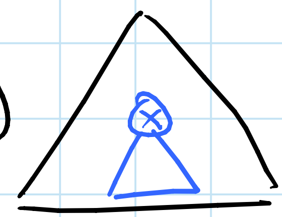
6.046	Lecture 9	Mar. 10, 2015

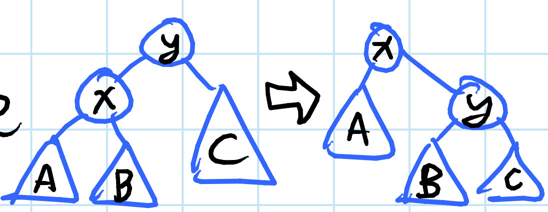
TODAY: Augmentation
- easy tree augmentation
- order-statistic trees
- finger search trees
- range trees

Idea: modify "off-the-shelf" data structure ↳ updates
to store additional information

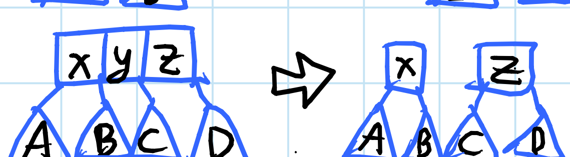
Easy tree augmentation:
- goal: store f(subtree rooted at x) at each node x in x.f



- suppose x.f can be computed → UPDATED in O(1) time from x, children, & children.f
- if modify set S of nodes (data, children) then costs O(# ancestors of nodes in S) to update f(x)'s (walk up from S)
- so O(lg n) updates in
  - AVL trees: e.g. rotate ⇒ update y then x



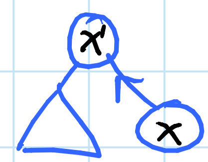
  - 2-3 trees: e.g. split ⇒ update x & z



(← also update up the tree)

Order-statistic trees: (from 6.006)

- ADT/interface: (Abstract Data Type)
  - insert(x) / delete(x) / successor(x)
  - rank(x): find x's index in sorted order (= # elements $< x$ if all distinct)
  - select(i): find element of rank i
- idea: use easy tree augmentation to store subtree size: f(subtree) = # nodes in it
  - $\Rightarrow$ x.size = 1 + sum(c.size for c in x.children)
- say, AVL trees $\Rightarrow$ binary (2-3 trees also work)
- rank(x):
  - rank = x.left.size + 1*
  - walk up to root from x
    - when go left ($x \to x'$): rank += x'.left.size + 1



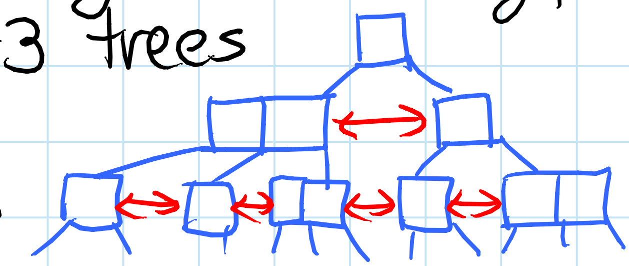
- select(i):
  - x = root
  - rank = x.left.size + 1*
  - if i = rank: return x
  - if i < rank: x = x.left
  - if i > rank: x = x.right
    i −= rank
  - repeat

*omit for indices starting at 0

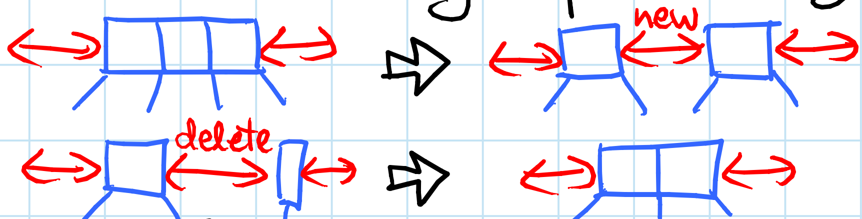
- e.g. can't maintain rank of each node: insert($-\infty$) would change all ranks

