Geometric DS ← also for multidimensional data

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

queries: exact match, partial match, range (interval), ...
Find(3,*,*), RangeQuery([3,7), (−∞,+∞), 1)

result: enumerate / count / aggregation ← (sum, max, ...)
"list"

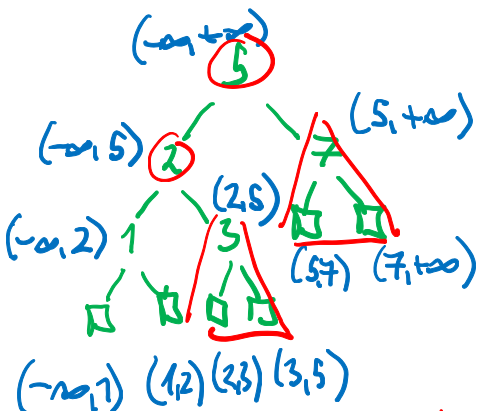
static / dynamic

(yellow = our focus)

1D search tree

- sorted array: RangeQuery([a,b]) by binary search ... $O(\log n)$ for count
  + p for list, p = #objects
  Build by sorting ... $O(n \log n)$
  static DS, optimal space $O(n)$, no aggregation

- balanced BSTs: same complexity + dynamic + aggregation queries



node

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

$\Rightarrow \text{int}(\text{root}) = \mathbb{R}$

left (right) child

$\text{int}(\ell(v)) = \text{int}(v) \cap (-\infty, \text{key}(v))$

$\text{int}(r(v)) = \text{int}(v) \cap (\text{key}(v), +\infty)$

⇒ open intervals, subtree of v in $\text{int}(v)$

node — any type of interval — assert: $Q \subseteq \text{int}(v)$

RangeQuery(v, Q)

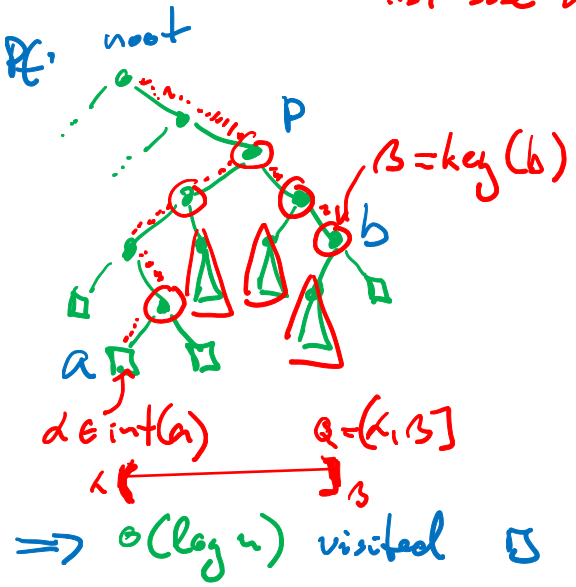
1. if v external, return. (no output) ← leaf
2. if $\text{int}(v) \subseteq Q$, report the subtree of v, return. ← good for aggregation
3. if $\text{key}(v) \in Q$, report key(v).
4. $Q_\ell := Q \cap \text{int}(\ell(v))$, $Q_r := Q \cap \text{int}(r(v))$ — possible subintervals
5. if $Q_\ell \neq \emptyset$, RangeQuery($\ell(v)$, Q_ℓ).
6. if $Q_r \neq \emptyset$, RangeQuery($r(v)$, Q_r).

} continue in children if any chance

ex: RangeQuery(root, [2,+∞))
report: 5, 2, △, ▽ (2 nodes, 2 subtrees)

Lem: Range Query visits $\mathcal{O}(\log n)$ nodes and subtrees if tree is balanced.
↑ list size not included (for enumerate)
↑ height $\mathcal{O}(\log n)$

Pf: root

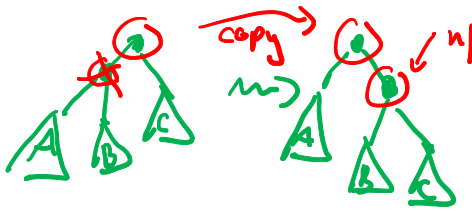


- $Q = (\alpha, \beta)$ range, a, b nodes where Find(α), Find(β) stops, $p := lcp(a,b)$
  ↑ lowest common predecessor
- visited nodes: on the paths to a, b (some are repeated)
- visited subtrees: right (left) subtrees on the path from p to a (resp. b)

$\Rightarrow \mathcal{O}(\log n)$ visited ☐

static BSTs: perfectly balanced $\Rightarrow$ Build in $\mathcal{O}(n \log n)$ time
aggregation ... precomputed $\mathcal{O}(n)$ time (by levels)
... queries in $\mathcal{O}(\log n)$ time
space $\mathcal{O}(n)$

dynamic BSTs: balancing (eg. AVL, RB, ...) + update precomputed aggregation answers when rotating an edge



