Recall: Range Search

Given a set of $n$ pts $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$, and class of shapes (range space), preprocess $P$ to answer range queries:

Given shape $Q$, count/report the pts in $P \cap Q$.

$P$ + shape = rectangles

**Data structure**

Ans: 10 pts of $P$ in $Q$

Last lecture: $kd$-trees

- $O(n)$ space / $O(n \log n)$ build time
- $O(\sqrt{n})$ query time (in $\mathbb{R}^2$)
- $O(n^{1-\frac{1}{d}})$ in $\mathbb{R}^d$

Today: Orthogonal Range Trees + Layered Data Structures

$O(\log^d n)$ query time

$O(n \log^{d-1} n)$ space
1-Dimensional Range Tree: (Review)

- Given set of scalars: \( P = \{ p_1, ..., p_n \} \subseteq \mathbb{R} \)
- Store as leaves in balanced search tree \( \rightarrow \mathcal{O}(n) \) space \( \rightarrow \mathcal{O}(n \log n) \) construct.

- Each node stores number of leaves: \( u.size \)

- Given query interval \( Q = [Q_{lo}, Q_{hi}] \)
- Identify \( \mathcal{O}(\log n) \) maximal subtrees that cover \( Q \)
- Add up sizes for all these nodes

Query answer = 1 + 2 + 4 + 2 + 1 = 10
Range counting algorithm:

Node u:

- **u.point**: point \(p\) (if u is leaf)
- **u.x**: split value (if u internal)
- **u.size**: # leaves (if u internal)
- **u.left**, **u.right**: children

range1Dx(Node u, Range Q, Interval C = \([x_0, x_1]\))

if (u is leaf)
    return \(\{1\text{ if } u\text{.point} \in Q\}

else if (\(C \cap Q = \emptyset\)) (no overlap)
    return 0

else if (\(C \subseteq Q\)) (contained)
    return u.size

else
    return range1Dx(u.left, Q, \([x_0, u.x]\))
    + range1Dx(u.right, Q, \([u.x, x_1]\))

\(\text{u.point} \quad \text{u.x} \quad \text{Q} \quad \text{C} \quad \text{Q} \quad \text{C} \quad \text{Q} \quad \text{C} \subseteq Q\)
Multi-Layered Structures:

Suppose your ranges are formed from composing multiple (independent) queries:

E.g. Find all patients of
- age between 25..35 : $Q_1$
- weight $\leq 200$ lbs : $Q_2$
- blood pressure $> 100$ : $Q_3$

Idea: Design a data structure for each query type + "merge them"

How to merge?
- Build range structure for age for $P$
  $\Rightarrow$ Canonical subsets: $P_1, P_2, \ldots, P_m$
- For each $P_i$, build a range structure for weight
  $\Rightarrow$ Canonical subsets: $P_{i1}, P_{i2}, \ldots$
- For each $P_{ij}$, build range structure for blood pressure
Multi-Layered Search Tree:

- Store data in leaves of tree
- Each node’s canonical subset consist of its leaves
- For each node, build a search tree for its canonical subset called its auxiliary tree

Example:

Q₁: Sorted by age

Q₂: Sorted by weight
Orthogonal (2-d) Range Tree:

- Given points $P = \{p_1, ..., p_n\} \subseteq \mathbb{R}^2$
- Build a 1-d range tree for $P$ based on $x$ only (data in leaves)
- For each internal node $u$, let $P(u)$ be points in its leaves (canonical subset)
  - Build a 1-d range tree for $P(u)$ sorted by $y$-coords.

![Diagram showing the construction of a 2D range tree with labeled ranges for $x$-values and $y$-values.](image)
To process query $Q = [Q_{lo}, Q_{hi}] = [Q_{lo.x}, Q_{hi.x}] \times [Q_{lo.y}, Q_{hi.y}]$

- Apply 1-d search in main tree with query $[Q_{lo.x}, Q_{hi.x}]$ to identify $O(\log n)$ maximal subtrees.
- For each root $u$ of one of these max. subtrees apply 1-d search in $u$.aux with query $[Q_{lo.y}, Q_{hi.y}]$.
- Return overall sum.

range 2D(Node $u$, Range $Q$, Interval $C = [x_0, x_1]$)

if ($u$ is leaf)
  if $u$.point $\in Q$
    return 1
  else
    return 0
else if ($Q.x \cap C = \emptyset$) (no $x$ overlap)
  return 0
else if ($C \subseteq Q.x$) (containment in $x$)
  return range 1D$y(u.aux, Q, [-\infty, +\infty])$
else (recursive)
  return range 2D($u$.left, $Q$, $[x_0, u.x]$) + range 2D($u$.right, $Q$, $[u.x, x_1]$)
Space + Preprocessing Time:

- Since each node stores $O(1)$ data, total space = size of main tree + total size of aux. trees
- A tree with $m$ leaves has size $O(m)$

\[
\text{Space} = n + \sum_u |P(u)|
\]

- Main tree’s height is $O(\log n)$
- Each leaf contributes a point to $u$.aux for each of its ancestors
  \[\Rightarrow \text{Each point appears in } O(\log n) \text{ aux. trees}\]
  \[\Rightarrow \sum_u |P(u)| = O(n \log n)\]

\[\Rightarrow \text{Total space is } O(n \log n)\]

Construction time:

Naive: $O(n \log^2 n)$
Better: Build aux trees bottom-up
- Two child sets can be merged in linear time

\[\Rightarrow O(n \log n)\]
Query Time:

Main tree: $O(\log n)$ time
- Identifies $O(\log n)$ maximal subtrees
  - each has $\leq n$ points
  - each searchable in $O(\log n)$ time

$\Rightarrow$ Total time = $O(\log n) \cdot O(\log n)$
= $O(\log^2 n)$

Thm: Using orthogonal range trees, 2-dim orthog. range (counting) queries can be answered in:
- $O(n \log n)$ space
- $O(n \log n)$ build time
- $O(\log^2 n)$ query time $\rightarrow$ tk for reporting

Thm: Using orthogonal range trees, d-dim orthog. range (counting) queries can be answered in:
- $O(n \log n)$ space
- $O(n \log^{d-1} n)$ build time
- $O(\log^d n)$ query time $\rightarrow$ tk for reporting
Can we do better?

You can shave off a $\log n$ factor for query times - Cascading Search

$2$-dim: $O(\log^2 n) \rightarrow O(\log^2 n)$
$d$-dim: $O(\log^d n) \rightarrow O(\log^{d-1} n)$

(See latex notes)

Idea:

- Final aux trees can be stored as sorted arrays (trees not needed)
- Always searching for same values: $Q_{low} \preceq Q_{hi,y}$
- Can exploit knowledge of answer in one array to find answer in another, without doing search from scratch.