Discuss midterm.
Discuss project:

- four goals: read, implement, test, and exposite.
- was described as 2 projects, but can be met in 1.
- choosing algorithm:
  - a nontrivial algorithm or data structure: Fib heap, bucket heap, VEB heap, splay tree, suffix tree, max-flow push/relabel, unusual shortest path, etc
  - source: advanced textbooks, or FOCS/STOC/SODA
  - maybe inherent interest, maybe useful in an application you are working on.
- read:
  - some article from a proceedings or journal, not yet digested in testbook.
  - goal: demonstrate you understood it (by explaining well).
- implement/test:
  - should not be too major to implement (don't write whole interior point algorithm!)
  - design/justify interesting inputs, or
  - use inputs from a real motivating application (still need to "design", but easier)
  - need a "control" experiment—dumb or prior implementation
  - testing should suggest changes. explain motivation and effect.
  - testing should suggest new inputs. explain motivation and effect.
  - perfectly all right for new version to work worse. But have to explain why.
- exposition:
  - clear explanation of algorith.
  - must demonstrate you understand it well
  - needn't include proofs in detail, but should give idea of "why works"
- meeting the goals: can either implement a new algorithm, or implement an old one but write about a new one. need to know can read journal article.

# 1 Interior Point

Ellipsoid has problems in practice ($O(n^6)$ for one). So people developed a different approach that has been extremely successful.
What goes wrong with simplex?

- follows edges of polytope
- complex stucture there, run into walls, etc
- interior point algorithms stay away from the walls, where structure simpler.
- Karmarkar did the first one (1984); we'll descuss one by Ye

## 1.1 Potential Reduction

Potential function:

- Idea: use a (nonlinear) potential function that is minimized at opt but also enforces feasibility
- use gradient descent to optimize the potential function.
- Recall standard primal $\{Ax = b, x \geq 0\}$ and dual $yA + s = c, s \geq 0$.
- duality gap $sx$
- Use *logarithmic barrier function*

  $$G(x, s) = q \ln xs - \sum \ln x_j - \sum \ln s_j$$

  and try to minimize it (pick $q$ in a minute)
- first term forces duality gap to get small
- second and third enforce positivity
- note barrier prevents from ever hitting optimum, but as discussed above ok to just get close.

Choose $q$ so first term dominates, guarantees good $G$ is good $xs$

- $G(x, s)$ small should mean $xs$ small
- $xs$ large should mean $G(x, s)$ large
- write $G = \ln(xs)^q / \prod x_j s_j$
- $xs > x_j s_j$, so $(xs)^n > \prod x_j s_j$. So taking $q > n$ makes top term dominate, $G > \ln xs$

How minimize potential function? Gradient descent.

- have current $(x, s)$ point.
- take linear approx to potential function around $(x, s)$
- move to where linear approx smaller $(-\nabla_x G)$
- deduce potential also went down.
- crucial: can only move as far as linear approximation accurate

Firs wants big $q$, second small $q$. Compromise at $n + \sqrt{n}$, gives $O(L\sqrt{n})$ iterations.
Must stay feasible:

- Have gradient $g = \nabla_x G$
- since potential not minimized, have reasonably large gradient, so a small step will improve potential a lot. **picture**
- want to move in direction of $G$, but want to stay feasilbe
- project $G$ onto nullspace$(A)$ to get $d$
- then $A(x + d) = Ax = b$
- also, for sufficiently small step, $x \geq 0$
- potential reduction proportional to length of $d$
- problem if $d$ too small
- In that case, move $s$ (actually $y$) by $g - d$ which will be big.
- so can either take big primal or big dual step

### 1.2 Path Following

Potential function:

- Define
$$P(\mu) = cx - \mu \sum \log x_i$$
- minimize over $Ax = b$
- When $\mu$ is tiny, barrier is negligible except right at edge of polytope
- so optimum is right near LP opt, just pushed away from boundary a bit.
- For each $\mu$, some optimum $x(\mu)$
- $\lim_{\mu \to 0} P(\mu)$ is LP opt.
- $P(\mu)$ as $\mu$ varies defines a function: *central path*

- starts where $\mu = \infty$, *analytic center* farthest from all boundaries.

Path following algorithm:

- repeatedly optimizes $P(\mu)$ for smaller and smaller $\mu$
- when $\mu$ small enough, round to (optimal) vertex
- need to start somewhere near central path—revise problem to make this easy.

Path following step:

- suppose have $x'(\mu)$ near $x(\mu)$
- want $x'(\overline{\mu}\mu)$ near $x(\overline{\mu})$ for $\overline{\mu} = (1 - \beta)\mu$
- take a (second order) taylor expansion of $P(\overline{\mu})$ near $x'(\mu)$
- since $x'(\mu)$ near $x(\overline{\mu})$, Taylor "accurate" (need $\beta \approx 1/\sqrt{n}$)
- take a "Newton step" towards minimizing $P(\overline{\mu})$
- takes us closer to $x(\overline{\mu})$
- update $\overline{\mu}$ and repear
- like potential method, $O(\sqrt{n}L)$ iterations.
- in practice, 9 iterations halve potential!

