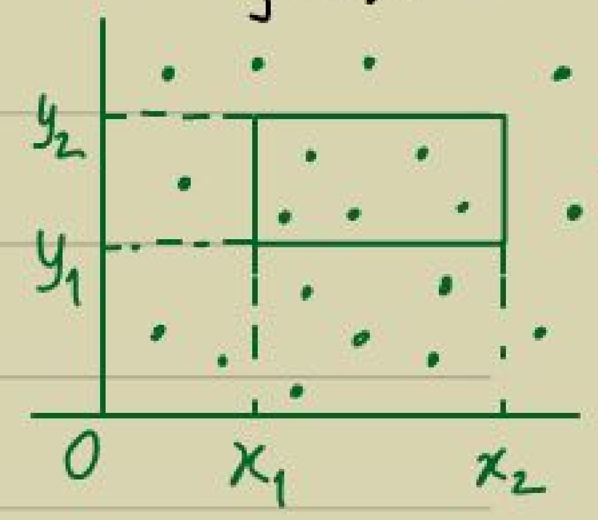
- Another example of augmented AVL tree:

Orthogonal Range Searching

- Input: A set of points $T \subset \mathbb{R}^2$ & a rectangle $(x_1, y_1, x_2, y_2) =: R$.

  Output: All points in the rectangle, i.e. $T \cap R$.



- Brute-force: It can be solved in $O(n)$ time.

  Qn: Can it be done significantly faster?

  Let $k := |T \cap R|$.

- Easier question: Find the points on the line (x_1, x_2)?
  - Store the points in an AVL tree wrt the x-coordinate.
  - Search x_1, x_2 in T & find the least common ancestor (lca) y.

▷ In the tree T the blue shaded nodes are exactly the points in $[x_1, x_2]$.

▷ If $|T \cap [x_1, x_2]| =: \ell$ then these points can be found in $O(\ell + \lg n)$ time.

- Next, how do we find the points in $T \cap R$?

- Ans: Augment each node v by adding another copy of tree(v) as: An AVL tree organized wrt y-coordinates.

Call this Ytree(v).

- This inspires the following pseudocode for RangeSearch(T, x_1, x_2, y_1, y_2):
  - For root v of each blue shaded subtree { Do RangeSearch(Ytree(v), y_1, y_2) }

We may make $O(\lg n)$ such calls.

  - For the other blue vertices v on the

search path: Check whether $v \in ? T \cap R$.
• Output the ones found in $T \cap R$.

▷ Orthogonal Range Search can be done in $O(k + \lg^2 n)$ time.

Pf: (Exercise) $\square$

- Note that preprocessing time taken is $O(n \lg n)$.

But, the query time is significantly lower!

- Note that the space required by the augmented AVL tree is $\approx$

$$\sum_{v \in T} |\text{tree}(v)| = \sum_{u \in T} \#(v \in T : v \text{ is an ancestor of } u)$$

$$\leq |T| \cdot \text{depth}(T)$$

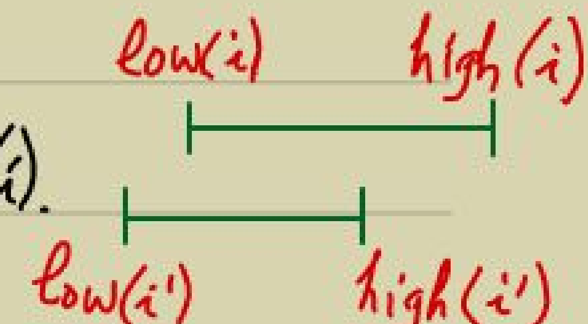
$$= O(n \lg n).$$

Interval Trees

- Computational geometry, or scheduling, problems require organization of intervals.

- Interval $i = [t_1, t_2]$ has the low endpoint $t_1 = low(i)$ & high endpoint $t_2 = high(i)$.

