Geometric data structures ← also multidimensional data

taxonomy of geometric DS by types of:

1) objects: points, lines, polygons, ... (in $\mathbb{R}^d$) ← supported dimension

2) queries: single object, partial match, range (intervals), ...

Find (x,y,z) (in $\mathbb{R}^3$) Find $(x,*,*)$ Range Query $([a_1,b_1],(-\infty,b_2),[a_3,a_3])$

$a_1 \le x \le b_1$, $y \le b_2$, $z = a_3$

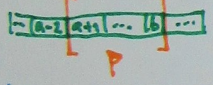
3) result: enumerate / count / aggregation
"list" ↑ sum, max, ...

complexity w.r.t. n ... # objects, d ... dimension, p ... objects listed (for enumeration queries)

4) static / dynamic (supports insert, delete)

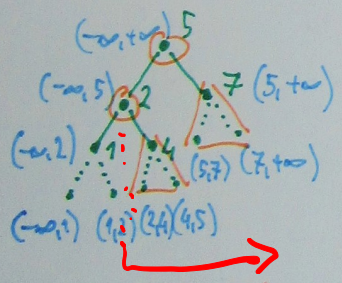
1D search trees and range queries (interval)

- sorted array: Range Query $([a,b])$ ↔ binary search ... $O(\log n + p)$



Build ... sorting $O(n \log n)$; static DS optimal space $O(n)$

- binary search trees: same complexity + dynamic + aggregation queries (balanced)



node def.

$int(v) = \{x \in \mathbb{R} \mid \text{Find}(x) \text{ visits } v\}$

$\Rightarrow int(root) = (-\infty, +\infty)$

left/right child: $int(l(v)) = int(v) \cap (-\infty, key(v))$, $int(r(v)) = int(v) \cap (key(v), +\infty)$ (key in v)

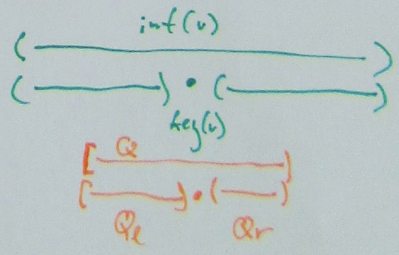
$\Rightarrow$ open interval (subtree of v in $int(v)$)

node range (interval)

Range Query (v, Q):

1. if v external (leaf), return. (no output)
2. if $int(v) \subseteq Q$, report subtree of v, return.
3. if $key(v) \in Q$, report $key(v)$.
4. $Q_l := Q \cap int(l(v))$, $Q_r := Q \cap int(r(v))$
5. if $Q_l \neq \emptyset$, Range Query $(l(v), Q_l)$
6. if $Q_r \neq \emptyset$, Range Query $(r(v), Q_r)$

assert: $Q \subseteq int(v)$

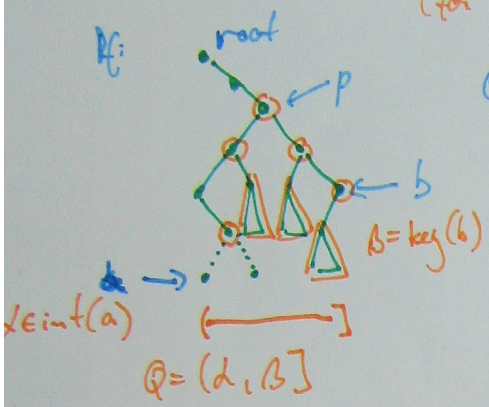


ex: Range $(root, [2,+\infty))$ depends on report type

report: 5, 2, △, △ → report type

Lem.: Range Query visits $O(\log n)$ nodes and subtrees if tree balanced. [depth $O(\log n)$]

(obs.) ↑ output size not counted (for enumerate)



$Q =: (\alpha, \beta)$ range, a, b nodes where $Find(\alpha)$, $Find(\beta)$ ends ← or $[,], -\infty, +\infty$

$p := lcp(a,b)$ ← lowest common predecessor

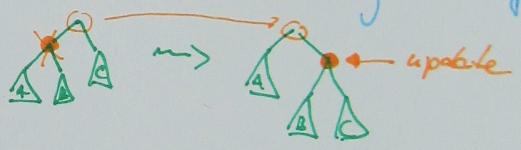
path to a, b ... $O(\log n)$ nodes ⇐ balanced tree

visited subtrees = right subtrees on path from p to a
+ left subtrees on the path from p to b.