## 2 Online algorithms

Motivation:

- till now, our algorithms start with input, work with it
- (exception: data structures—come back later)
- now, suppose input arrives a little at a time, need instant response
- eg stock market, paging
- question: what is a "good" algorithm.
- depends on what we measure.
- if knew whole input $\sigma$ in advance, easy to optimize $C_{MIN}(\sigma)$
- ski rental problem: rent 1, buy $T$. don't know how often.
- notice that on some inputs, can't do well! (stock market that only goes down, thrashing in paging)

- problem isn't to decide fast, rather what to decide.

Definition: competitive ratio

- compare to full knowledge optimum
- $k$-competitive if for all sequences etc. $C_A(\sigma) \leq kC_{MIN}(\sigma)$
- sometimes, to ignore edge effects, $C_A(\sigma) \leq kC_{MIN}(\sigma) + O(1)$.
- idea: "regret ratio"
- analyze ski rental
- we think of competitve analysis as a (zero sum) game between algorithm and adversary. want to find best strategy for algorithm.
- supposed to be competitive against all sequences. So, can imagine that adversary is adapting to algorithm's choices (to get worst sequence)

Paging problem

- define
- LRU, FIFO, LIFO, Flush when full, Least freq use
- LIFO, LFU not competititive
- LRU, FIFO $k$-competitive.
- will see this is best possible (det)

LRU is $k$-competitive

- note we prove this without knowing opt!
- assume start with same pages in memory (adds const)
- phase: $k$ page faults, ending with last fault (start counting after first fault)
- show 1 fault to MIN in each phase
- case 1: two faults on $p$ in 1 phase
  - then had accesses to $k$ other pages between faults to $p$
  - so $k+1$ pages accessed in phase—MIN must fault once.
- case 2: $k$distinct faults
  - let $p$ be last fault of previous phase
  - case 2a: fault to $p$ in phase. Then argue as before, $k$ pages between $p$ faults

- case 2b: no fault to $p$. immediately after first $p$-fault, MIN has $p$ in memory, other $k-1$ pages. $k$ new pages accessed in phase. Deduce one faults MIN.

- Notice: in case 2, fault we charge to phase might happen before phase.
  - but, happens after last fault-for-LRU in previous phase
  - so is different fault than the one deduced for previous phase.

Observations:

- proved without knowing optimum
- instead, derived *lower bound* on cost of *any* algorithm
- same argument applies to FIFO.

Lower bound: no online algorithm beats $k$-competitive.

- set of $k+1$ pages
- always ask for the one $A$ doesn't have
- faults every time.
- so, just need to show can get away with 1 fault every $k$ steps
- have $k$ pages, in memory. When fault, look ahead, one of $k+1$ isn't used in next $k$, so evict it.
- one fault every $k$ steps
- so $A$ is only $k$-competitive.

Observations:

- lb can be proven without knowing OPT, often is.
- competitive analysis doesn't distinguish LRU and FIFO, even though know different in practice.
- still trying to refine competitive analysis to measure better: new SODA paper: "LRU is better than FIFO"
- applies even if just have $k+1$ pages!

Optimal offline algorithm: Longest Forward Distance

- evict page that will be asked for farthest in future.
- suppose MIN is better than LFD. Will make NEW, as good, agrees more with LFD.
- Let $\sigma_i$ be first divergence of MIN and LFD (at page fault)

- LFD discards $q$, MIN discards $p$ (so $p$ will be accessed before $q$ after time $i$)
- Let $t$ be time MIN discards $q$
- revise schedule so MIN and LFD agree up to $t$, yielding NEW
- NEW discards $q$ at $i$, like LFD
- so MIN and NEW share $k-1$ pages. will preserve till merge
- in fact, $q$ is unique page that MIN has that new doesn't
- case 1: $\sigma_i, \ldots, \sigma_t, \ldots, p, \ldots, q$
  - until reach $q$
  - let $e$ be unique page NEW has that MIN doesn't (init $e = p$)
  - when get $\sigma_l \neq e$, evict same page from both
  - note $\sigma_l \neq q$, so MIN does fault when NEW does
  - both fault, and preserves invariant
  - when $\sigma_l = e$, only MIN faults
  - when get to $q$, both fault, but NEW evicts $e$ and converges to MIN.
  - clearly, NEW no worse than MIN
- case 2: $t$ after $q$
  - follow same approach as above till hit $q$
  - since MIN didn't discard $q$ yet, it doesn't fault at $q$, but
  - since $p$ requested before $q$, had $\sigma_l = e$ at least once, so MIN did *worse* than NEW. (MIN doesn't have $p$ till faults)
  - so, fault for NEW already paid for
  - still same.
- prove that can get to LFD without getting worse.
- so LFD is optimal.

## 2.1 Randomized Online Algorithms

We've seen online as a game between adversary input and algorithm.
Well known that optimum strategies require *randomization*.

- alg has random bits, makes random decisions.
- effect: random choice from among det algorithms.
- for given sequence, get expected performance, compare to opt

- find worst over all inputs for competitive ratio.
- oblivious adversary: doesn't know alg choices (equive: must choose seq in advance)
- adaptive: knows coin tosses up to $n$ before making move $n$
- seems like det, but not: can't back $A$ into a corner.
- will see that with randomization, can get $\log k$ competitive.
- marking algorithm