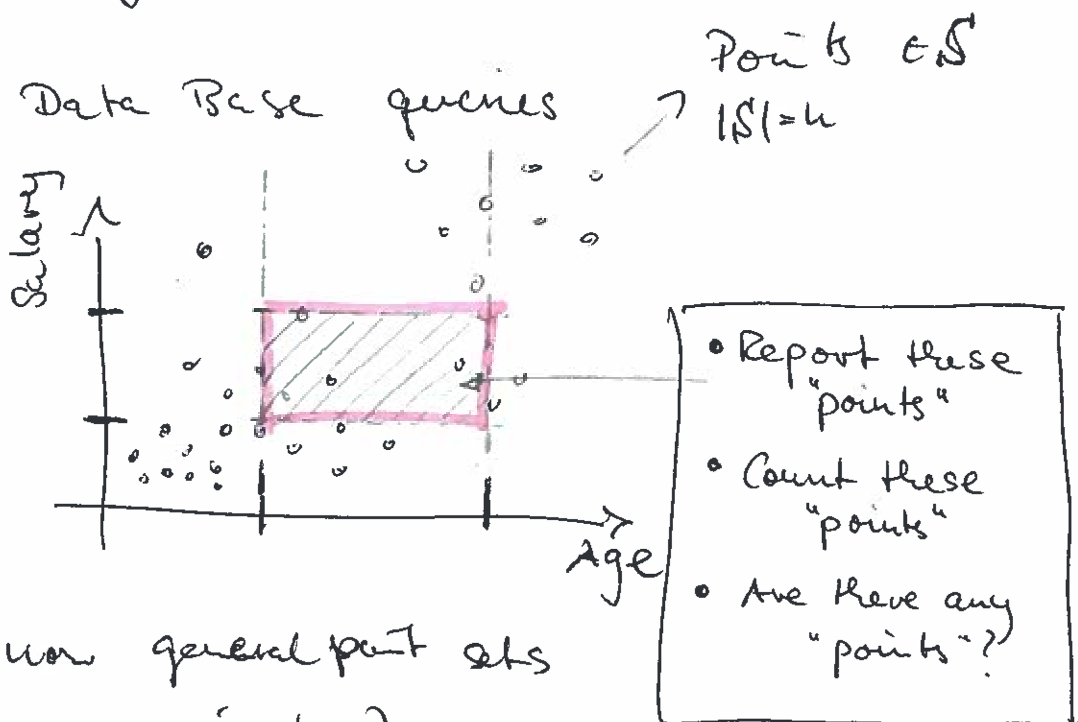
6.851 ADVANCED DATA STRUCTURES

(ANDRÉ SCHULZ NOTES)

3rd Lecture

Orthogonal Range Queries

Motivation: Data Base queries



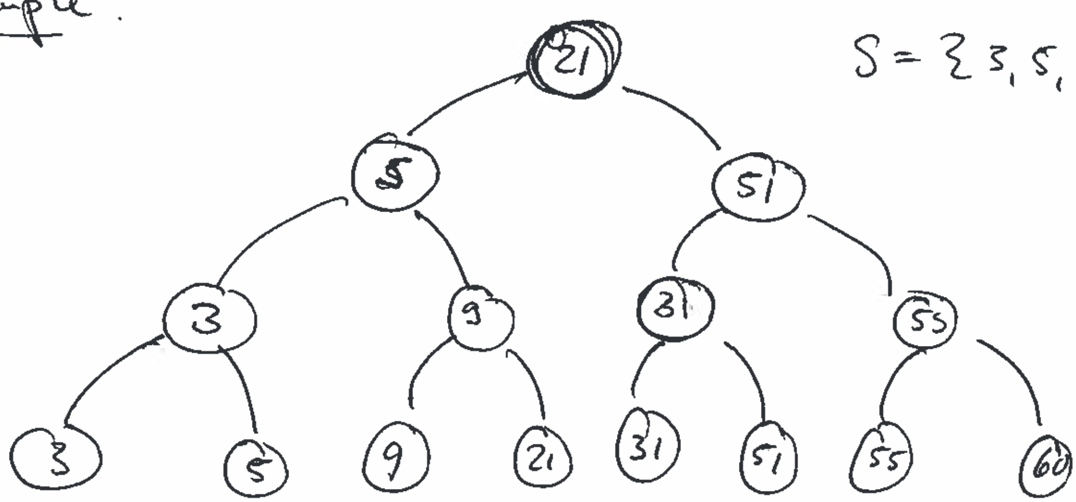
We assume for now general point sets (distinct x & y-coordinates)

