上海科技大学
ShanghaiTech University

CS240 Algorithm Design and Analysis

Lecture 23

Randomized Algorithms

Quan Li
Fall 2023
2023.12.28

Bloom Filters

Approximate Sets

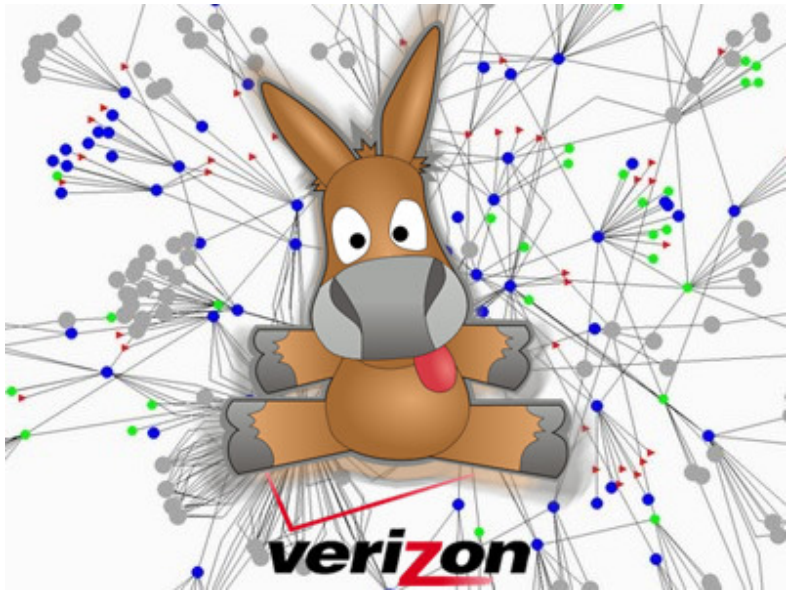
- A Bloom filter is a data structure that can implement a set.
  - It only keeps track of which keys are present, not any values associated to keys.
  - It supports insert and find operations.
  - It doesn't support delete operations.
- Bloom filters use less memory than hash tables or other ways of implementing sets.
- However, Bloom filters are approximate.
  - It can produce false positives: it says an element is present even though it's not.
    - We can bound the probability of false positives.
  - But it doesn't produce false negatives: if it says an element isn't present, then it's not.

Bloom Filter Applications

- Suppose we have a big database and querying it to check if an item is present is expensive.
- We store the set of items in the database using a Bloom filter.
  - This tells us whether an item is in database or not.
- If filter says an item's not present, it's definitely not in the database.
  - So, no need to do an expensive query.
- If filter says an item is present, then either item is present, or there's false positive.
  - When we query the database, there's a small probability we waste time querying for a nonexistent item.
- Overall, we save time by checking Bloom filter first before querying database.

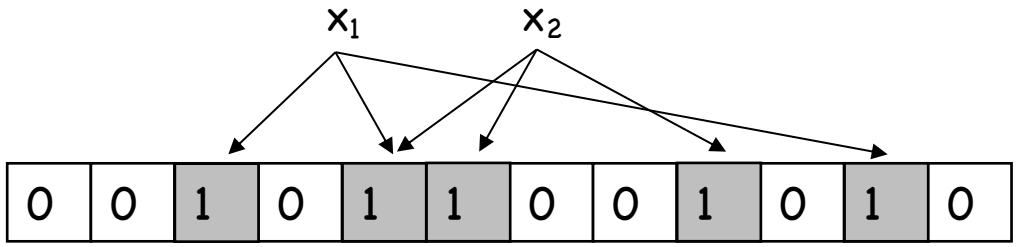
Bloom Filter Applications

- Consider a P2P network, where each node stores some files.
- If you want to get a file, you need to know which nodes have it.
- Keeping a list of all items stored at each node is too expensive.
- Instead, for every other node, keep a Bloom filter of its files.
- If filter says no for a node, it definitely doesn't have the file.
- If filter says yes, then either node has the file, or there's false positive and we make a useless request.
- Overall, we save space, and also won't waste much communication because we rarely make useless requests.



