Interval Trees and Segment Trees

slides by Andy Mirzaian

(a subset of the original slides are used here)

References:

- [M. de Berge et al] chapter 10

Data Structures:

- Interval Trees
- Priority Search Trees
- Segment Trees

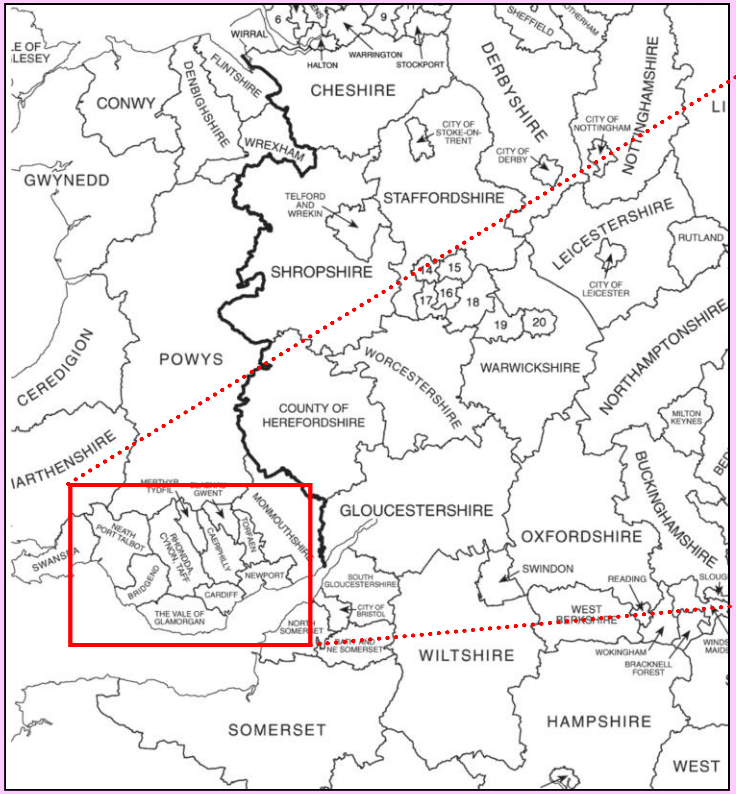
Applications:

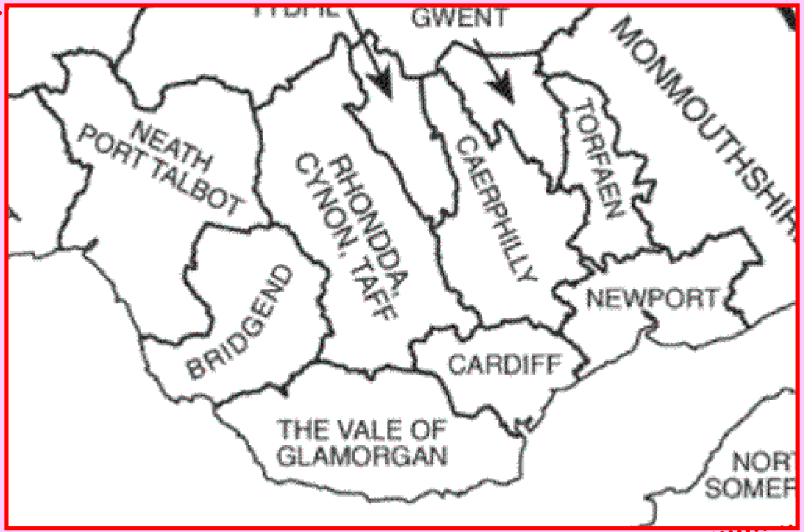
- Windowing Queries
- Vehicle navigation systems
- Geographic Information Systems
- Flight simulation in computer graphics
- CAD/CAM of printed circuit design

Windowing

PROBLEM 1: Preprocess a set S of non-crossing line-segments in the plane for efficient query processing of the following type:

Query: given an axis-parallel rectangular query window W, report all segments in S that intersect W.

