Announcements: Homework 6 due tonight at 11:59 PM

Practice Final Exam Released on Piazza

Course Evaluation Survey: Live until 4/28/2019 at 11:59PM. Your feedback is valued! (Current Response Rate: 25%)
13.9 Chernoff Bounds
Chernoff Bounds (above mean)

**Theorem.** Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

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

sum of independent 0-1 random variables
is tightly centered on the mean

**Pf.** We apply a number of simple transformations.

- For any $t > 0$, $\Pr[X > (1 + \delta)\mu] = \Pr[e^{tX} > e^{t(1+\delta)\mu}]$

- Let $Y = e^{tX}$ be a random variable $f(x) = e^{tx}$ is monotone in $x$

$$\Pr[X > (1 + \delta)\mu] = \Pr[Y > e^{t(1+\delta)\mu}] \leq \frac{E[Y]}{e^{t(1+\delta)\mu}} = e^{-t(1+\delta)\mu} \times E[e^{tx}]$$

Markov's inequality: $\Pr[Y > a] \leq E[Y]/a$
Theorem. Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

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

\[\uparrow\]

sum of independent 0-1 random variables is tightly centered on the mean

Pf. We apply a number of simple transformations.

- For any $t > 0$,
  \[\Pr[X > (1 + \delta)\mu] \leq e^{-t(1+\delta)\mu} \times E[e^{tX}] = e^{-t(1+\delta)\mu} \prod_i E[e^{tx_i}]\]

- Now
  \[E[e^{tX}] = E[e^{t\sum_i x_i}] = \prod_i E[e^{tx_i}]\]

  \[\uparrow\]

  definition of $X$

  \[\uparrow\]

  independence
Chernoff Bounds (above mean)

**Pf. (cont)**

- Let $p_i = \Pr[x_i = 1]$. Then,
  \[ E[e^{tx_i}] = p_i \times e^t + (1 - p_i)e^0 = 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)} \]
  for any $\alpha \geq 0$, $1 + \alpha \leq e^\alpha$

- Combining everything:
  \[
  \Pr[X > (1 + \delta)\mu] \leq e^{-t(1+\delta)\mu} \prod_i E[e^{tx_i}] \\
  \quad \leq e^{-t(1+\delta)\mu} \prod_i e^{p_i(e^t - 1)} \\
  \quad = e^{-t(1+\delta)\mu} \times e^{(e^t - 1)\Sigma_i p_i} \\
  \quad \leq e^{-t(1+\delta)\mu} \times e^{(e^t - 1)\mu}
  \]

- Finally, choose $t = \ln(1 + \delta)$. 
Chernoff Bounds (above mean)

Theorem. Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

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

sum of independent 0-1 random variables is tightly centered on the mean

Pf. (cont) We had derived for any $t > 0$

$$\Pr[X > (1 + \delta)\mu] \leq e^{-t(1+\delta)\mu} \times e^{(e^t-1)\mu}$$

Plugging in $t = \ln(1 + \delta)$. We have

$$e^{-t(1+\delta)\mu} \times e^{(e^t-1)\mu} = \left[\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right]^\mu$$
Chernoff Bounds (below mean)

**Theorem.** Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \leq E[X]$ and for any $0 < \delta < 1$, we have

$$\Pr[X < (1 - \delta)\mu] < e^{-\delta^2 \mu / 2}$$

**Pf idea.** Similar.

**Remark.** Not quite symmetric since only makes sense to consider $\delta < 1$. 
13.10 Load Balancing
Load Balancing

Load balancing. System in which m jobs arrive in a stream and need to be processed immediately on n identical processors. Find an assignment that balances the workload across processors.

Centralized controller. Assign jobs in round-robin manner. Each processor receives at most \( \lceil m/n \rceil \) jobs.

Decentralized controller. Assign jobs to processors uniformly at random. How likely is it that some processor is assigned "too many" jobs?
Load Balancing

Analysis. (m=n jobs)

- Let $X_i$ = number of jobs assigned to processor $i$.
- Let $Y_{ij} = 1$ if job $j$ assigned to processor $i$, and 0 otherwise.
- We have $E[Y_{ij}] = 1/n$
- Thus, $X_i = \sum_j Y_{ij}$, and $\mu = E[X_i] = 1$.
- Applying Chernoff bounds with $\delta = c - 1$ yields $\Pr[X_i > c] < \frac{e^{c-1}}{c^c}$

**Theorem.** Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

$$\Pr[X > (1 + \delta)\mu] < \left[\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right]^\mu$$
Load Balancing

Analysis. \(m=n\) jobs

- Let \(X_i\) = number of jobs assigned to processor \(i\).
- Let \(Y_{ij}\) = 1 if job \(j\) assigned to processor \(i\), and 0 otherwise.
- We have \(E[Y_{ij}] = 1/n\)
- Thus, \(X_i = \sum_j Y_{ij}\), and \(\mu = E[X_i] = 1\).
- Applying Chernoff bounds with \(\delta = c - 1\) yields \(\Pr[X_i > c] < \frac{e^{c-1}}{c^c}\).

