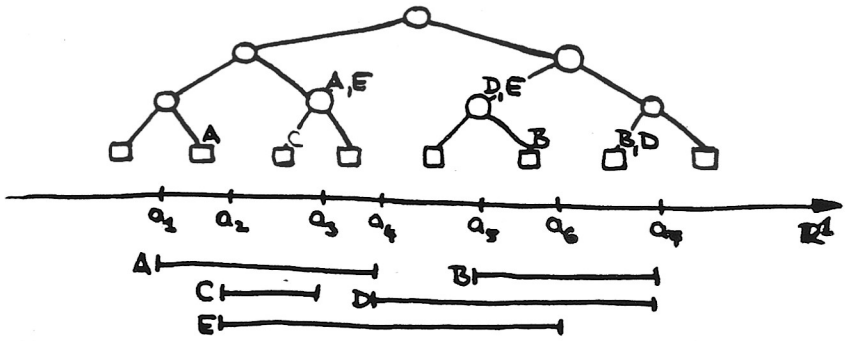

Figure 1.1: A segment tree for 5 intervals defining 7 different endpoints. Each node stores a set or list of intervals.

atomic segment, if κ is a leaf, and it is $\mathrm{s}(\kappa) = \mathrm{s}(\mu) \cup \mathrm{s}(\nu)$, if $\mu = \mathrm{l}(\kappa)$ and $\nu = \mathrm{r}(\kappa)$ are its left and right children. In other words, $\mathrm{s}(\kappa)$ is the union of all atomic segments associated to leaves descending from κ. Each node κ stores a subset of the intervals in S, namely all intervals that contain its segment but not the segment of its parent, $\mathrm{p}(\kappa)$. Formally,

$$L(\kappa) = \{\iota \in S \mid \mathrm{s}(\kappa) \subseteq \iota \text{ and } \mathrm{s}(\mathrm{p}(\kappa)) \not\subseteq \iota\}.$$

Properties. Consider the tree itself first. It has $k+1 \leq 2n+1$ leaves and thus $2k+1$ nodes in total. Its height is therefore $\lceil \log_2(k+1) \rceil \leq 2 + \log_2 n$. It follows that it has $1 + \lceil \log_2(k+1) \rceil \leq 3 + \log_2 n$ levels.

Next the segments. Let ν and κ be two nodes in the segment tree. If they lie on a common path from the root to a leaf, say ν is a descendent of κ, then $\mathrm{s}(\nu) \subseteq \mathrm{s}(\kappa)$. Otherwise, $\mathrm{s}(\nu) \cap \mathrm{s}(\kappa) = \emptyset$ because there is no leaf that descends from both nodes. This is summarized for later reference.

(i) Any two of the $2k+1$ segments defined by the segment tree are either nested or disjoint.

(ii) The segments of nodes on a path from the root to a leaf form a nested sequence.

(iii) The segments of the nodes in a level partition $\mathbf{R}^1$.

Finally, consider the sets L. It is interesting to see which sets $L(\kappa)$ contain an interval ι. Consider the two atomic segments to the left and right of ι, but not contained in ι. The paths from the root to the corresponding leaves, λ_1 and λ_2, coincide up to a node μ and then branch off. Of course, ι does not belong to L of any node on either path. Of the other nodes κ, $\iota \in L(\kappa)$ precisely if κ is the right child of a node on the path from μ to λ_1 or the left child of a node on the path from μ to λ_2. In other words, these nodes κ are connected to the double path by an edge and they lie strictly between its two branches. Note that these nodes κ are the roots of maximal subtrees whose atomic segments are contained in ι, see figure 1.2. It is now easy to see the following three properties of

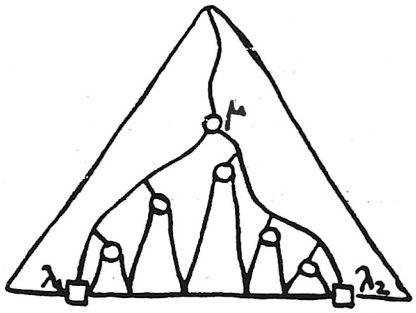

Figure 1.2: An interval belongs to the sets of the nodes connected to and between two paths.

segment trees.

(iv) An interval belongs to the sets L of at most $2 + 2\log_2 n$ nodes.