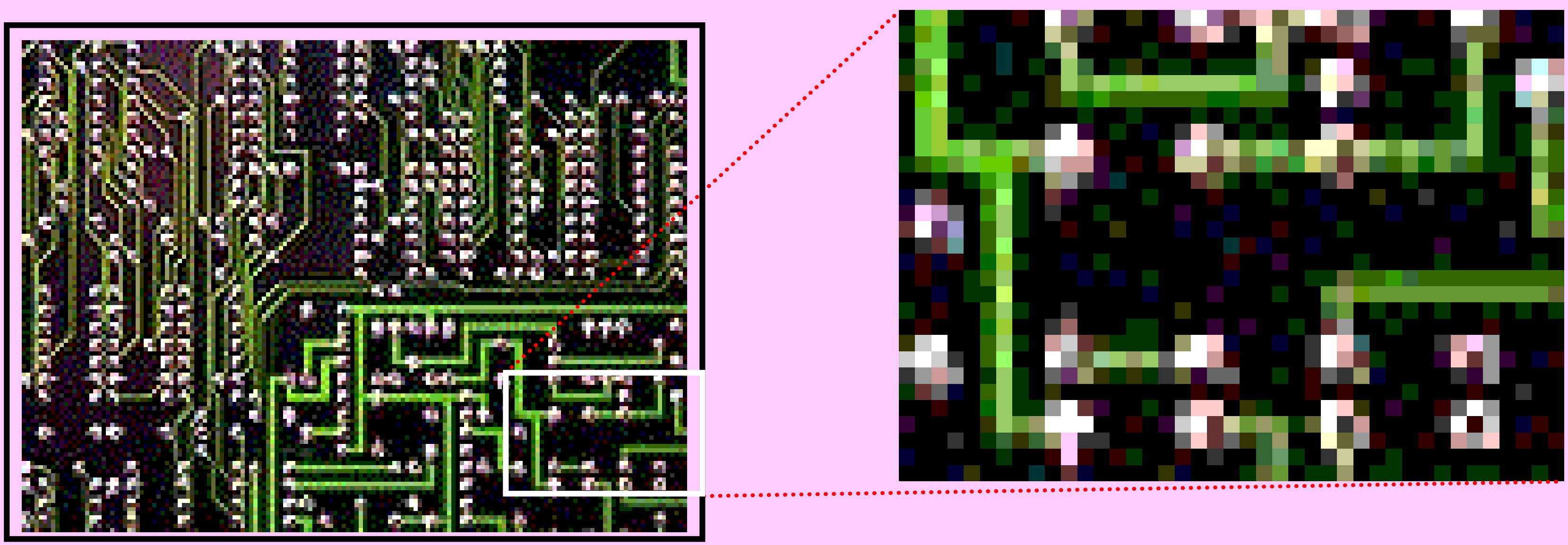




Windowing

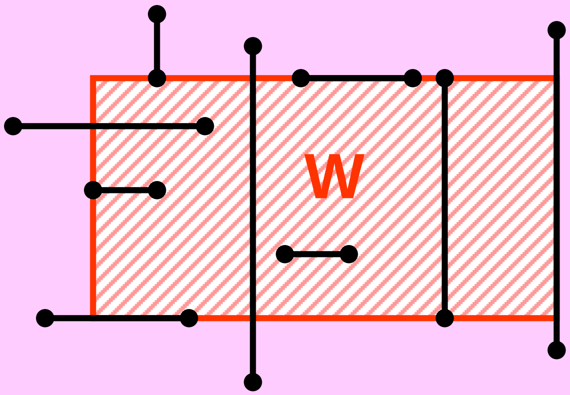
PROBLEM 2: Preprocess a set S of horizontal or vertical line-segments in the plane for efficient query processing of the following type:

Query: given an axis-parallel rectangular query window W, report all segments in S that intersect W.



INTERVAL TREES

PROBLEM 2: Preprocess a set S of horizontal or vertical line-segments in the plane for efficient query processing of the following type:
Query: given an axis-parallel rectangular query window W, report all segments in S that intersect W.



INTERVAL TREES

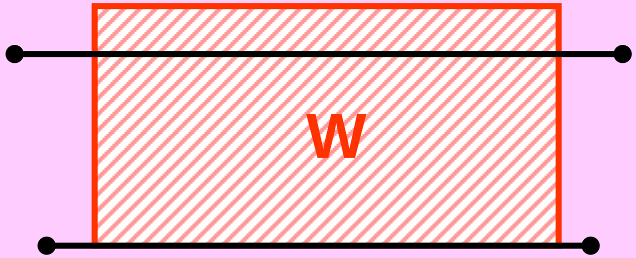
<u>SUB-PROBLEM 1.1 & 2.1</u>:
Let S be a set of n line-segments in the plane. Given an axis-parallel query window W, the segments of S that have at least one **<u>end-point inside</u>** W can be reported in O(K + log n) time with a data structure that uses O(n log n) space and O(n log n) preprocessing time, where K is the number of reported segments.

Method:
Use 2D Range Tree on segment end-points and fractional cascading.

INTERVAL TREES

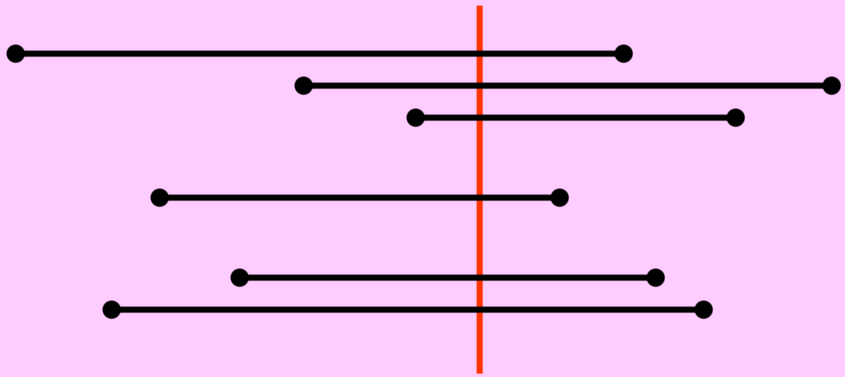
Now consider horizontal (similarly, vertical) segments in S that intersect W, but their end-points are **outside** W.

They must all cross the left edge of W.