Bloom Filters

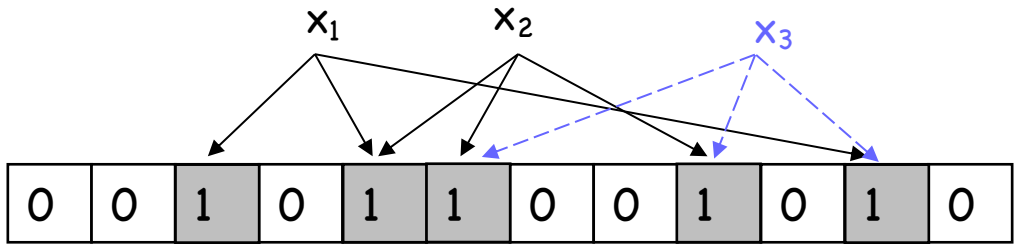
- A Bloom filter consists of
  - An array A of size m, initially all 0's.
  - k independent hash functions $h_1,\ldots,h_k$, each mapping from keys to $\{1,\ldots,m\}$.
- To store key x
  - Set $A[h_1(x)]$, $A[h_2(x)]$, ..., $A[h_k(x)]$ all to 1.
  - Some locations can get set to 1 multiple times; that's fine.
- To check if key x is in the set
  - Read array locations $A[h_1(x)]$, $A[h_2(x)]$, ..., $A[h_k(x)]$.
  - If all the values are 1, output "x is in set".
  - Otherwise, output "x is not in set".



A Bloom filter with k=3 hash functions storing 2 items.

Correctness

- Let's look at the correctness of the search function.
- If search for x returns no, then at least one of $A[h_1(x)],\ldots, A[h_k(x)]$ equals 0.
  - So x cannot be in the set, because if x had been inserted into the set, then we would have $A[h_1(x)]=\ldots=A[h_k(x)]=1$.
  - So there are no false negatives.
- If search for x returns yes, then $A[h_1(x)]=\ldots=A[h_k(x)]=1$.
  - So either x was inserted into the set.
  - Or we inserted some keys that hashed to the same k locations as x.
    - So it looks as if x was inserted, even though it wasn't.
    - This is a false positive. We'll bound the probability this happens.



False Positive Probability 1

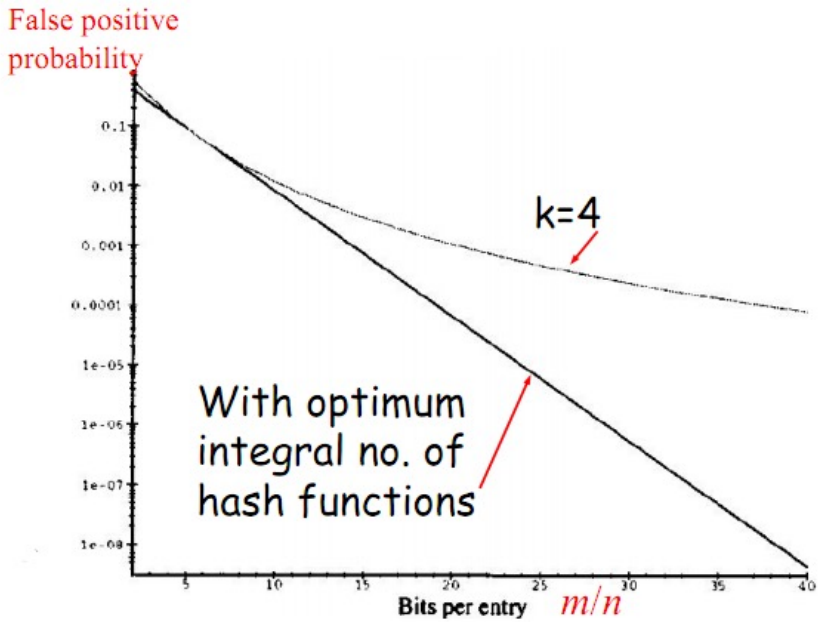
- False positive probability depends on k (number of hash functions), m (size of table) and n (number of keys inserted).
- Assume hash functions hash keys to random locations.
- When inserting one key, we set k random locations to 1.
- Fix any position i. Probability i is set to 1 by a hash function is 1/m, so probability i stays 0 is 1-1/m.
  - After k hashes, probability i still 0 is $(1-1/m)^k$.
  - To insert n items, we used nk hashes. So, probability i still 0 after all these is $p=(1-1/m)^{nk}$.
- We now use an approximation $\left(1-\frac{1}{m}\right)^{nk} \approx e^{-\frac{nk}{m}}$.

