Orthogonal Range Search using Range trees

Data: A set $P$ of $n$ points in $d$-dimensional space. Assume that no two points have identical values for the same coordinate.
Query: $[x_1, x_2, \ldots, x_d], [x'_1, x'_2, \ldots, x'_d]$.
Output: All points in $P$ that fall within the $d$-dimensional rectangle $[x_1, x_2, \ldots, x_d] \times [x'_1, x'_2, \ldots, x'_d]$.

1-Dimensional range search

- Organize the data as a balanced binary search tree with all points at the leaves.
- The internal nodes contain the value of the largest element in the left subtree.
- This can be done in $O(n \log n)$ time.
\( v \leftarrow \text{root}(T) \)
\[
\begin{aligned}
&\text{while } v \text{ is not a leaf and } (x' \leq x_v \text{ or } x > x_v) \\
&\quad \text{if } x' \leq x_v \text{ then} \\
&\quad \quad v \leftarrow \text{leftchild}(v) \\
&\quad \text{else} \\
&\quad \quad v \leftarrow \text{rightchild}(v) \\
&\text{return } v
\end{aligned}
\]

\( v_{\text{split}} \leftarrow \text{FindSplitNode}(T, x, x') \)
\[
\begin{aligned}
&\text{if } v_{\text{split}} \text{ is a leaf} \\
&\quad \text{check if } v_{\text{split}} \text{ is in the range, and report it if it is} \\
&\quad \text{else} \\
&\quad \quad v \leftarrow \text{leftchild}(v_{\text{split}}) /* \text{search left subtree} */ \\
&\quad \quad \text{while } v \text{ is not a leaf} \\
&\quad \quad \quad \text{if } x \leq x_v \text{ then} \\
&\quad \quad \quad \quad \text{1-D RangeQuery}(T_{\text{associated}}(\text{rightchild}(v)), y, y') \\
&\quad \quad \quad \quad v \leftarrow \text{leftchild}(v) \\
&\quad \quad \quad \text{else} \\
&\quad \quad \quad \quad v \leftarrow \text{rightchild}(v) \\
&\quad \quad \text{Report } v \text{ if it is in the range} \\
&\quad /* \text{Similar search in the right subtree} */
\end{aligned}
\]

**Lemma 5.1:** The above algorithm reports exactly those points that lie in the query region.

**Lemma 5.2:** \( P \) can be stored in a balanced BST taking \( O(n) \) storage and \( O(n \log n) \) time to construct, such that the points in the query range can be reported in \( O(\log n + k) \) time, where \( k \) is the number of points in the range. Note that the time complexity is output sensitive.

**Range trees in 2-D**

- The main tree is a balanced BST on the \( x \) coordinates, built as in the 1-D case. The points are stored in the leaves.
- The interior nodes contain an additional pointer to another balanced BST, which contains \( P(v) \) in its leaves, where \( P(v) \) is the set of points in the leaves of the subtree rooted at \( v \). This associated structure of \( v \) is built on the \( y \) coordinates of \( P(v) \).
- This can be constructed in \( O(n \log n) \) time and uses \( O(n \log n) \) storage.

\( v_{\text{split}} \leftarrow \text{FindSplitNode}(T, x, x') \)
\[
\begin{aligned}
&\text{if } v_{\text{split}} \text{ is a leaf} \\
&\quad \text{check if } v_{\text{split}} \text{ is in the range, and report it if it is} \\
&\quad \text{else} \\
&\quad \quad v \leftarrow \text{leftchild}(v_{\text{split}}) /* \text{search left subtree} */ \\
&\quad \quad \text{while } v \text{ is not a leaf} \\
&\quad \quad \quad \text{if } x \leq x_v \text{ then} \\
&\quad \quad \quad \quad \text{1-D RangeQuery}(T_{\text{associated}}(\text{rightchild}(v)), y, y') \\
&\quad \quad \quad \quad v \leftarrow \text{leftchild}(v) \\
&\quad \quad \quad \text{else} \\
&\quad \quad \quad \quad v \leftarrow \text{rightchild}(v) \\
&\quad \quad \text{Report } v \text{ if it is in the range} \\
&\quad /* \text{Similar search in the right subtree} */
\end{aligned}
\]

**Lemma 5.7:** Time complexity of the search is \( O(\log^2 n + k) \) time to report \( k \) points.

Proof: Time spent in each 1-D RangeQuery call is \( O(\log n + k_v) \), if there are \( k_v \) points reported under \( v \).

Total time = \( \sum_{v \text{ in 1-D calls}} O(\log n + k_v) \). Since there are \( O(\log n) \) vs, with points in them being distinct, the total time is \( O(\log^2 n + k) \).