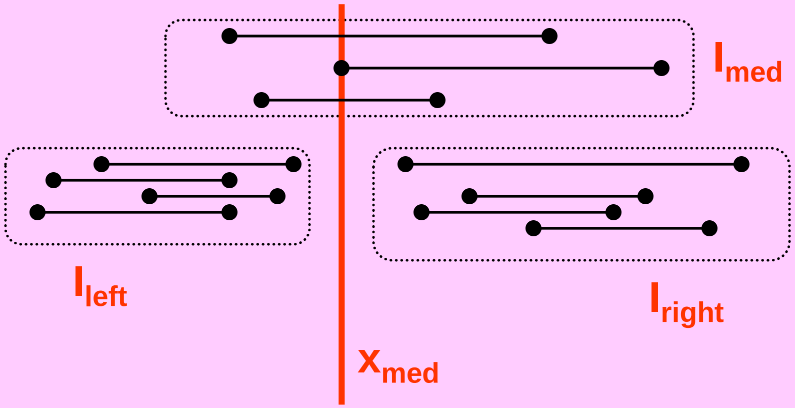


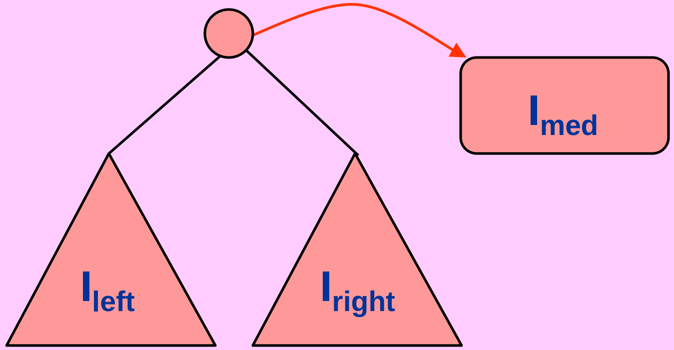
SUB-PROBLEM 2.2:
Preprocess a set S_H of horizontal line-segments in the plane, so that the subset of S_H that intersects a query **vertical line** can be reported efficiently.

Method: Use Interval Trees.



INTERVAL TREES

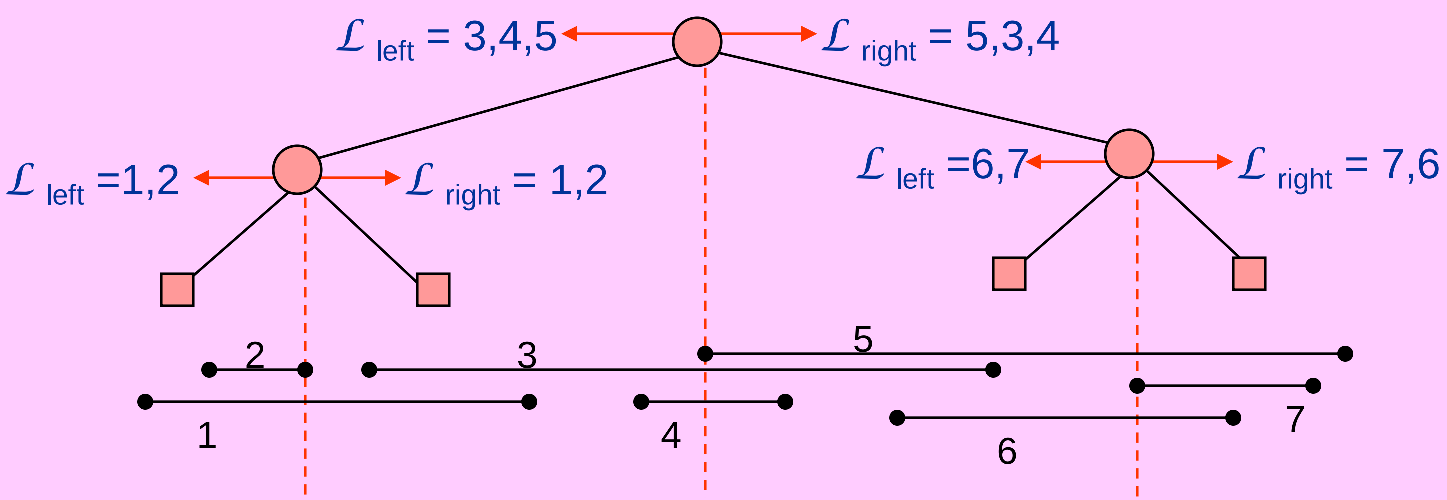




Associated structure for I_{med} :

$\mathcal{L}_{left}$ = list of segments in I_{med} sorted by their left end-points,

$\mathcal{L}_{right}$ = list of segments in I_{med} sorted by their right end-points.



INTERVAL TREES

THEOREM: Interval Tree for a set of n horizontal intervals:

- O(n) storage space
- O(n log n) construction time
- O(K + log n) query time

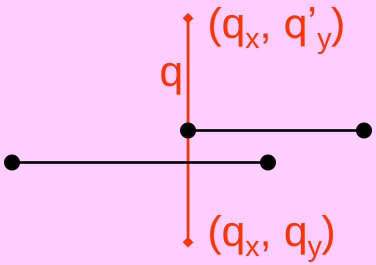
[report all K data intervals that contain a query **x-coordinate**.]

INTERVAL TREES

SUB-PROBLEM 2.3:

Now instead of the query being on a vertical line, suppose it is on a vertical **line-segment**.

