#### Augmenting AVL trees

#### How we've thought about trees so far



## Other kinds of uses?

- Any thoughts?
- Finding a minimum/maximum...
  - (heaps are probably just as good or better)
- Finding an average?
- More complicated things?!!!11one

Enter: idea of augmenting a tree

## Augmenting

- Can quickly compute many global properties that seem to need knowledge of the whole tree!
- Examples:
  - size of any sub-tree
  - height of any sub-tree
  - averages of keys/values in a sub-tree
  - min+max of keys/values in any sub-tree, …
- Can quickly compute **any function f(u)** so long as you only need to know f(u.left) and f(u.right)!

#### Augmenting an AVL tree

- Can augment any kind of tree
- Only balanced trees are guaranteed to be fast
- After augmenting an AVL tree to compute f(u), we can still do all operations in O(lg n)!

## Simple first example

- We are going to do one
- Then, you will help with

A regular AVL tree already does this

- Problem: augment an AVL tree so we can do:
  - Insert(key): add key in O(lg n)
  - **Delete(key)**: remove key in O(lg n)
  - Height(node): get height of sub-tree rooted at node in O(1)

How do we do this?

Store some extra data at each node... but what?

#### Can we compute this function quickly?

• Function we want to compute: Height(u) = H(u)

u

 $H(u_R)$ 

 $U_R$ 

H(u)=?

u

H(u<sub>∟</sub>)

- If someone gives us H(u<sub>L</sub>) and H(u<sub>R</sub>), can we compute H(u)?
- What formula should we use?
- If u is a leaf then
  - -H(u) = 0
- Else

 $- H(u) = max{H(u_L), H(u_R)}+1$ 

#### Augmenting AVL tree to compute H(u)



## Algorithm idea:

• From the last slide, we develop an algorithm

#### Insert(key):

- 1 BST search for where to put key
- 2 Insert **key** into place like in a regular AVL tree
- 3 Fix balance factors and rotate as you would in AVL insert, but **fix heights at the same time**.

(Remember to fix heights all the way to the root. Don't stop before reaching the root!)

• (When you rotate, remember to fix heights of all nodes involved, just like you fix balance factors!)

#### Harder problem: scheduling conflicts

- Your calendar contains a bunch of time intervals [lo,hi] where you are busy
- We want to be able to quickly tell whether a new booking conflicts with an earlier booking.

[AmDann	Search Calendar		۹.				
Calendar	1 K > 0	day 2012			they m	on Month Theys Age	na mer Ör
CHARTE .	Sun .	Non	Tat.	Wed	714	Pri .	fut.
* May 2012 5 3 6 M T W T F 6 24 30 1 2 3 4 5 6 7 8 9 10 11 10 13 14 85 16 17 10 19	21 P	20 Sp triffe Tutude 8.35p Vital Heading 7.35p Vital	Mae 1 PAD Date Wat bil Bast 2009 Sekjar tagawifran	2 2209 Hillywaing	2 Buy Book App 0.0.7	4 3p investo linpart 5.33p Class Physic 5.33p Class Physic 33p Con Class Physic 3p Con Class Physic 3p Linp State Physic 4p Linp State Physic 4p North	S Copin Brethidae Voga Visucher Explores Cresso de Meyer 19 Licour Indoo 11:30 Voga 10 mare
20 21 22 23 24 25 24 27 28 29 38 34 1 2 3 4 5 6 7 0 0 • My calenders	8 Lahdooter Papanvah R38 Smarthat with Col R38 Smarthat with Col R39 Lanch with Fod, Alass	7 DONE Sand Hum Cant Hip URDERCESSION BHIP Direct Deput BHIP Direct Deput	PAD Cull Phane Bit 79 Distance Cose	8 29 Jane View Tech-Montajo 29 Gentlement Profer Ware	18 21% of multiDay Flyins ID margins Place Due 19 Bay M/TW Toolets 19 Onnari-oth Hitsten	11 Shen Bavers definend 3a Vinnes Expert 633p Onle Wynt 633p Onle Wynt	F2 ROB MCT Muterary or Organ ROB MCT Muterary or Organ Sp Strin fortisting See Date Sp Strin fortisting See Date Sp Strin fortisting See Date
Tapa	Og Lanch with Post, Alson	E The Drives Dan Proper	-4	4	By Dener with Hitslen		DUNELEDS AD DCA.
Other calendars	Nother's Day No Mathon's Disclosure	Holidas Check Geored Send Ded Cord	CuttCut	Care2/Field Date 79-Cares w/9-Earl	Abertulo Side Parking Sa PAD Natama Orie Due	Faceback PO Aprilation Prepat	Cod SD my
Diana Villiant	Ra Multive's Day Lonin	Reader Proceeding		The second distance	Partiel's Instance with Potent Cue By Cross and Leah Dener By Cross and Leah Dener	By Allian Going Away	12prillar Meeting 4p Hanging Outwith Klund, 4p Hanging Outwith Klund, ~2604
Diana Viliand Diana Viliand Phases of the Nove VS Haldwar Piesebur	19 Mather's Cherlands 20 Mather's Cherlands Road: Cherland Cherl Payment Road: Thank Cherla Due 16 Carlier or Capa 19 Ersteit with Engline, etc. 49 Mathericate	Continue Sure American Proceedians 21 Stant LAPTOP Mag Country Frait Dat 39 Dres w Data in Stracty 39 Dres w Data in Stracty	22 Ay Densi with Julie O	2019 Handlog och Will av 20 Carel Passe Das 29 Carel a Dave	Particle Survey a week Proces Oue Par Deep sectures Dense Particles Dense Particles Dense 24 29 LB mission Woor	<ul> <li>Attain Going Away</li> <li>25</li> <li>36 Meetin Propiet</li> </ul>	<ul> <li>Calvina Meeting</li> <li>Earlinging Cut-with Mand, </li></ul>

