Range Tree

Range Search 1-D

Given $S \subset \mathbb{R}$ with $|S| = n$
Query: interval $I = [l, h] = \{x \in S \mid l \le x \le h\}$
Report: $S \cap I$

Solution: Balanced binary search tree
with all elements stored in leaves
link leaves from left to right.
Search for l. Report all leaves to the right $\le h$
Called a 1-D range tree

$P(n) = O(n \log n)$
$Q(n) = O(\log n + A)$ $\quad A$ = size of answer
$S(n) = O(n)$

Range Search 2-D

Given $S \subset \mathbb{R}^2$ with $|S| = n$
Query: rectangle $R = [l_1, h_1] \times [l_2, h_2]$.
Report: $S \cap R$

Solution Naive
2 1-D Range Trees, one on x coord, one on y-coord
Find all pts in vertical range } take intersection
Find all pts in horizontal range }

would be bad.

$P(n) = O(n \log n)$
$Q(n) = O(|S_1| + |S_2| + \log n)$
$S(n) = O(n)$

Wasteful
For all $\binom{n+1}{2}$ possible vertical strips, with endpoints in S,
create a 1-D range tree for the points in that strip
For a query rectangle R, do a 1-D query on widest vertical strip contained by extending R vertically
$P(n) = O(n^3)$ $\quad Q(n) = O(\log n + A)$ $\quad S(n) = O(n^3)$

Better

segment tree idea: Decompose the plane into a small number of disjoint standard answer strips which correspond to subtrees of a balanced tree for S.

T balanced binary search tree for S

$\forall v \in T$, $S_v = \{p \in S \mid p$ lies in subtree rooted at $v\}$

$strip_v = [l_v, h_v] \times \mathbb{R}$ = vertical strip containing exactly S_v.

Each possible answer strip $\leq 2 \log_2 n$ standard answer strips.

Each v has a 1-D range tree for S_v on x_2 coord.

if $[l_v, h_v] \subset [l_1, h_1]$ then 1-D query$([l_2, h_2], T)$

else

if $[l_1, h_1] \cap [l_v, h_v] = \emptyset$, then return

2-D query $[l_1, h_1] \times [l_2, h_2]$ leftson(v)

" " rightson(v)

$P(n) = O(n \log n)$

$Q(n) = O(\log^2 n + A)$

$S(n) = O(n \log n)$

} later we will see a way to reduce query time to $O(\log n + A)$

Problem d-Dimensional Range Searching

Given $S \subset \mathbb{R}^d$, $|S| = n$

Query: rectangle $R = l_1 \times l_2 \times \ldots \times l_d$

Report: $R \cap S$

Generalize the solution to 2-D range searching.

Balanced binary tree T on the x_1 coordinate

$\forall v \in T$, associate a secondary $(d-1)$ dim range tree on $(x_2, \ldots x_d)$ coords.

$P(n) = O(n \log^{d-1} n)$

$Q(n) = O(\log^d n + A)$

$S(n) = O(n \log^{d-1} n)$

} can be improved to $O(\log^{d-1} n + A)$

Inverse Range Queries
Segment Trees
Interval Trees
Unbounded Rectangular Ranges and Treaps

Last time, given a set of pts in $\mathbb{R}^n$, constructed a data structure to report which points lay within a query range
This time, given a set of ranges, construct a data structure to help us report on which of them a query point lies.

1-D Problem
Given a finite set $S = \{ [l_i, h_i] \mid i = 1 \text{ to } n \}$ ← line segments on the real # line
Query: $x \in \mathbb{R}$
Report: set of segments containing x
$\{ s \in S \mid x \subset s \}$

Segment Tree
on ~~endp~~ intervals defined on endpoints
atomic intervals:
~~$E(S) = \bigcup \{l_i, h_i\}$~~ $2n+1$ intervals
Store all segments.
Each node contains list of all segments covering it but not its parent.

query(x, v)
if $x \notin I_v$ then return
report S_v
query$(x, \text{lchild}(v))$

$P(n) = O(n \log n)$
$S(n) = O(n \log n)$
$Q(n) = O(\log n + A)$ where A is the size of the answer.