Easy start: 1D Range Queries with Range Trees

Idea: Store the coordinates $(x_1, x_2, x_3 \ldots x_n)$ in a balanced binary tree

Example:

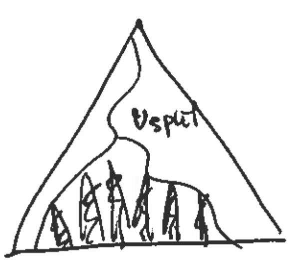
$S = \{3, 5, 9, 21, 31, 51, 55, 6$



- Every node v of the tree encodes a sub set of S ($\{x_i, x_{i+1} \dots x_{j-1}\ x_j\}$) which we name $R(v)$;
  ↑ ordered from $x_i \dots x_j$
- We store in every node the maximal key contained in the subtree of its <u>left</u> child
- Such a tree can be constructed in $O(n \log n)$ time (first sort) and needs $O(n)$ space

<u>How to perform a query?</u>

- Query : Find all elements s_i of S with
  $$a \leq s_i \leq b$$
- Algorithm: (1) Find the "split node" in the tree, that is the node v_{split} with
  $$R(v_{split}) \supseteq [a,b]$$
  $$R(\underset{\uparrow \text{ Left child}}{LC}(v_{split})) \not\supseteq [a,b]$$

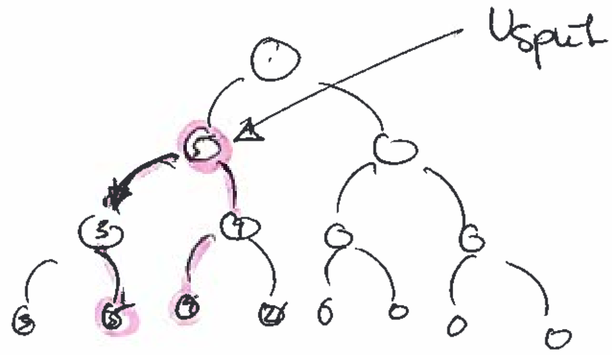


(2) If v_{split} is a leaf, check if $R(v_{split}) \in [a,b]$

(3a) Follow the path to a and report the right subtrees

(3b) Follow the path to b and Report the left subtrees

<u>Example</u> $a = 5$, $b = 17$



Always check for the leaves!

<u>"Running" time</u>: It only makes sense to study the ~~Running~~ query time "output-sensitive". We denote the length of the output by k!

<u>Facts</u>:
- a subtree of size n' can be reported in $O(n')$
- ~~the node v_{split} can be found in~~
- other than reporting subtrees, we search for a and b

$\Rightarrow$ Reporting takes $O(k)$ time, Searching for a, b $O(\log n)$

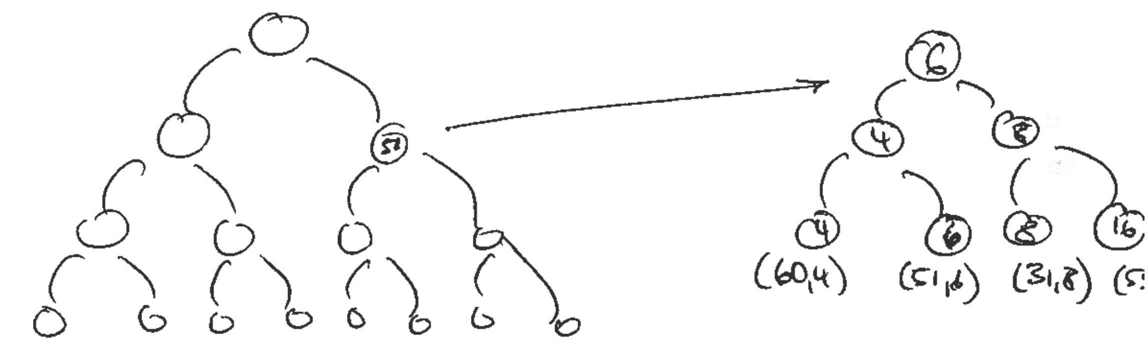
$\Rightarrow$ Query time $O(k + \log n)$

Now 2D!

