Range Tree

CS 5633 Analysis of Algorithms

Computer Science
University of Texas at San Antonio

October 14, 2024

The Range Search Problem

The Range Search Problem

- Today we will discuss **orthogonal range searching** in which we are given *n* points in a *d* dimensional geometric space.
- We will perform queries with an axis-aligned box (a rectangle in 2D), and we wish to answer questions regarding the points contained inside of the box.
  - How many points are in the box?
  - List all of the points in the box.
  - For example, given points (2,2) (1,1) (3,2), how many points are in the box (0,0)-(4,4)?

The Range Search Problem Example

- This is an important application in databases. Suppose we have a database with n records each with d numeric fields. Each record will be represented as a point in the d dimensional geometric space, and value assigned to field i is the coordinate of the point in the ith dimension.
  - For example, select the person who has a salary between S_1 and S_2, and has worked for at least y years.

Solving The Range Search Problem

- We want to maintain a data structure which will allow us to answer queries quickly.
- We will consider a *static data structure* (i.e. we will not add or delete points from our data structure).
- We are allowed to spend time preprocessing our input (i.e. the preprocessing time is one time operation and so it can be more expensive than the query time).

1-D Range Search Problem

The 1D Range Search Problem

- Consider the problem in 1D (all of the points are along a line).

- What are the values that fall in the range of (1, 5)?

A Solution to The 1D Range Search Problem

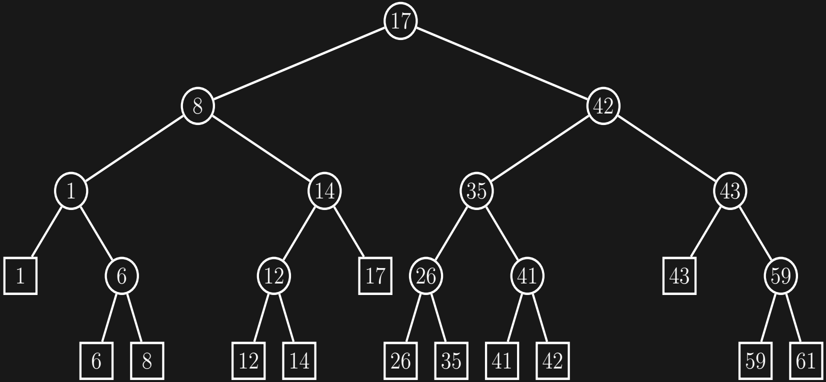
- Since we are allowed preprocessing time, we can sort the points according to their left to right ordering in $O(n \log n)$ time.
  - Since the points are sorted, we can find the first point by binary search in $O(\log n)$ time.
  - If there are k points in the query range, we can step through the sorted list and report all of the k points in $O(k)$ time.
  - Total running time is $O(\log n + k)$.
- Nice simple idea, but it does not generalize to higher dimensions, and the ultimate goal is to get something that works in any d dimensional space.

The 1D Range Tree

- Another approach to the 1D problem: use a binary search tree.
- Store each of the points in the *leaves* of the tree.
- To help us search the tree, each internal node x stores $x.key$ which is the maximum key of any leaf in the left subtree of x.
- For this sub-tree,
  - For each node y in the left subtree of x, we have $y \leqslant x$.
  - For each node y in the right subtree of x, we have $y > x$.

An Example of 1D Range Tree

- Note that leaves nodes have the actual values, while internal nodes record the maximum value of the left sub-tree.



The Intuition of 1D Range Tree

- Consider a search range of $[l, h]$.
- For any interval node with key v
  - Its left sub-tree has leaves with values less or equal to v.
  - Its right sub-tree has leaves with values larger or equal to v.
  - Therefore,
    - If $v < l$, only its right sub-tree may have values in (l, h).
    - If $v \geqslant h$, only its left sub-tree may have values in (l, h).
    - If $l \leqslant v < h$, when both sub-trees may have values in (l, h). We call this type node, a split node.

The Intuition of 1D Range Tree cont.

- For any interval node with key v.
  - If this node on the left sub-tree of the split node.
  - If $l \leqslant v$, all its right sub-tree is in (l, h).
- For any interval node with key v.
  - If this node on the right sub-tree of the split node
  - If $v \leqslant h$, all its left sub-tree is in (l, h).

1D Range Tree Search Algorithm

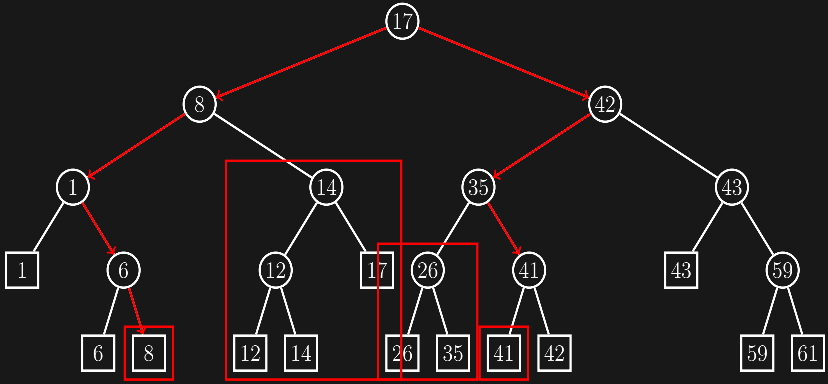
- Based on the previous intuitions, we can implement the search algorithm.