⇒ $O(\log n)$ visited nodes and subtrees □

Static BSTs: perfectly balanced tree (e.g.) ⇒ Build in $O(n \log n)$ time
counting + aggregate ... precomputed for subtrees ⇒ $O(\log n + P)$
space $O(n)$ time for Range Queries (enum.)

dynamic BSTs: balancing (eg. AVL, RB, ...) + update precomputed count/aggregate answers when rotating an edge ⇒ Insert, Delete $O(\log n)$

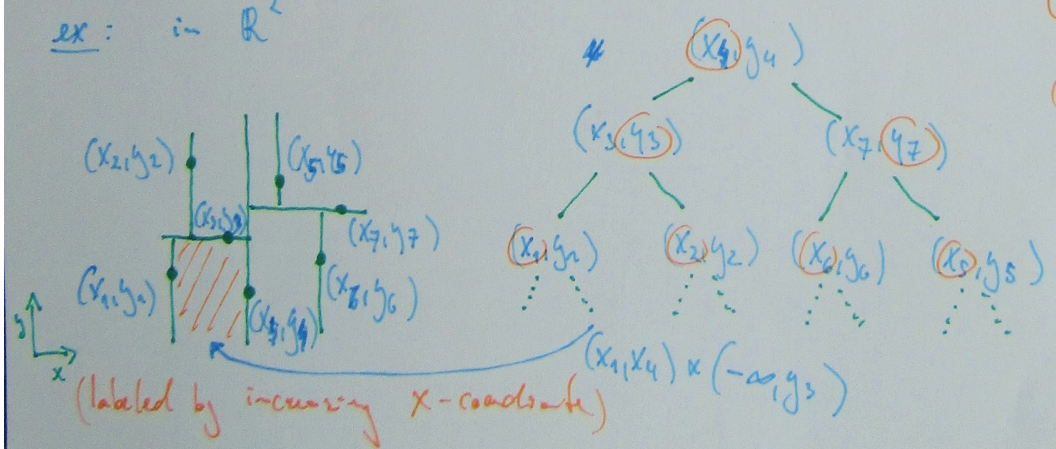


k-d search trees and range queries

idea: keys = points in $\mathbb{R}^d$, in level i split by the coordinate $i \pmod d$

assumption: no two points share the same value in a one coordinate
(no two points on the same x line or y line)
(simple case)

ex: in $\mathbb{R}^2$



external nodes = open regions (boxes)

Build (for static version): (dynamic version in DS2)

d sorted arrays of points by each coordinate

ex: $x_1, x_2, x_3, x_4, x_5, x_6, x_7$

$y_4, y_1, y_6, y_3, y_7, y_5, y_2$

recursively: root of subtree = median by the current coordinate (by levels) split the arrays + continue with the next coordinate

x_1, x_2, x_3 | x_5, x_6, x_7

y_1, y_3, y_2 | y_6, y_7, y_5

⇒ median of n items in $O(n)$ time ⇒ $O(n)$ by level, $\lceil \log n \rceil$ levels

(median-of-medians algorithm)

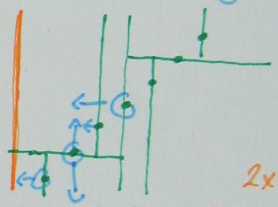
⇒ $O(n \log n)$ time, $O(n)$ space for tree

Range Query: same algorithm as 1D, alternate coordinates, $\text{int}(v)$ region open (product of open intervals)

Lem: Range Query in 2-d tree has $\Omega(\sqrt{n})$ worst-case time complexity.

