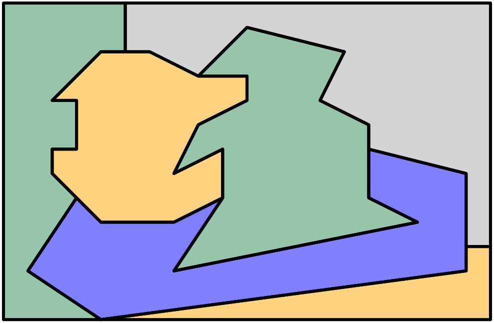
Augmented Trees (part 2)

Comp 160: algorithms

Tufts University

Warm-up Question

Say you have an interactive map. User clicks on a pixel.



How to determine which country was selected?

Previously

Augmenting is a standard idea in CS

Explicitly store a derived attribute
(i.e., current bank balance after many transactions)

Reduces runtime

Beware: with duplicate information must update all copies!

When augmenting AVL/BST trees:

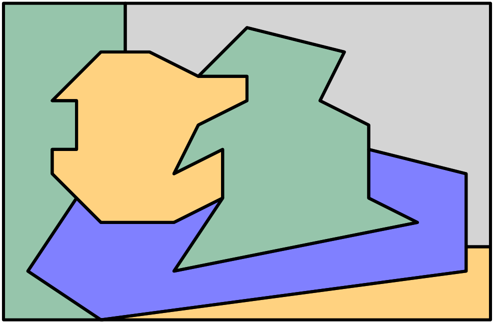
Data should depend on descendants

Explain how to update on rotations

Might add extra operations on initialize/insert/delete

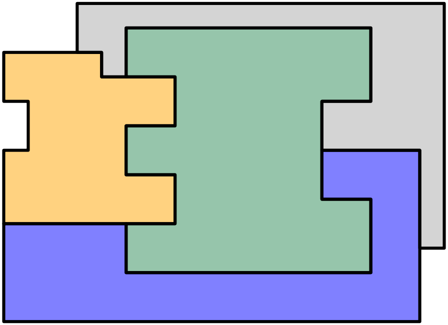
Point location problem

Interactive map. User clicks on a pixel. What country is it?



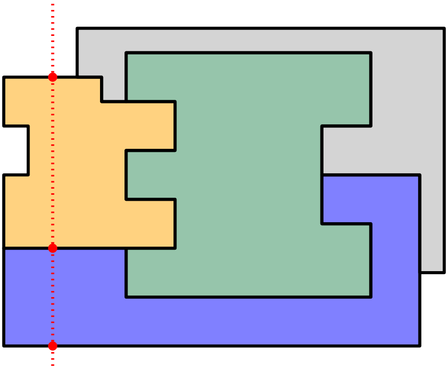
- Full algorithm in Comp 163/Math 181
- Let's discuss easier case

Orthogonal Point Location



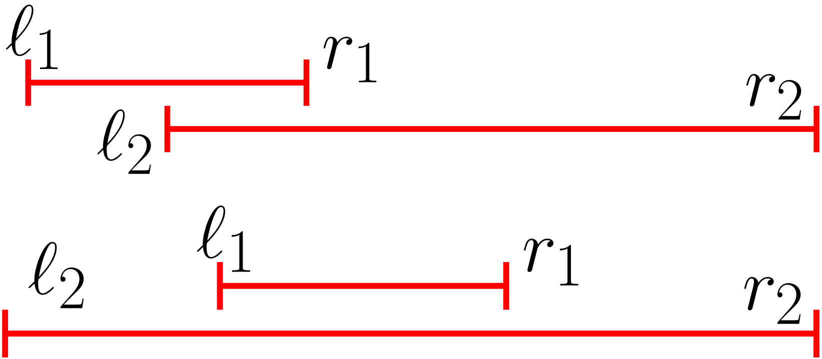
- To define *country* we need to find intersections
- Technique called *plane sweep*

Plane sweep



- Horizontal segments are added to DS
- Vertical segments are *queries* (want to find all intersections)
- More details in recitation

Cheat of the day

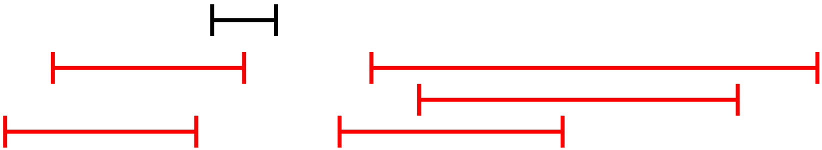


Two intervals $A_1 = (\ell_1, r_1)$ and $A_2 = (\ell_2, r_2)$ overlap if and only if:

$\ell_1 \leq \ell_2 \leq r_1$ or $\ell_2 \leq \ell_1 \leq r_2$

We can check this in $O(1)$ time

Problem definition



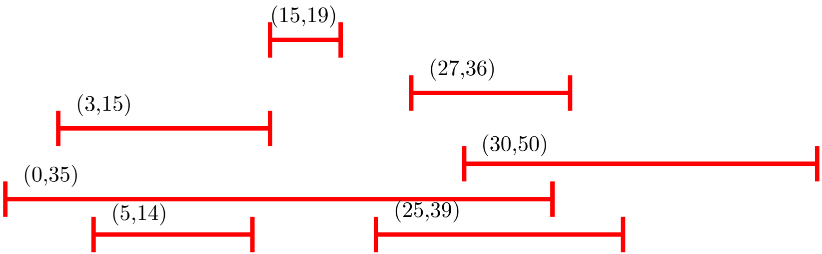
We want to preprocess intervals to answer **intersection** queries.

Given a query interval, can you find any that intersects?

Want a **dynamic** DS (additions/deletions)

How fast is the algorithm? Must explain **all** runtimes

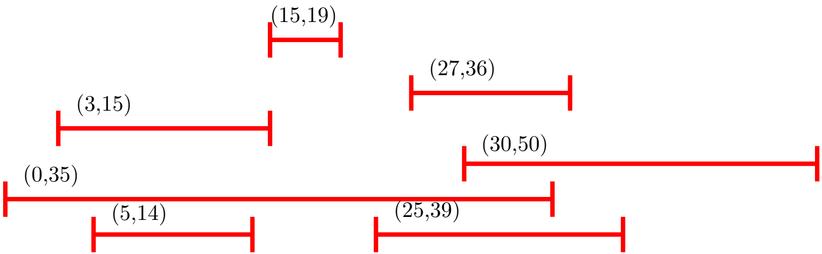
Choosing the augmentation



Let's store intervals in an AVL tree:

- We want unique key ⇒ let's index by left endpoint
- Two intervals are checked for intersection in $O(1)$ time
- How to compare segment versus subtree? ⇒ we store the span

Span



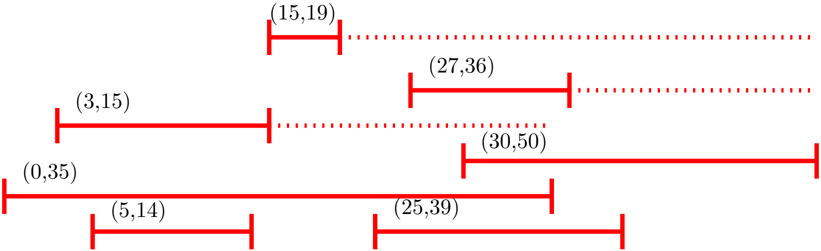
Span(node) = Interval(node.ℓ, largest right endpoint in subtree)

Intuitive idea: if query avoids span $\Rightarrow$ we can prune intervals

If query intersects span $\Rightarrow$ ... its complicated

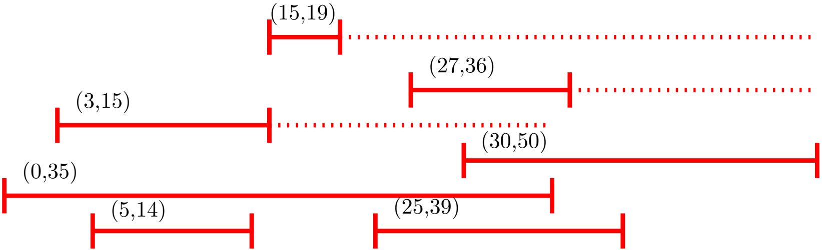
Case analysis incoming!

Case 1a: $q \cap \text{span(root)} = \emptyset$ and q is to the right



No intersection with tree!

