Today: Geometry I (of 2)
- point location
  - static via persistence
  - dynamic via retroactivity
- orthogonal range searching
  - range trees
  - layered range trees
- dynamization with augmentation via weight balance
  - fractional cascading

Planar point location: given planar map, (planar) graph drawn in plane with straight edges & without crossings
  - support query: which face contains point \((x, y)\)?
    - e.g. which GUI element got clicked?
    - which city are these GPS coords. in?
  - static: preprocess map
  - dynamic: add/delete edges (& deg.-\(\emptyset\) vertices)

Vertical ray shooting: given planar map, support
  - query: which edge first hit by ray \(\uparrow(x, y)\)
    - implies (static) solution to point location:
      - maintain pointer from edge to face below it
    - also dynamic reduction with \(+O(lg n)\) overhead
Line sweep: technique traditionally used for line-segment intersection

- maintain order of intersection with vertical line which sweeps right
- left/right endpoints are inserts/deletes
- order swaps are crossings

- typically intersection DS = balanced BST
⇒ line-segment intersection in \(O(n \log n + k)\) output size

- if we use partially persistent balanced BST then successor\((y)\) query at time \(t\)
  \(=\) upward ray shooting query from \((t,y)\)
⇒ \(O(\log n)\) query after \(O(n \log n)\) preprocessing
  [Dobkin & Lipton—SIAM J. Comp. 1976]

(part of the initial motivation for persistence)
- if we use fully retroactive balanced BST
  then $\text{Insert/Delete}(t_1, \text{insert}(y))$
  $+$ $\text{Insert/Delete}(t_2, \text{delete}(y))$
  $= \text{insert/delete edge } (t_1, y) \rightarrow (t_2, y)$

$\Rightarrow O(lg n)$ dynamic vertical ray shooting among horizontal line segments

[Giora & Kaplan – T. Alg. 2009]
also: [Blelloch – SODA 2008] (later)

- also reduces back to retroactive successor

[OPEN]: $O(lg n)$ dynamic vertical ray shooting in general planar map?

- $O(lg n \log lg n)$ query & insert; $O(lg^2 n)$ delete
  [Baumgarten, Jung, Mehlhorn – J. Alg. 1994]
- $O(lg n)$ query, $O(lg^{1+\epsilon} n)$ insert, $O(lg^{2+\epsilon} n)$ delete
  [Arge, Brodal, Georgiadis – FOCS 2006]

[OPEN]: $O(lg n)$ static ray shooting (not vertical)

- $O\left(\frac{n}{\sqrt{s}} \text{ poly} lg n\right)$ query & $O(s^{1+\epsilon})$ space
  for any $1 \leq s \leq n$ [Agarwal – SICOMP 1992]
- conjectured nearly optimal
- 3D even harder e.g. [Agarwal & Sharir – SICOMP 1996]
- motivation: ray tracing
Orthogonal range searching:

- maintain n points in d dimensions subject to query: given box \([a_1, b_1] \times \cdots \times [a_d, b_d]\), report existence/count/k points in box
- static: preprocess points; dynamic: insert/delete
- motivation: query in database table with d cols.

Range trees: \(O(lg^d n + k)\) query

(see de Berg, Cheong, van Kreveld book)

[ Bentley - IPL 1979; Lee & Wong - TOS 1980; Lueker - FOCS 1978; Willard - TR 1979]

- 1D: balanced BST on leaves = points
  - internal node key = \(\text{max(left subtree)}\)
  - query([a,b]): search(a); search(b)

\(\text{BST}
\)

\(\text{LCA(pred(a), succ(b))}\)

pred(a) \(\rightarrow\) results in \(O(lg n)\) subtrees

\(\text{succ(b)}\)

- augment with subtree sizes to get count

(can also do this with regular BST but messier, especially to generalize)
- 2D: 1D tree on \( x \) + each subtree links to 1D tree on \( y \) on same points

- each point appears in \( \Theta(lg n) \) structures
  \[ \Rightarrow \Theta(n lg n) \text{ space} \]
- query \( ([a_1,b_1] \times [a_2,b_2]) \):
  - \( x \) query \( [a_1,b_1] \) \( \Rightarrow \Theta(lg n) \times \text{subtrees} \)
  - follow pointers \( \Rightarrow \Theta(lg n) \times \text{trees} \)
  - \( \Theta(lg n) \) \( y \) queries \( \Rightarrow \Theta(lg^2 n) \times \text{subtrees} \)
  \[ \Rightarrow \Theta(lg^2 n + k) \text{ time} \]
- augment \( y \) trees with subtree sizes for count