- Interval i, i' overlap if $i \cap i' \neq \phi$.
  Equivalently,
  $low(i) \leq high(i')$ & $low(i') \leq high(i)$.



- Qn: Is there a data structure where an overlapping interval can be searched in $O(\lg n)$ time? (given an i)

- Ans: Let T be the set of n intervals. Organize T into an AVL tree wrt the low endpoints,

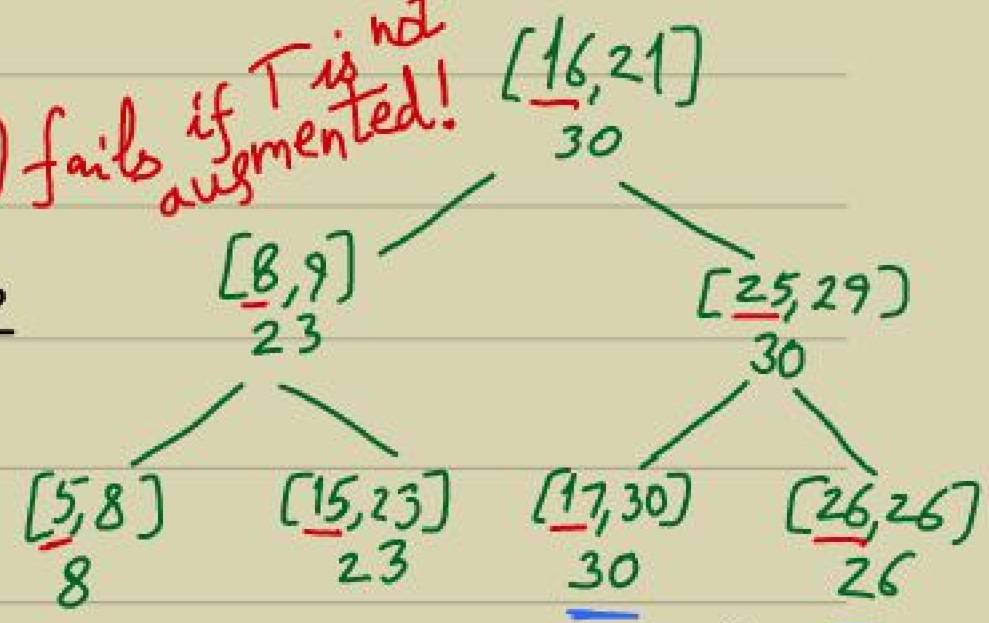
- We now need to implement:
  1) Insert (T, i): insert i into T.
  2) Delete (T, i): delete i from T.
  3) Search (T, i): return a pointer to a node $x \in T$ that overlaps with i.

- To search for i, just having $low(v)$, in every node $v \in T$, is not enough.

Augmented AVL

We also store $max(v)$:= the maximum value across all the intervals in tree (v).

Example: Searching for $[30,30]$ fails if T is not augmented!



▷ Insert (T, i) can be done in $O(\lg n)$ time.

Pf: One needs to change $max(v)$ in only depth(T) many ancestors, while inserting i. *

□

▷ Similarly, for Delete (T, i).

* Not a bst wrt $max(v)$.

- The pseudocode for Search(T, i) is mainly guided by "$low(i) \leq max(left(v))$":

• $v \leftarrow root(T)$;
• while (i does not overlap $int(v)$) {
high(i) → not used?
  • if ($low(i) \leq max(left(v))$)
  then $v \leftarrow left(v)$;
  else $v \leftarrow right(v)$; }
• return v;

- Caution: Handle the boundary conditions like – $v = NULL$ or $left(v) = NULL$ or $right(v) = NULL$.

Exercise: Show that it correctly finds a $v \in T$ s.t. $int(v) \cap i \neq \phi$ in $O(\lg n)$ time.

Hint 1: Loop invariant – If i overlaps with some interval in T, then " " " " " " tree(v).