False Positive Probability 2

- So, probability any position i is 1 after n keys inserted is $1 - p \approx 1 - e^{-\frac{nk}{m}}$.
- Since there are m positions in the array, assume there are (1-p)m positions that are 1.
  - This isn't quite correct. The actual number of 1's in the array is a random variable, whose expectation is (1-p)m.
  - However, we can make the argument rigorous by showing that the actual number of 1's is $(1-p)m \pm \sqrt{m \log m}$ with high probability.
- We only get a false positive if when we check k random locations, they're all 1.
  - Probability is $f = (1-p)^k \approx \left(1 - e^{-\frac{nk}{m}}\right)^k$.

False Positive Probability 3

- Notice the false prob. $\left(1-e^{-\frac{nk}{m}}\right)^k$ is a function of k, the number of hash functions we use.
- We find k to minimize the false positive prob. by differentiating f wrt k and solving.
- The optimum k is $\frac{m\ln(2)}{n}$, which leads to $f=\left(\frac{1}{2}\right)^k \approx 0.6185^{\frac{m}{n}}$.
  - Notice that m/n is the average number of bits per item. So error rate decreases exponentially in space usage.



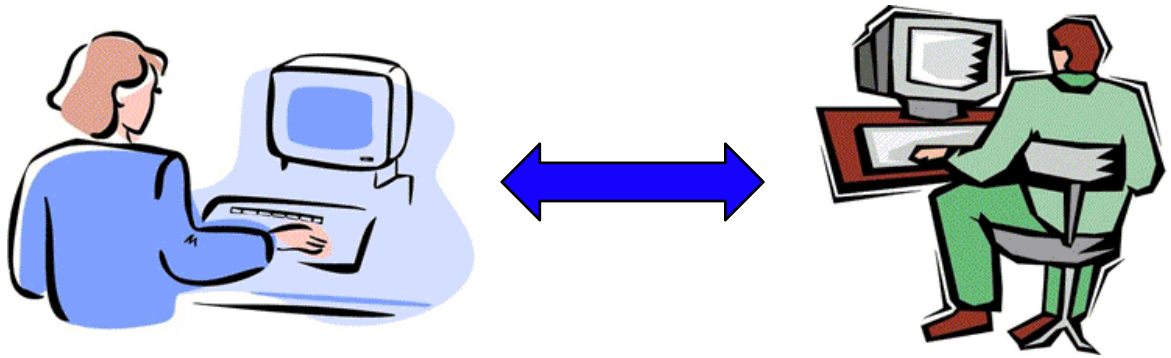
Improvements

- Right now, Bloom filters can't handle deletes.
  - Say keys k_1, k_2 hash to two overlapping sets of locations. If you delete k_1 by setting some of its locations to 0, you could also delete k_2.
- Deletes can be done by storing a count of how many keys hashed to that location, and inc / dec the counts when inserting or deleting.
  - But this uses more memory.
  - Also, what if the counts overflow?

