CSC3160: Design and Analysis of Algorithms

Week 4: Randomized Algorithms

Instructor: Shengyu Zhang

Randomized Algorithms

- We use randomness in our algorithms.
- You've seen examples in previous courses
  - quick sort: pick a random pivot.

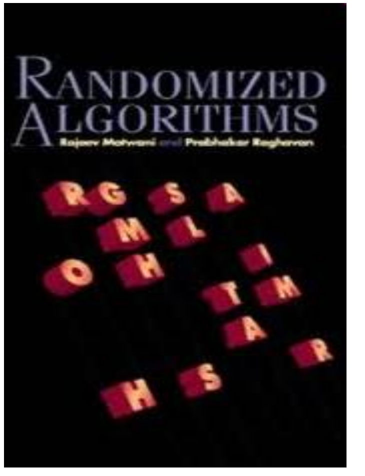
- We'll see more in this week.

Motivation

- Why randomness?
  - Faster.
  - Simpler.
- Price: a nonzero error probability
  - Usually can be controlled to arbitrarily small.
  - Repeating k times drops the error probability to c^{-k} for some constant $c > 1$.
    - Second part of the lecture.

General references

- Randomized Algorithms,
  Rajeev Motwani and Prabhakar Raghavan, *Cambridge University Press*, 1995.



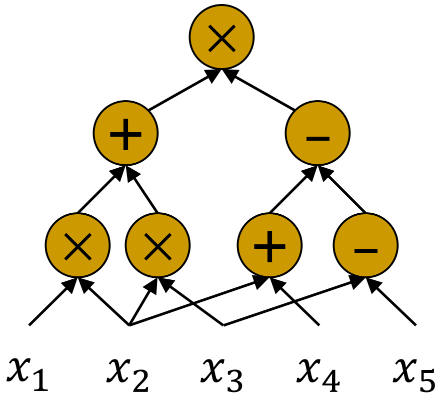
- Probability and Computing,
  Michael Mitzenmacher and Eli Upfal,
  Cambridge University Press,
  2005.

Part 1: Examples

Example 1: Polynomial Identity Testing

Question

- Given two polynomials p_1 and p_2 (by arithmetic circuit), decide whether they are equal.
- Arithmetic circuit:

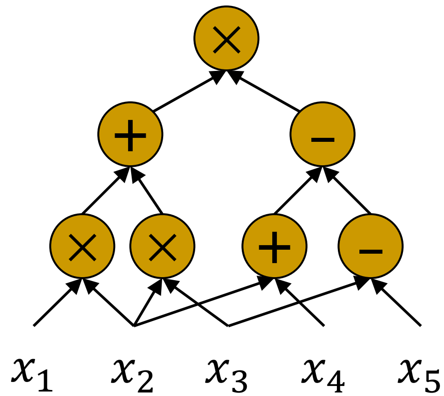


polynomial computed:

$(x_1x_2 + x_2x_3)((x_2 + x_4) - (x_3 - x_5))$

- Question: Given two such circuits, do they compute the same polynomial?

Naïve algorithm?



polynomial computed:

$(x_1x_2 + x_2x_3)((x_2 + x_4) - (x_3 - x_5))$

- We can expand the two polynomials and compare their coefficients
- But it takes too much time.
  - Size of the expansion can be exponential in the number of gates.
  - Can you give such an example?

Key idea

- *Schwartz-Zippel Lemma*. If $p(x_1, \ldots, x_n)$ is a polynomial of total degree d over a field $\mathbb{F}$, then $\forall S \subseteq \mathbb{F}$,

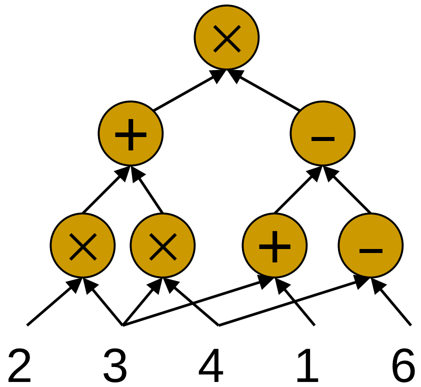
$$\Pr_{a_i \leftarrow_R S}[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}.$$

  - *total degree of a monomial* $x_1^2 x_2^3 x_5^7: 2 + 3 + 7 = 12$
  - *total degree of a polynomial*: the max total degree of its monomials.
  - $a_i \leftarrow_R S$: pick each a_i from S uniformly at random. (Different a_i's are picked independently.)