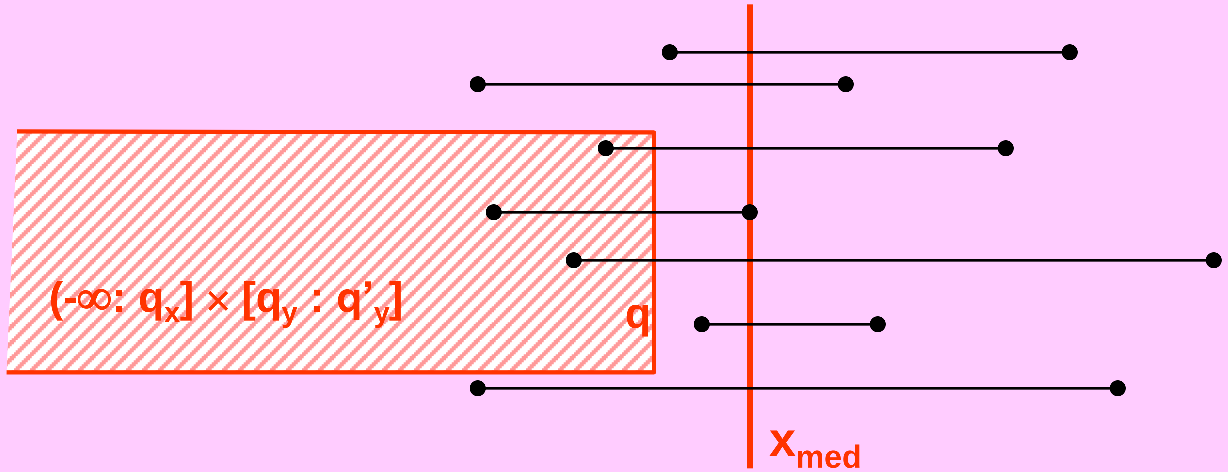
The primary structure of Interval Trees is still valid. Modify the associated secondary structure.



SOLUTION:

$\mathcal{L}_{\text{left}}$ = **Range Tree** on left end-points of I_{med} ,

$\mathcal{L}_{\text{right}}$ = **Range Tree** on right end-points of I_{med} .



INTERVAL TREES

THEOREM: Interval Tree for a set of n horizontal intervals:

- $O(n \log n)$ storage space
- $O(n \log n)$ construction time
- $O(K + \log^2 n)$ query time

[report all K data intervals that intersect a query **vertical line-segment**.]

COROLLARY: Let S be a set of n horizontal or vertical line-segments in the plane. We can preprocess S for axis-parallel rectangular query window intersection with the following complexities:

- $O(n \log n)$ storage space
- $O(n \log n)$ construction time
- $O(K + \log^2 n)$ query time

[report all K data intervals that intersect the query **window**.]

PRIORITY SEARCH TREES

Improving the previous solution:
the associated structure can be implemented by Priority Search Trees, instead of Range Trees.

$P = \{p_1, p_2, \ldots , p_n \} \subseteq \Re^2$.

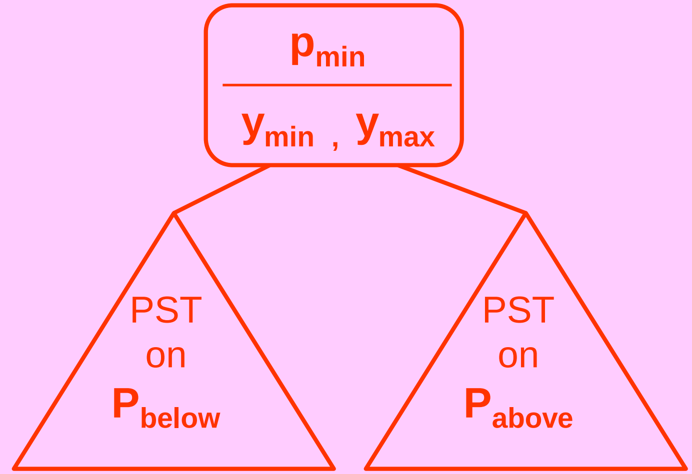
A Priority Search Tree (PST) $\mathcal{T}$ on P is:

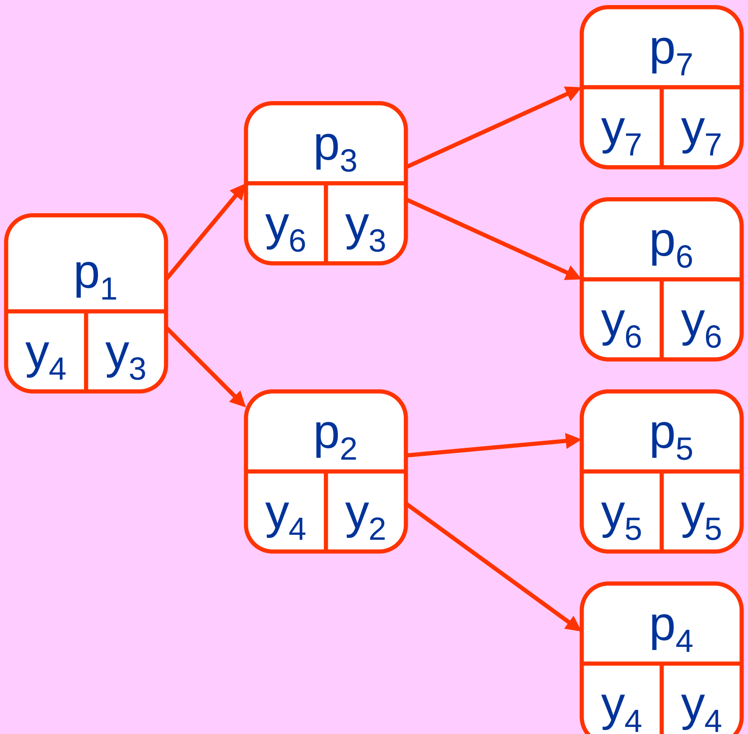
- a binary tree, one point per node,
- heap-ordered by x-coordinates,
- (almost) symmetrically ordered by y-coordinates.

