Randomized incremental construction

Special sampling idea:

- Sample all *except* one item
- hope final addition makes small or no change

Method:

- process items in order
- average case analysis
- randomize order to achieve average case
- e.g. binary tree for sorting

Backwards analysis

- compute expected time to insert $S_{i-1} \to S_i$
- backwards: time to delete $S_i \to S_{i-1}$
- conditions on S_i
- but generally analysis doesn't care what S_i is.

Trapezoidal decomposition:

Motivation:

- manipulate/analayze a collection of n segments
- assume no degeneracy: endpoints distinct
- (simulate touch by slight crossover)
- e.g. detect segment intersections
- e.g., point location data structure
- Basic idea:
  - Draw verticals at all points and intersects
  - Divides space into slabs
  - binary search on x coordinate for slab
  - binary search on y coordinate inside slab (feasible since lines noncrossing)
  - problem: $\Theta(n^2)$ space

Definition.

- draw altitudes from each endpoints and intersection till hit a segment.
- trapezoid graph is *planar* (no crossing edges)
- each trapezoid is a *face*
- show a face.
- one face may have many vertices (from altitudes that hit the *outside* of the face)
- but max vertex degree is 6 (assuming nondegeneracy)
- so total space $O(n + k)$ for k intersections.
- number of faces also $O(n + k)$ (at least one edge/face, at most 2 face/edge)
- (or use Euler's theorem: $n_v - n_e + n_f \geq 2$)
- standard clockwise pointer representation lets you walk around a face

Randomized incremental construction:

- to insert segment, start at left endpoint
- draw altitudes from left end (splits a trapezoid)
- traverse segment to right endpoint, adding altitudes whenever intersect
- traverse again, erasing (half of) altitudes cut by segment

Implementation

- clockwise ordering of neighbors allows traversal of a face in time proportional to number of vertices
- for each face, keep a (bidirectional) pointer to all not-yet-inserted left-endpoints in face
- to insert line, start at face containing left endpoint
- traverse face to see where leave it
- create intersection,
  - update face (new altitude splits in half)
  - update left-end pointers
- segment cuts some altititudes: destroy half
  - removing altitude merges faces
  - update left-end pointers
  - (note nonmonotonic growth of data structure)

Analysis:

- Overall, update left-end-pointers in faces neighboring new line
- time to insert s is

$$\sum_{f \in F(s)} (n(f) + \ell(f))$$

  where

  - $F(s)$ is faces s bounds after insertion
  - $n(f)$ is number of vertices on face f boundary
  - $\ell(f)$ is number of left-ends inside f.

- So if S_i is first i segments inserted, expected work of insertion i is

$$\frac{1}{i} \sum_{s \in S_i} \sum_{f \in F(s)} (n(f) + \ell(f))$$

- Note each f appears at most 4 times in sum since at most 4 lines define each trapezoid.
- so $O(\frac{1}{i} \sum_f (n(f) + \ell(f)))$.
- Bound endpoint contribution:
  - note $\sum_f \ell(f) = n - i$
  - so contributes n/i
  - so total $O(n \log n)$ (tight to sorting lower bound)
- Bound intersection contribution
  - $\sum n(f)$ is just number of vertices in planar graph
  - So $O(k_i + i)$ if k_i intersections between segments so far
  - so cost is $E[k_i]$
  - intersection present if both segments in first i insertions
  - so expected cost is $O((i^2/n^2)k)$
  - so cost contribution $(i/n^2)k$
  - sum over i, get $O(k)$
  - **note:** adding to RIC, assumption that first i items are random.
- Total: $O(n \log n + k)$

Search structure

Starting idea:

- extend all vertical lines infinitely
- divides space into slabs
- binary search to find place in slab
- binary search in slab feasible since lines in slab have total order
- $O(\log n)$ search time

Goal: apply binary search in slabs, without n^2 space

- Idea: trapezoidal decom is "important" part of vertical lines
- problem: slab search no longer well defined
- but we show ok

The structure:

- A kind of search tree
- "x nodes" test against an altitude
- "y nodes" test against a segment
- leaves are trapezoids
- each node has two children
- **But** may have many parents

Inserting an edge contained in a trapezoid

- update trapezoids
- build a 4-node subtree to replace leaf

Inserting an edge that crosses trapezoids

- sequence of traps Δ_i
- Say Δ_0 has left endpoint, replace leaf with x-node for left endpoint and y-node for new segment
- Same for last Δ
- middle Δ:
  - each got a piece cut off

  - cut off piece got merged to adjacent trapezoid
  - Replace each leaf with a y node for new segment
  - two children point to appropriate traps
  - merged trap will have several parents—one from each premerge trap.

Search time analysis

- depth increases by one for new trapezoids
- RIC argument shows depth $O(\log n)$
  - Fix search point q, build data structure
  - Length of search path increased on insertion only if trapezoid containing q changes
  - Odds of top or bottom edge vanishing (backwards analysis) are $1/i$
  - Left side vanishes iff **unique** segment defines that side and it vanishes
  - So prob. $1/i$
  - Total $O(1/i)$ for i^{th} insert, so $O(\log n)$ overall.

Treaps

Dictionaries for **ordered** sets

- New Operations.
  - enumerate in order
  - successor-of, predecessor-of (even if not in set)
  - join(S, k, T), split, paste(S, T)

Binary tree.

- child and parent pointers
- endogenous: leaf nodes empty.
- *balanced* if depth $O(\log n)$
- average case.
- worst case

Tree balancing

- rotations (show)
- implementing operations.
- red/black, AVL

- splay trees.
  - drawbacks in geometry:
  - auxiliary structure on nodes in subtree (eg, for remaining dimensions)
  - rebuild on rotation

Returning to average case:

- Assign random "arrival orders" to keys
- Build tree **as if** arrived in that order
- Average case applies
- No rotations on searches

Choosing priorities

- define arrival by random priorities
- assume continuous distribution, fix.
- eg, use $2 \log n$ bits, w.h.p. no collisions

Treaps.

- tree has keys in heap order of priorities
- unique tree given priorities—follows from insertion order
- implement insert/delete etc.
- rotations to maintain heap property

Depth $d(x)$ analysis

- Tree is trace of a quicksort
- We proved $O(\log n)$ w.h.p.

lemma: for x rank k, $E[d(x)] = H_k + H_{n-k+1} - 1$

- $S^- = \{y \in S \mid y \leq x\}$
- Q_x = ancestors of x
- Show $E[Q_x^-] = H_k$.
- to show: $y \in Q_x^-$ iff inserted before all z, $y < z \leq x$.
- deduce: item j away has prob $1/j$. Add.
- Suppose $y \in Q_x^-$.

  - Then inserted before x
  - Suppose some z between inserted before y
  - Then y in left subtree of z, x in right, so not ancestor
  - Thus, y before every z
- Suppose y first
  - then x follows y on all comparisons (no z splits
  - So ends up in subtree of y

Rotation analysis

- Insert/Delete time
  - define spines
  - equal left spine of right sub plus right spine of left sub
  - proof: when rotate up, one spine increments, other stays fixed.
- R_x length of right spine of left subtree
- $E[R_x] = 1 - 1/k$ if rank k
- To show: $y \in R_x$ iff
  - inserted after x
  - but before all z, $y < z < x$
  - sinceif z before y, then y goes left, so not on spine
- deduce: if r elts between, $r!$ of $(r+2)!$ permutations work.
- So probability $1/(r+1)(r+2)$.
- Expectation $\sum 1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots = 1 - 1/k$
- subtle: do analysis only on elements inserted in real-time before x, but now assume they arrive in random order in virtual priorities.