Range Counting

Count (or enumerate) objects in a given range

1D:

2D:
1D:

USE ARRAY: $O(\log n)$ to place $L, R \rightarrow$ to count.
$O(k + \log n)$ to enumerate/report.

2 problems:
- doesn't generalize to 2D (no array)
- not dynamic ... insert, delete data: $O(n)$
store size of subtree in each node
$Q_i \rightarrow \text{count } 1$
$O(\log n)$ nodes visited
- 2 paths root→leaf
- 1 neighbor off path per node

○: always "inside"
×: always "outside"
2D:

k-d tree (k=2)
recursively cut
median, alternating ↩️ ↔️

first cut

first node
2D:
2D:

until every cell has 1 point
Building a 2-d tree
at each level we compute medians.
(at disjoint) $\Sigma = O(n)$
$O(n \log n)$
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

We will only query the crossing leaves explicitly.
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

Algorithm:

Step 1: compare rectangle to the 2 rectangles of 1st partition

intersects both: proceed on both sides of tree
3 Types of Leaves (Rectangles)

- Inside
- Outside
- Crossing

Step 2: Compare to 4 rectangles (children)
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

Step 3: Finally one rectangle doesn't overlap. Ignore it and its children
3 Types of Leaves (Rectangles)

- Inside
- Outside
- Crossing

Level 4:
- No comparison of □ to □
- Ignore new external boxes: □
- Found internal □

Count total points inside & ignore in next levels
3 Types of Leaves (Rectangles)
- inside
- outside
- crossing

Similar to 1D,
- we implicitly count #points in maximally contained rectangles & ignore points in exterior rectangles

Unlike 1D,
- can branch a lot. But it's still a balanced BST, so work ~ # leaves reached (leaves ~ internal)
3 Types of leaves (Rectangles)
- inside 
- outside 
- crossing 

Work $\sim$ # crossing leaves

Other leaves are either
- in an ignored rectangle
- already counted (ancestor)

$\#\text{crossings} \leq 4 \times \text{max # rectangles stabbed by a line}$
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.
Every vertical node: must visit both children.

For vertical stabbing line, opposite: h-nodes ↔ v-nodes.
must visit both subtrees

Notice we don't care if \( \times \) means that a rectangle is in or out. Just counting leaves

horizontal stab

guaranteed to skip one subtree
Never visit more than 2 grandchildren of a visited node.

Compress every other level in tree: every path takes 2 of 4 subtrees.
Both trees:
- Take half of subtrees
- # internal nodes \(\approx\) # leaves (x2 at every level)
  \(\Rightarrow\) total work \(\sim\) # leaves
- O(1) work per node

Branching factor changes analysis:

# leaves: \(S(n) \leq 2 \cdot S\left(\frac{n}{4}\right) \leq 2^2 \cdot S\left(\frac{n}{16}\right) \leq 2^2 \cdot 2 \cdot S\left(\frac{n}{64}\right) \leq \ldots \leq 2^k \cdot S\left(\frac{n}{4^k}\right)\)

...ends when \(k \sim \log_4 n\) \(\Rightarrow\) \(S(n) \leq 2^{\log_4 n} \cdot S(1) = O(\sqrt{n})\)
So any rectangle can intersect only $O(\sqrt{n})$ edges of a 2-d tree.

- 2D range query (counting points in $\Box$): $O(\sqrt{n})$ time.
- Reporting: $O(k + \sqrt{n})$

Updating the 2-d tree

- Each branch represents median (by $x$ or by $y$)
- $\pm$ one point shifts median over by 1
- Can be complicated to rebalance

But possible

And this works in dimension $k$
Another intuitive idea: search $X$, then search $Y$
Another intuitive idea: search $X$, then search $Y$
Another intuitive idea: search X, then search Y.

Clearly we can't Y-sort every possible X-range.

$O(n \log n) \cdot O(n^2)$
Remember: any $x$-range can be described using $O(\log n)$ nodes in a tree.

For each node, store a new tree, sorting all contents of subtree by $y$.

(size of aux. trees)

\[
\frac{1}{2} \cdot n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + 8 \cdot \frac{n}{8} + \ldots + n \cdot 1 \sim n \log n \text{ space}
\]

-or-

Every point can only be represented in $O(\log n)$ nodes
- Building the tree of trees
  - Build primary tree, by $x \rightarrow O(n \log n)$ time
  - Build aux. trees starting from leaves: mergesort
  - Otherwise, each aux tree separately

$$n \log n + 2 \left( \frac{n}{2} \log \frac{n}{2} \right) + 4 \left( \frac{n}{4} \log \frac{n}{4} \right) + \ldots + n \cdot O(1) \approx n \log^2 n$$

- Query
  - Search $X$: identify $O(\log n)$ nodes $\implies O(\log^2 n)$
  - Search $Y$: in $O(\log n)$ aux. trees.
  - Union the secondary searches $O(\log n) + O(\log \frac{n}{2}) + O(\log \frac{n}{4}) + \ldots + O(1)$

- Updating
  - $O(\log n)$ primary nodes affected; $O(\log^2 n)$ total