- Let \(\gamma(n)\) be number \(x\) such that \(x^x = n\), and choose \(c = e^{\gamma(n)}\).

\[
\Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left(\frac{e}{c}\right)^c = \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)} < \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)} = \frac{1}{n^2}
\]

- Union bound \(\Rightarrow\) with probability \(\geq 1 - 1/n\) no processor receives more than \(e^{\gamma(n)} = \Theta(\log n / \log \log n)\) jobs.

Fact: this bound is asymptotically tight: with high probability, some processor receives \(\Theta(\log n / \log \log n)\) jobs.
Theorem. Suppose the number of jobs $m = 16n \ln n$. Then on average, each of the $n$ processors handles $\mu = 16 \ln n$ jobs. With high probability every processor will have between half and twice the average load.

Pf.

- Let $X_i$, $Y_{ij}$ be as before.
- Applying Chernoff bounds with $\delta = 1$ yields
  
  $\Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16 \ln n} \ll \left(\frac{1}{e}\right)^{2 \ln n} = \frac{1}{n^2}$

  $\Pr[X_i < \frac{\mu}{2}] < e^{2 \ln n} = \left(\frac{1}{e}\right)^{2 \ln n} = \frac{1}{n^2}$

- Union bound $\Rightarrow$ every processor has load between half and twice the average with probability $\geq 1 - 2/n$. $\blacksquare$
Load Balancing with Asymmetry

**Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most \( \lceil \frac{m}{n} \rceil \) jobs.

- Suppose each job has cost between 1 and 4
- Round-Robin assignment may be highly unbalanced
- E.g., (n=4 processors): 1,2,3,4,1,2,3,4,…
  - Processor 1 total cost: \( \frac{m}{4} \)
  - Processor 2 total cost: \( \frac{m}{2} \)
  - Processor 3 total cost: \( \frac{3m}{4} \)
  - Processor 4 total cost: \( m \)
  - Fair: \( 2.5\frac{m}{4} \) per processor

**Decentralized controller.** Assign jobs to processors uniformly at random. How likely is it that some processor is assigned "too many" jobs?

- Still works well in the above scenario (bounded costs)
- Workload on each processor would be \( \approx 2.5\frac{m}{4} \) (w.h.p) in above example
12.4 Maximum Cut
Maximum Cut

Maximum cut. Given an undirected graph $G = (V, E)$ with positive integer edge weights $w_e$, find a node partition $(A, B)$ such that the total weight of edges crossing the cut is maximized.

$$w(A, B) := \sum_{u \in A, v \in B} w_{uv}$$

Toy application.
- $n$ activities, $m$ people.
- Each person wants to participate in two of the activities.
- Schedule each activity in the morning or afternoon to maximize number of people that can enjoy both activities.
  - Nodes: Activities
  - Edge Weights: $w_{uv} =$ \#people who want to participate in both activities

Real applications. Circuit layout, statistical physics.
Maximum Cut

Single-flip neighborhood. Given a partition \((A, B)\), move one node from \(A\) to \(B\), or one from \(B\) to \(A\) if it improves the solution.

Greedy algorithm.

Max-Cut-Local \((G, w)\) {
    Pick a random node partition \((A, B)\)

    while (\(\exists\) improving node \(v\)) {
        if \((v\) is in \(A)\) move \(v\) to \(B\)
        else move \(v\) to \(A\)
    }

    return \((A, B)\)
}
Theorem. Let \((A, B)\) be a locally optimal partition and let \((A^*, B^*)\) be optimal partition. Then \(w(A, B) \geq \frac{1}{2} \sum e \ w_e \geq \frac{1}{2} w(A^*, B^*)\).