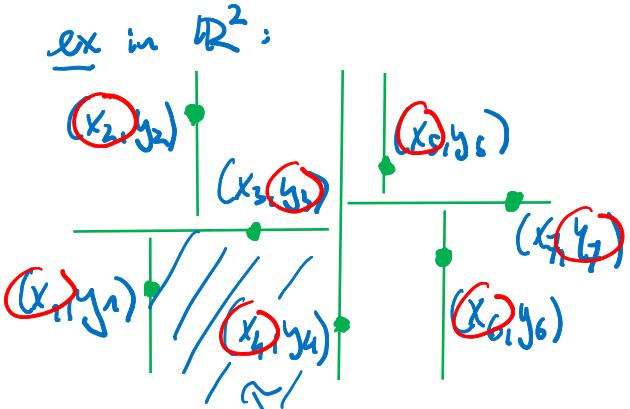
$\Rightarrow$ Insert / Delete in $\mathcal{O}(\log n)$ time

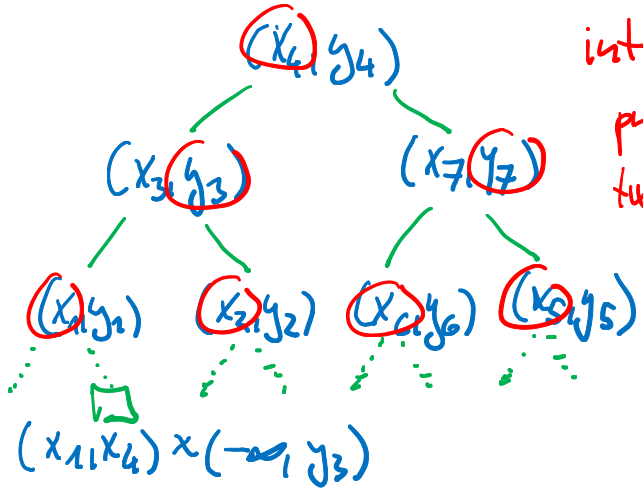
k-D search trees

idea: keys = points in $\mathbb{R}^k$, on level i split by i-th coordinate (mod k)

assume: no shared coordinates

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





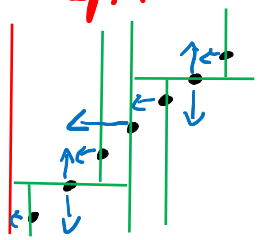
int(v) -- product of (k in general) two open intervals (open region)

Build (static DS): • on level i select **median** value of i-th coord. ($\mod k$)
for a new root, split, recurse
• median of n items in $O(n)$ time
$\Rightarrow$ $O(n)$ per level, $\lceil \log n \rceil$ levels $\Rightarrow$ $O(n \log n)$ time, space $O(n)$

Range Query: same alg. as in 1D, alternate coordinates,
$int(v)$ ... open region (product of open intervals),
recurse until all coordinates fall into target intervals

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

Pf: points $= \{(i,i) \mid 1 \le i \le n\}$, $n = 2^t - 1$ ← diagonal, ← depth

Range Query $(\{0\} \times \mathbb{R})$ → 

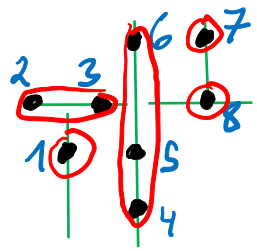
x-coordinate $\Rightarrow$ go to the left

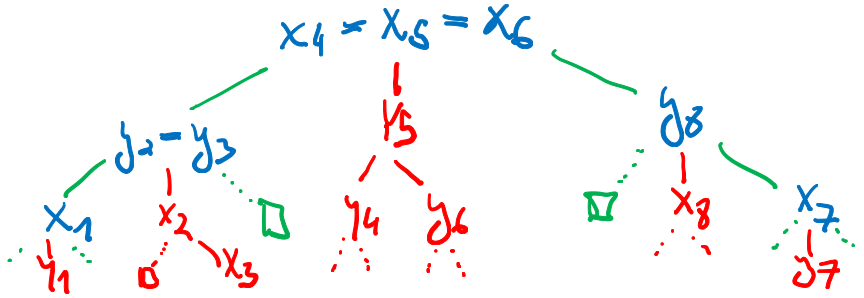
y-coordinate $\Rightarrow$ go **both** left and right $\Rightarrow$ $2^{t/2} \approx \sqrt{n}$ visited nodes (every second level) □

Note: Actually, $O(\sqrt{n})$ is also an upper bound. (without proof)
For general k-D trees, the query complexity is $\Theta(n^{\frac{k-1}{k}})$
(gen. lower bound for points $(i,\dots,i)$, range $\{0\} \times \mathbb{R}^{k-1}$)

Q: How to allow shared values in the same coord. of multiple points?
A: Attach $(k-1)$-D subtree to each node for all points with this shared coordinate.

ex:





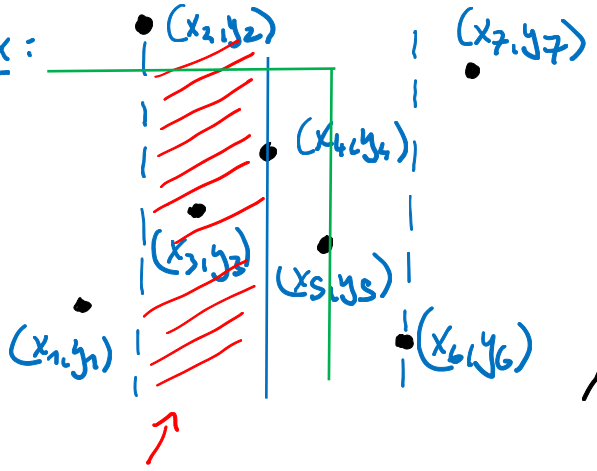
2D range trees ($k=2$ first)

goal: poly-logarithmic range queries, not space optimal
($O(\log^k n)$, k dim.) $O(n \log^{k-1} n)$ vs $O(n)$

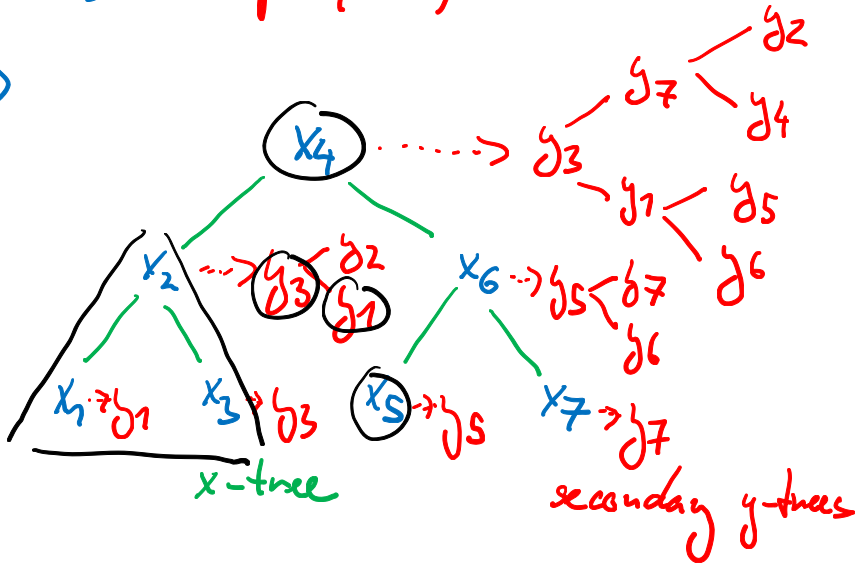
idea: primary 1D search tree for x-coordinates of all points (x-tree)
+ secondary 1D search tree in each node v for y-coordinates of all points in $\text{int}(v) \times \mathbb{R}$ ("band")

assume: no shared coordinates (simple first)

ex:



$(x_2, x_4) \times \mathbb{R}$ band



y_i is only in secondary subtrees on the path to x_i
$\Rightarrow O(n \log n)$ space if x-tree balanced

Range Query $([a_1, b_1] \times [a_2, b_2])$: (corresponding to subtrees)

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

- points (x_i, y_i): if $y_i \in [a_2, b_2]$, report.
- bands: use corresponding y-tree.RangeQuery$([a_2, b_2])$

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

report: points x_4, x_5 + band $(-\infty, x_4) \Rightarrow$ $(x_4, y_4), (x_5, y_5)$, $(x_3, y_3), (x_1, y_1)$

$O(\log n)$ time in x-tree, $O(\log n)$ time in each y-tree
$\Rightarrow O(\log^2 n)$ time (aggregation, if list then $+p$)

Build: 2 sorted arrays of x / y coordinates ... $O(n \log n)$
find median in x-array ... $O(1)$, build y-tree for it ... $O(n)$ per level
split arrays ... $O(n)$ per level, recurse

$\lceil \log n \rceil$ levels $\Rightarrow$ $O(n \log n)$ time (again)

Q: How to allow shared values in the same coordinate for multiple points?
A: Attach y-tree for points with the same x-coordinate
(2 y-trees: for bound $int(v) \times \mathbb{R}$ + for $leg(v) \times \mathbb{R}$)

k-D range trees (straightforward generalization)

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

space: every point in $O(\log n)$ secondary range trees $\Rightarrow$
$O(n \log^{k-1} n)$ in total (multiply by $\log n$ $(k-1)$-trees)

Range Query $([a_1, b_1] \times \dots \times [a_k, b_k])$:
x-tree.RangeQuery$([a_1, b_1]) \Rightarrow O(\log n)$ points + bands
check points + secondary y-trees.RangeQuery$([a_2, b_2] \times \dots \times [a_k, b_k])$
$\Rightarrow O(\log^k n)$ time (+p if enumeration)

Build: same as 2D, k sorted arrays, recursively on dimension
$\Rightarrow O(n \log^{k-1} n)$ time (constant is dependent on k)

Notes
- dynamic version – possible but not with balanced BSTs (exercises) with balancing based on edge rotations
  $\Rightarrow$ use loosely balanced trees
- speed up in queries by factor $\log n$ (fractional cascading)