String Equality and Fingerprinting

String Equality and Fingerprinting

- Alice and Bob both have copies of a database.
- They want to keep the database consistent, so they want to check if their copies are the same.
  - If you think of the databases as strings, they want to check if their strings are equal.
- But transferring the entire database is expensive.
- Instead, they calculate a small value called a fingerprint of their databases.
  - If the fingerprints are the different, then their databases are definitely different.
  - If the fingerprints are the same, then the databases are probably the same; but there's a small probability they're actually different.
- Transferring the fingerprint is much cheaper than the database.



Fingerprinting

- Let Alice and Bob's databases be the bit sequences $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$.
- View these as n-bit integers $a = \sum_{i=1}^{n} a_i * 2^{i-1}$ and $b = \sum_{i=1}^{n} b_i * 2^{i-1}$.
- The fingerprint $F(a) = a \bmod p$, for a specially chosen prime number p.
  - Alice transfers F(a) to Bob, and Bob compares it to his fingerprint F(b)=b mod p.
  - Since F(a) < p, transferring the fingerprint only takes O(log p) bits, instead of n.

Correctness

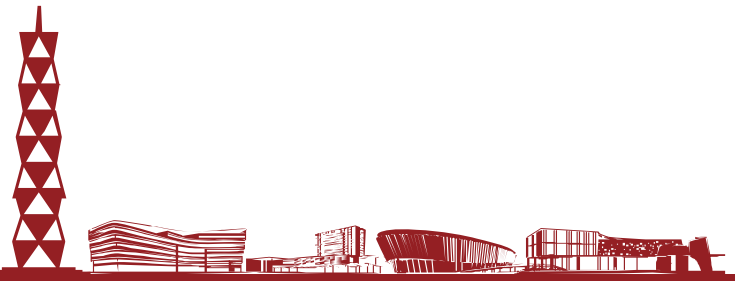
- No false positives (positive means "$a \neq b$").
  - If F(a) ≠ F(b), then a ≠ b.
- False negatives are possible.
  - If F(a)=F(b), then a mod p = b mod p.
  - So either a=b, or a ≠ b but p divides (a-b).
- We can't avoid false negatives. But we can minimize the probability it occurs.
- Pick a random p.
  - Bigger p decreases false negative probability.
  - But we don't want to make p too big, since we have to transfer O(log p) bits.

Correctness

- To analyze the false negative probability, we use two facts from number theory.
- Lemma Any number t has at most $\log_2(t)$ distinct prime divisors.
- Proof Each divisor is ≥ 2, and their product is $\leq t$. If there were more than $\log_2(t)$ divisors, their product would be $> 2^{\log_2(t)} = t$, contradiction.
- Recall $a = \sum_{i=1}^{n} a_i * 2^{i-1}$ and $b = \sum_{i=1}^{n} b_i * 2^{i-1}$
- So $a-b<2^n$, and so a-b has at most n distinct prime divisors.

Correctness

- Prime Number Theorem Given any number t, the number of primes smaller than t is ~ t / ln(t).
- The PNT allows us to efficiently generate a random prime.
  - Picking a number less than t at random, it has a 1/ln(t) probability of being prime.
  - We can check if a number is prime using the Rabin-Miller primality test.
    - If number is prime, it always passes the test.
    - If number is composite, there's small probability it's declared a prime.
    - Run the test few more times to exponentially decrease false positive probability.
  - So with high probability, we can tell if a number is prime.



Correctness

- Let $t = n^2 \ln(n)$. The number of primes less than t is $\approx \frac{t}{\ln(t)} = \frac{n^2 \ln(n)}{2\ln(n)+\ln\ln(n)} = O(n^2)$.
- Pick a random prime p less than t.
- We get a false negative if a≠b but p divides (a-b).
  - We saw earlier that a-b has < n prime divisors, and p must be one of these.
  - But p is randomly chosen from $O(n^2)$ primes less than t.
  - So false negative probability ≤ $n/O(n^2) = O(1/n)$.
- We transfer log(p) ≤ log(t) = O(log n) bits.
- Transferring O(log n) bits gets O(1/n) probability of error. If we want perfect accuracy, we need to transfer the entire database, O(n) bits.