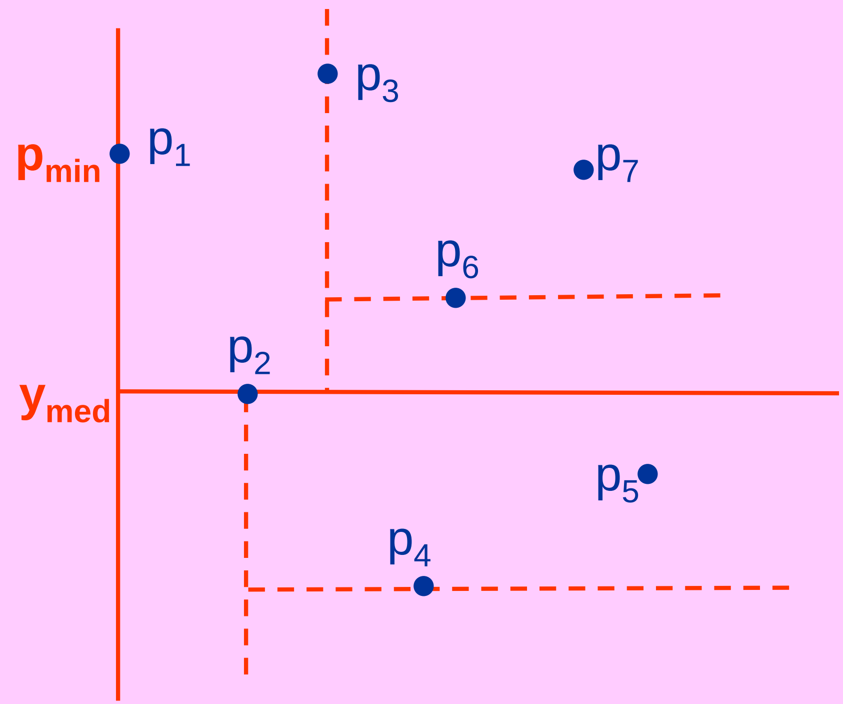
PRIORITY SEARCH TREES

$p_{min} \leftarrow$ point in P with minimum x-coordinate.
$y_{min} \leftarrow$ min y-coordinate of points in P
$y_{max} \leftarrow$ max y-coordinate of points in P

$P' \leftarrow P - \{p_{min}\}$
$y_{med} \leftarrow$ y-median of points in P'
$P_{below} \leftarrow \{ p \in P' \mid p_y \leq y_{med} \}$
$P_{above} \leftarrow \{ p \in P' \mid p_y > y_{med} \}$







PRIORITY SEARCH TREES

Priority Search Tree $\mathcal{T}$ on n points in the plane requires:

- O(n) storage space
- O(n log n) construction time:
  - either recursively, or
  - pre-sort P on y-axis, then construct $\mathcal{T}$ in O(n) time bottom-up. (How?)

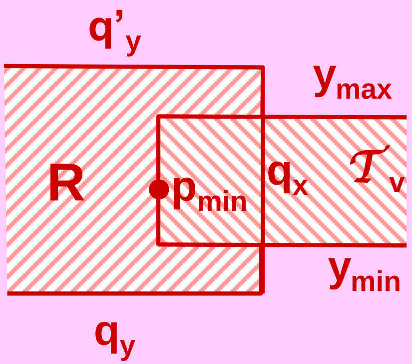
Priority Search Trees can replace the secondary structures (range trees) in **Interval Trees**.

- simpler (no fractional cascading)
- linear space for secondary structure.

How to use PST to search for a query range $R = (-\infty: q_x] \times [q_y : q'_y]$?

ALGORITHM QueryPST (v, R)

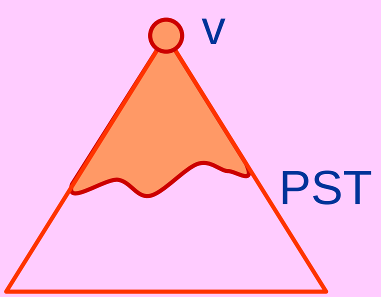
if $v = nil$ or $p_{min\ x}(v) > q_x$ or $y_{min}(v) > q'_y$ or $y_{max}(v) < q_y$
  then return
if $p_{min\ x}(v) \le q_x$ and $q_y \le y_{min}(v) \le y_{max}(v) \le q'_y$
  then Report.In.Subtree (v, q_x)
  else do
    if $p_{min\ x}(v) \in R$ **then** report $p_{min\ x}(v)$
    QueryPST (lc(v), R)
    QueryPST (rc(v), R)
  end else
end



PROCEDURE Report.In.Subtree (v, q_x)

if $v=nil$ **then return**
if $p_{min\ x}(v) \le q_x$ **then do**
  report $p_{min\ x}(v)$
  Report.In.Subtree (lc(v), q_x)
  Report.In.Subtree (rc(v), q_x)
end if
end

Truncated Pre-Order on the Heap: $O(1 + K_v)$ time.