Few other observations

- A polynomial is easy to evaluate on any point by following the circuit.

- The (formal) degree of an polynomial is easy to obtain.



Randomized Algorithm

On input polynomials p_1 and p_2:

- $d = \max\{\deg(p_1), \deg(p_2)\}$
- $a_1, \ldots, a_n \leftarrow_R \{1,2, \ldots, 100d\}$
- Evaluate $p_1(a_1, \ldots, a_n)$ and $p_2(a_1, \ldots, a_n)$ by running the circuits on $(a_1, \ldots, a_n)$.
- **if** $p_1(a_1, \ldots, a_n) = p_2(a_1, \ldots, a_n)$,

    output "$p_1 = p_2$".

  else

    output "$p_1 \neq p_2$".

Correctness

- If $p_1 = p_2$, then $p_1(a_1, \dots, a_n) = p_2(a_1, \dots, a_n)$ is always true, so the algorithm outputs $p_1 = p_2$.
- If $p_1 \neq p_2$: Let $p = p_1 - p_2$. Recall that
  - we picked $a_1, \dots, a_n \leftarrow_R S \stackrel{\text{def}}{=} \{1,2, \dots, 100d\}$,
  - Lemma. $\Pr_{a_i \leftarrow_R S}[p(a_1, \dots, a_n) = 0] \leq \frac{d}{|S|}$.
  - So $p_1(a_1, \dots, a_n) = p_2(a_1, \dots, a_n)$ w/ prob. only 0.01.
  - The algorithm outputs $p_1 \neq p_2$ w/ prob. ≥ 0.99.

Catch

- One catch is that if the degree d is very large, then the evaluated value can also be huge.
  - Thus unaffordable to write down.
- Fortunately, a simple trick called "fingerprint" handles this.
  - Use a little bit of algebra; omitted here.

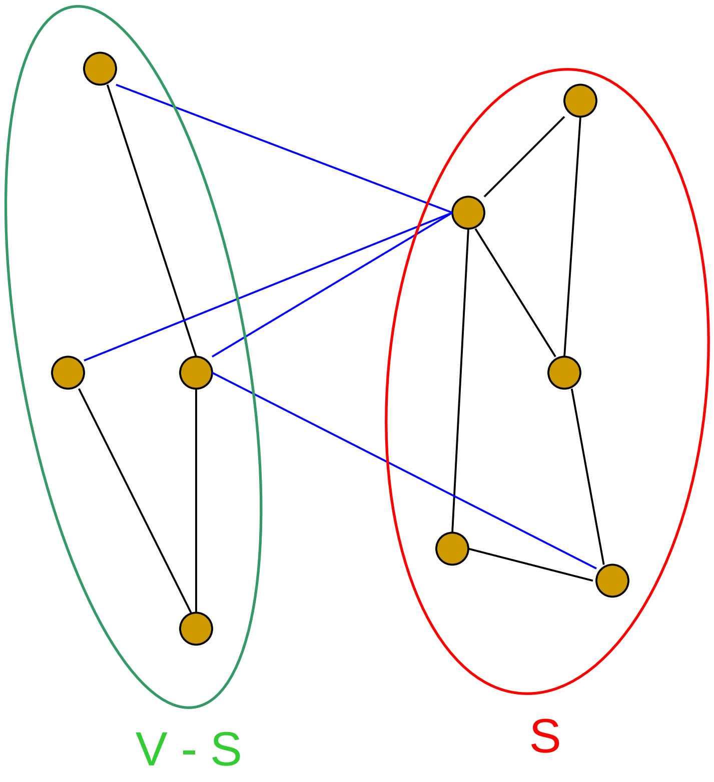
- Questions for the algorithm?

