Computational Geometry

Geometric Data Structure (Lecture 1, 2 by Amit Kumar Sir)

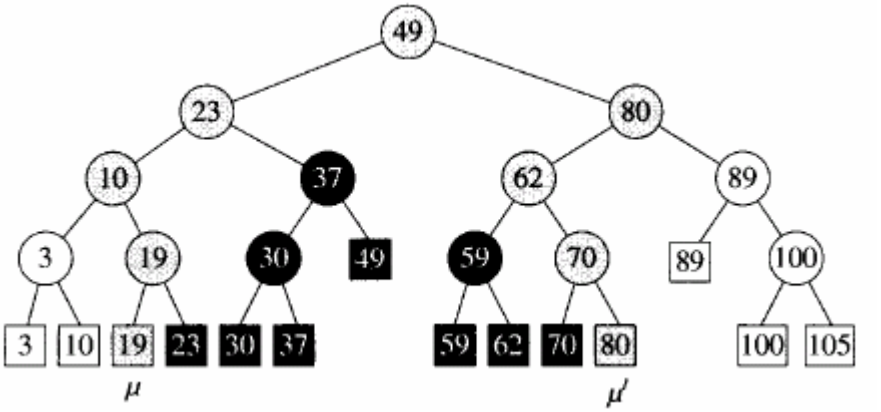
Scribe- Gaurav Gupta
2002435

Range Queries

One Dimensional Range Searching

*Given a one dimensional space, query asks for the points inside an interval *[x:x]'*.
Let $P = \{p_1, p_2, \ldots, p_n\}$ be the given set of points on the real line. We can solve the 1-dimensional range searching problem using a balanced binary search tree T . The leaves of T store the point P and internal nodes of T store splitting values to guide the search. Let the value at node v is x_v. Left subtree of a node v contains all the points smaller than of equal to x_v and right subtree contains all the points strictly greater than x_v.

To report the points in the range query *[x:x']* we search with x and x' in the tree T. let u and u' be the leaves where the searches end. Then the points in the interval *[x:x']* are ones stored between the leaves u and u'. e.g. search with the interval *[18:77]* –

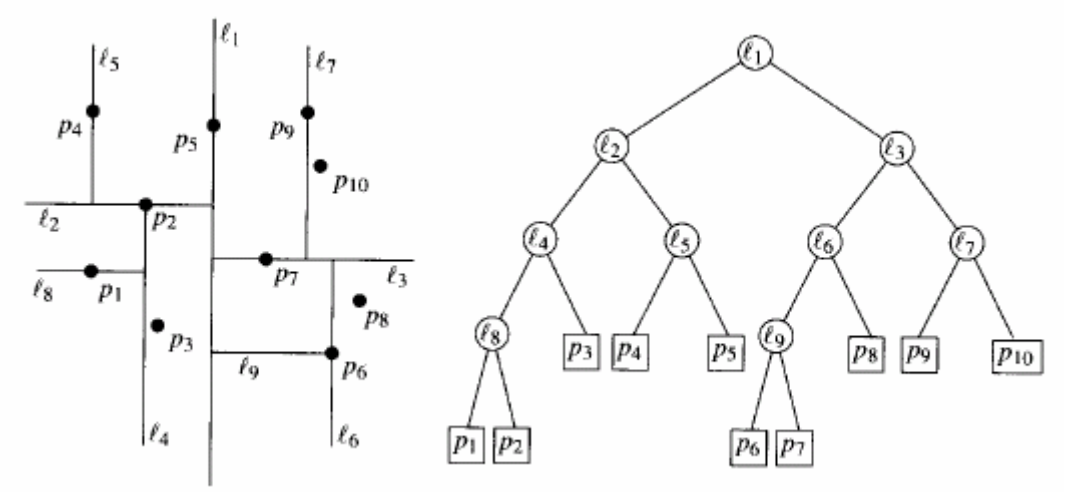


Start from the root and recursively call the function on the left and right subtree if it satisfies certain conditions. It's easy to see that we can build a balanced tree using median as the split vertex.

Complexity: - The tree takes $O(n)$ space and $O(\log n)$ time in preprocessing. Each query take $O(\log n + k)$ time. Where k is the number of favorable cases.

2 Dimensional Range Queries

Each point has two values: its x coordinate and its y coordinate. Therefore we split on x-coordinate and next on y-coordinate, then again on x coordinate and so on. In general we split with a vertical line at nodes whose depth is even and we split with a horizontal line at nodes whose depth is odd. The time to build the 2-D tree is as follows.