2-D Case

Given set of rectangles, each defined as the cross product of an x and y interval

$$R_i = [lx_i, hx_i] \times [ly_i, hy_i]$$

1-D segment tree on x-intervals

Each node has a set S_v containing rectangles
Organize each S_v into a segment tree on the y intervals

Locate x-position of the query point in the primary segment tree
Locate y-position in the secondary segment tree of each vertex in which we pass.

reporting the rectangles found in the vertices of the secondary segment trees

Query Time: $O(\log^2 n + A)$
Space: $O(n \log^2 n)$
~~P(n)~~ $O(n \log^2 n)$

only sort rectangles once along each coordinate

each segment has $O(\log n)$ appearances in primary tree, each causes $O(\log n)$ appearances in a secondary segment tree

Interval Trees

1-D orthogonal range searching problem ← more space efficient data structure

Given set of segments S
Query pt $x \in \mathbb{R}$
Report segments containing x

Take endpoints of S. ← multisets (may have multiple copies of a point)
Take median m
Partition $S = S_l, S_r, S_{cut}$

segments containing m

At node v containing m
Store two lists: (sorted)
one of left endpoints of S_{cut}
one of right endpoints of S_{cut}

Answer query:
use median values stored at each vertex to locate the query point as with binary search tree

At each node, search list from outside report segments containing it.

$P(n) = O(n \log n)$
$Q(n) = O(\log n + A)$
$S(n) = O(n)$ ← improvement in space.

2-D range query problem: query ranges are unbounded rectangles open on the top with sides extending to infinity

Could ~~g~~ use range trees to answer such queries but

McCreight "treaps" hybrid of trees and heaps use linear space
Priority search trees.

Build a treap for a set $S \subseteq \mathbb{R}^2$ as follows.

0) If $S = \emptyset$ do nothing
1) Find $p \in S$ with max y coord
2) Remove p from S.
   Compute median m of x coords of remaining points
   Divides set into S_l, S_r (left, right of median)
3) Make a vertex v with fields for p and m, left, right child pointers to treaps for S_l, S_r respectively

A treap has height $O(\lg n)$
Every point is stored exactly once as the p field of some vertex.
A treap is a heap as far as the y-coordinate is concerned. With respect to the x-coordinate, it is "nearly" sorted in the sense that for a given path from root to leaf the contents of all subtrees to the left of the path are less than the leaf node and the contents of those to the right are greater.

A query of the form $[lx_i, hx_i] \times [y_i, \infty]$ is processed:

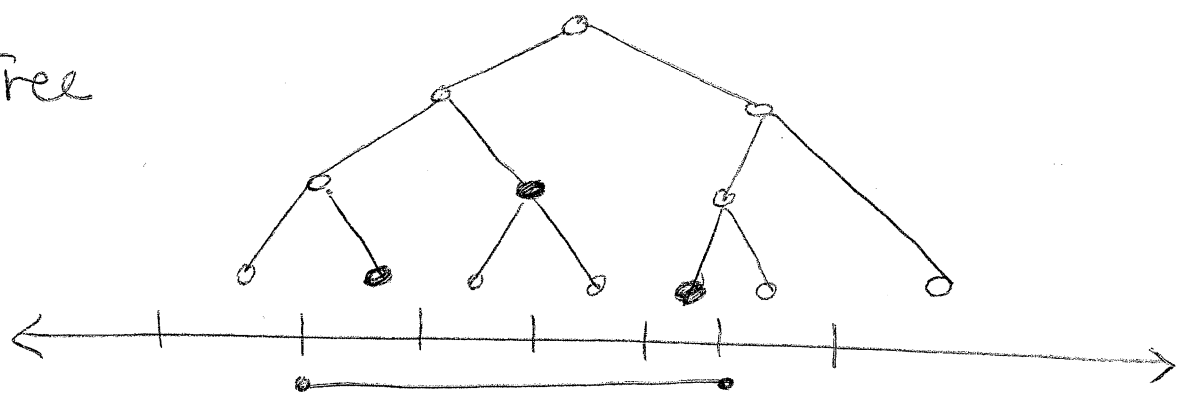
1) Locate the two leaves of the tree that would follow lx and precede hx, respectively, branching on the median values stored at the tree nodes.
For each node along the two traced root-leaf paths, report the point stored in the p field if it is contained in the query range. ~~Search the~~

2) Search all subtrees between the two paths, reporting all points stored in the φ fields with y-coordinate not less than y_i.