Hint 2: If $low(i) \leq max(left(v))$ & i overlaps with some interval in tree(v), then i overlaps with someone in left(v).

Proof:

- Otherwise, it means that $\forall u \in left(v)$, $i \cap int(u) = \phi$.

$\Rightarrow low(i) > high(int(u))$ OR
$high(i) < low(int(u))$

$\Rightarrow \exists u \in left(v)$, $high(i) < low(int(u))$ [$\because low(i) \leq max(left(v))$]

$\Rightarrow low(i) < high(i) < low(int(u))$.

- This means that $high(i) < low(int(u'))$, $\forall u' \in tree(right(v)) \cup \{v\}$. [$\because$ T uses low endpoints]
- Hence, i does not overlap with any interval in tree(v). $\square$

! It's a tricky proof, as it deduces a lot about high(i) !

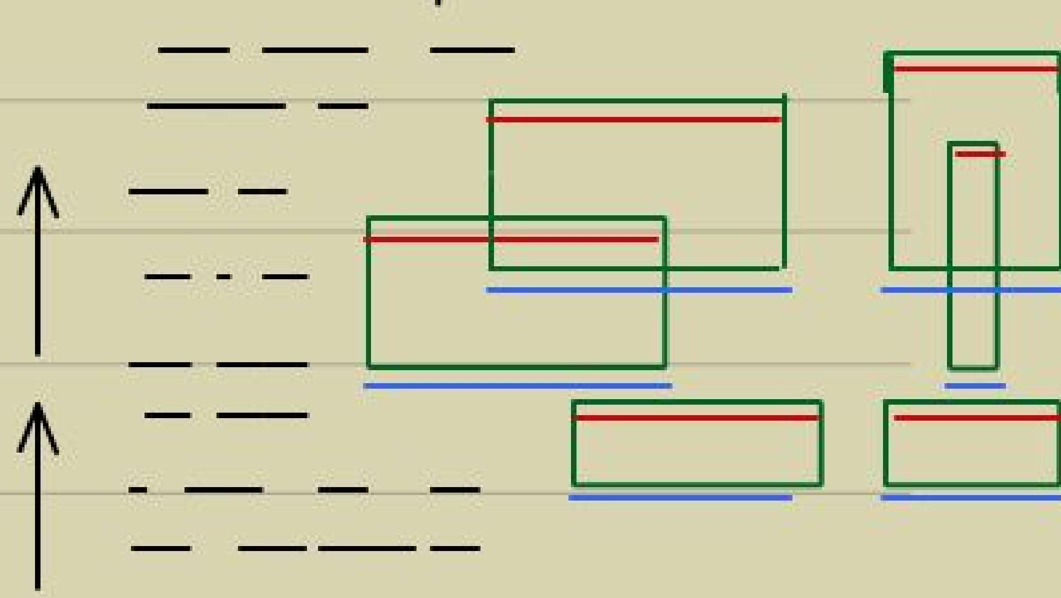
Apply to Rectangle Overlap

- Input: A list L of axis-parallel rectangles (n of them via $2n$ points).

- Output: YES if two of them overlap

Qn: Can you solve in time less than $O(n^2)$?

- Idea: (Virtual line sweep!)
    - Order the red & blue edges wrt y-coordinates in an array A.



    - Pick an edge e from A, in order.
    - If e is blue: Check whether e overlaps with an edge in an interval tree T; if no, Insert (T, e).

- If e is red: Let e′ be the associated blue edge. Search & Delete e′ from T. If $e' \notin T$ then OUTPUT OVERLAP. Else goto the next e in A.

Exercise: Write the pseudocode & prove that it works in time $O(n \lg n)$.

- Invariant 1: T has non-overlapping blue intervals with their reds yet to be swept.
- Invariant 2: Unmatched red means an overlap.

- Eg. B R B R
- Eg. B B R R
- Eg. B B R R