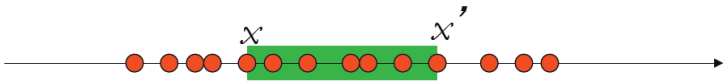
Geometric Data Structures

Motivation

Say we have a data table of *employee(age, salary, start date, address)*, and we want to find the records with $salary \in [40k, 50k]$ AND $age \in [25,40]$. Doing this linearly is easy but very inefficient.

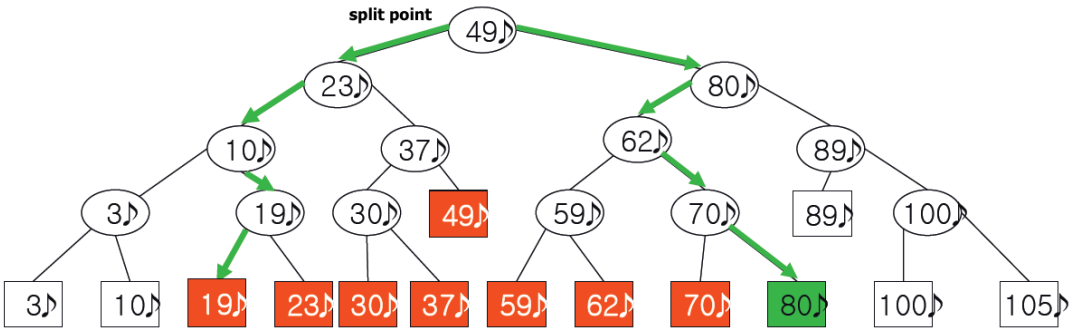
1-D Range Search



Since we are interested in the data itself, if k is the number of output records, it has to be a part of the complexity. In 1-D it is easy: binary search to find the lower and upper limits - $\lg n$ – and then k to traverse all.

We can use a tree or a skip-list to allow a balanced binary search and have insertion and deletion cost $O(\lg n)$. When using a BST all values are in the leafs. The query $[x, x']$:

- Locate x, x' in the tree T: $2 \cdot \lg n$
- Do an in-order traversal from x to x'
- The split point – the lowest common ancestor, finding it - $\lg n$



The pseudo code for finding the split point is in the slides. Due to the property of a split point, we are guaranteed that all points in between will be printed.

The traversal is actually between u, u' where u is the biggest node in the tree with value $\leq x$, and u' is the smallest node in the tree with value $\geq x'$.

Algorithm: 1-D-Range-Query(T, x, x'): (full pseudo code in the slides)

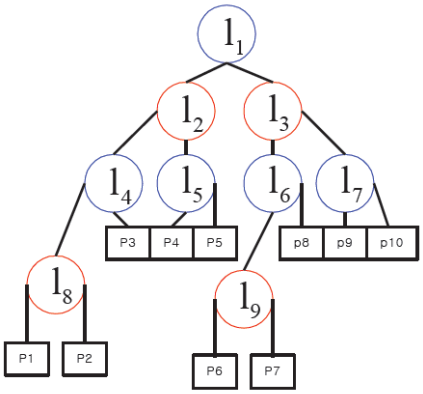
- Report-subtree prints the inner nodes.
- Going up from u to the split point, we print the right-trees; going down from it to u' we print the left-trees.
- For any subtree with k_i values in the nodes, since each subtree is a balanced subtree, the total number of nodes we will traverse is k_i (nodes) + k_i (all inner nodes), so the total for all nodes that need to be printed is $2k$.
- Total running point: $\Theta(k + \lg n)$.

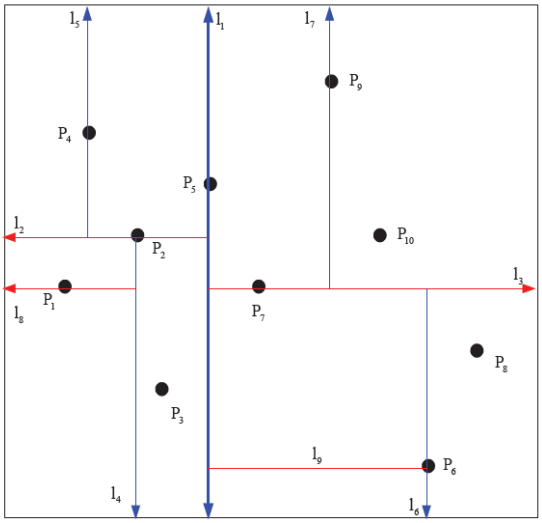
K-D Tree:

This is a generalization of the above to the K-dimension. Start with a 2-D example:

- We project all points to the x axis, and split them in the middle, with $\frac{n}{2}$ points at each side. Call that separating line l_1.
- On the next step we do the same with respect to the y axis, such that on each subtree we have a substitution into $\frac{n}{4}$ on each side.
- Continue this way interchangeably.

In every split we divide the sizes by 2, so the maximum depth would be $\lg n$.





- The lines on some path define a region in which each node at the end of that path resides.