\(\uparrow\)

weights are nonnegative

\(\text{Pf.}\)

- Local optimality implies that for all \(u \in A\): \(\sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv}\)

Adding up all these inequalities yields:

\[
2 \sum_{\{u,v\} \subseteq A} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)
\]

- Similarly \(2 \sum_{\{u,v\} \subseteq B} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)\)

- Now, each edge counted once

\[
\sum_{e \in E} w_e = \sum_{\{u,v\} \subseteq A} w_{uv} + \sum_{u \in A, v \in B} w_{uv} + \sum_{\{u,v\} \subseteq B} w_{uv} \leq 2w(A, B)
\]
**Maximum Cut: Big Improvement Flips**

**Local search.** Within a factor of 2 for MAX-CUT, but not poly-time!

**Big-improvement-flip algorithm.** Only choose a node which, when flipped, increases the cut value by at least \( \frac{2\varepsilon}{n} w(A, B) \)

**Claim.** Upon termination, big-improvement-flip algorithm returns a cut \((A, B)\) with \((2 + \varepsilon) w(A, B) \geq w(A^*, B^*)\).

**Pf idea.** Add \( \frac{2\varepsilon}{n} w(A, B) \) to each inequality in original proof.

**Claim.** Big-improvement-flip algorithm terminates after \( O(\varepsilon^{-1} n \log W) \) flips, where \( W = \sum_e w_e \).

- Each flip improves cut value by at least a factor of \((1 + \varepsilon/n)\).
- After \( n/\varepsilon \) iterations the cut value improves by a factor of 2.
- Cut value can be doubled at most \( \log W \) times.

\[
\text{if } x \geq 1, (1 + 1/x)^x \geq 2
\]
Maximum Cut: Context

**Theorem.** [Sahni-Gonzales 1976] There exists a $\frac{1}{2}$-approximation algorithm for MAX-CUT.

- In fact a random cut will cut $\frac{1}{2}$ of all edges in expectation!

**Theorem.** [Goemans-Williamson 1995] There exists an $0.878567$-approximation algorithm for MAX-CUT.

\[
\begin{align*}
\min_{0 \leq \theta \leq \pi} & \quad \frac{2}{\pi} \cdot \frac{\theta}{1 - \cos \theta} \\
& < 0.941176
\end{align*}
\]

**Theorem.** [Håstad 1997] Unless P = NP, no $16/17$ approximation algorithm for MAX-CUT.
12.5 Neighbor Relations
Neighbor Relations for Max Cut

1-flip neighborhood. (A, B) and (A', B') differ in exactly one node.

k-flip neighborhood. (A, B) and (A', B') differ in at most k nodes.
  Θ(n^k) neighbors.

KL-neighborhood. [Kernighan-Lin 1970]
  To form neighborhood of (A, B):
  - Iteration 1: flip node from (A, B) that results in best cut value (A_1, B_1), and mark that node.
  - Iteration i: flip node from (A_{i-1}, B_{i-1}) that results in best cut value (A_i, B_i) among all nodes not yet marked.
  Neighborhood of (A, B) = (A_1, B_1), ..., (A_{n-1}, B_{n-1}).
  Neighborhood includes some very long sequences of flips, but without the computational overhead of a k-flip neighborhood.

Practice: powerful and useful framework.
Theory: explain and understand its success in practice.
12.3 Hopfield Neural Networks
Hopfield Neural Networks

**Hopfield networks.** Simple model of an associative memory, in which a large collection of units are connected by an underlying network, and neighboring units try to correlate their states.

**Input:** Graph $G = (V, E)$ with integer edge weights $w$.

**Configuration.** Node assignment $s_u = \pm 1$.

**Intuition.** If $w_{uv} < 0$, then $u$ and $v$ want to have the same state; if $w_{uv} > 0$ then $u$ and $v$ want different states.

**Note.** In general, no configuration respects all constraints.

![Graph diagram with edge weights 5, 7, and 6]
Def. With respect to a configuration $S$, edge $e = (u, v)$ is **good** if $w_e s_u s_v < 0$. That is, if $w_e < 0$ then $s_u = s_v$; if $w_e > 0$, $s_u \neq s_v$.

Def. With respect to a configuration $S$, a node $u$ is **satisfied** if the weight of incident good edges $\geq$ weight of incident bad edges.

$\sum_{v: e=(u,v) \in E} w_e s_u s_v \leq 0$

Def. A configuration is **stable** if all nodes are satisfied.