Finger search trees: [Brown & Tarjan 1980]

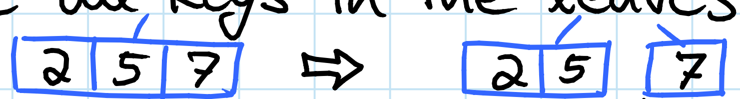
- goal: if already found y, search(x from y) should only take $O(\lg |\text{rank}(x) - \text{rank}(y)|)$
- idea: level-linked 2-3 trees
  - each node points to next & previous on same level



- maintain during split/merge:

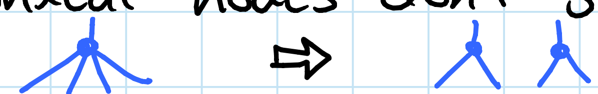


- store all keys in the leaves:



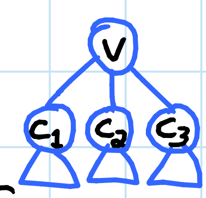
no promotion of 5

  - nonleaf nodes don't store keys



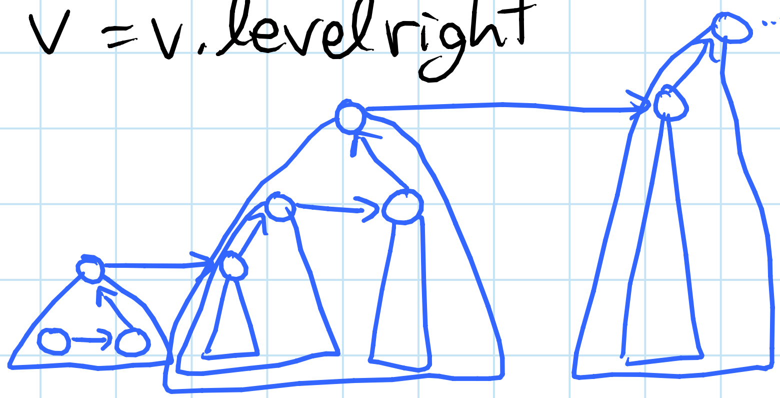
adds child to parent

- maintain min & max of each subtree (via easy tree augmentation)



$\Rightarrow$ can still do (top-down) search(x):
- say at vertex v with children c_1, c_2, c_3
- look at min & max of each child c_i
- if $c_i.\text{min} \le x \le c_i.\text{max}$: go down to c_i
- if $c_i.\text{max} < x < c_{i+1}.\text{min}$: return $c_i.\text{max}$ (predecessor) or $c_{i+1}.\text{min}$ (successor)

- search(x from y):
  - v = leaf containing y (given)
  - if $v.\text{min} \le x \le v.\text{max}$:
    do top-down search for x from v
    (i.e. within rooted subtree at v)
  - if $x < v.\text{min}$: $v = v.\text{levelleft}$
  - elif $x > v.\text{max}$: $v = v.\text{levelright}$
  - $v = v.\text{parent}$
  - repeat



Analysis:
- start at leaf level (height $\emptyset$)
- each round, go up 1 level

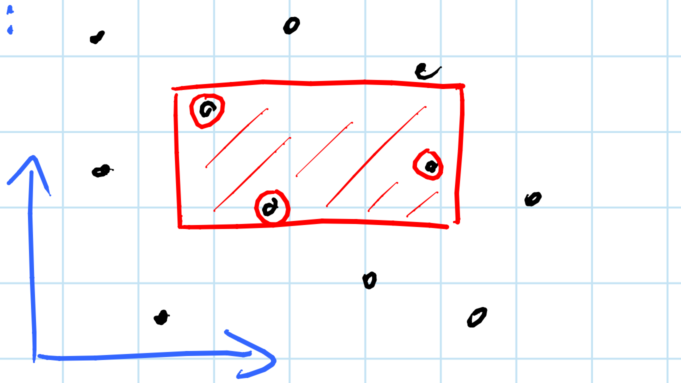
$\Rightarrow$ at step i, level link (height i) skips $\approx c^i$ keys/ranks, where $c \in [2, 3]$