Pf: $(x_i, y_i) = (i, i)$ for $1 \le i \le n$, $n = 2^t - 1$ (height)

Range Query $(\{0\} \times \mathbb{R})$ (answer is $\emptyset$)



2x every second level

x-coordinate ⇒ go to the left

y-coordinate ⇒ go both left and right

⇒ $2^{t/2} \approx \sqrt{n}$ visited nodes □

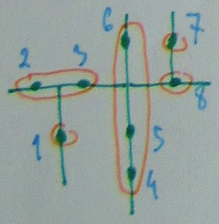
Note:

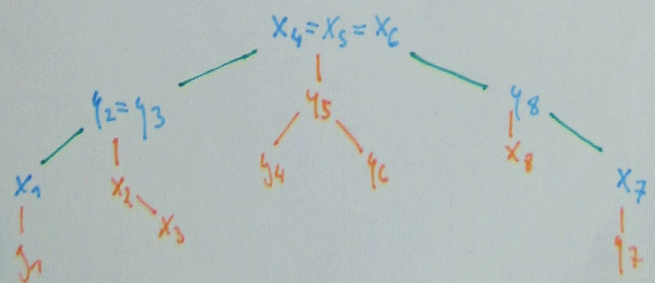
- Actually, $O(\sqrt{n})$ is also an upper bound. (proof omitted)
- For general k-d search trees, the query complexity is $\Theta(n^{\frac{k-1}{k}})$.

(Same argument with points $(i, i, \dots, i)$ and range $\{0\} \times \mathbb{R}^{k-1}$.)

Q: How to allow same values in a same coordinate in more points?

A: attach $(k-1)$-d subtree to each node to store all point with the same coordinate





2-d range trees

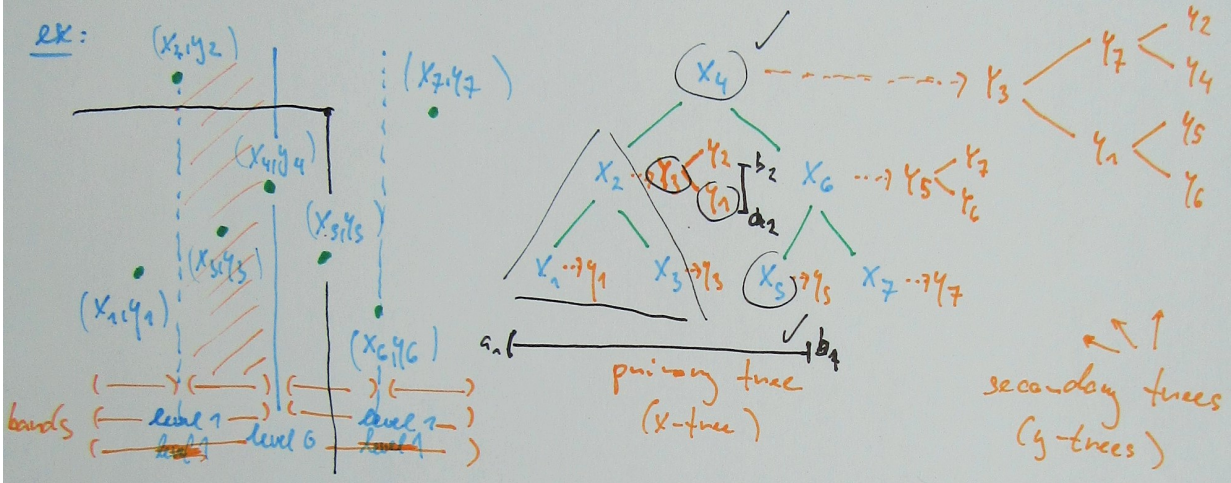
aim: poly-logarithmic range queries, not space optimal

$O(\log^k n)$ k dimensions $O(n \log n)$ vs $O(n)$

idea: a primary 1D search tree on x-coordinate + in each node v)
for all points
a secondary 1D search tree on y-coordinate for points in $int(v) \times \mathbb{R}$
"band"

assumption: no two points have same value in the x coordinate (simple first)

ex:



x-tree balanced $\Rightarrow$ every point in $O(\log n)$ y-trees $\Rightarrow$ $O(n \log n)$ space

RangeQuery $([a_1, b_1] \times [a_2, b_2])$:

ex: RangeQuery $((-\infty, x_5) \times (-\infty, y_7))$

* x-tree. RangeQuery $([a_1, b_1]) \Rightarrow$ $O(\log n)$ points + bands (corresponding to subtrees of x-tree)