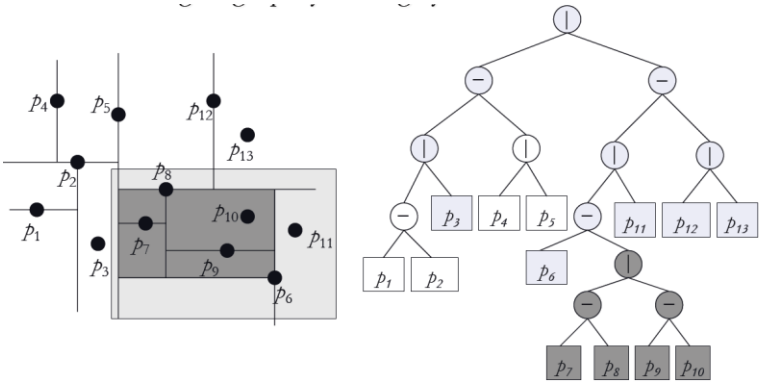
In K-dimension we do the same interchangeably, going through levels $1,2,\ldots,K$.

The space is linear; the time is $O(n \lg n)$ – we can find the median at each part and we divide $\lg n$ times, the total is $n \lg n$.

The recurrence for any dimension is $T(n) = O(n) + 2T\left(\frac{n}{2}\right) = O(n \lg n)$.

<u>What is a search region in this setup?</u>

A search (in 2-D) is a **rectangle** x, x', y, y'. In the general case it could be any kind of object. The algorithm we will look at works the same for any constant number of sides. we have to check all light-grey areas (and once we got to the dark-grey – everything in it is in our query):



The running time of this algorithm is $\sqrt{n}$.

Observations:

- We have to search all subtrees rooted at v if the range R intersects $region(v)$.
- If $region(v)$ is contained in R, all the leafs of the subtree of v are in the region.

SearchKD-Tree(v, R):

- To find whether a leaf is within a rectangle region is constant time.
- If it is not a leaf, we can find the intersection of R with $region(v)$ (which can be a bounded rectangle or unbounded).
  - If that region is entirely in R, return all its leafs.
  - Otherwise we need to check both left and right regions for intersections with R and continue recursively to any of the sides that are relevant (could be both right and left regions).

Lemma:

The running time is $O(k + \sqrt{n})$.

Proof:

We will be reviewing at most $\sqrt{n}$ intermediate nodes. The total time of ReportSubtree is $O(k)$, we have seen it already. So we need to show that at most $\sqrt{n}$ intermediate nodes are encountered in the traversal.

Each region is basically 4 lines (upper, lower, left, right). To bound the number of such nodes we need to bound the number of regions intersected by a vertical line. The number of intersection is bounded by the total number of regions it will be intersecting.

Define $Q(n)$ the number of intersected regions in an n point KD tree whose root contains vertical splitting lines. We know: $Q(n) = 2 + 2Q\left(\frac{n}{4}\right)$ since two levels created 4 regions, and the edge of the rectangle will intersect at most 2 of these 4 regions. The same happens for the right and the left, that's why it is 2 times $Q\left(\frac{n}{4}\right)$. By Master's Theorem: $Q(n) = O(\sqrt{n})$.

In the mix of all edges of the rectangle we will do that 4 times much, concluding to still $O(\sqrt{n})$.

General case:

Construction:

- Storage: $O(dn)$
- Time: $O(dn \lg n)$

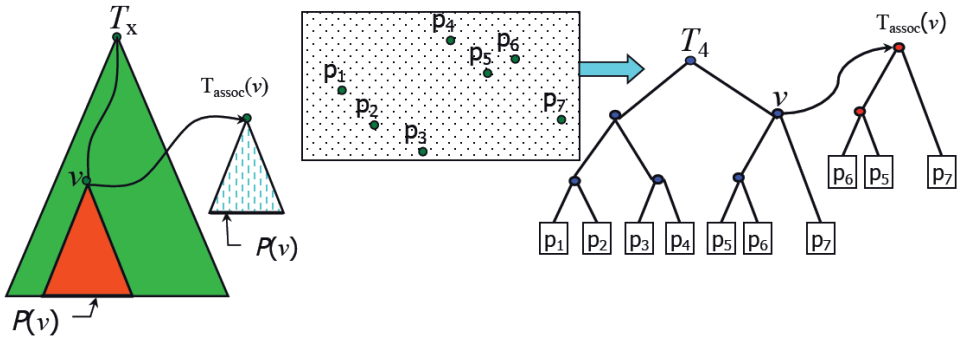
Query: $O\left(n^{1-\frac{1}{d}} + k\right)$.

Range Trees

In the previous structure any value appeared some constant number of times. Here the expected number of appearances is $O(\lg n)$. Let P denote a set of n points and R the $[x, x'] \times [y, y']$ query rectangle.

The structure is a binary tree with respect to x, and at each node there is the same subtree sorted with respect to y. Denote the leaves if a subtree rooted at v as $P(v)$.

Any point in the range $[x, x']$ is part of at most $\lg n$ $P(v)$ sets along the path. We are interested only in the nodes in $P(v)$ that have y values in $[y, y']$.



Query:

- First we identify the $O(\lg n)$ subtrees to search in, with respect to the x value.
- For each of those trees we have a $\lg n$ search with respect to the y value.

Running time:

Explanations in the slides - $O(k + \lg^2 n)$.