In this lecture, we will continue our discussion on the orthogonal range reporting problem defined as follows. Let $P$ be a set of $n$ points in $\mathbb{R}^2$. Given an axis-parallel rectangle $q$, a range query reports all the points in $P \cap q$. We want to preprocess $P$ into a data structure so that all queries can be answered efficiently. We have learned a range tree structure of $O(n \log n)$ space that answers a query in $O(\log n + k)$, where $k$ is the number of points reported. In this lecture, we will improve the space to $O(n \log n / \log \log n)$. The resulting structure is optimal among all the pointer machine structures—it can be proved that every pointer machine structure must use $\Omega(n \log n / \log \log n)$ space to ensure $O(\log n + k)$ query time.

We will first describe two other data structures, the interval tree and the priority search tree, in the next two sections. The improved range tree is described in Section 3.

1 Interval Tree

Let us take a short break from range reporting by focusing first on the stabbing problem. Let $I$ be a set of $n$ intervals in $\mathbb{R}$. We want to pre-process $I$ into a structure so that, given a query value $q \in \mathbb{R}$, all the intervals in $I$ containing $q$ can be reported efficiently (see the figure below). Such a query is called a stabbing query. The interval tree, presented next, uses $O(n)$ space, and solves a query in $O(\log n + k)$ time, where $k$ is the number of intervals reported.

**Figure 1: A stabbing query reporting $s_1$ and $s_4$**

**Structure.** Let $X$ be the set of (both left and right) endpoints of the intervals in $I$. Create a binary search tree $T$ on $X$ (all the endpoints in $X$ are stored at the leaves of $T$). Recall that each node $u$ in $T$ stores a value $x_u$ such that all the endpoints in the right subtree of $u$ are at least $x_u$, while those in the left subtree of $u$ are less than $x_u$.

Each node $u$ in $T$ is associated with a set $S_u$ of intervals in $I$ determined as follows: an interval $I \in I$ is assigned to the stabbing set of the highest node $u$ satisfying $x_u \in I$. We also associate $u$ with two linked lists $L_u$ and $R_u$, where $L_u$ ($R_u$) stores the left (right) endpoints of the intervals in $S_u$, sorted by their coordinates. We refer to $S_u$ as the stabbing set of $u$. Since each interval in $I$ appears in the $S_u$ of only one node $u$ in $T$, the overall space is $O(n)$.
The priority search tree was designed to solve the so-called 3-sided orthogonal range reporting, which is a restricted version of the orthogonal range reporting problem we ultimately want to solve. Once again, let \( P \) be a set of \( n \) points in \( \mathbb{R}^2 \). A rectangle is said to be 3-sided if it has the form \([x_1, x_2] \times [y, \infty)\). Given a 3-sided rectangle \( q \), a query reports all the points in \( P \cap q \). A priority search tree on \( P \) uses \( O(n) \) space, and answers each query in \( O(\log n + k) \) time, where \( k \) is the number of points reported.

**Structure.** Create a binary search tree \( T \) on the x-coordinates of the points in \( P \) (with all the coordinates stored at the leaves). Each node \( u \) naturally corresponds to a vertical slab \( \sigma(u) = [x, x') \times (-\infty, \infty) \) defined as follows. If \( u \) is a leaf, then \( x \) is the x-coordinate stored at \( u \), and \( x' \) is the x-coordinate stored at the leaf succeeding \( u \) (\( x' = \infty \) if no such leaf exists). If \( u \) is an internal node, then \( \sigma(u) \) unions those of its child nodes.

We associate each node \( u \) with a pilot point, denoted as \( \text{pilot}(u) \). If \( u \) is the root of \( T \), \( \text{pilot}(u) \) is the highest point in \( P \) (i.e., the one with the maximum y-coordinate). In general, \( \text{pilot}(u) \) is the highest point among all the points stored in the subtree of \( u \), excluding the prior points of the proper ancestors of \( u \). It Notice that every point in \( P \) is a pilot point of exactly one node. The overall space consumption is therefore \( O(n) \).