Part 1: Examples

Example 2: minimum cut

Min-cut for undirected graphs

- Given an undirected graph, a global min-cut is a cut $(S, V-S)$ minimizing the number of crossing edges.
  - Recall: a crossing edge is an edge (u, v) s.t. $u \in S$ and $v \in V-S$.

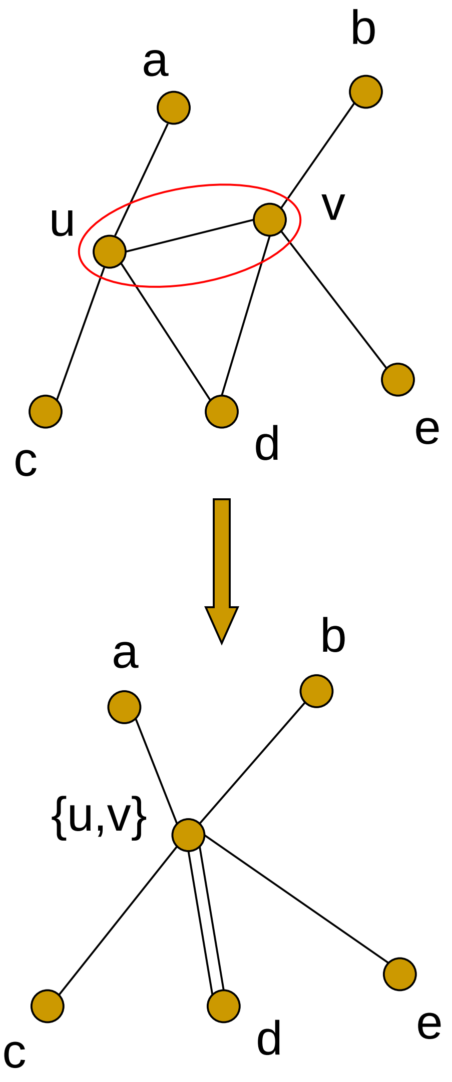


A simple algorithm

- We'll introduce Karger's *Contraction Algorithm*.
- It's surprisingly simple.

Graph Contraction

- For an undirected graph G and two vertices u, v.
- We contract u and v and form a new graph G':
  - u and v merge into one vertex $\{u, v\}$
  - Naturally, the edge (u, v) disappears.
  - Other edges incident to u or v in G naturally change to edges incident to $\{u, v\}$ in G'.
  - Now we may have more than one edge between two vertices. But well… that's fine. We just keep them there.



Karger's algorithm

- **for** $i = 1$ **to** $100n^2$

  repeat

  randomly pick an edge (u, v)

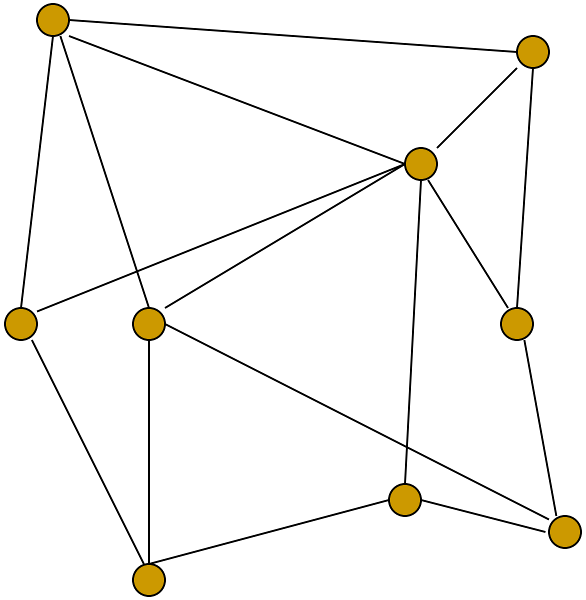
  contract u and v

  until two vertices are left

  $c_i \leftarrow$ the number of edges between them
- Output $\min_i c_i$

Example

- See an example on board.



key fact

- If we keep contracting a random edge until two vertices are left, then

  # of edges between them = min cut

  with prob. $\Omega(1/n^2)$.
  - $n = |V|$
- Thus repeating this $O(n^2)$ times and taking minimum give the min-cut with high prob.

