**CS 580: Algorithm Design and Analysis**

Jeremiah Blocki  
Purdue University  
Spring 2018

**Reminder:** Homework 6 has been released.

---

**Weighted Vertex Cover**

**Theorem.** 2-approximation algorithm for weighted vertex cover via LP rounding.

**Theorem.** [Dinur-Safra 2001] If $P \neq NP$, then no $\rho$-approximation for $\rho < 1.3607$, even with unit weights.

10 \( \sqrt{5} \) - 21

**Open research problem.** Close the gap.

**Unique Games Conjecture:** Implies there is no $\rho$-approximation for $\rho < 1.99999$, even with unit weights

Disagreement among about validity of this conjecture

---

**11.8 Knapsack Problem**

**Polynomial Time Approximation Scheme**

**PTAS.** $$(1 + \varepsilon)$$-approximation algorithm for any constant $\varepsilon > 0$.

- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora 1996]

**Consequence.** PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

**This section.** PTAS for knapsack problem via rounding and scaling.
Knapsack Problem

Knapsack problem.
- Given \( n \) objects and a "knapsack."
- Item \( i \) has value \( v_i > 0 \) and weighs \( w_i > 0 \).
- Knapsack can carry weight up to \( W \).
- Goal: fill knapsack so as to maximize total value.

Ex: \( \{3, 4\} \) has value 40.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

\( W = 11 \)

Knapsack Problem: Dynamic Programming 1

Def. \( \text{OPT}(i, w) = \max \text{ value subset of items } 1, \ldots, i \text{ with weight limit } w. \)
- Case 1: \( \text{OPT} \) does not select item \( i \).
  - \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) using up to weight limit \( w \)
- Case 2: \( \text{OPT} \) selects item \( i \).
  - new weight limit = \( w - w_i \)
  - \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) using up to weight limit \( w - w_i \)

\[
\text{OPT}(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
\text{OPT}(i-1, w) & \text{if } w_i > w \\
\max \{ \text{OPT}(i-1, w), v_i + \text{OPT}(i-1, w-w_i) \} & \text{otherwise}
\end{cases}
\]

Running time. \( O(n W) \).
- \( W = \) weight limit.
- \( \text{Not polynomial} \) in input size!

Knapsack Problem: Dynamic Programming II

Def. \( \text{OPT}(i, v) = \min \text{ weight subset of items } 1, \ldots, i \text{ that yields value exactly } v \).
- Case 1: \( \text{OPT} \) does not select item \( i \).
  - \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) that achieves exactly value \( v \)
- Case 2: \( \text{OPT} \) selects item \( i \).
  - consumes weight \( w_i \), new value needed = \( v - v_i \)
  - \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) that achieves exactly value \( v \)

\[
\text{OPT}(i, v) = \begin{cases} 
0 & \text{if } v = 0 \\
\infty & \text{if } i = 0, v > 0 \\
\text{OPT}(i-1, v) & \text{if } v_i > v \\
\min \{ \text{OPT}(i-1, v), w_i + \text{OPT}(i-1, v-v_i) \} & \text{otherwise}
\end{cases}
\]

Running time. \( O(n V^*) = O(n^2 V_{\max}) \).
- \( V^* = \text{optimal value} = \max \text{ value } v \text{ such that } \text{OPT}(n, v) \leq W. \)
- \( \text{Not polynomial} \) in input size!
Knapsack: FPTAS

Intuition for approximation algorithm.
- Round all values up to lie in smaller range.
- Run dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934,221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5,956,342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17,810,013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21,217,800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27,343,199</td>
<td>7</td>
</tr>
</tbody>
</table>

$W = 11$

original instance

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\tau = \left\lceil \frac{v_i}{\theta} \right\rceil, 0$

Theorem. If $S$ is solution found by our algorithm and $S^*$ is any other feasible solution then

$\sum_{i \in S^*} v_i \geq \sum_{i \in S} \theta \left\lceil \frac{v_i}{\theta} \right\rceil$

Pf. Let $S^*$ be any feasible solution satisfying weight constraint.