$\Rightarrow$ $O(\log n)$ time in x-tree
$O(\log n)$ time in each y-tree

$y_i \in [a_2, b_2]$? report if yes

report points in the corresponding y-tree RangeQuery $([a_2, b_2])$

$\Downarrow$

$O(\log^2 n + p)$ time

p: if p points enumerated

Build: 2 sorted arrays by x/y coordinates
(as before) find median in x-array + build y-tree for it
split arrays + proceed recursively
$\Big\}$ $O(n \log n)$ time again

Q: How to allow same value in x-coordinate in multiple points?

A: another y-tree for points with the same x-coordinate

(2 y-trees: for the entire band $int(v) \times \mathbb{R}$ + for $key(v) \times \mathbb{R}$)

k-dimensional range trees ← straightforward generalization

primary x-tree (1-d search tree) + secondary trees for each band $int(v) \times \mathbb{R}^{k-1}$ ((k-1)-dim. range tree)

(as above) (k-dimensional)

Space: every point in $O(\log n)$ secondary range trees (multiply by this factor k-1 times)

$\Rightarrow O(n \log^{k-1} n)$ space in total

↑ constant dependent on k (hidden)

RangeQuery $([a_1,b_1] \times \dots \times [a_k,b_k])$:

x-tree.RangeQuery $([a_1,b_1]) \rightarrow O(\log n)$ points + bands

(as for 2D)

check if in Q, report if yes

secondary (k-1)-trees: RangeQuery $([a_2,b_2] \times \dots \times [a_k,b_k])$

$\Rightarrow O(\log^k n + p)$ time

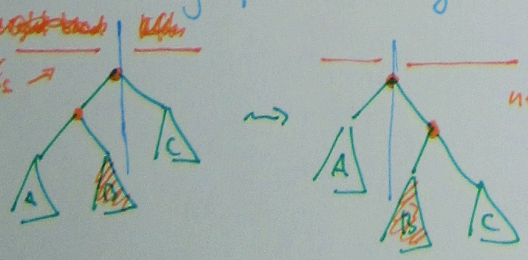
Build: same as for 2D, recursively on dimension, k sorted arrays

$\Rightarrow O(n \log^{k-1} n)$ time

Dynamic range trees ← Insert, Delete without Build

Q: Can we use balanced trees based on rotation (AVL, RB, ...)?

A: No, too many points change band (cannot be updated efficiently): $\Omega(n)$ in worst case



Dynamic version by lazily balanced trees (for k=2)

← lecture 2, $O(\log n)$ amortized bds for Insert, Delete

Insert (x, y): 1x Insert(x) to x-tree + $O(\log n)$ Insert(y) to y-trees

Rebuild of y-tree: $O(n)$ time worst-case, amortized $O(\log n)$

Rebuild of x-tree: $O(n \log n)$ time, amortized $O(\log^2 n)$

↑ each point in $O(\log n)$ y-trees

Similarly for general k: Delete:

Build ... $O(n \log^{k-1} n)$

RangeQuery ... $O(\log^k n + p)$

Insert ... $O(\log^k n)$ amortized bds

Delete ... $O(\log^k n)$ amortized bds

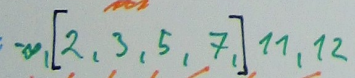
Fractional cascading

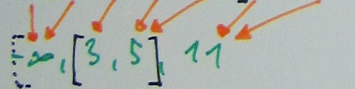
goal: for k=2, $O(\log n + p)$ query complexity, static version with some precomputation ← instead of $\log^2 n$

idea: use search in parent y-trees to speed up searches in child y-trees by precomputed links

↑ to $O(n)$

for simplicity (static version): y-tree ~ sorted array

→ parent: 

→ child: 

link: pointer to a "predecessor" of a (add $-\infty$ so always defined) ← largest b s.t. $b \le a$

ex: RangeQuery([1,9]) ↷

RangeQuery child: use links from parent + adjust indices by ±1

⇒ in general k-dim range trees, $O(\log^{k-1} n + p)$ time for queries

↑ instead $\log^k n$