Why?

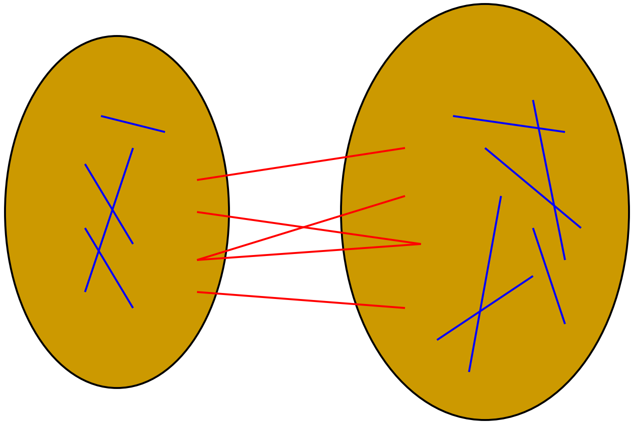
- One trial finds the min cut with probability $p = c/n^2$ for some constant c.
- If we make kn^2/c trials, then the probability that none of these finds the min cut is at most

$$\left(1 - \frac{c}{n^2}\right)^{\frac{kn^2}{c}} \approx e^{-k}$$

  - $\left(1 - \frac{1}{n}\right)^n \approx e^{-1}$
- Choose $k = 10$ makes this error probability < 0.001.

Analysis of the key fact

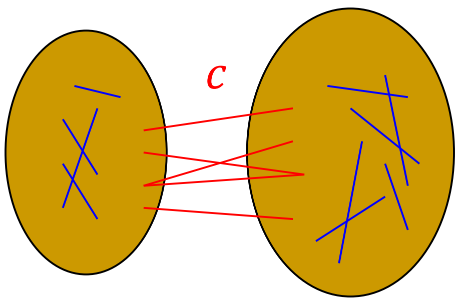
- Fix a min cut $(S, V - S)$:
  If we never pick a crossing edge in algorithm, then ok.
  - i.e. then finally the number of edges between two last vertices is the correct answer.



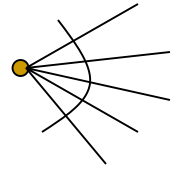
- Intuitively, a min cut has few crossing edges.
  - Thus it's likely that we don't pick them.
- Let's formally analyze the probability step by step.

Step 1

- In step 1: what's the prob. that a crossing edge is not picked?
- $(|E| - c)/|E|$.
  - c: the number of edges of min cut.



- Let's analyze this quantity:
  - By def of min cut, we know that each vertex v has degree at least c.
    - Otherwise the cut $(\{v\}, V - \{v\})$ is lighter.
  - Thus $|E| \geq nc/2$
  - And $(|E| - c)/|E| = 1 - c/|E| \geq 1 - 2/n$.



Step 2

- Similarly, in step 2,
- Pr [no crossing edge picked] ≥ $1 - 2/(n-1)$
  - assuming no crossing edge is picked in step 1
  - Note that now the number of vertices is $n - 1$.
- …
- In general, in step j,
- Pr [no crossing edge picked] $\geq 1 - 2/(n - j + 1)$

Together

- What's the prob. that all the $n-2$ steps didn't contract a crossing edge?
  - $\Pr[\text{step 1 didn't}]$