- $\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \theta \left\lceil \frac{v_i}{\theta} \right\rceil$ always round up
- $\sum_{i \in S} \theta \left\lceil \frac{v_i}{\theta} \right\rceil$ solve rounded instance optimally
- $\sum_{i \in S} (\theta \left\lceil \frac{v_i}{\theta} \right\rceil)$ never round up by more than $\theta$
- $\sum_{i \in S} v_i + n\theta$ $|S| \leq n$
- $\sum_{i \in S} v_i + n\theta$ $\theta \leq \sum_{i \in S} v_i + n\theta$ of alg can take $v_{\max}$
- $(1 + \epsilon) \sum_{i \in S} \theta \left\lceil \frac{v_i}{\theta} \right\rceil$ if alg can take $v_{\max}$

$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \frac{n}{\epsilon}$

Intuition. $\tau$ close to $v$ so optimal solution using $\tau$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm is fast.

Running time. $O(n^3 / \epsilon)$
- Dynamic program II running time is $O(n^2 \hat{v}_{\max})$, where $\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \frac{n}{\epsilon}$
Randomization

Algorithmic design patterns.
• Greedy.
• Divide-and-conquer.
• Dynamic programming.
• Network flow.
• Randomization.

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.

13.1 Contention Resolution

Contestation Resolution in a Distributed System

Contestation resolution. Given n processes P1, ..., Pn, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can’t communicate.

Challenge. Need symmetry-breaking paradigm.

Contention Resolution: Randomized Protocol

Protocol. Each process requests access to the database at time t with probability p = 1/n.

Claim. Let S(i, t) = event that process i succeeds in accessing the database at time t. Then 1/(e · n) ≤ Pr[S(i, t)] ≤ 1/(2n).

Pf. By independence, Pr[S(i, t)] = p (1-p)n-1.

• Setting p = 1/n, we have Pr[S(i, t)] = 1/n (1 - 1/n)n-1.

Useful facts from calculus. As n increases from 2, the function:
• (1 - 1/n)n converges monotonically from 1/4 up to 1/e
• (1 - 1/n)n-1 converges monotonically from 1/2 down to 1/e.
**Contestation Resolution: Randomized Protocol**

**Claim.** The probability that process \( i \) fails to access the database in \( en \) rounds is at most \( 1/e \). After \( en(\ln n) \) rounds, the probability is at most \( n^{-c} \).

**Pf.** Let \( F[i, t] \) = event that process \( i \) fails to access database in rounds \( 1 \) through \( t \). By independence and previous claim, we have

\[
Pr[F(i, t)] \leq (1 - 1/(en))^t.
\]

- Choose \( t = \lceil e \cdot n \rceil \): \( Pr[F(i, t)] \leq \left(1 - \frac{1}{en}\right)^{\lceil e \cdot n \rceil} \leq \left(1 - \frac{1}{en}\right)^n \leq \frac{1}{e} \)
- Choose \( t = \lceil e \cdot n \rceil (\ln n) \): \( Pr[F(i, t)] \leq \left(\frac{1}{e}\right)^{\ln n} = n^{-c} \)

---

**Global Minimum Cut**

**Global min cut.** Given a connected, undirected graph \( G = (V, E) \) find a cut \( (A, B) \) of minimum cardinality.

**Applications.** Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

**Network flow solution.**
- Replace every edge \((u, v)\) with two antiparallel edges \((u, v)\) and \((v, u)\).
- Pick some vertex \( s \) and compute \( \min s-v \) cut separating \( s \) from each other vertex \( v \in V \).

**False intuition.** Global min-cut is harder than min s-t cut.
**Contraction Algorithm**

**Contraction Algorithm.** [Karger 1995]

- Pick an edge \( e = (u, v) \) uniformly at random.
- **Contract** edge \( e \):
  - replace \( u \) and \( v \) by single new super-node \( w \)
  - preserve edges, updating endpoints of \( u \) and \( v \) to \( w \)
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes \( v_1 \) and \( v_2 \).
- Return the cut (all nodes that were contracted to form \( v_j \)).

**Claim.** The contraction algorithm returns a min cut with prob \( \geq 2/n^2 \).

**Pf.** Consider a global min-cut \((A^*, B^*)\) of \( G \). Let \( F^* \) be edges with one endpoint in \( A^* \) and the other in \( B^* \). Let \( k = |F^*| = \text{size of min cut} \).

