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 \(+ O(\log n)\) 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
  $\Rightarrow$ 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 $(t, y)$

$\Rightarrow O(\log n)$ query after $O(n \log n)$ preprocessing

[Dobkin & Lipton - SICOMP 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?
- \( O(\lg n \lg \lg n) \) 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)
- \( O\left(\frac{n}{\sqrt{s}}\right) \) polylog \( n \) query & \( O(s^{1+\epsilon}) \) space
  [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:** $O(\log^d n + k)$ query
(see de Berg, Cheong, van Kreveld book)

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

```
results in $O(\log n)$ subtrees
```

```
\[ \text{lca}(\text{pred}(a), \text{succ}(b)) \]
```

- 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
  $\Rightarrow O(n lg n)$ space
- query $[[a_1,b_1] \times [a_2,b_2]]$:
  - $x$ query $[[a_1,b_1]] \Rightarrow O(lg n) \times$ subtrees
  - follow pointers $\Rightarrow O(lg n)$ y trees
  - $O(lg n)$ y queries $\Rightarrow O(lg^2 n)$ y subtrees
  $\Rightarrow O(lg^2 n + k)$ time
- augment y trees with subtree sizes for count

- dD: recurse on d
  - $O(lg^d n)$ query
  - $O(n lg^{d-1} n)$ space & preprocessing
  - $O(lg^d n)$ 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)
  - 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[\alpha] trees: [Nievergelt & Reingold - STOC 1972]
  - for each node x:
    size(left(x)) & size(right(x)) are \geq \alpha \cdot size(x)
  \implies 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
  \implies O(lg n) amortized cost
- applied to range tree: [idea in Lueker - FOCS 1979; Willard - SICOMP 1985]
  - rebuild costs \Theta(k lg^{d-1} k)
  \implies O(lg^d n) amortized update
  - update also updates O(lg n) y-trees x O(lg n) z-trees \times \ldots \times O(lg n) 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 \( O\left(\frac{n \cdot lg^{d-1} n}{lg lg n}\right) \) [Chazelle - SICOMP 1986]
- for \( d \geq 3 \), can improve query to \( O(lg^{d-2} n) \)
- \( O(n \cdot lg^{d} n) \) space via fractional cascading [Chazelle & Guibas - Alg. 1986 \times 2]
- \( O(n \cdot lg^{d-1+\varepsilon} n) \) space [Alstrup, Brodal, Rauhe - FOCS 2000]
Fractional cascading: [Chatelie & Guibas - Alg. 1986 x 2]
Dynamic: [Mehlhorn & Naher - Alg. 1990]

Warmup: predecessor/successor search for x "1.5D" among k lists each of length n
- O(k lg n) trivial (k binary searches)
- O(k + lg n) solution:
  - Let \( L'_i = L_i \) + every other element of \( L'_{i+1} \)
  \( \Rightarrow |L'_i| = n + \frac{1}{2} |L'_{i+1}| = O(n) \) (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
  - If amid \( L_1 \), follow pointers to neighbors in \( L_1 - L_1 \) (else stay)
  - Walk down to \( L_2 \)
  - Repeat
General: graph where each
- vertex contains set of elements
- edge labeled with range \([a,b]\)
- \underline{locally bounded degree}: \# incoming edges whose labels \(\geq x\) is \(\leq c\).
- \text{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 \(O(k + \log n)\)

idea: same as warmup
new: cycles in graph
but very few items go around cycles