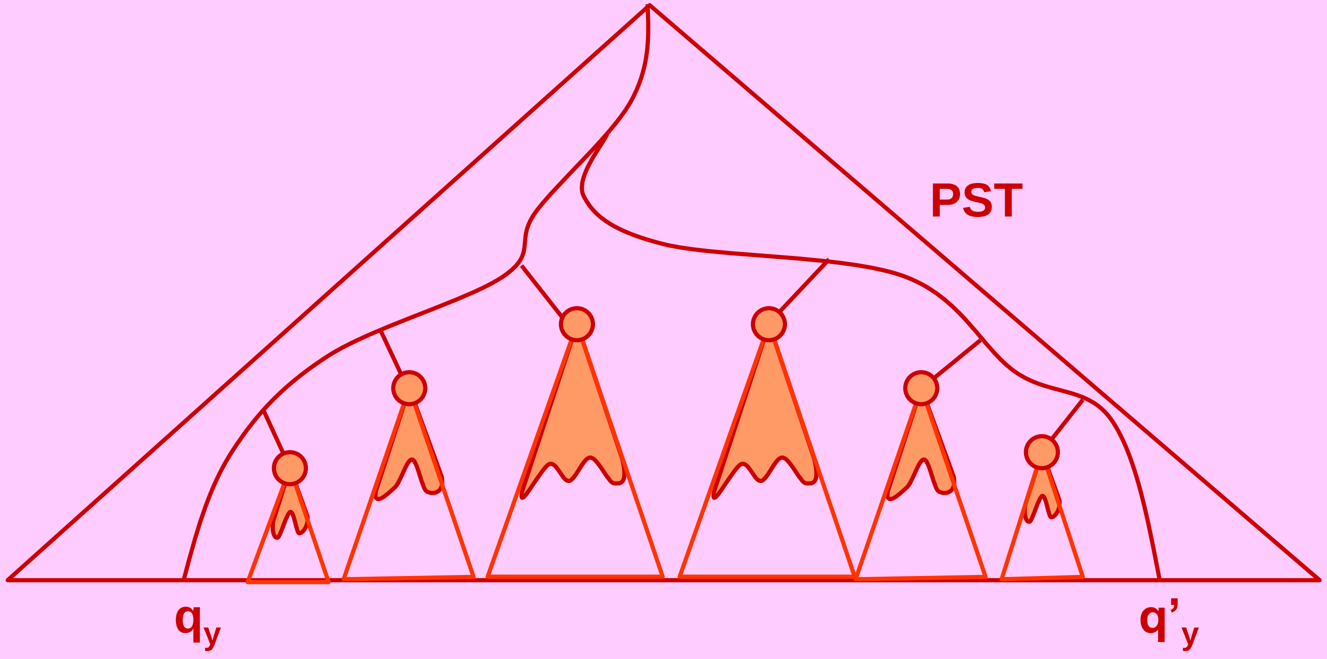


LEMMA: Report.In.Subtree(v, q_x) takes $O(1 + K_v)$ time to report all points in the subtree rooted at v whose x-cooridnate is $\le q_x$, where K_v is the number of reported points.

THEOREM: Priority Search Tree for a set P of n points in the plane has complexities:

- $O(n)$ Storage space
- $O(n \log n)$ Construction time
- $O(K + \log n)$ Query time

[report all K points of P in a query range $R = (-\infty: q_x] \times [q_y : q'_y]$.]

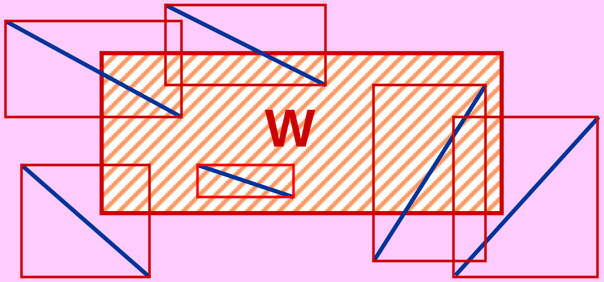


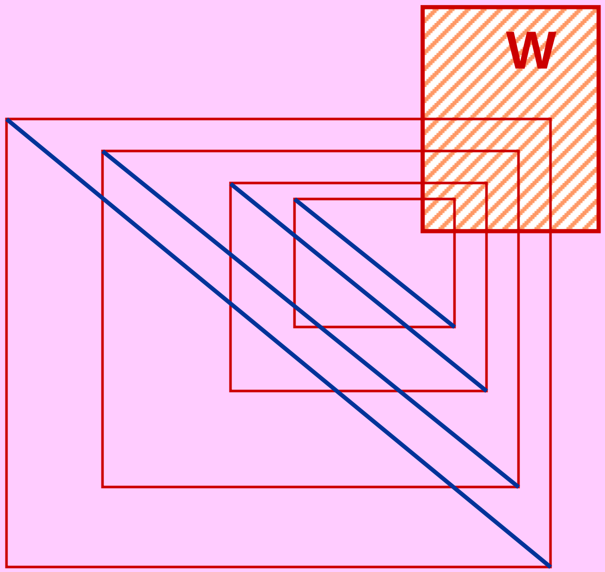
SEGMENT TREES

Back to Problem 1: Arbitrarily oriented line segments.

Solution 1: Bounding box method.

Bad worst-case.
Many false hits.





SEGMENT TREES

Back to Problem 1: Arbitrarily oriented line segments.

Solution 2: Use Segment Trees.

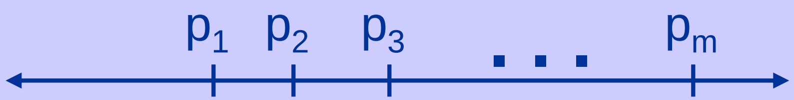
a) Segments with end-points in W can be reported using range trees (as before).

b) Segments that intersect the boundary of W can be reported by Segment Trees.