**Goal.** Find a stable configuration, if such a configuration exists.
Hopfield Neural Networks

**Goal.** Find a stable configuration, if such a configuration exists.

**State-flipping algorithm.** Repeated flip state of an unsatisfied node.

```plaintext
Hopfield-Flip(G, w) {
    S ← arbitrary configuration

    while (current configuration is not stable) {
        u ← unsatisfied node
        S_u = -S_u
    }

    return S
}
```
State Flipping Algorithm

unsatisfied node
10 - 8 > 0

(a)

(b)

(c)

unsatisfied node
8 - 4 - 1 - 1 > 0

(d)

(e)

(f)

stable
Hopfield Neural Networks

Claim. State-flipping algorithm terminates with a stable configuration after at most $W = \sum e \mid w_e \mid$ iterations.

Pf attempt. Consider measure of progress $\Phi(S) = \#\text{ satisfied nodes}$.

**Conclusion:** Some local flips actually decrease $\#\text{ satisfied nodes}$. 
Hopfield Neural Networks

**Claim.** State-flipping algorithm terminates with a stable configuration after at most $W = \sum |w_e|$ iterations.

**Pf.** Consider measure of progress $\Phi(S) = \sum_{e \text{good}} |w_e|$.

- Clearly $0 \leq \Phi(S) \leq W$.
- We show $\Phi(S)$ increases by at least 1 after each flip.

When $u$ flips state:
- all good edges incident to $u$ become bad
- all bad edges incident to $u$ become good
- all other edges remain the same

$$\Phi(S') = \Phi(S) - \sum_{e: e = (u, v) \in E \text{ bad}} |w_e| + \sum_{e: e = (u, v) \in E \text{ good}} |w_e| \geq \Phi(S) + 1$$

\[\text{u is unsatisfied}\]
Complexity of Hopfield Neural Network

Hopfield network search problem. Given a weighted graph, find a stable configuration if one exists.

Hopfield network decision problem. Given a weighted graph, does there exist a stable configuration?

Remark. The decision problem is trivially solvable (always yes), but there is no known poly-time algorithm for the search problem.

\[\text{polynomial in } n \text{ and } \log W\]
13.6 Universal Hashing
Dictionary Data Type

**Dictionary.** Given a universe $U$ of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in $S$ is efficient.

**Dictionary interface.**

- **Create()**: Initialize a dictionary with $S = \emptyset$.
- **Insert(u)**: Add element $u \in U$ to $S$.
- **Delete(u)**: Delete $u$ from $S$, if $u$ is currently in $S$.
- **Lookup(u)**: Determine whether $u$ is in $S$.

**Challenge.** Universe $U$ can be extremely large so defining an array of size $|U|$ is infeasible.

**Applications.** File systems, databases, Google, compilers, checksums, P2P networks, associative arrays, cryptography, web caching, etc.
Hashing

Hash function. \( h : U \rightarrow \{ 0, 1, ..., n-1 \} \).

Hashing. Create an array \( H \) of size \( n \). When processing element \( u \), access array element \( H[h(u)] \).

Collision. When \( h(u) = h(v) \) but \( u \neq v \).
- A collision is expected after \( \Theta(\sqrt{n}) \) random insertions. This phenomenon is known as the "birthday paradox."
- Separate chaining: \( H[i] \) stores linked list of elements \( u \) with \( h(u) = i \).
Ad Hoc Hash Function

Ad hoc hash function.

```java
int h(String s, int n) {
    int hash = 0;
    for (int i = 0; i < s.length(); i++)
        hash = (31 * hash) + s[i];
    return hash % n;
}
```

hash function ala Java string library

Deterministic hashing. If \(|U| \geq n^2\), then for any fixed hash function \(h\), there is a subset \(S \subseteq U\) of \(n\) elements that all hash to same slot. Thus, \(\Theta(n)\) time per search in worst-case.

Q. But isn't ad hoc hash function good enough in practice?
Algorithmic Complexity Attacks

When can't we live with ad hoc hash function?

- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

malicious adversary learns your ad hoc hash function (e.g., by reading Java API) and causes a big pile-up in a single slot that grinds performance to a halt

Real world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.
Hashing Performance

**Idealistic hash function.** Maps m elements uniformly at random to n hash slots.
- Running time depends on length of chains.
- Average length of chain = $\alpha = m / n$.
- Choose $n \approx m \Rightarrow$ on average $O(1)$ per insert, lookup, or delete.

**Challenge.** Achieve idealized randomized guarantees, but with a hash function where you can easily find items where you put them.

