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

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

Easy tree augmentation:
- goal: store \( f(\text{subtree rooted at } x) \) at each node \( x \) in \( x.f \)
- suppose \( x.f \) can be computed in \( O(1) \) time from \( x \), children, & children.f
- if modify set \( S \) of nodes (data, children) then costs \( O(\#\text{ancestors of nodes in } S) \) to update \( f(x) \)’s (walk up from \( S \))
- so \( O(\lg n) \) updates in
  - AVL trees: e.g. rotate
    \( \Rightarrow \) update \( y \) then \( x \)
  - 2-3 trees: e.g. split
    \( \Rightarrow \) update \( x \) & \( z \)
(→ also update up the tree)
Order-statistic trees: (from 6.006)

- ADT/interface: (Abstract Data Type)
  - \texttt{insert}(x) / delete(x) / \texttt{successor}(x)
  - \texttt{rank}(x): find x's index in sorted order (= \# elements < x if all distinct)
  - \texttt{select}(i): find element of rank i

- Idea: use easy tree augmentation to store subtree size: $f(\text{subtree}) = \# \text{nodes in it}$
  \Rightarrow $x.\text{size} = 1 + \text{sum}(c.\text{size} \text{ for } c \text{ in } x.\text{children})$

- Say, AVL trees \Rightarrow binary (2-3 trees also work)

- \texttt{rank}(x):
  - \texttt{rank} = x.\text{left.\text{size}} + 1^*
  - Walk up to root from $x$
    - when go left ($x \rightarrow x'$):
      \texttt{rank} += $x'.\text{left.\text{size}} + 1$

- \texttt{select}(i):
  - $x = \text{root}$
  - \texttt{rank} = $x.\text{left.\text{size}} + 1^*$
  - If $i = \text{rank}$: return $x$
    - If $i < \text{rank}$: $x = x.\text{left}$
    - If $i > \text{rank}$: $x = x.\text{right}$
      \texttt{i} := \text{rank}
  - Repeat

- E.g. can't maintain \texttt{rank} of each node: \texttt{insert}(\text{-}\infty) would change all ranks
Finger search trees: [Brown & Tarjan 1980]
- **goal**: if already found y, search(x from y) should only take \(O(\log |\text{rank}(x) - \text{rank}(y)|)\)
- **idea**: level-linked 2-3 trees
  - each node points to next & previous on same level
- maintain during split/merge:
  - ![Diagram of split/merge]
- store all keys in the leaves:
  - ![Diagram of key distribution]
  - nonleaf nodes don't store keys
- maintain min & max of each subtree (via easy tree augmentation)

\[\Rightarrow \text{can still do (top-down) search}(x):\]
- say at vertex v with children \(c_1, c_2, c_3\)
- look at min & max of each child \(c_i\)
- if \(c_i, \text{min} \leq x \leq c_i, \text{max}\): 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):
  - \( v = \text{leaf containing } y \) \hspace{1em} (given)
  - if \( v.\min \leq x \leq v.\max \):
    - do top-down search for \( x \) from \( v \)
      - i.e. within rooted subtree at \( v \)
    - if \( x < v.\min \): \( v = v.\text{levelleft} \)
    - else if \( x > v.\max \): \( v = v.\text{levelright} \)
    - \( v = v.\parent \)
  - 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 \( O(\lg k) \) 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 points in given axis-aligned box (rectangle in 2D) OR count # points

2D:

3D:

1D: query = interval

- 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 secondary 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 $\rightarrow$ secondary $\rightarrow$ ...

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

Chazelle's improvement:
- $O(\lg^{d-1} n + k)$
- $O(n (\frac{\lg n}{\lg \lg n})^{d-1})$

(see 6.851)