(v) The segments $s(\kappa)$ of nodes κ with $\iota \in L(\kappa)$ form a partition of ι.

(vi) For a point $x \in \mathbf{R}^1$, let $\rho = \kappa_0, \kappa_1, \ldots, \kappa_h = \lambda$ be the path starting at the root, ρ, and ending at the leaf, λ, with $x \in s(\lambda)$. Then $\{\iota \in S \mid x \in \iota\} = \bigcup_{i=0}^{h} L(\kappa_i)$.

Note that (iv) follows from the observation that an interval belongs to sets L of at most two nodes per level and no node of the top two levels, and (v) is a direct consequence of the definitions. Furthermore, (vi) follows from (ii) and (iii). Indeed, if $x \in \iota \in S$ then ι belongs to the highest node on the path from λ to ρ whose segment is still contained in ι.

Analysis. The amount of storage required for n intervals is $O(n \log n)$, see (iv).[1] The intervals containing a query point x can be reported in time $O(\log n + t)$, where t is the number of intervals found, see (vi).[2] Given the initial tree for a fixed set of endpoints, an interval (with endpoints in this set) can be added in time $O(\log n)$. From this it follows that the segment tree for a given set of n intervals can be constructed in time $O(n \log n)$.[3]

Homework exercises

1.1 Let S be a set of n intervals. For a node κ of a segment tree for S, define $H(\kappa) = \{\iota \in S \mid s(\kappa) \cap \iota \neq \emptyset \text{ and } s(\kappa) \not\subseteq \iota\}$. Take the sum of the sizes of these sets, over all nodes κ of the segment tree, and prove that $\sum_\kappa \operatorname{card} H(\kappa) = O(n \log n)$.

[1] In good cases, the amount of storage needed is only $O(n)$, but the average case is almost as bad as the worst case, namely $\Theta(n \log n)$, see [2]. More precisely, the total size of all sets L is $n \log_2 n + o(n \log n)$, on the average and in the worst case.

[2] By storing $\operatorname{card} L(\kappa)$ rather than $L(\kappa)$, the amount of storage decreases to $O(n)$. With this information it is still possible to count the intervals that contain a point x in time $O(\log n)$.

[3] There are in fact several different algorithms that construct a segment tree for n intervals in time $O(n \log n)$. The one indicated above would construct the tree first and then add the intervals, one at a time. Alternatively, the tree can be built in pre-order, constructing κ and $L(\kappa)$ simultaneously, and passing the relevant intervals to the recursive construction of the left and right subtrees.

1.2 Modify the segment tree of a finite set S of intervals as follows. Each node κ stores card $L(\kappa)$ and the total length of the part of $s(\kappa)$ covered by intervals in $\bigcup_\nu L(\nu)$, over all descendents ν of κ (this includes κ). Consider adding and removing intervals without changing the set of endpoints defining the tree structure. Show that an interval can be added or removed in time $O(\log n)$, where n is the number of endpoints.
(Remark. The root of the tree stores the length (one-dimensional measure) of the union of intervals currently in the tree.)

1.3 Consider a set R of n rectangles in $\mathbf{R}^2$, each the Cartesian product of an interval on the x_1-axis and an interval on the x_2-axis. Show how the version of the segment tree in exercise 1.2 can be used to compute the area of $\bigcup R = \bigcup_{r \in R} r$ in time $O(n \log n)$.

1.4 Store the rectangles in R in a (modified) segment tree so that for a point $x \in \mathbf{R}^2$, the t rectangles that contain x can be reported in time $O(\log^2 n + t)$. The amount of storage allowed is $O(n \log^2 n)$.

References

[1] J. L. Bentley. Algorithms for Klee's rectangle problem. Unpublished notes, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, 1977.

[2] W. Bucher and H. Edelsbrunner. On expected and worst-case segment trees. In *Advances in Computing Research, Vol. 1*, ed. F. P. Preparata, Jai Press, London, 1983, 109–125.

[3] D. E. Knuth. *Sorting and Searching. The Art of Computer Programming, Vol. 3.* Addison-Wesley, Reading, 1973.

[4] F. P. Preparata and M. I. Shamos. *Computational Geometry – an Introduction.* Springer-Verlag, New York, 1985.