**Query.** Let \( q = [x_1, x_2] \times [y, \infty) \) be the query rectangle. Let \( z_1, z_2 \) be the two leaves whose slabs contain \( x_1, x_2 \), respectively. Let \( \ell \) be the horizontal line intersecting the y-axis at \( y \).

We follow two root-to-leaf paths \( \Pi_1, \Pi_2 \) leading to \( z_1, z_2 \), respectively. We first check the pilot points of the nodes on \( \Pi_1 \cup \Pi_2 \). For each such point \( p \), report it if \( p \in q \). Let \( \hat{u} \) be the lowest common ancestor of \( z_1, z_2 \). Define \( \Pi'_1 \) (\( \Pi'_2 \), resp.) to be the path from \( z_1 \) (\( z_2 \), resp.) to the child of \( \hat{u} \) that is an ancestor of \( z_1 \) (\( z_2 \), resp.).
Let \( u \) be a node that is a right child of a node on \( \Pi'_1 \), or a left child of a node on \( \Pi'_2 \) (there are \( O(\log n) \) such nodes). We invoke the following algorithm \( \text{report-above-line}(u, \ell) \) to report all the pilot points that are stored in the subtree of \( u \), and are on or above \( \ell \). First, \( \text{report-above-line}(u, \ell) \) checks whether the pilot point \( p \) of \( u \) is on or above \( \ell \). If not, the algorithm terminates right away (think: why?). Otherwise, the algorithm reports \( p \), and finishes if \( u \) is a leaf. If \( u \) is an internal node with child nodes \( v_1, v_2 \), the algorithm recursively invokes \( \text{report-above-line}(v_1, \ell) \) and \( \text{report-above-line}(v_2, \ell) \).

Now we prove that the query time is \( O(\log n + k) \). It suffices to prove that \( \text{report-above-line}(u, \ell) \) costs \( O(1 + k') \) time, where \( k' \) is the number of points reported by this algorithm. This can be seen easily as follows. A node \( v \) is accessed if and only if a point has been reported at the parent of \( v \). Hence, the number of such nodes is \( O(1 + k') \).

3 Improved Range Tree

Let us return to the orthogonal range reporting problem. Recall that the input set \( P \) contains \( n \) points in \( \mathbb{R}^2 \). A query is given a rectangle \( q = [x_1, x_2] \times [y_1, y_2] \) and needs to report all the points in \( P \cap q \).

**Structure.** Create an \( f \)-ary search tree \( \mathcal{T} \) on the \( x \)-coordinates of \( P \) where \( f = \log n \) (with all the \( x \)-coordinates stored at the leaves). The height of \( \mathcal{T} \) is \( O(\log_{\log n} n) = O(\log n / \log \log n) \). As before, each node \( u \) of \( \mathcal{T} \) naturally corresponds to a vertical slab \( \sigma(u) \) in \( \mathbb{R}^2 \). Denote by \( P(u) \) the set of points whose \( x \)-coordinates are stored in the subtree of \( u \). In other words, \( P(u) = \sigma(u) \cap \mathcal{T} \).

Each internal \( u \) is associated with several secondary structures. Let \( u_1, \ldots, u_f \) be the child nodes of \( u \). For each child node \( u_i \), associate with \( u \) two priority search trees \( \Pi_\square(u, u_i) \) and \( \Pi_\sqsubset(u, u_i) \) on \( P(u_i) \), which support 3-sided range queries whose search regions have the shapes \( \square \) and \( \sqsubset \), respectively. Furthermore, for each \( i \in [1, f] \), create a linked list \( \Xi(u, u_i) \) storing the points of \( P(u_i) \) sorted by \( y \)-coordinate. Finally, \( u \) is also associated with an interval tree \( \Gamma(u) \) built on a set \( S(u) \) of 1d intervals obtained as follows. For each \( i \in [1, f] \), let \( Y_1, \ldots, Y_m \) be the \( y \)-coordinates of the points of \( P(u_i) \) in ascending order, where \( m = P(u_i) \). Generate a set \( S_i(u) \) of \( m \) 1d intervals where the \( j \)-th (\( 1 \leq j \leq n \)) equals \( (Y_{j-1}, Y_j] \) (defining a dummy \( Y_0 = -\infty \)). The interval carries a pointer to the node of \( \Xi(u, u_i) \) where the point corresponding to \( Y_j \) is stored, so that once \( (Y_{j-1}, Y_j] \) is retrieved, we can jump to that node in \( O(1) \) time. \( S(u) \) is the union of \( S_1(u), \ldots, S_f(u) \).