Chernoff Bounds

Beating Expectations

- Given a random variable X, the expected value E[X] is the value we're "supposed" to get when we take a sample of X.
- We won't always get E[X].
- What is the probability the sample is very different from E[X]?
- Ex If we flip a fair coin 100 times, we expect to get 50 heads. What's the probability we get 60 heads? 80 heads? 100 heads?

Beating Expectations

- Markov's Inequality Given a positive random variable X, $\Pr[X \geq a] \leq E[X]/a$ for any $a > 0$.
- Ex X = no. heads in 100 flips. $\Pr[X \geq 60] \leq 50/60 = 5/6$.
- Proof Suppose $\Pr[X \geq a] > E[X]/a$. By definition, we have

$$E[X] = \sum_x x \cdot \Pr[X = x] \geq \sum_{x \geq a} x \cdot \Pr[X = x] \geq$$

$$a \cdot \Pr[X \geq a] > a \cdot E[X]/a = E[X], \text{ contradiction.}$$

- Markov's inequality is weak.
  - Using the previous example, it states $\Pr[X \geq 101] \leq 50/101 \approx 0.495$, which is quite obvious!

Beating Expectations

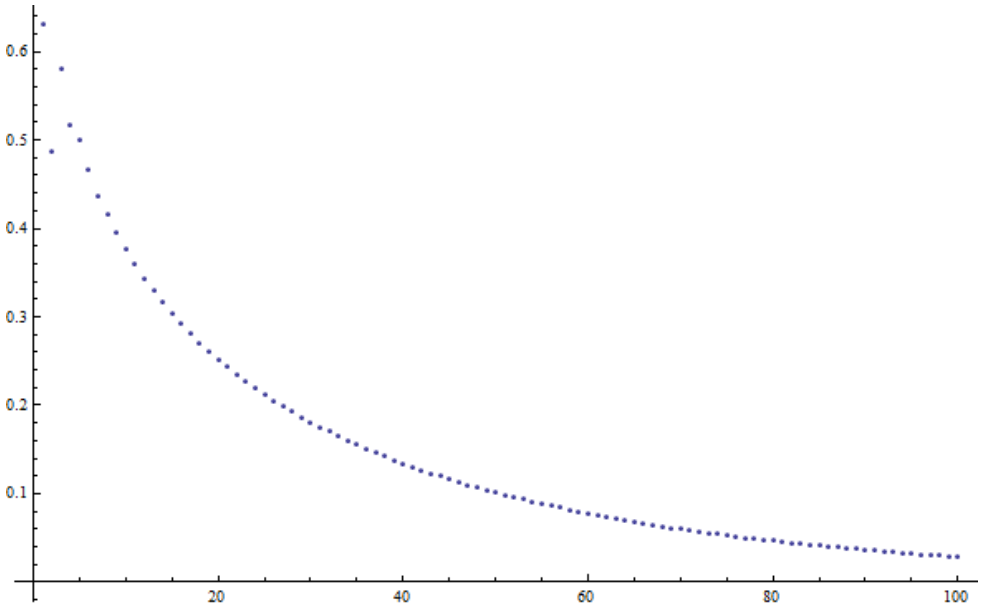
- But Markov's inequality is general.
  - X can be any positive random variable. It doesn't need to satisfy any special properties, as for some other inequalities.
  - Under some circumstances it can be used to prove stronger statements.
- Chebychev's Inequality Given a random variable X and any $a > 0$, we have $\Pr[|X - E[X]| \geq a] \leq Var[X]/a^2$.
- Proof $Var[X] = E[(X - E[X])^2]$, so by Markov's inequality,

$\Pr[|X - E[X]| \geq a] = \Pr[(X - E[X])^2 \geq a^2] \leq Var[X]/a^2$.