SUB-PROBLEM 1.1: Preprocess a set S of n non-crossing line-segments in the plane into a data structure to report those segments in S that intersect a given **vertical** query segment $q = q_x \times [q_y : q'_y]$ efficiently.

SEGMENT TREES

Elementary x-intervals of S



$(-\infty: p_1), [p_1 : p_1], (p_1 : p_2), [p_2 : p_2], \ldots , (p_{m-1} : p_m), [p_m : p_m], (p_m : +\infty).$

Build a balanced search tree with each **leaf** corresponding (left-to-right) to an elementary interval (in increasing x-order).

Leaf v:

Int(v) = set of intervals (in S) that contain the elementary interval corresponding to v.

IDEA 1: Store Int(v) with each leaf v.

Storage O(n²), because intervals in S that span many elementary intervals will be stored in many leaves.

SEGMENT TREES

IDEA 2:

∀ internal node v:

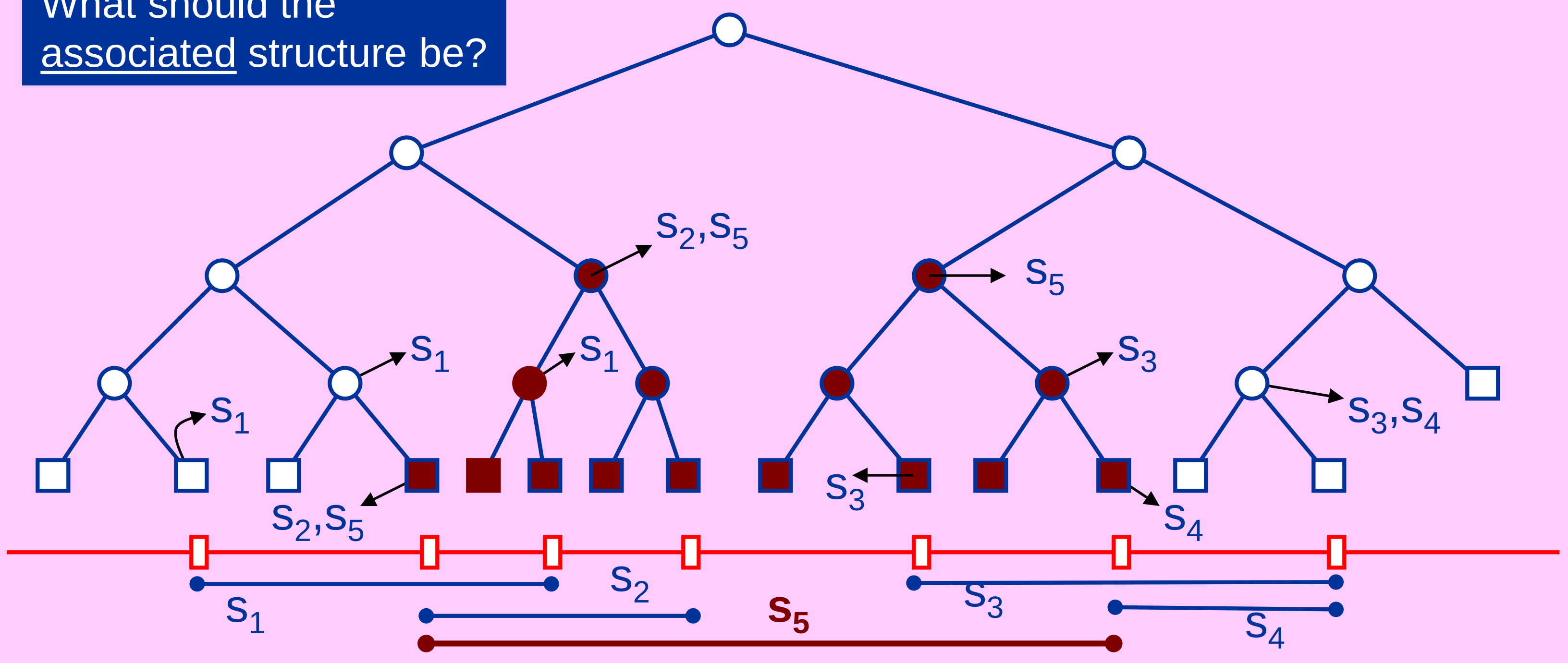
Int(v) = union of elementary intervals corresponding to the leaf-descendents of v.

Store an interval [x:x’] of S at a node v iff Int(v) ⊆ [x:x’] but Int(parent(v)) ⊄ [x:x’].