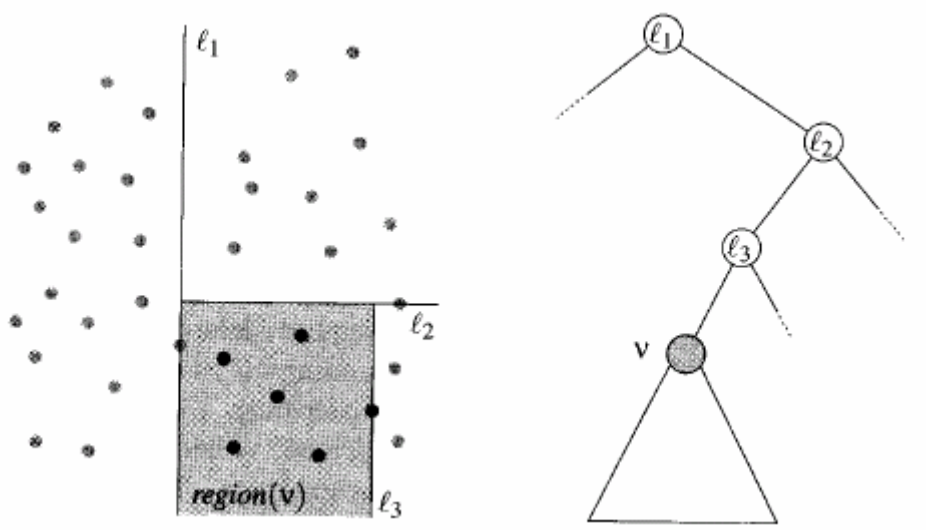


The median can be found in *O(n)* time. Time to build the 2-D tree is

$$T(n) = \begin{cases} O(1), & \text{if } n = 1, \\ O(n) + 2T(\lceil n/2 \rceil), & \text{if } n > 1, \end{cases}$$

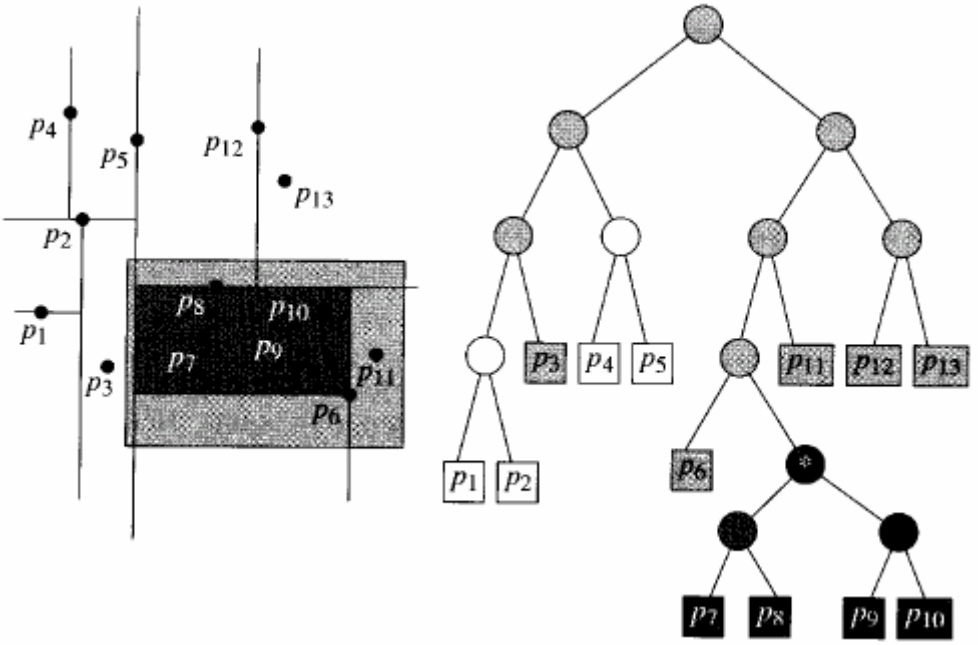
Thus *T(n) = O(n log n)* and space required is *O(n) = O(number of elements in the dataset)*

The region corresponding to a node *v* is a rectangle which can be bounder or unbounded on one or more sides. For example :-



Thus we have to search the subtree rooted at *v* iff the query rectangle intersects *region(v)*. We traverse the 2-D tree, but visit only nodes whose region is intersected by the query rectangle. When a region is fully contained in the query rectangle, we can report all the points stored in its subtree. When the traversal reaches a leaf, we have to check whether the point stored at the leaf is contained in the query region and, if so, report it. The grey nodes are visited when we query with the grey rectangle. The node marked with a star corresponds to a region that is completely contained in the query rectangle; in the figure this rectangular region is shown darker. Hence, the dark grey subtree rooted at this node is traversed and all points stored in it are reported. The other leaves that are visited

correspond to regions that are only partially inside the query rectangle. Hence, the points stored in them



Must be rested for inclusion in the query range; this results in points p_6 and p_{11} being reported, and points p_3, p_{12} and p_{13} not being reported.

