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.

Convex Hulls

Define

- assume no 3 points on straight line.
- output:
  - points and edges on hull
  - in counterclockwise order
  - can leave out edges by hacking implementation

$\Omega(n \log n)$ lower bound via sorting
algorithm (RIC):

- random order p_i
- insert one at a time (to get S_i)
- update $conv(S_{i-1}) \to conv(S_i)$
  - new point stretches convex hull
  - remove new non-hull points

  - revise hull structure

Data structure:

- point p_0 inside hull (how find?)
- for each p, edge of $conv(S_i)$ hit by $\vec{p_0 p}$
- say p *cuts* this edge
- To update p_i in $conv(S_{i-1})$:
  - if p_i inside, discard
  - delete new non hull vertices and edges
  - 2 vertices v_1, v_2 of $conv(S_{i-1})$ become p_i-neighbors
  - other vertices unchanged.
- To implement:
  - detect changes by moving out from edge cut by $\vec{p_0 p}$.
  - for each hull edge deleted, must update cut-pointers to $\vec{p_i v_1}$ or $\vec{p_i v_2}$

Runtime analysis

- deletion cost of edges:
  - charge to creation cost
  - 2 edges created per step
  - total work $O(n)$
- pointer update cost
  - proportional to number of pointers crossing a deleted cut edge
  - BACKWARDS analysis
    * run backwards
    * delete random point of S_i (**not** $conv(S_i)$) to get S_{i-1}
    * same number of pointers updated
    * expected number $O(n/i)$
      · what Pr[update p]?
      · Pr[delete cut edge of p]
      · Pr[delete endpoint edge of p]
      · $2/i$
    * deduce $O(n \log n)$ runtime
- Book studies 3d convex hull using same idea, time $O(n \log n)$, also gets voronoi diagram and Delauney triangulations.

Trapezoidal decomposition:

Motivation:

- manipulate/analayze a collection of *segments*
- e.g. detect segment intersections
- e.g., point location data structure
  - Draw verticals at all points
  - binary search for slab
  - binary search inside slab
  - problem: $O(n^2)$ space

Definition.

- draw altitudes from each 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)
- max vertex degree is 6 (assuming nondegeneracy)
- so total space $O(n + k)$ for k intersections.
- number of faces also $O(n + k)$ (each face needs one 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

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 in face f
  - $\ell(f)$ is number of left-ends in f.
- So if S_i is first i segmenets 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
- so $O(\frac{1}{i} \sum_f (n(f) + \ell(f)))$.
- Bound endpoint contribution:
  - note $\sum l(f) = n - i$
  - so contributes n/i
  - so total $O(n \log n)$
- Bound intersection contribution
  - $\sum n(f)$ is $O(k_i + i)$ if k_i intersections

- 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: $O(n \log n + k)$