day5 1/26/98 Quote of the day: "Just leave it out."

1 Linear Programming

The constraints can be divided into 3 sets:

- $H^+ = \{i : y \geq \alpha_i x + \beta_i\}$
- $H^- = \{i : y \leq \alpha_i x + \beta_i\}$
- $H^0 = \{i : \alpha_i x = \beta_i\}$ (vertical line constraints)

In general only 2 of the constraints in H^0 can be active at one time.

Define:

- $l_t^+ \in H^+$ with largest intersection on line $x = t$
- $l_t^- \in H^-$ with smallest intersection on line $x = t$

By looking at the slopes of l_t^+ and l_t^- you can decide whether the solution must lie to the left or to the right.

1.1 Algorithm

This algorithm is dual to the bridge finding algorithm in the convex hull problem. Start with a window(a,b) in which the solution must lie.

1. Pair up lines in H^+, Pair up lines in H^-.
2. compute $\approx n/2$ intersections
3. pick t = median x value from the intersections. Let l_t^- = the smallest y-value which any solution must lie below and l_t^+ = the largest y value which any solution must lie above.
4. case 1: $l_t^- > l_t^+$ (there exists a feasible region) if slope(l_t^+)=0 then you are done, else if slope(l_t^+)>0, pick window(a,t) else pick window(t,b)
5. case 2: $l_t^- < l_t^+$ if $slope(l_t^+) - slope(l_t^-) > 0$ then pick window(t,b) else pick window(a,t).
6. case 3: :$l_t^- = l_t^+$ degenerate nastiness.
7. For each pair with an intersection outside the new window discard a constraint.
8. repeat with new window and remaining constraints.

1.2 Timing:

Timing is the same as the bridge algorithm.
$T(n) = T(3/4 * n) + c * n$

2 Amortized analysis

2.1 Binary counter

How many bits change on average?

- Let C_i = counter at time i. C_0= all zeros
- potential function: $\#(C_i) = \#$ of ones in counter.
- $\widehat{cost}_i = cost_i + \#(C_{i+1}) - \#(C_i)$ = amortized cost

To figure out the total cost, do $(\sum \widehat{cost}_i + \#C_{begin} - \#C_{end})/n$ = amortized cost. The amortized cost for a the binary counter is 2. One way to see this is to notice that when the counter is updated there is a $p = 1$ chance of 1 bit changing, a $p = 1/2$ chance of another bit changing, a $p = 1/4$ chance of another bit changing.... the series sums to 2.

2.2 Splay trees

2.2.1 binary search trees

Common to all ordered trees

- each of the nodes stores at least one key.
- keys are "inorder"
- member(i,S)
- insert(i,S)
- delete(i,S)
- join(S,S')
- split(i,S)

examples

- AVL-trees (everything with log(n) or log(n)+1 trees)
- 2-3-4 trees (similar)
- red-black trees (depth varies between log(n) and 2*log(n))

- splay trees (move interesting elements to the root of the tree)
- random trees

The important method of transforming a binary tree is rotation of a child to a parent. Splay trees rotate any element looked at to the top of the tree.