- In first step, algorithm contracts an edge in \( F^* \) with prob \( k / |E| \).
- Every node has degree \( \geq k \) since otherwise \((A^*, B^*)\) would not be min-cut. \( \Rightarrow |E| \geq \frac{1}{2}kn \).
- Thus, algorithm contracts an edge in \( F^* \) with prob \( \leq 2/n \).

**Amplification.** To amplify the probability of success, run the contraction algorithm many times.

**Claim.** If we repeat the contraction algorithm \( n^2 \ln n \) times with independent random choices, the probability of failing to find the global min-cut is at most \( 1/n^2 \).

**Pf.** Consider a global min-cut \((A^*, B^*)\) of \( G \). Let \( F^* \) be edges with one endpoint in \( A^* \) and the other in \( B^* \). Let \( k = |F^*| = \text{size of min cut} \).

- Let \( G' \) be graph after \( j \) iterations. There are \( n' = n - j \) supernodes.
- Suppose no edge in \( F^* \) has been contracted. The min-cut in \( G' \) is still \( k \).
- Since value of min-cut is \( k \), \( |E'| \geq \frac{1}{2}kn' \).
- Thus, algorithm contracts an edge in \( F^* \) with prob \( \leq 2/n' \).
- Let \( E_j \) be event that an edge in \( F^* \) is not contracted in iteration \( j \).
  \[
  \Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}] = \Pr[E_1] \times \Pr[E_2 | E_1] \times \cdots \times \Pr[E_{n-2} | E_1 \cap E_2 \cdots \cap E_{n-3}]
  \geq \left( \frac{1}{2} \right)^{n-2} \left( \frac{1}{2} \right)^{n-2} \cdots \left( \frac{1}{2} \right)^{n-2}
  = \left( \frac{1}{2} \right)^{n-2} \left( \frac{1}{2} \right) \left( \frac{1}{2} \right) \left( \frac{1}{2} \right)
  = \frac{1}{2^{n-2}} \left( \frac{1}{2} \right)^{3/2}
  \geq \frac{1}{n^2}
  \]

Global Min Cut: Context

**Remark.** Overall running time is slow since we perform $\Theta(n^2 \log n)$ iterations and each takes $\Omega(m)$ time.

**Improvement.** [Karger-Stein 1996] $O(n^2 \log^3 n)$.
- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm until $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

**Extensions.** Naturally generalizes to handle positive weights.

**Best known.** [Karger 2000] $O(m \log^3 n)$.

13.3 Linearity of Expectation

**Expectation.** Given a discrete random variables $X$, its expectation $E[X]$ is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$$

**Waiting for a first success.** Coin is heads with probability $p$ and tails with probability $1-p$. How many independent flips $X$ until first heads?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j \cdot (1-p)^{j-1} \cdot p = \frac{p}{1-p} \cdot \frac{1}{1-p} = \frac{1}{p}$$

**Expectation: Two Properties**

**Useful property.** If $X$ is a 0/1 random variable, $E[X] = \Pr[X = 1]$.

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \Pr[X=1]$$

**Linearity of expectation.** Given two random variables $\hat{X}$ and $\hat{Y}$ defined over the same probability space, $E[\hat{X} + \hat{Y}] = E[\hat{X}] + E[\hat{Y}]$.

Decouples a complex calculation into simpler pieces.
Guessing Cards

**Game.** Shuffle a deck of $n$ cards; turn them over one at a time; try to guess each card.

**Memoryless guessing.** No psychic abilities; can’t even remember what’s been turned over already. Guess a card from full deck uniformly at random.

**Claim.** The expected number of correct guesses is 1.

**Pf.** (surprisingly effortless using linearity of expectation)
- Let $X_i = 1$ if $i$th prediction is correct and 0 otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1/n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1$.

Coupon Collector

**Coupon collector.** Each box of cereal contains a coupon. There are $n$ different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have $\geq 1$ coupon of each type?