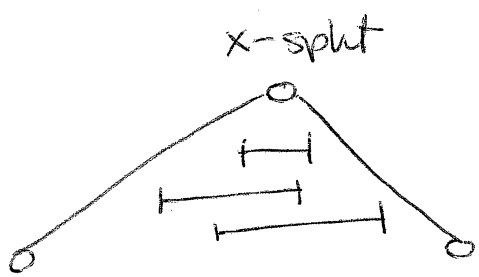
By the heap property, we can stop searching a path when we encounter a point with y-value below y. So the vertices actually examined will form a forest of binary trees with A internal nodes, where A = size of answer, hence ~~some~~ containing $O(A)$ total nodes.
The entire query process takes time

$$Q(n) = O(\log n + k)$$
$$P(n) = O(n \log n)$$
$$S(n) = O(n).$$

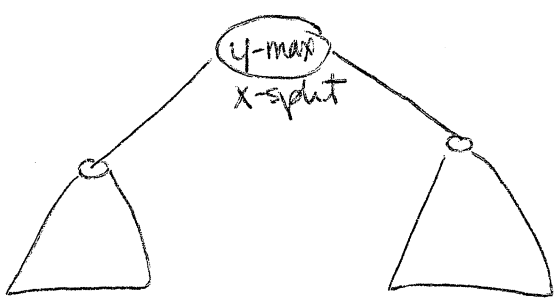
Segment Tree



Interval Tree



Treap



Problem

S = set of intervals on $\mathbb{R}$

Need data structure to answer:

a) given query interval I_0, return $\{I \in S \mid I \cap I_0 \neq \emptyset\}$

b) $\{I \in S \mid I_0 \subset I\}$

c) $\{I \in S \mid I \subset I_0\}$

Solution $I \in S$, $I = [a, b]$ define

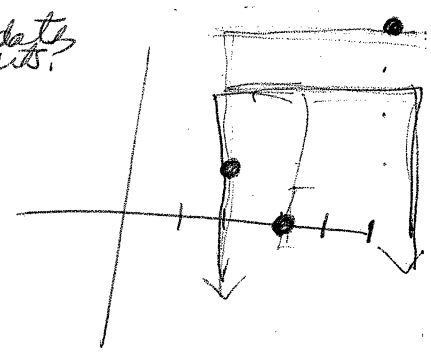
$\bar{S} = \{(\underset{\text{end}}{b}, \underset{\text{start}}{a}) \mid [a, b] \in S\}$ pts in $\mathbb{R}^2$

$I_0 = [x_0, y_0]$

a) query $\bar{S}$ with open rectangle

$(x_0, \infty; y_0)$ ← does it end after candidate starts?

did it start before candidate ends?



i.e all points $x_0 \leq b \leq \infty$ and $a \leq y_0$

b) open rectangle $(y_0, \infty; x_0)$

does it end before candidate ends

and

does it start

$y_0 \leq b \leq \infty$ and $a \leq x_0$

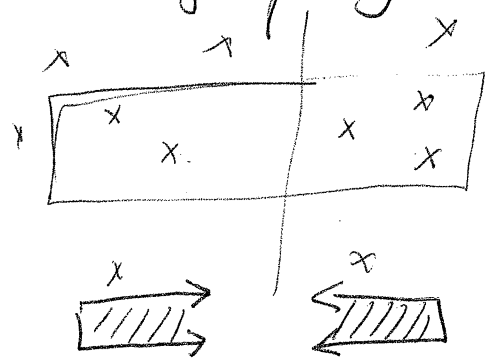
c) $S^* = \{(a, b) \mid [a, b] \in S\}$

query with open rectangle $(x_0, y_0; y_0)$

i.e all points ∋ $x_0 \leq a \leq y_0$ and $b \leq y_0$.

2-D Range search.

If separating line cuts through query rectangle:



S = set of pts in $\mathbb{R}^2$

Build a balanced binary search tree T for S wrt x-coord.

T is primary data structure.

Let $v \in T$. Define

key(v)

$S_l = \{p \in S \mid p$ stored in left subtree of $v\}$

$S_r = \{p \in S \mid p$ stored in right subtree of $v\}$

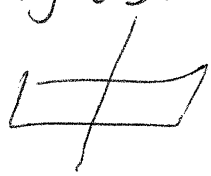
Build secondary data structure.

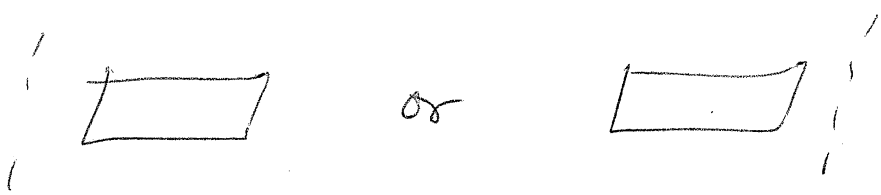
T_l = priority search tree for S_l to answer 

T_r = — — S_r —

Each point occurs once at each level of the tree T

$S(N) = O(N \log N)$.

Case i)  do query in secondary structure



or

One subtree is irrelevant. Search in other subtree.

$Q(N) = O(\log N + k)$ where k is answer size