    $\cdot \Pr[\text{step 2 didn't} \mid \text{step 1 didn't}]$

    $\cdot \Pr[\text{step 3 didn't} \mid \text{step 1,2 didn't}]$

    $\ldots$

    $\cdot \Pr[\text{step } (n-2) \text{ didn't} \mid \text{step } 1,2,\ldots,n-3 \text{ didn't}]$

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\ldots\left(1-\frac{2}{3}\right)$$

$$= \frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2}\ldots\frac{2}{4}\frac{1}{3} = \frac{2\cdot 1}{n(n-1)} = \Omega\left(\frac{1}{n^2}\right)$$

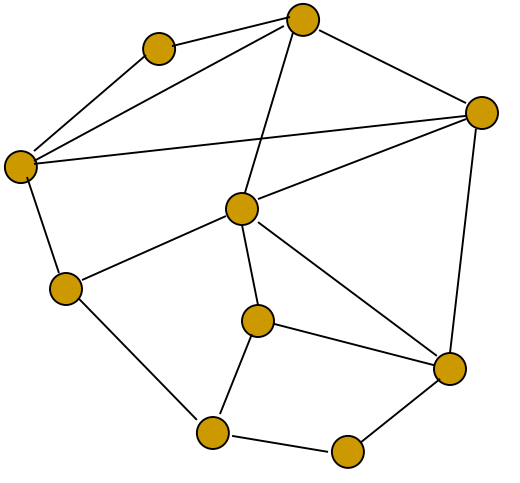
Part 1: Examples

Example 3. connectivity and 2-SAT by random walk

Random walk on graphs

- Graph G.
- Starting vertex v_0
- Each step:
  - Go to a random neighbor.

- Simple but powerful.



Typical questions about random walk

- Hitting time: How long it takes to hit a particular vertex?
  - $H(s,t)$: Expected time needed to hit t, starting from s
  - General graph: $H(s,t) = O(n^3)$
  - On a line $(v_1, \ldots, v_n)$: $H(v_1, v_n) = \Theta(n^2)$
- Covering time: How long it takes to visit all other vertices (at least once)?
  - $C(s)$: Expected time needed to visit all other vertices, starting from s.
  - General graph: $C(s) = O(n^3)$.
  - On a line $(v_1, \ldots, v_n)$: $H(v_i) = \Theta(n^2), \forall i$.

Connectivity

- *st*-Connectivity: Given an undirected graph G and two vertices s and t in it, decide whether there is a path from s to t in G.
- BFS can solve it, but needs $O(n)$ space.
- Here is an algorithm using only $O(\log n)$ space.
  - Starting from s, do random walk $O(n^3)$ steps
  - If never seen t, output NO; otherwise output YES.
- Space: $O(\log n)$, because one only needs to remember the current vertex.
- Correctness: Recall that the hitting time $H(s,t) = O(n^3)$ for any G and any s, t.

Algorithm for 2-SAT

- 2SAT: each clause has two variables /negations

$$(x_1 \lor x_2) \land (x_2 \lor \neg x_3) \land (\neg x_4 \lor x_3) \land (x_5 \lor x_1)$$

- Papadimitriou's Algorithm:
  - Pick any assignment
  - Repeat $O(n^2)$ time
    - If all satisfied, done
    - Else
      - Pick any unsatisfied clause
      - Pick one of the two literals each with ½ probability, and flip the assignment on that variable