$\Rightarrow$ if $|\text{rank}(x) - \text{rank}(y)| = k$
then reach x in $\Theta(\lg k)$ steps
(and downward search also $O(\lg k)$)

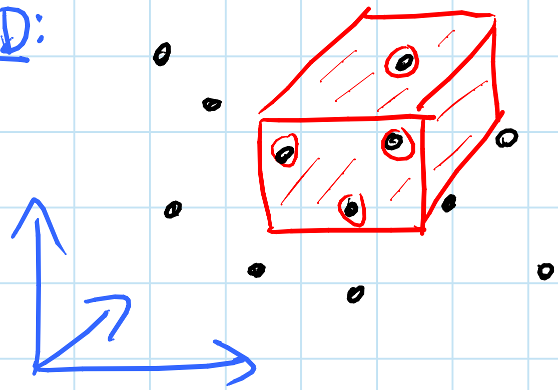
Orthogonal range searching:

preprocess n points in d dimensions
into a (static) data structure
supporting range query:
find k points in given axis-aligned box (rectangle in 2D)
OR count # points

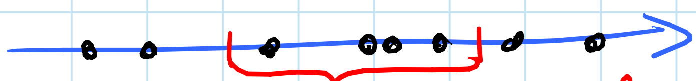
2D:



3D:



1D:

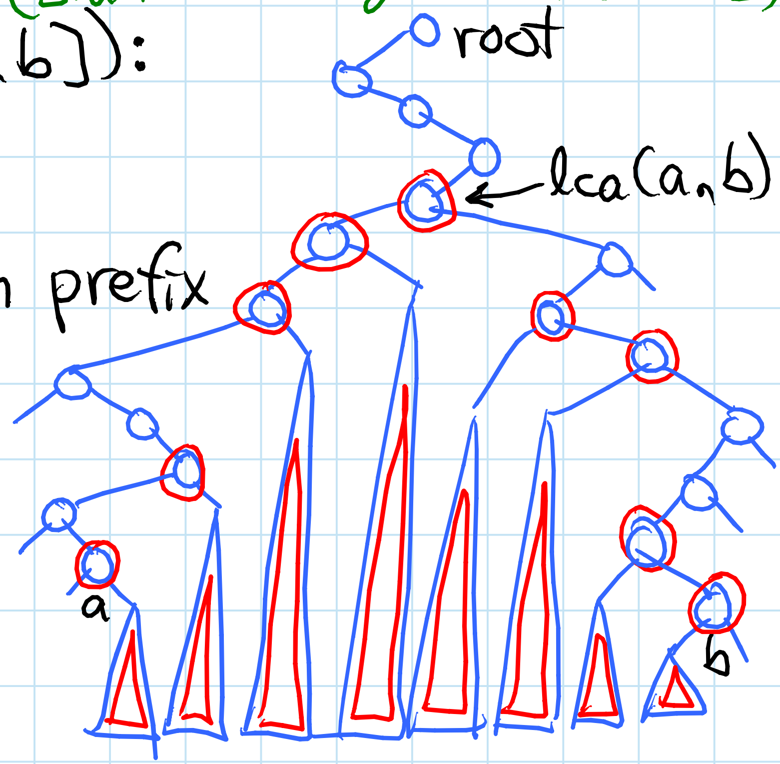


- sorted array: binary search, walk right
  $\Rightarrow O(\lg n + k)$ to report k
  (count in $O(\lg n)$ via 2 binary searches + subtract)

- finger search tree: (dynamic)
  search, finger search right by 1, ...
  $\Rightarrow O(\lg n + k)$ also
  (counting harder...)

