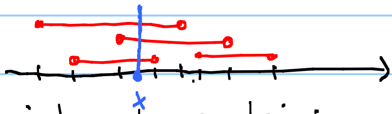
Interval trees / Priority search trees / Segment trees

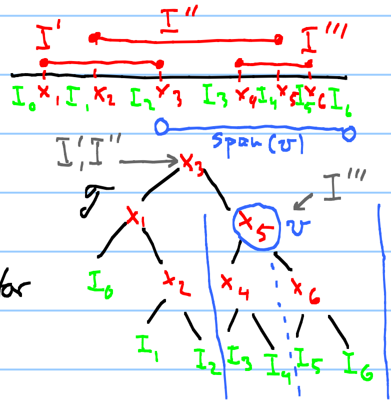
Notetitel 05-11-2008

Problem: Store n intervals $[x_i, x_i']$

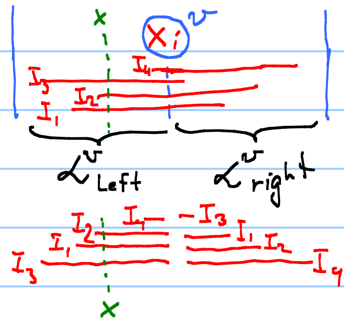


Query: Given x, report all intervals containing x.

Solution: Interval tree



- Build a binary tree $\mathcal{T}$ over the endpoints in internal nodes – leaves store the open intervals between consecutive endpoints.
- Each node spans an interval = union of subtree
- Store each interval at the lowest common ancestor of its endpoints
- Store each segment at v in two lists $\mathcal{L}^v_{left}$ and $\mathcal{L}^v_{right}$, that are sorted w.r.t. the left and right endpoints respectively



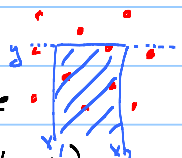
- A query x in the span of v will intersect a set of intervals that either is a prefix of $\mathcal{L}^v_{left}$ or a suffix of $\mathcal{L}^v_{right}$. If k_v intervals at v intersect x, then these can be found in $O(1+k_v)$ time
- Query: Follow search path in $\mathcal{T}$ for the query point ~ and report all intersecting intervals found on search path $\Rightarrow$ $O(\log n + k)$ time

  height $\mathcal{T}$ ↗ ↖ output size

  Space: $O(n)$

  Preprocessing: $O(n \cdot \log n)$

Problem: Store n points in $\mathbb{R}^2$ using $O(n)$ space

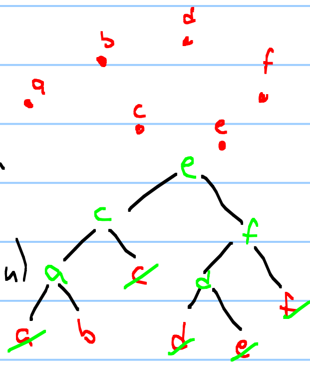


Query: Report all points within a 3-sided range

Note: Range trees solves the problem with $O(n \cdot \log n)$ space and query time $O(\log n + k)$.

Solution: Priority Search trees

- Sort points w.r.t. x-value, and store them at the leafs of a balanced binary tree

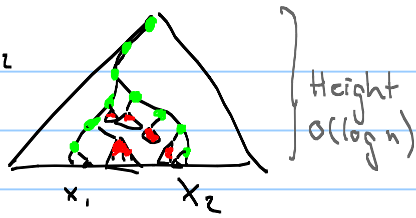


- Fill internal nodes top-down: Move lowest point w.r.t. y in a subtree to the root of the subtree (and remove it from the leaf) $\Rightarrow$ Space $O(n)$, Preprocessing $O(n \cdot \log n)$
- Properties: 1. Resulting trees satisfies heap order w.r.t. y

  2. A point from a leaf can only have moved to an ancestor

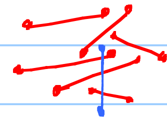
- Queries: - Report points on the paths to x_1 and x_2 which are within the query range (check each point both w.r.t. x and y).
  - For each subtree between x_1 and x_2, report top-down all points below y

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



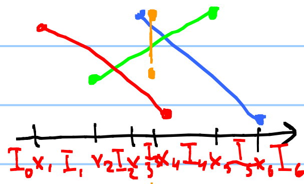
Problem: Store a set of n non-intersecting segments

Query: Report segments intersecting a vertical segment

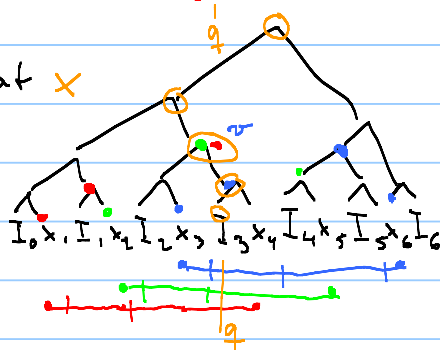


Solution:

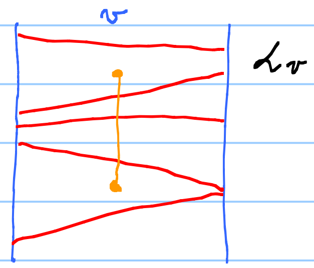
- Build a balanced binary tree over endpoints + intervals between endpoints as leaves (projection onto x)



- split a segment into $O(\log n)$ subsegments, such that each subsegment spans a complete subtree
- Segments intersecting vertical query line at x are stored along search path to leaf containing x



- All segments stored at a node v are stored in a sorted list L_v
- Query: for each node v on query path do a binary search + output segments

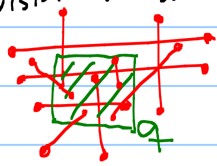


$\Rightarrow$ query time $O(\log^2 n + k)$

space $O(n \cdot \log n)$

preprocessing time: $O(n \cdot \log n)$

Application: Windowing queries for axis parallel segments – report all segments visible within a query rectangle.



Solution:
- Find all segments with at least one endpoint in q using range tree $\Rightarrow$ space $O(n \cdot \log n)$, time $O(\log n + k)$
- Find all segments crossing q's boundaries by quering a segment tree for each side of q $\Rightarrow$ space $O(n \cdot \log n)$, time $O(\log n + k)$
- Remove doublicates from output $\Rightarrow$ $O(k)$ time

(A segment can be reported because of 6 reasons: Each endpoint can be within q, and each side of q can be intersected once – by ranking the 6 reasons, only the highest applicable reason for a segment should generate the actual output)