- Ex Let X=no. heads in 100 flips. Var[X]=100/4. So $\Pr[|X - 50| \geq 10] \leq 25/100 = 1/4$ by Chebychev.
  - Since X is symmetric about 50, $\Pr[X \geq 60] \leq 1/8$, which is much better than Markov's inequality.

Sum of independent random variables

- We'll consider a special kind of random variable X that is the sum of n independent random variables, for some n. We look at how likely X is to deviate from its expectation.
- Intuitively, as n gets larger, X should approach E[X], in relative terms, very quickly.
  - In fact, the convergence is exponential.
- Consider flipping a fair coin n times. What's the probability we get at least 60% heads?
  - For n=10, the probability is 37.7%.
  - For n=20, the probability is 25.2%.
  - For n=30, the probability is 18.1%.
  - For n=40, the probability is 13.4%.
  - For n=100, the probability is 2.84%.



Chernoff Bounds

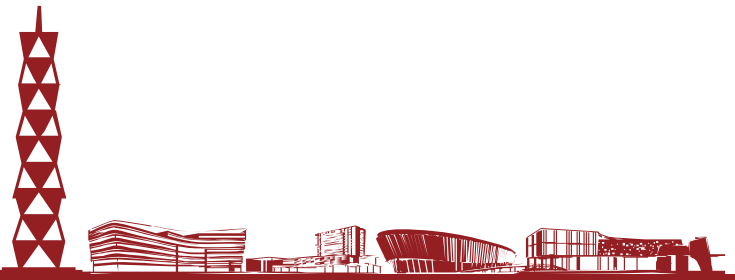
- **Thm 1** Let $X_1,\ldots,X_n$ be independent random variables, s.t. $E[X_i] = p_i$ for all i. Let $X = \sum_i X_i$ and $\mu = E[X] = \sum_i p_i$. Then

  - For $0 < \delta \leq 1$, $\mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$.
  - For $\delta > 1$, $\mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\mu\delta \ln \delta/3}$.
  - For $0 \leq \delta \leq 1$, $\mathbf{Pr}[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$.

- Chernoff bounds say the probability the sum of a set of independent random variables is more than $1 \pm \delta$ times its expectation m decreases exponentially in δ and μ.

Chernoff Bounds

- Chernoff bounds are very useful in the analysis of randomized algorithms.
  - Often the algorithm does things in independent stages.
  - It's often easy to compute the expected cost of each stage.
  - The total cost is the sum of the cost of all the stages.
  - We use Chernoff bounds to show this is unlikely to be too big or small compared to its expectation.
- There are some variants of Chernoff's bound that differ in the precise bounds. Some giver tighter bounds for certain values of δ. Pick the best one to use for the situation.



Chernoff Bounds

- **Thm 2** Let X, $X_1,\ldots,X_n$ be defined as earlier. Then for any δ >0 we have

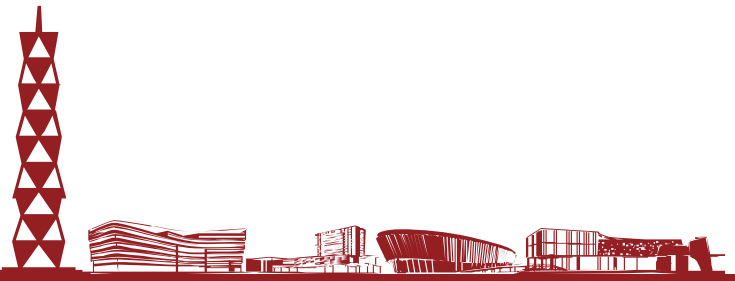
$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- **Thm 3** Let $X_1,\ldots,X_n$ be independent {-1,1} valued random variables, with $\Pr[X_i=1]= \Pr[X_i=-1]=1/2$ for all i. Let $X = \sum_i X_i$. Then for any $\delta \geq 0$, $\Pr[X \geq \delta] = \Pr[X \leq -\delta] \leq e^{-\frac{\delta^2}{2n}}$.