x_1, x_2, x_3, x_4, x_5

~~0~~ 1, 0, 1, 0

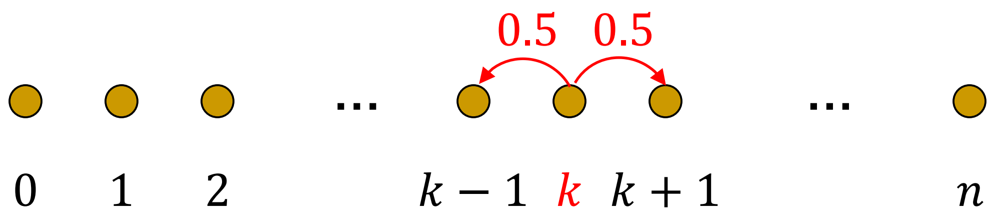
1

Analysis

- $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_4 \vee x_3) \wedge (\underline{x_5 \vee x_1})$
  - x_1, x_2, x_3, x_4, x_5
  - 0, 1, 0, 1, 0
- If unsatisfiable: never find a satisfying assignment
- If satisfiable: there exists a satisfying assignment x
  - If our initially picked assignment x' is satisfying, then done.
  - Otherwise, for any unsatisfied clause, *at least one of the two variables is assigned a value different than that in x*
  - Randomly picking one of the two variables and flipping its value increases $\{i : x_i = x_i'\}$ by 1 w.p. ≥ ½.

Analysis (continued)

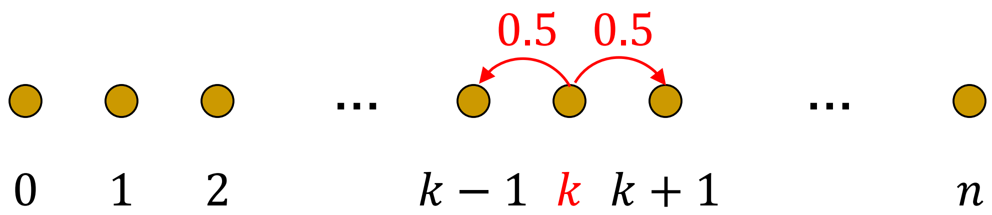
- Consider a line of $n+1$ points,



- Point k: we've assigned k variables correctly
  - "correctly": the same way as x
  - $k = n$: we've made $x' = x$ and thus found a satisfying assignment!
- Recall effect of flipping the value of a random variable (in a "bad" clause): increases $\{i: x_i = x_i'\}$ by 1 w.p. ≥ ½.

Analysis (continued)

- Consider a line of $n + 1$ points,



- Thus the algorithm is actually a random walk on the line of $n + 1$ points, with **$\mathbf{Pr}$[going right] ≥ ½.**
  - Recall hitting time ($i \to n$): $O(n^2)$.
- So by repeating this flipping process $O(n^2)$ steps, we'll reach n with high probability.
  - And thus find x, if such a satisfying assignment exists.

Part II: Basic analytical tools

Concentration and tail bounds

- In many analysis of randomized algorithms, we need to study how concentrated a random variable X is close to its mean $E[X]$.
  - Many times $X = X_1 + \cdots + X_n$.
- Upper bounds of

$$\Pr[X \text{ deviates from } E[X] \text{ a lot}]$$

is called *tail bounds*.

Markov's Inequality: when you only know expectation

- [Thm] If $X \geq 0$, then

$$\mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}.$$

In other words, if $E[X] = \mu$, then

$$\mathbf{Pr}[X \geq k\mu] \leq \frac{1}{k}.$$

- Proof. $\mathbf{E}[X] \geq a \cdot \mathbf{Pr}[X \geq a]$.
  - Dropping some nonnegative terms always make it smaller.

Moments

- Def. The k^{th} moment of a random variable X is

$$\mathbf{M}_k[X] = \mathbf{E}[(X - \mathbf{E}[X])^k]$$

- $k = 2$: variance.

$$\begin{aligned}\mathbf{Var}[X] &= \mathbf{E}[(X - \mathbf{E}[X])^2] \\ &= \mathbf{E}[X^2 - 2X \cdot \mathbf{E}[X] + \mathbf{E}[X]^2] \\ &= \mathbf{E}[X^2] - 2\mathbf{E}[X] \cdot \mathbf{E}[X] + \mathbf{E}[X]^2 \\ &= \mathbf{E}[X^2] - \mathbf{E}[X]^2\end{aligned}$$

Chebyshev's Inequality: when you also know variance

- [Thm] $\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a] \leq \frac{\mathbf{Var}[X]}{a^2}$.