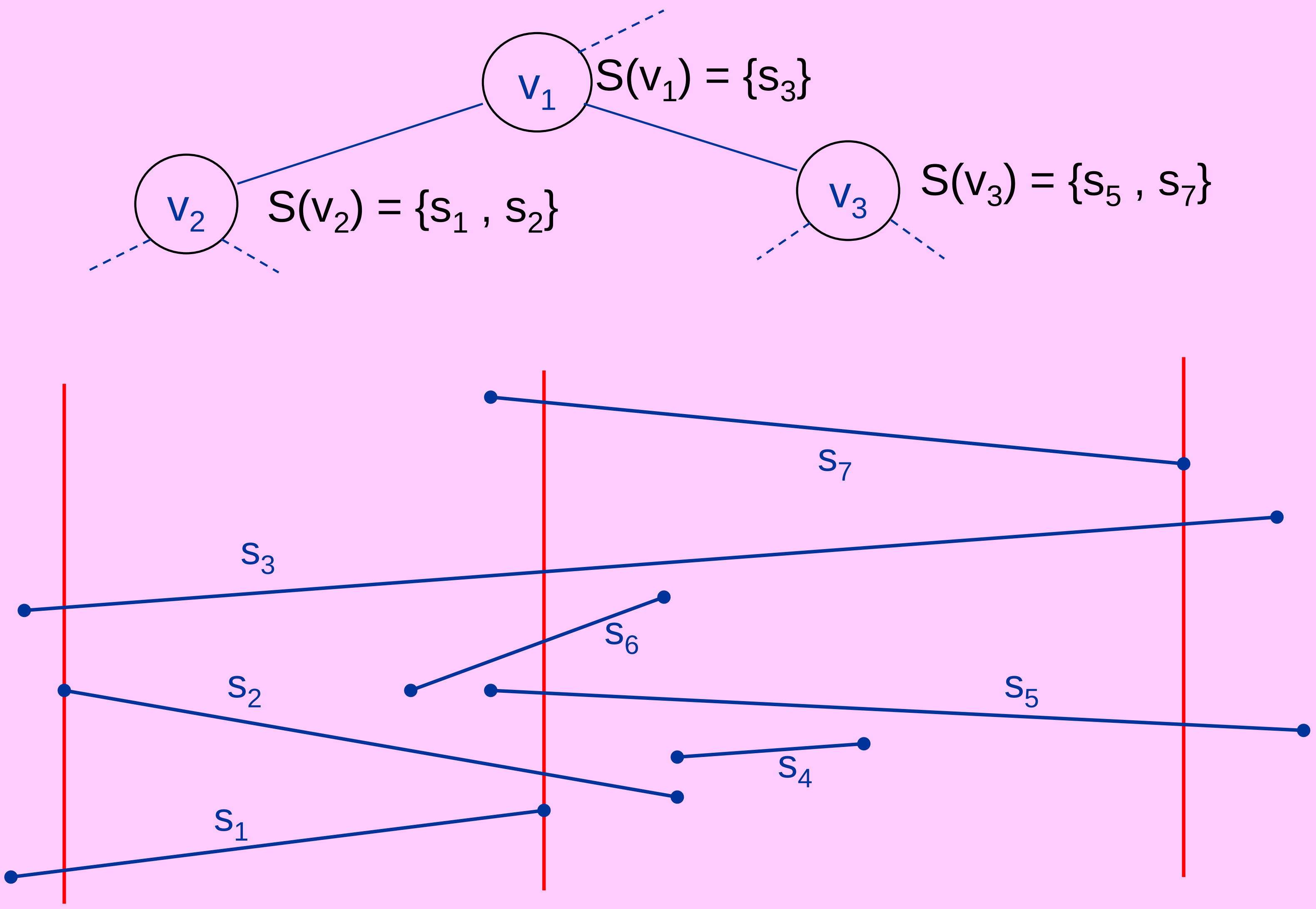
Each interval of S is stored in at most 2 nodes per level (i.e., O(log n) nodes).

Thus, storage space reduces to O(n log n).

What should the <u>associated</u> structure be?



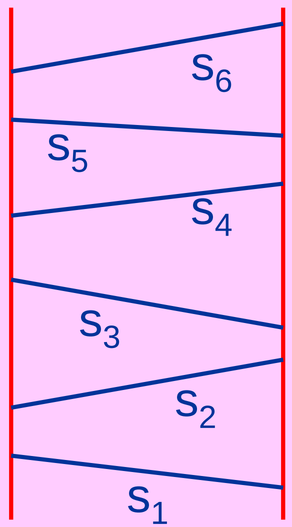
SEGMENT TREES

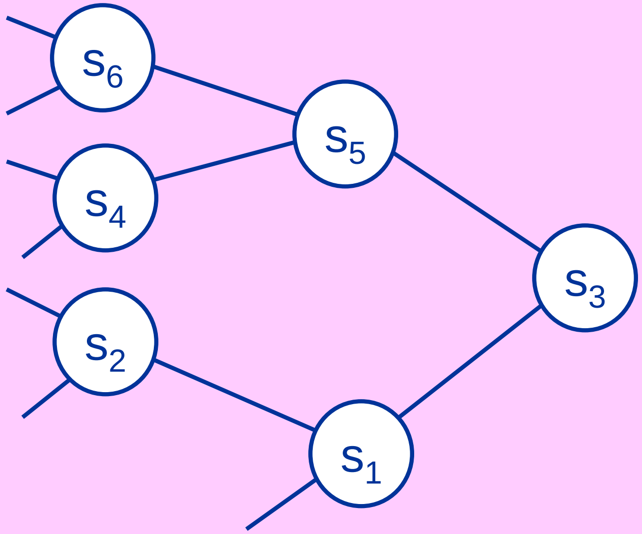


SEGMENT TREES

Associated structure

is a balanced search tree based on the vertical ordering of segments S(v) that cross the slab Int(v) × (-∞ : +∞).





SEGMENT TREES

THEOREM:

Segment Tree for a set S of n non-crossing line-segments in the plane:

- $O(n \log n)$ Storage space
- $O(n \log n)$ Construction time
- $O(K + \log^2 n)$ Query time
  [report all K segments of S that intersect a vertical query line-segment.

COROLLARY:

Segment Trees can be used to solve Problem 1 with the above complexities. That is, the above complexities applies if the query is with respect to an axis-parallel rectangular window.