**Approach.** Use randomization in the choice of $h$.  

| adversary knows the randomized algorithm you're using, but doesn't know random choices that the algorithm makes |
Universal Hashing

Universal class of hash functions. [Carter-Wegman 1980s]
- For any pair of elements $u, v \in U$, $\Pr_{h \in H} [ h(u) = h(v) ] \leq 1/n$
- Can select random $h$ efficiently.
- Can compute $h(u)$ efficiently.

Ex. $U = \{ a, b, c, d, e, f \}$, $n = 2$.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1(x)$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$h_2(x)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$H = \{ h_1, h_2 \}$
$\Pr_{h \in H} [ h(a) = h(b) ] = 1/2$
$\Pr_{h \in H} [ h(a) = h(c) ] = 1$ not universal
$\Pr_{h \in H} [ h(a) = h(d) ] = 0$
...

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1(x)$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$h_2(x)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$h_3(x)$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$h_4(x)$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$H = \{ h_1, h_2, h_3, h_4 \}$
$\Pr_{h \in H} [ h(a) = h(b) ] = 1/2$
$\Pr_{h \in H} [ h(a) = h(c) ] = 1/2$
$\Pr_{h \in H} [ h(a) = h(d) ] = 1/2$
$\Pr_{h \in H} [ h(a) = h(e) ] = 1/2$
$\Pr_{h \in H} [ h(a) = h(f) ] = 0$
...

universal
Universal Hashing

Universal hashing property. Let $H$ be a universal class of hash functions; let $h \in H$ be chosen uniformly at random from $H$; and let $u \in U$. For any subset $S \subseteq U$ of size at most $n$, the expected number of items in $S$ that collide with $u$ is at most 1.

Pf. For any element $s \in S$, define indicator random variable $X_s = 1$ if $h(s) = h(u)$ and 0 otherwise. Let $X$ be a random variable counting the total number of collisions with $u$.

$$E_{h \in H}[X] = E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr[X_s = 1] \leq \sum_{s \in S} \frac{1}{n} = |S| \frac{1}{n} \leq 1$$

- linearity of expectation
- $X_s$ is a 0-1 random variable
- universal (assumes $u \not\in S$)
Designing a Universal Family of Hash Functions

**Theorem.** [Chebyshev 1850] There exists a prime between $n$ and $2n$.

**Modulus.** Choose a prime number $p \approx n$. ← no need for randomness here

**Integer encoding.** Identify each element $u \in U$ with a base-$p$ integer of $r$ digits: $x = (x_1, x_2, \ldots, x_r)$.

**Hash function.** Let $A =$ set of all $r$-digit, base-$p$ integers. For each $a = (a_1, a_2, \ldots, a_r)$ where $0 \leq a_i < p$, define

$$h_a(x) = \left( \sum_{i=1}^{r} a_i x_i \right) \mod p$$

**Hash function family.** $H = \{ h_a : a \in A \}$. 
Theorem. $H = \{ h_a : a \in A \}$ is a universal class of hash functions.

Pf. Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of $U$. We need to show that $Pr[h_a(x) = h_a(y)] \leq 1/n$.

- Since $x \neq y$, there exists an integer $j$ such that $x_j \neq y_j$.
- We have $h_a(x) = h_a(y)$ iff

$$a_j (y_j - x_j) + \sum_{i \neq j} a_i (x_i - y_i) \mod p$$

- Can assume $a$ was chosen uniformly at random by first selecting all coordinates $a_i$ where $i \neq j$, then selecting $a_j$ at random. Thus, we can assume $a_i$ is fixed for all coordinates $i \neq j$.

- Since $p$ is prime, $a_j z = m \mod p$ has at most one solution among $p$ possibilities. $\leftarrow$ see lemma on next slide

- Thus $Pr[h_a(x) = h_a(y)] = 1/p \leq 1/n$. ▪
**Number Theory Facts**

**Fact.** Let $p$ be prime, and let $z \neq 0 \mod p$. Then

$$\alpha z = m \mod p$$
has at most one solution $0 \leq \alpha < p$.

**Pf.**
- Suppose $\alpha$ and $\beta$ are two different solutions.
- Then $(\alpha - \beta)z = 0 \mod p$; hence $(\alpha - \beta)z$ is divisible by $p$.
- Since $z \neq 0 \mod p$, we know that $z$ is not divisible by $p$; it follows that $(\alpha - \beta)$ is divisible by $p$.
- This implies $\alpha = \beta$. □

**Bonus fact.** Can replace "at most one" with "exactly one" in above fact.

**Pf idea.** Euclid's algorithm.
Extra Slides