  In other words,

  $$\mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq k \cdot \sqrt{\mathbf{Var}[X]}\right] \leq \frac{1}{k^2}.$$

- Proof.

  $$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a]$$

  $$= \mathbf{Pr}[|X - \mathbf{E}[X]|^2 \geq a^2]$$

  $$= \mathbf{Pr}[(X - \mathbf{E}[X])^2 \geq a^2]$$

  $$\leq \mathbf{E}[(X - \mathbf{E}[X])^2]/a^2$$ // Markov on $(X - \mathbf{E}[X])^2$

  $$= \mathbf{Var}[X]/a^2$$ // recall: $\mathbf{Var}[X] = \mathbf{E}[(X - \mathbf{E}[X])^2]$

Inequality by the k^{th}-moment (k: even)

- [Thm] $\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a] \leq \mathbf{M}_k[X]/a^k$.
- Proof.

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a]$$
$$= \mathbf{Pr}[|X - \mathbf{E}[X]|^k \geq a^k]$$
$$= \mathbf{Pr}[(X - \mathbf{E}[X])^k \geq a^k] \quad \text{// } k \text{ is even}$$
$$\leq \mathbf{E}[(X - \mathbf{E}[X])^k]/a^k \text{ // Markov on } (X - \mathbf{E}[X])^k$$
$$= \mathbf{M}_k[X]/a^k$$

Chernoff's Bound

- [Thm] Suppose $X_i = \begin{cases} 1 & \text{with prob. } p \\ 0 & \text{with prob. } 1-p \end{cases}$

  and let

  $$X = X_1 + \cdots + X_n.$$

  Then

  $$\mathbf{Pr}[|X - \mu| \geq \delta\mu] \leq e^{-\delta^2\mu/3},$$

  where $\mu = np = \mathbf{E}[X]$.

Some basic applications

- One-sided error: Suppose an algorithm for a decision problem has
  - $f(x) = 0$: no error
  - $f(x) = 1$: output $f(x) = 0$ with probability 1/2
- We want to decrease this ½ to ε. How?
- Run the algorithm $\left\lceil \log_2\left(\frac{1}{\varepsilon}\right) \right\rceil$ times. Output 0 iff all executions answer 0.

Two-sided error

- Suppose a randomized algorithm has two-sided error
  - $f(x) = 0$: output $f(x) = 0$ with probability > 2/3
  - $f(x) = 1$: output $f(x) = 1$ with probability > 2/3

- How?
- Run the algorithm $O(\log(1/\varepsilon))$ steps and take a majority vote.

Using Chernoff's bound

- Run the algorithm n times, getting n outputs. Suppose they are $X_1, \dots, X_n$.

- Let $X = X_1 + \cdots + X_n$
  - if $f(x) = 0$: $X_i = 1$ *w.p.* $p < \frac{1}{3}$, thus $\mathbf{E}[X] = np < \frac{n}{3}$.
  - if $f(x) = 1$: $X_i = 1$ *w.p.* $p > \frac{2}{3}$, so $\mathbf{E}[X] = np > \frac{2n}{3}$.

- Recall Chernoff: $\mathbf{Pr}[|X-\mu| \geq \delta\mu] \leq e^{-\delta^2\mu/3}$.
- If $f(x)=0$: $\mu = \mathbf{E}[X] < \frac{n}{3}$.
  - $\delta\mu = \frac{n}{2} - \frac{n}{3} = \frac{n}{6}$, so $\delta = \frac{n/6}{n/3} = \frac{1}{2}$.
- $\mathbf{Pr}\left[X \geq \frac{n}{2}\right] \leq \mathbf{Pr}\left[|X-np| \geq \frac{n}{6}\right] \leq e^{-\frac{\delta^2\mu}{3}} = 2^{-\Omega(n)}$.
- Similar for $f(x)=1$.
- The error prob. decays exponentially with # of trials!

Summary

- We showcased several random algorithms.
  - Simple and fast
- We also talked about some basic tail bounds.
  - Concentration of a random variable around its mean.