#### Breaking the problem down

- You must **design a data structure** D to efficiently do:
  - Insert(D; x): Insert interval x into D.
  - **Delete(D; x):** Delete interval x from D.
  - Search(D; x): If D contains an interval that overlaps with x, return *any* such interval.
     Otherwise, return null.

The hard part

All functions must run in O(lg n)

#### Figuring out the data structure - 1

- Iterative process; <u>HARD</u> to get right the first time!
- Need a way to insert intervals into the tree
   Use low end-point of interval as the key
- Example tree:



## Figuring out the data structure - 2

- What function do we want to compute?
  - Does an interval x intersect any interval in the tree?
- What info should we store at each node u?
  - Mhi(u) = Maximum high endpoint of any node in the subtree.
- How can we use the info stored at each node to compute the desired function (by looking at a <u>small</u> number of nodes)?
- Start by computing whether an interval x intersects any interval **in a subtree**.

Returns an interval in the subtree rooted at u that intersects [lo, hi]

if u is null then return null

Search(lo, hi, u):<

Returns an interval in the subtree rooted at u that intersects [lo, hi]

if u is null then return null

Search(lo, hi, u):

if [lo, hi] intersects [lo(u), hi(u)] then return [lo(u), hi(u)]

Returns an interval in the subtree rooted at u that intersects [lo, hi]

Search(lo, hi, u):

if [lo, hi] intersects [lo(u), hi(u)] then return [lo(u), hi(u)]

else (no intersection)

if lo < lo(u) return Search(lo, hi, left(u))</pre>



Returns an interval in the subtree rooted at u that intersects [lo, hi]

if u is null then return null

if [lo, hi] intersects [lo(u), hi(u)] then return [lo(u), hi(u)]

else (no intersection)

Search(lo, hi, u):

if lo < lo(u) return Search(lo, hi, left(u))</pre>

else (lo  $\geq$  lo(u))

if lo > Mhi(left(u)) then return Search(lo, hi, right(u))



#### Algorithm for Search within a subtree Returns an interval in the subtree rooted at u Search(lo, hi, u): that intersects [lo, hi] if u is null then return null if [lo, hi] intersects [lo(u), hi(u] u else (no intersection) lo(u) hi(u) if lo < lo(u) return Search(l else (lo $\geq$ lo(u)) lo hi V if lo > Mhi(left(u)) ther lo(v) hi(v) = Mhi(left(u)) else ( $lo \leq Mhi(left(u))$ return Search(lo, hi, left(u))

## Final algorithm for Search

#### Search(lo, hi, u):

if u is null then return null

if [lo, hi] intersects [lo(u), hi(u)] then return [lo(u), hi(u)]

else (no intersection)

if lo < lo(u) return Search(lo, hi, left(u))</pre>

else ( $lo \ge lo(u)$ )

if lo > Mhi(left(u)) then return Search(lo, hi, right(u))

else ( $lo \leq Mhi(left(u))$ 

return Search(lo, hi, left(u))

Search(D, x=[lo, hi]):

return Search(lo, hi, root(D))

## Algorithms for Insert and Delete

- Insert(D, x=[lo, hi]):
  - Do regular AVL insertion of key **lo**, also storing **hi**.
  - Set Mhi of the new node to **hi**.
  - Fix balance factors and perform rotations as usual, but also update Mhi(u) whenever you update the balance factor of a node u.
  - Update Mhi(u) for all ancestors, and for every node involved in a rotation, using formula: Mhi(u) = max{hi(u), Mhi(left(u)), Mhi(right(u))}.
- Delete(D, x=[lo, hi]): similar to Insert

# Why O(lg n) time?

- Insert/Delete: normal AVL operation = O(lg n)
  - PLUS: update Mhi(u) for each u on path to the root
    - Length of this path ≤ tree height, so O(lg n) in an AVL tree
  - PLUS: update Mhi(u) for each node involved in a rotation
    - At most O(lg n) rotations (one per node on the path from the root ≤ tree height)
    - Each rotation involves a constant number of nodes
    - Therefore, constant times O(lg n), which is O(lg n).
- Search
  - Constant work + recursive call on a child
  - Single recursive call means O(tree height) = O(lg n)