**Claim.** The expected number of steps is $\Theta(n \log n)$.

**Pf.**
- Phase $j = \text{time between } j \text{ and } j+1 \text{ distinct coupons}.$
- Let $X_j = \text{number of steps you spend in phase } j$.
- Let $X = \text{number of steps in total} = X_0 + X_1 + \ldots + X_{n-1}$.

$$E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n H(n)$$

$$\ln(n+1) < H(n) < 1 + \ln n$$
Maximum 3-Satisfiability

**MAX-3SAT.** Given 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

\[
\begin{align*}
C_1 &= x_2 \lor x_3 \lor \overline{x_4} \\
C_2 &= x_2 \lor x_3 \lor \overline{x_4} \\
C_3 &= \overline{x_1} \lor x_2 \lor x_4 \\
C_4 &= \overline{x_1} \lor x_2 \lor x_4 \\
C_5 &= x_1 \lor x_2 \lor \overline{x_4}
\end{align*}
\]

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \(\frac{1}{2}\), independently for each variable.

---

Maximum 3-Satisfiability: Analysis

**Claim.** Given a 3-SAT formula with \(k\) clauses, the expected number of clauses satisfied by a random assignment is \(7k/8\).

**Pf.** Consider random variable \(Z_j\) such that:

- Let \(Z\) = weight of clauses satisfied by assignment \(Z_j\)

\[
\begin{align*}
E[Z] &= \sum_{j=1}^{k} E[Z_j] \\
&= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \\
&= \frac{7}{8}k
\end{align*}
\]

---

The Probabilistic Method

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \(7/8\) fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time. 

---

Probabilistic method. We showed the existence of a non-obvious property of 3-SAT by showing that a random construction produces it with positive probability!

---

Maximum 3-Satisfiability: Analysis

**Q.** Can we turn this idea into a \(7/8\)-approximation algorithm? In general, a random variable can almost always be below its mean.

**Lemma.** The probability that a random assignment satisfies \(\geq 7k/8\) clauses is at least \(1/(8k)\).

**Pf.** Let \(p_j\) be probability that exactly \(j\) clauses are satisfied; let \(p\) be probability that \(\geq 7k/8\) clauses are satisfied.

\[
\begin{align*}
\frac{7}{8}k &= E[Z] = \sum_{j=0}^{k} j p_j \\
&= \sum_{j=7/8k}^{k} j p_j + \sum_{j<7/8k} j p_j \\
&\leq \left(\frac{7}{8} - \frac{1}{2}\right) \sum_{j=7/8k}^{k} p_j + k \sum_{j=7/8k}^{k} p_j \\
&\leq \left(\frac{7}{8} - \frac{1}{2}\right) \cdot 1 + 1 \cdot p \\
\end{align*}
\]

Rearranging terms yields \(p \geq 1/(8k)\).
Maximum 3-Satisfiability: Analysis

**Johnson’s algorithm.** Repeatedly generate random truth assignments until one of them satisfies $\geq 7k/8$ clauses.

**Theorem.** Johnson’s algorithm is a 7/8-approximation algorithm.

**Pf.** By previous lemma, each iteration succeeds with probability at least $1/(8k)$. By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most $8k$.

---

Monte Carlo vs. Las Vegas Algorithms

**Monte Carlo algorithm.** Guaranteed to run in poly-time, likely to find correct answer.

**Ex:** Contraction algorithm for global min cut.

**Las Vegas algorithm.** Guaranteed to find correct answer, likely to run in poly-time.

**Ex:** Randomized quicksort, Johnson’s MAX-3SAT algorithm.

Stop algorithm after a certain point

**Remark.** Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

---

Maximum Satisfiability

**Extensions.**
- Allow one, two, or more literals per clause.
- Find max weighted set of satisfied clauses.

**Theorem.** [Asano-Williamson 2000] There exists a 0.784-approximation algorithm for MAX-SAT.

**Theorem.** [Karloff-Zwick 1997, Zwick+computer 2002] There exists a 7/8-approximation algorithm for version of MAX-3SAT where each clause has at most 3 literals.

**Theorem.** [Håstad 1997] Unless $P = NP$, no $\rho$-approximation algorithm for MAX-3SAT (and hence MAX-SAT) for any $\rho > 7/8$.