Load Balancing

- Suppose we have n computers. A set of equal sized jobs arrive online. We need to assign each to a computer for processing.
- To make all jobs finish fast, we want to give all computers (almost) same number of jobs, i.e., we want to balance their load.
- A simple way is to assign jobs round-robin. Keep giving next job to next computer, wrapping around if necessary.
- But this requires communicating with a centralized controller, which can be a bottleneck for large n.
- Instead, we do randomized load balancing.

❖ **Algorithm** Given a new job, assign it to a random computer.



Load Balancing

- How well does this balance the load?
- If there are m jobs, then in expectation every computer gets m/n jobs.
- Let X_i be the number of jobs computer i gets and let $X=\max_i \{X_i\}$ be max number of jobs any computer gets.
  - We'll bound probability that X_i or X are too large compared to the expectation m/n.
  - This shows the load is roughly balanced with high probability.
- Let Y_{ij} be a random variable that's 1 if the j'th job is assigned to computer i, and 0 otherwise.
  - $\Pr[Y_{ij}=1]=1/n$, since the jobs are assigned randomly.
  - $X_i = \sum_j Y_{ij}$ is the number of jobs computer i gets.
  - X_i is the sum of independent random variables Y_{ij}, so we can apply the Chernoff bound to X_i.

Analysis

- First consider the case $m = n$.
- We have $\mu = \mathbf{E}(X_i) = \mathbf{E}[\sum_{j=1}^{m} Y_{ij}] = \sum_{j=1}^{m} \mathbf{E}[Y_{ij}] = m\frac{1}{n} = 1$ for every i.
- We'll show $\mathbf{Pr}[X > O(\frac{\ln n}{\ln \ln n})] < \frac{1}{n}$.
  - So with high probability ($> 1 - 1/n$), every computer gets at most $O(\frac{\ln n}{\ln \ln n})$ times its expected number (=1) of jobs.
- Let's focus on X_i for some particular i. Let $\delta = O\left(\frac{\ln n}{\ln \ln n}\right) > 1$.
- By part 2 of Theorem 1, $\mathbf{Pr}[X_i > (1+\delta)\mu] \leq e^{-\delta \ln \delta/3}$, since $\mu = 1$.
  - So $e^{-\delta \ln \delta/3} = e^{O(\ln n)} \leq \frac{1}{n^2}$ for some choice of the constant in the big-O.
- We have $\mathbf{Pr}[X_i > O(\frac{\ln n}{\ln \ln n})] \leq \frac{1}{n^2}$ for every i. Hence by the union bound $\mathbf{Pr}[X_i > O(\frac{\ln n}{\ln \ln n})$ for any $i] \leq n\frac{1}{n^2} = \frac{1}{n}$.
- So $\mathbf{Pr}[X = \max_i X_i > O(\frac{\ln n}{\ln \ln n})] < \frac{1}{n}$.

Analysis

- We see that when there are n jobs for n computers, the load can be quite unbalanced.
- Suppose now we have more jobs, $m = 16n \ln n$. Then $\mu = 16 \ln n$.
- By Theorem 2, $\mathbf{Pr}[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16 \ln n} < \left(\frac{1}{e^2}\right)^{\ln n} = \frac{1}{n^2}$.
- Thus, by the union bound $\mathbf{Pr}[\max_i X_i > 2\mu] \leq n \frac{1}{n^2} = \frac{1}{n}$.
- By part 3 of Theorem 1, $\mathbf{Pr}[X_i < \frac{1}{2}\mu] \leq e^{-(\frac{1}{2})^2 16 \ln n/2} = e^{-2 \ln n} = \frac{1}{n^2}$.
- So by the union bound, $\mathbf{Pr}[\min_i X_i < \frac{1}{2}\mu] \leq \frac{1}{n}$.
- Thus, we have that with high probability, every computer has between half and twice the average load.
- These results hold for any $m > 16n \ln n$. So the more jobs there are, the better the load balancing random allocation achieves. This is similar to the phenomenon where the more coins we flip, the more likely we are to get the expected number of heads (in relative terms).