**Space.** The space of all the secondary structures of an internal node \( u \) is \( O(|P(u)|) \). Hence, the secondary structures of all the nodes at an internal level of \( \mathcal{T} \) occupy \( O(n) \) space in total. The overall space consumption is therefore \( O(n \log n / \log \log n) \).

**Query.** Given a query with search region \( q = [x_1, x_2] \times [y_1, y_2] \), we first find the lowest node \( u \) whose slab contains \( q \). Let \( u_1, \ldots, u_f \) be the child nodes of \( u \). Suppose that \( x_1 = (x_2, \text{resp.}) \) falls in some \( \sigma(\alpha) \) (\( \sigma(\beta) \), resp.), where \( 1 \leq \alpha < \beta \leq f \). Clearly, only the points in \( P(u_\alpha), P(u_{\alpha+1}), \ldots, P(u_\beta) \) can possibly fall in \( q \). We find \( P(u_\alpha) \cap q \) by performing a 3-sided range query on \( \Pi_\square(u_\alpha) \) with the search region \( q \cap \sigma(u_\alpha) \), and similarly, find \( P(u_\beta) \cap q \) by performing a 3-sided range query on \( \Pi_\sqsubset(u_\beta) \). The cost of the two queries is \( O(\log n + k_1) \) where \( k_1 \) is the number of points retrieved by them.

It remains to report the qualifying points in \( P(u_{\alpha+1}), P(u_{\alpha+2}), \ldots, P(u_{\beta-1}) \). For this purpose, we search \( \Gamma(u) \) by issuing a stabbing query with value \( y_1 \), which reports at most one interval from each \( S_i(u) \) where \( 1 \leq i \leq f \). The cost is \( O(\log n + f) = O(\log n) \) time. Now consider each \( i \in [\alpha + 1, \beta - 1] \), and let \( (Y_{j-1}, Y_j] \) be the interval fetched from \( S_i(u) \). Then, the point \( p \) that corresponds to \( Y_j \) is the lowest point in \( P(u_i) \) on or above the horizontal line \( y = y_1 \). We jump
to the node of $\Xi(u, u_i)$ storing $p$ in $O(1)$ time, and then scan in $\Xi(u, u_i)$ the points of $P(u_i)$ in ascending order of their $y$-coordinates starting from $p$, until seeing the first point with a coordinate greater than $y_2$. All the scanned points falling in $q$ are reported. Figure 3 illustrates the above process assuming $f = 8$. Here, $\alpha = 2$ and $\beta = 6$. The stabbing query fetches $p_i$ from each $\sigma_i$ ($1 \leq i \leq 8$). For each $\sigma_j$ for $3 \leq j \leq 5$, we scan $P(u_j)$ bottom up from $p_j$ until reaching a point outside the upper boundary of $q$.

In total, we spend $O(\log n + f + k_2) = O(\log n + k_2)$ time reporting the points in $P(u_{\alpha+1})$, $P(u_{\alpha+2})$, ..., $P(u_{\beta-1})$, where $k_2$ is the number of these points. Overall, the total cost is $O(\log n + k_1 + k_2) = O(\log n + k)$. 

Figure 3: Answering a range query at $u$ ($\alpha = 2$, $\beta = 6$, $f = 8$)