Idea: First do a 1D search (say for the x-coordinates) and then, while reporting, filter for the right y-coordinates

By Example $S = \{ (3,1), (5,18), (9,7), (21,11), (31,8), (51,6), (55,16), (60,4) \}$

Multi level DS



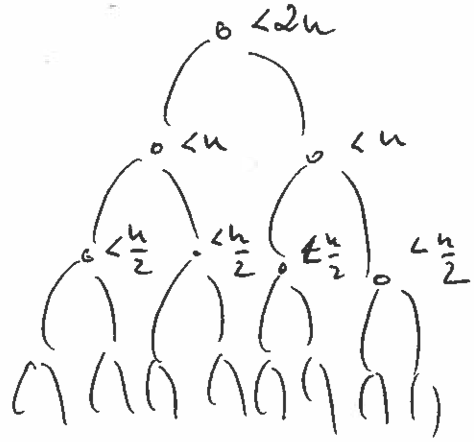
First level tree

one of the 2nd level tree

- For every node v in the First level tree (the one for the x-coordinates) we store a ~~2nd~~ balanced binary tree with the y-coordinates of $R(v)$
- Every ~~node~~ leaf in a 2nd level tree stores x and y coordinates

Storage :

1st level : Storage / node



Σ / level

$2n$

$2n$

$2n$

$\vdots$

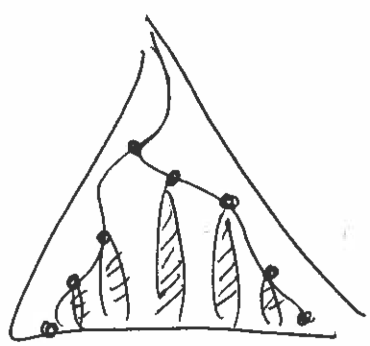
$\sum_{\text{level}} 2n \log n$

Preprocessing time :

When the y-coordinates are pre-sorted, the 2nd level trees can be build in linear time

→ same analysis as for Storage

Query time



For every node on $v_{split} \to a_x$ & $v_{split} \to b_x$ we perform a 1D range query

$$\Rightarrow \text{Query time} = \sum_{\substack{v\,(v \in v_{split}\to a_x \\ \vee\, v_{split}\to b_x)}} O(\log n_v + k_v)$$

$$= O(\log^2 n + k)$$ since $\sum_v k_v = k$ and the sum runs over $O(\log n)$ summands

Higher dimensions $\mathbb{R}^d$

- Idea : Apply the idea from 1D → 2D again (and again)

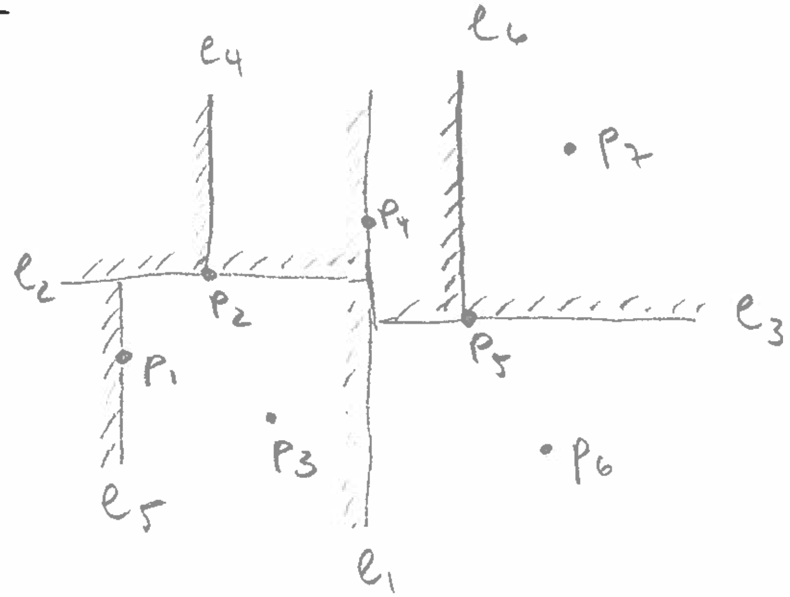
- Data structure with:

  Storage : $O(n \log^{d-1} n)$
  preproc : $O(n \log^{d-1} n)$
  query time : $O(\log^d n + k)$

An Alternative : kd-trees [Bentley '75]

- Use only 1 Tree and subdivide the x and y coordinates alternating

Example

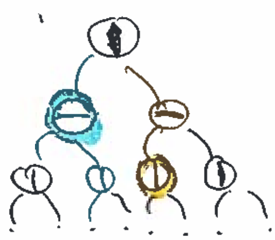


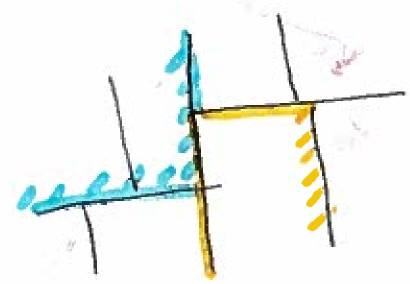
Construction : Storage : $O(n)$

preprocessing : $O(n \log n)$

(presort coordinates simplifies the median search)

Queries Every node v in the tree covers a ~~range~~ region (v)





"Scan" the tree for points

↳ I can stop at v if (i) Region (v) contained in the query range or (ii) Region (v) and query range are disjoint

↳ otherwise recurse

Query time Step (i) can be done in linear time in k

→ How often do I have to recurse?

Answer : # of regions containing a part of the query region

Theorem : ~~Every~~ A query region contains at most $O(\sqrt{n})$ regions partially

proof skiped (not too hard)

<u>consequence</u> Query time : $O(\sqrt{n} + k)$

Back to Range trees : FRACTIONAL CASCADING (Chazell + Guibas 1986)

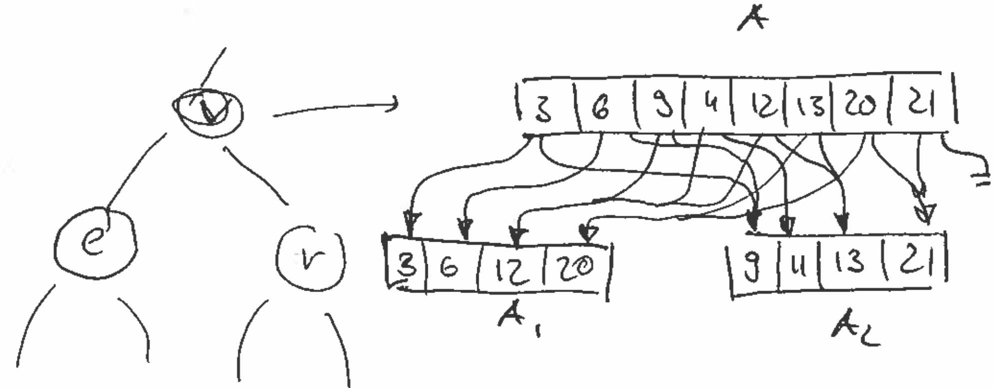
Speed up in the query time from $O(\log^2 n + k)$ to $O(\log n + k)$

Basic Idea : • use the range tree as 1st level DS

• 2nd level DS's are stored as arrays

• we "wire" the different arrays to reuse information

Example

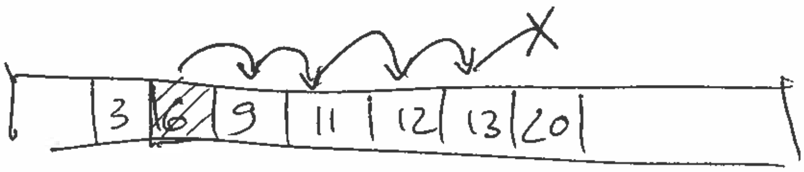


<u>Rule</u> : Pointer $\underset{A}{X} \rightarrow \underset{A_1}{Y}$, s.t. $y \geq x \wedge \forall y' \in A_1 : y' < y : y' < x$

[The smallest entry in $A_{1/2}$ greater equal X]

How does this help?