- \( d \)-D: recurse on \( d \)
  - \( O(lgd n) \) query
  - \( O(n lg^{d-1} n) \) space & preprocessing
  - \( O(lgd^2 n) \) update: recursively update each node along root-to-leaf path
Layered range tree: $O(lg^{d-1} n)$ query for $d > 1$

- 2D: search in $x$
  as before

- store $y$ structures as arrays (sorted by $y$)
- search once in root $y$ structure $\sim O(lg n)$
- carry those search results down to result subtree roots
- from one level down:
  store pointers to corresponding spots (successors)
  $\Rightarrow$ find start & end in $O(lg n)$ $y$ arrays
  in $O(1)$ per level, $O(lg n)$ overall
- can still compute counts & report $k$ points

- $dD$: same as before, just use 2D base case
- $O(n lg^{d-1} n)$ space & preprocessing
Dynamization with augmentation via weight balance

- **BB[x]** trees: [Nievergelt & Reingold - STOC 1972]
  - for each node \( x \):
    
    \[
    \text{size}(\text{left}(x)) \& \text{size}(\text{right}(x)) \text{ are } \geq \alpha \cdot \text{size}(x)
    \]
    \[\Rightarrow \text{height} \leq \log_{\frac{1}{\alpha}} n\]
  - when node is unbalanced, can afford to perfectly rebuild entire subtree of size \( k \):
    - charge to \( \Theta(k) \) of additive imbalance
    - update gets charged \( \Theta(lg n) \) times
    \[\Rightarrow O(lg n) \text{ amortized cost}\]
  - applied to range tree:
    - rebuild costs \( \Theta(k \ lg^{d-1} k) \)
    \[\Rightarrow O(lg^d n) \text{ amortized update}\]
    - update also updates \( O(lg n) \) \( y \)-trees \( \times \)
    \[O(lg n) \text{ } z\text{-trees } \times \ldots \times O(lg n) \text{ time } = O(lg^d n) \]
  - layered range trees: hard to update pointers;
    cost \( O(lg^d n) \) on average if array \( \rightarrow \) BST at root, linked list elsewhere [Willard - SICOMP 1985]
    average case inputs

Static improvement:

- can reduce space to \( O(\frac{n \ lg^{d-1} n}{lg lg n}) \) [Chazelle - SICOMP 1986]
  - for \( d \geq 3 \), can improve query to \( O(lg^{d-2} n) \)
  - \( O(n \ lg^d n) \) space via fractional cascading [Chazelle & Guibas - Alg. 1986 x2]
  - \( O(n \ lg^{d-1+\varepsilon} n) \) space [Alstrup, Brodal, Rauhe - FOCS 2000]
Fractional cascading: \([\text{Chazelle \& Guibas - Alg. 1986} \times 2]\)

dynamic: \([\text{Mehlhorn \& N"aher - Alg. 1990}]\)

Warmup: predecessor/ successor search for \(x\) among \(k\) lists each of length \(n\)

- \(O(k \lg n)\) trivial \((k\) binary searches\)
- \(O(k + \lg n)\) solution:
  - let \(L_i' = L_i + \text{every other element of } L_{i+1}\)
  \(\Rightarrow |L_i'| = n + \frac{1}{2} |L_{i+1}| = O(n)\) \(\text{(geometric)}\)
  - link between identical elements in \(L_i\) \& \(L_{i+1}\)
  - each element in \(L_i\) stores pointer to previous/next element in \(L_i' - L_i\)
  - each element in \(L_i' - L_i\) stores pointer to previous/next element in \(L_i\)

- search(\(x\)):
  - binary search in \(L_1\) \(\Rightarrow O(\lg n)\)
  \(\Rightarrow\) if amid \(L_1' - L_1\), follow pointers to neighbors in \(L_1\) to solve \(L_1\) problem
  \(\Rightarrow\) if amid \(L_1\), follow pointers to neighbors in \(L_1' - L_1\) \(\text{(else stay)}\)
  - walk down to \(L_2\)
  - repeat
General: graph where each
- vertex contains set of elements
- edge labeled with range \([a, b]\)
- \textbf{locally bounded degree}: \# incoming edges whose labels \(\geq x\) is \(\leq c\).
- \textbf{search}(x) wants to find \(x\) in \(k\) vertices' sets
  found by navigating (online) from any vertex,
  along edges whose labels \(\geq x\)
- improve \(O(k \log n)\) to \(O(k + \log n)\)

\textbf{idea}: same as warmup
  new: cycles in graph
  but very few items go around cycles