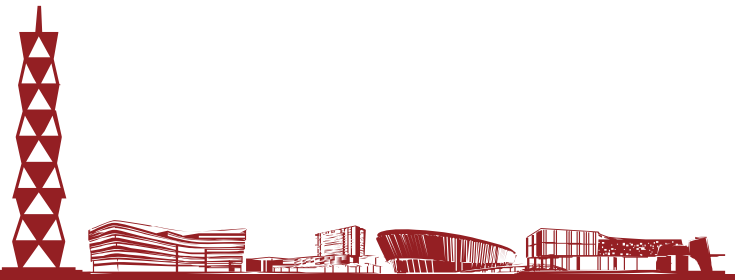
Set Balancing

- Suppose we have m sets, each a subset of {1,2,...,n}.
  - We can represent each set as an n-bit vector.
  - Ex If n=4 and S={1,3,4}, we can represent it as [1,0,1,1].
- We want to divide the sets into two groups, such that the sums of the sets in the groups are roughly equal.
- Ex S_1=[1,0,1,1], S_2=[1,1,1,0], S_3=[0,0,1,1], S_4=[0,1,1,0]. Then $S_1+S_4=[1,1,2,1]=S_2+S_3$.
- And exact balancing might not exist. We look for one that's as good as possible.
- Let G, G′ denote the two groups. We want to minimize the max imbalance $|\sum_{i\in G} S_i - \sum_{i\in G'} S_i|_\infty$.
  - Here $|v|_\infty$ denotes the L_∞ norm and is equal to the max component of v in absolute value. E.g. $|[1,2,-3,1]|_\infty = 3$.
  - Ex S_1=[1,1,0,1], S_2=[1,0,0,1], S_3=[1,0,1,0], S_4=[1,1,0,0].
  - There's no exact balancing, but S_1+S_3=[2,1,1,1] and S_2+S_4=[2,1,0,1], so the max imbalance is $|[2,1,1,1]-[2,1,0,1]|_\infty = |[0,0,1,0,]|_\infty = 1$.

Set Balancing

- Finding the grouping that minimizes the max imbalance is hard. Brute force would try 2^m possible groupings.
- We give a randomized algorithm that ensures the max imbalance is $\sqrt{4m \ln n}$ with high probability.
  - Max possible imbalance is m.

❖ Algorithm Assign each set to a random group.



Analysis

- Consider an item i. Suppose i belongs to k different sets.
- For the j'th such set S, define X_j=1 if S is in the first group, and X_j=-1 if it's in the other group.
- The imbalance from item i is simply $B_i = \sum_{j=1}^{k} X_j$.
- Since sets are assigned randomly, $X_1,\ldots,X_k$ are independent {1,-1} valued random values, we can apply Thm 3 to bound $\max_i B_i$.
- If $k \le \sqrt{4m\ ln\ n}, then\ B_i \le \sqrt{4m \ln n}$
- If $k > \sqrt{4m \ln n}$, then by Theorem 3 we have

$$\begin{aligned}\Pr[B_i > \sqrt{4m \ln n}] &\le e^{-\left(\sqrt{4m \ln n}\right)^2/(2k)} \\ &= e^{-4m \ln n/(2k)} \\ &\le e^{-4m \ln \frac{n}{2m}} \\ &= e^{-2 \ln n} \\ &= \frac{1}{n^2}\end{aligned}$$

- We showed the probability item i's balance is $\ge \sqrt{4m \ln n}$ is $\le \frac{1}{n^2}$. By the union bound, the probability that any of the n items' imbalance is $\ge \sqrt{4m \ln n}$ is $\frac{1}{n}$. So with high probability the max imbalance is $\le \sqrt{4m \ln n}$

Next Time:
Randomized
&
Approximation algorithms