Algorithm 1: 1-D Range Tree Search

```
Function OneD_Search(root r, range [l, h])
    T = r;
    // find the split node;
    while (T is not leaf) and (l > T.key or h ⩽ T.key) do
        if h ⩽ T.key then
            T = T.left;
        else
            T = T.right;
    if T is leaf then
        if l ⩽ T.val ⩽ h then
            output(T.val);
        return;
    X = T.left;
    // handle the left sub-tree of split node;
    while X is not leaf do
        if l ⩽ X.key then
            output_subtree(X.right); X = X.left;
        else
            X = X.right;
    X = T.right;
    . . . ; // handle the right sub tree of split node. Code not shown;
```

1D Range Tree Search Example.

- Suppose we want to search for values in range $[7, 41]$.
- Red arrows indicate search path.
- Red rectangles circle the sub-trees that are directly outputted.



Run Time of 1D Range Tree Search

- Tree is balanced, so the depth is $O(\lg n)$.
- Finding the split node traverses the tree and takes at most $O(\lg n)$ time.
- The search of the left and right sub-trees of the split node takes at most $O(\lg n)$ time.
- The run-time of outputting sub-tree is linear of the leaf node count. Therefore, the output time is $O(k)$, when there are k values within range.
  - For any binary tree with k leaves, the total node count is no larger than $2k$.
  - Therefore the traversal cost of a binary tree is no larger than $2k$.
- The total cost of 1D range tree search is then $O(k + \lg n)$.

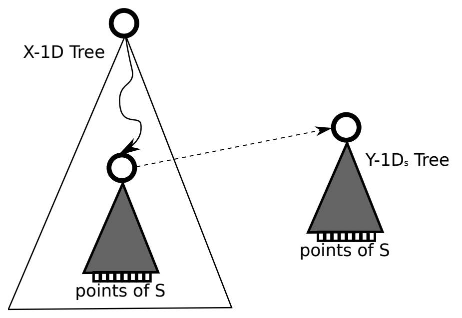
Extending Range Tree to *d* Dimensions

Idea of 2D Range Search

- For 2D search, the range represents a box and is defined by two points (l_x, l_y) and (h_x, h_y).
- The idea is to do two 1D search on each dimensions
  1. First search on the x-coordinates to find points within range $[l_x, h_x]$.
  2. Then do a second search on points found in the first search using y-coordinates to find points within range $[l_y, h_y]$.
  3. The points found from the 2nd search is then the points within the 2D range.

Extending Range Tree to 2D

- We first build the 1D range tree using *x*-coordinates.
  - Let's name this tree $\mathrm{X\text{-}1D}$ tree.
- For each interval node *T* of the $\mathrm{X\text{-}1D}$ tree, its leaves represent a set of points. Let this set be *S*.
  - For the points of *S*, build a 1D range tree using their *y*-coordinates. Let's name this tree $\mathrm{Y\text{-}1D}_S$.
  - Add a pointer in node *T*, which points to the root of $\mathrm{Y\text{-}1D}_S$ tree.



2D Range Tree Search

1. Search the $\mathrm{X\text{-}1D}$ tree using *x*-coordinates, using the 1D search algorithm.
2. When it is determined that the sub-tree at node *T* in the $\mathrm{X\text{-}1D}$ tree has points within range $[l_x, h_x]$, switch to search *T*'s $\mathrm{Y\text{-}1D}_s$ tree.
3. Search *T*'s $\mathrm{Y\text{-}1D}_s$ tree using *y*-coordinates. Output the keys within range $[l_y, h_y]$ in $\mathrm{Y\text{-}1D}_s$.
4. Repeat the search until $\mathrm{X\text{-}1D}$ tree is completely searched.

Run Time of 2D Range Tree Search

- Searching the $\mathrm{X\text{-}1D}$ tree takes $O(\lg n)$ time, as searching any 1D range tree.
- Searching the $\mathrm{X\text{-}1D}$ may find at most $O(\lg n)$ sub-trees, which may contain points within targeted range.
  - A second 1D range tree search will be conducted on these sub-trees' $\mathrm{Y\text{-}1D}$ trees. That is, there are at most $O(\lg n)$ $\mathrm{Y\text{-}1D}$ trees to search.
  - Each $\mathrm{Y\text{-}1D}$ tree is at most $O(\lg n)$ tall. Therefore, the cost of searching one $\mathrm{Y\text{-}1D}$ tree is $O(\lg n)$.
  - The cost of searching $\mathrm{Y\text{-}1D}$ trees is then $O(\lg^2 n)$.
- The output cost is still $O(k)$.
- The total search cost is then $O(\lg n + \lg^2 n + k) = O(\lg^2 n + k)$.

2D Range Tree Space Cost

- $\mathrm{X\text{-}1D}$ tree has n leaves, so it takes $O(n)$ spaces.
  - For any binary tree with k leaves, the total node count is no larger than $2k$.
- At each level of $\mathrm{X\text{-}1D}$ tree,
  - The $\mathrm{Y\text{-}1D}$ trees connected to this level has at most n leaves.
    - Each $\mathrm{Y\text{-}1D}$ trees at the same level of $\mathrm{X\text{-}1D}$ tree has different leaves.
    - There are no more than n leaves (values).
  - The $\mathrm{Y\text{-}1D}$ trees connected to this level takes $O(n)$ spaces.
- All $\mathrm{Y\text{-}1D}$ trees at all levels of $\mathrm{X\text{-}1D}$ tree take at most $O(n \lg n)$ space.
- Total space cost of 2D range tree is then $O(n \lg n + n) = O(n \lg n)$.

Extending Range Tree to *d*-Dimension

- For 3D points, we can add additional *z*-coordinate 1D trees to $\mathrm{Y\text{-}1D}$ trees.
- We can add extra 1D trees for each added dimension.
- The 1D trees are searched one dimension at a time.
- The cost of search a *d*-dimension range tree is $O(\lg^d n + k)$.