Assume we know the "place" of y_{min} in the Array



$y_{min} = 5$ $\Rightarrow$ Search for entry "6" and report all cells from this until y_{max} is reached

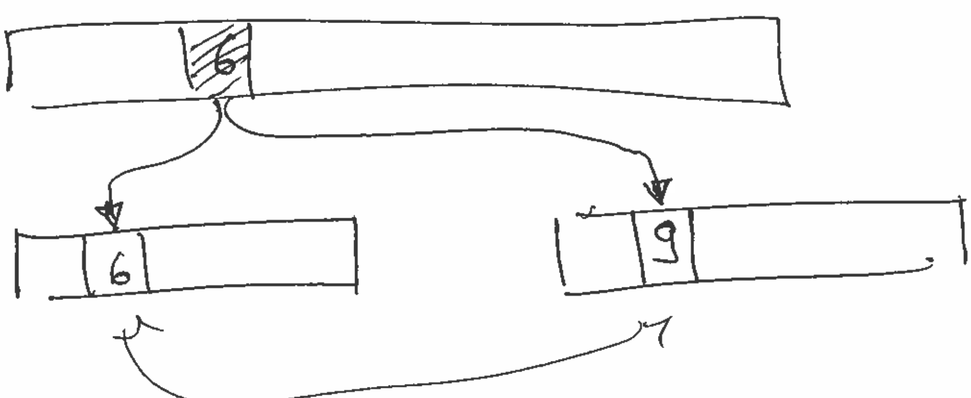
$y_{max} = 15$

Expensive is the search for "y_{min}" the rest is $O(k)$

BUT WE CAN REUSE THE RESULT OF THE SEARCH IN LATER SEARCHES

$\Downarrow$

Thats why we have the pointers!



New Startpositions for reporting

Analysis

(1) Search for v_{split} $O(\log n)$

(2a) Follow to the left and "report" all right subtrees

(2b) Follow to the right and "report" all left subtrees ?

The "report" has to scan for the right y-coordinates, but with help of the fractional cascading pointers we have to "search for y_{min}" only once

Takes $O(\log n)$ and the reporting only $O(k)$

$\Rightarrow$ $O(\log n + k)$ query time
$O(n \log n)$ space
$O(n \log n)$ preprocessing

Chazelle '86 improved this idea to $O(n \log n / \log\log n)$ Storage

which is optimal

See also Chazelle for higher dimensions

General Point Sets

up to know we assumed the all coordinates are distinct!

How to handle general point sets?

Use composite numbers

$(x, y) \in \mathbb{R}^2 \longrightarrow (x|y) \in CoN \leftarrow$ composite number space

CoN has a natural order (lexicographic)

$(x_1|y_1) < (x_2|y_2) \iff x_1 < x_2 \text{ or } (x_1 = x_2 \wedge y_1 < y_2)$

Lemma let $R = \cancel{[(x_1, x_2)}\ [x_1, x_2] \times [y_1, y_2]$ a query region than

$$p \in R \iff \hat{p} \in \hat{R}$$

$= ((p_x|p_y), (p_y|p_x))$

with $\hat{p}$ ~~composite number of p and~~

$$\hat{R} = [(x_1|-\infty), (x_2|+\infty)] \times [(y_1|-\infty), (y_2|+\infty)]$$

Proof:

$x_1 \le p_x \le x_2 \iff (x_1|-\infty) \le (p_x|p_y) \le (x_2|+\infty)$

$y_1 \le p_y \le y_2 \iff (y_1|-\infty) \le (p_y|p_x) \le (y_2|+\infty)$

$\square$