Search(Q, v)

If $R_v \in Q$ *then report points in* R_v

Else

Let R_v^l *and* R_v^r *be rectangle associated with the children*

If $R_v^l \bigcap Q \neq \phi$

$Search(Q, R_v^l)$

If $R_v^r \bigcap Q \neq \phi$

$Search\ (Q, R_v^r)$

Let *Q(n)* be the number of interested region in 2-D tree storing *n* points. To write recurrence for *Q(n)* we go two step down in the tree. Each of the four nodes at depth two corresponds to a region containing *n/4* points. Two of the four nodes correspond to the intersected region, so we have to count the number of interested region the subtree. Hence *Q(n)* satisfy the recursive equation :-

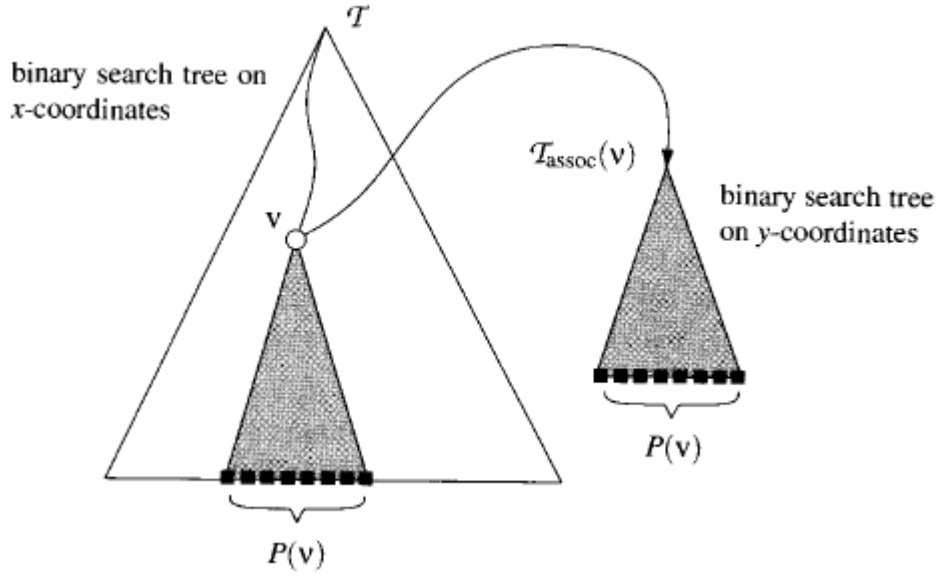
$$Q(n) = \begin{cases} O(1), & \text{if } n = 1, \\ 2 + 2Q(n/4), & \text{if } n > 1. \end{cases}$$

This recurrence solves to $Q(n) = O(\log \sqrt{n})$

Range Trees:-

2-D trees have use less space but have high query time. Range tree have higher space complexity $O(n \log n)$ but have low query time of $O(\log^2 n + k)$ for two dimensions and $O(\log^d n + k)$ for d-dimentions.

Let $[x:x'] * [y:y']$ be the range query. We first concentrate on finding a point whose x – coordinate lies in $[x:x']$ and worry about y coordinate later. Thus this query is exactly similar to the query in 1-Dimension. Lets call the subset of points stored in the leaf of the subtree rooted at a node v the canonical subset of $v = P(v)$. We are not interested in all the points in $P(v)$ but only in those points whose y-coordinate lies in the range $[y:y']$. Thus we can solve this provided we have a binary search tree on y-coordinate v.
For any internal or leaf node v in T, the canonical subset $P(v)$ is stored in a balanced binary tree $T_{assoc}(v)$ on the y-coordinate of the points. The node v stores a pointer to the root of $T_{assoc}(v)$ whish is called the associated structure of v.



A range tree on a set of n points in a plane requires $O(n \log n)$ space.
Because a point can appear in one of the associated structures in a row and there are $O(\log n)$ row. The space required is $O(n \log n)$.

A search query takes $O(\log^2 n + k)$ time.
In an Associated structure it takes a maximum $O(\log n + k')$ time. Since there can be maximum $O(\log n)$ subtree be search for, the query takes $O(\log^2 n + k)$ time.

For *d* -Dimensions

Let P be the set of n points in d-dimensional space, where $d \geq 2$. A range tree for P uses $O(n \log^{d-1} n)$ storage and it can be constructed in $O(n \log^{d-1} n)$ time. One can report the points in P that lie in a rectangular query range in $O(\log^d n + k)$ time, where k is the number of reported points.