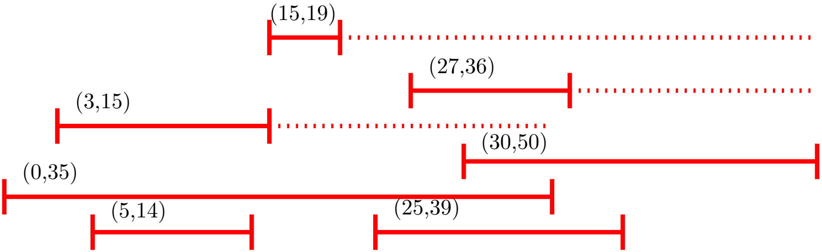
Case 1b: $q \cap$ span(root)$= \varnothing$ and q is to the left



Cannot intersect with right subtree

Recurse left subtree

Case 2: $q \cap span(root) \neq \emptyset$



Let L= Span(root $\rightarrow$ left child)

If $q \cap$ L $\neq \emptyset \Rightarrow$ Must intersect left Recurse left

If $q \cap$ L $= \emptyset \Rightarrow$ Cannot intersect left Recurse right

Summary

$\textsc{Cross}$(query interval q, Node n)

If n is null pointer $\Rightarrow$ No intersection

If q intersects $n \Rightarrow$ return n

If $q \cap Span(n) = \varnothing \wedge q$ is to the right $\Rightarrow$ No intersection

Else if $q \cap Span(n) = \varnothing \Rightarrow$ Return $\textsc{Cross}(q, n \rightarrow \text{left})$

(from here down we know $q \cap Span(n) \neq \varnothing$)

If $q \cap Span(n \rightarrow \text{left}) \neq \varnothing \Rightarrow$ Return $\textsc{Cross}(q, n \rightarrow \text{left})$

Else $\Rightarrow$ Return $\textsc{Cross}(q, n \rightarrow \text{right})$

Note: algorithm assumes Span(null pointer)$=\varnothing$

Runtime?

Lemma

Given an augmented tree with n segments, we can find if a query segment q intersects any of them in $\Theta(\log n)$ time.

Done? No! Need to handle updates

Updating Augmented Tree

Span(node) = Interval(node.ℓ, largest right endpoint in subtree)

Left endpoint never changes

We need only update largest right endpoint in subtree

Initialization No changes. Create empty tree

Insertion Insert recursively as usual

Along path: Span(node) $\leftarrow$ max$\{oldSpan, new.right\}$

Deletion Delete recursively as usual.

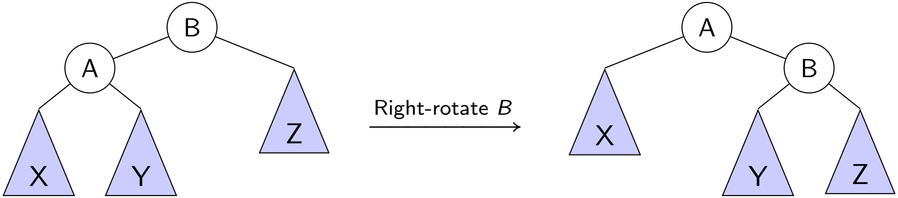
Beware! Must recompute Span

Recursion to the rescue!

Transmit information upwards!

Rotations Beware! Extra work to maintain Span

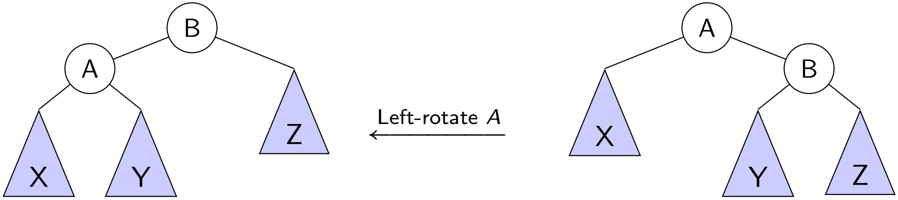
Rotations



Span(A)$\leftarrow$ (A.ℓ, oldSpan(B).r)

Span(B)$\leftarrow$ ($B.\ell$, max$\{B.r, Span(Y).r, Span(Z).r\}$)

Rotations



Span(B)$\leftarrow$ (B.ℓ, oldSpan(A).r)

Span(A)$\leftarrow$($A.\ell$, max$\{A.r, Span(X).r, Span(Y).r\}$)

... or just say reverse of right rotate.

Summary

Augmenting is a ~~new~~ old tool

Good when you want a derived attribute

- Do you need smallest/largest/count of many items?
- Repeated queries after some preprocessing?
- Or a dynamic setting?

Make sure to:

- First choose what exactly to augment
- Use that info to speed up algorithm
- Verify that maintenance does not impact runtime