1D range tree:

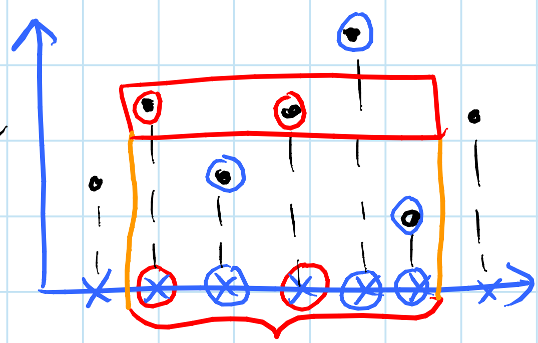
- complete BST (static ~ for dynamic, use AVL)
- range-query([a, b]):
  - search(a)
  - search(b)
  - trim common prefix
  - return $O(\lg n)$ nodes & subtrees "in between"



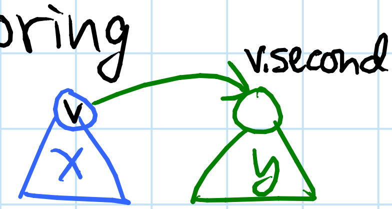
- $O(\lg n)$ to implicitly represent answer
- $O(\lg n + k)$ to traverse k outputs
- $O(\lg n)$ count via subtree size augmentation

2D range tree:

- primary 1D range tree keyed on x coordinate storing all points



- every node v in primary x-tree stores <u>secondary</u> 1D range tree, keyed on y coordinate, storing all points in v's subtree

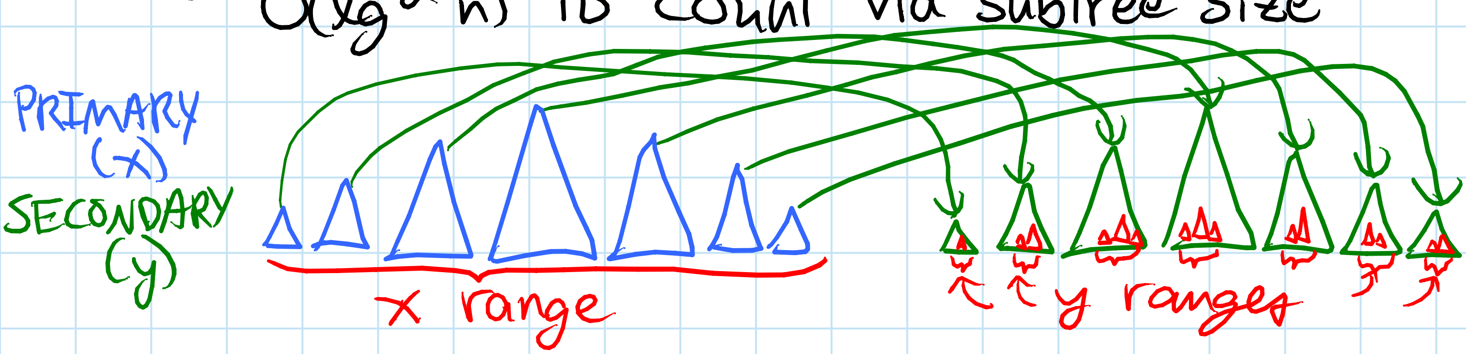


- range-search:
  - use primary x-tree to find points in correct x range (implicitly)
    - $O(\lg n)$ points: check manually
    - $O(\lg n)$ subtrees: for each v, use v's secondary y-tree to find points in correct y range (implicitly)

$\Rightarrow$ implicit representation as $O(\lg^2 n)$ nodes & subtrees (of secondary trees)

$\Rightarrow$ $O(\lg^2 n + k)$ to report k answers

- $O(\lg^2 n)$ to count via subtree size



Space: $O(n \lg n)$

- $O(n)$ for primary tree
- each point appears in $O(\lg n)$ secondary trees (one per ancestor)

OR: each level of primary tree stores all points in secondary trees

d-D range trees:

- recurse from primary → secondary → ...

Chazelle's improvement:

- query: $O(\lg^d n + k)$ — $O(\lg^{d-1} n + k)$
- space: $O(n \lg^{d-1} n)$ — $O\left(n\left(\frac{\lg n}{\lg \lg n}\right)^{d-1}\right)$

(see 6.851)