---

Monte Carlo

**Monte Carlo algorithm.** Guaranteed to run in poly-time, likely to find correct answer.

**Ex:** Contraction algorithm for global min cut.

---

Las Vegas

**Las Vegas algorithm.** Guaranteed to find correct answer, likely to run in poly-time.

**Ex:** Randomized quicksort, Johnson’s MAX-3SAT algorithm.

Stop algorithm after a certain point

**Remark.** Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

---

Monte Carlo vs. Las Vegas Algorithms

**Monte Carlo algorithm.** Guaranteed to run in poly-time, likely to find correct answer.

**Ex:** Contraction algorithm for global min cut.

**Las Vegas algorithm.** Guaranteed to find correct answer, likely to run in poly-time.

**Ex:** Randomized quicksort, Johnson’s MAX-3SAT algorithm.

Stop algorithm after a certain point

**Remark.** Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

---

RP and ZPP

**RP.** [Monte Carlo] Decision problems solvable with one-sided error in poly-time.

**One-sided error.**
- If the correct answer is no, always return no.
- If the correct answer is yes, return yes with probability $\geq \frac{1}{2}$.

**ZPP.** [Las Vegas] Decision problems solvable in expected poly-time.

**Remark.** Can decrease probability of false negative to $2^{-100}$ by 100 independent repetitions running time can be unbounded, but on average it is fast

**Theorem.** $P \subseteq ZPP \subseteq RP \subseteq NP$.

**Fundamental open questions.** To what extent does randomization help? Does $P \subseteq ZPP$? Does $ZPP \subseteq RP$? Does $RP \subseteq NP$?
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 $\Theta(n)$ 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;
}
```

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.

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 $n = m \implies$ 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 learns 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>$h(a)$</th>
<th>$h(b)$</th>
<th>$h(c)$</th>
<th>$h(d)$</th>
<th>$h(e)$</th>
<th>$h(f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$H \ni h \implies Pr_{h \in H}(h(a) = h(b)) = 1/2$
$Pr_{h \in H}(h(a) = h(c)) = 1/2$
$Pr_{h \in H}(h(a) = h(d)) = 1/2$
$Pr_{h \in H}(h(a) = h(e)) = 1/2$
$Pr_{h \in H}(h(a) = h(f)) = 0$

Universal 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 $u$.

$$E[X] = E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} Pr[X_s = 1] = \sum_{s \in S} \frac{1}{2} = |S| \frac{1}{2} \leq 1$$

(assumes $u \notin S$)
Designing a Universal Family of Hash Functions

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

**Modulus.** Choose a prime number $p \approx n$.

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

**Hash function.** Let $A$ be the set of all $r$-digit, base-$p$ integers. For each $a = (a_1, a_2, ..., a_r)$ 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.** $H = \{ h_a : a \in A \}$.

Designing a Universal Class of Hash Functions

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

**Pf.** Let $x = (x_1, x_2, ..., x_r)$ and $y = (y_1, y_2, ..., y_r)$ 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

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

- 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 $a_j$ 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.
- 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.

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

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

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

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

- $f(x) = e^t$ is monotone in $x$.
- Markov's inequality: $P[X > a] \leq E[X]/a$.

- Now, $E[e^{tX}] = E[e^{t\sum X}] = \prod_i E[e^{tX_i}]$
  $$= e^{t\sum pi} = e^{t\mu}.$$

- Definition of $X$ and independence.

- Combining everything:
  $$P[X > (1 + \delta)\mu] \leq e^{-t(1+\delta)\mu} \cdot \prod_i E[e^{tX_i}] \leq e^{-t(1+\delta)\mu} \cdot e^{t\mu} \leq e^{-t(1+\delta)\mu} \cdot e^{t\mu}.$$

- Finally, choose $t = \ln(1 + \delta)$.

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

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

**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 \( \lceil \frac{m}{n} \rceil \) jobs.

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

Load Balancing: Many 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

\[
\Pr[X_i > 2\mu] < \left( \frac{e}{4} \right)^{\mu/2} < \left( \frac{1}{2} \right)^{1/n} = \frac{1}{n}
\]

\[
\Pr[X_i < \frac{1}{2}\mu] < e^{2\mu \gamma(n)} = \frac{1}{n}
\]
- Union bound \( \Rightarrow \) every processor has load between half and twice the average with probability \( \geq 1 - 2/n \).

Analysis.
- 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

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

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

\[
\Pr[X_i > e^{c-1}] < \frac{e^{c-1}}{c} < \left( \frac{1}{\gamma(n)} \right)^{\mu/c} < \left( \frac{1}{\gamma(n)} \right)^{\mu/c} = \frac{1}{n}
\]
- Union bound \( \Rightarrow \) with probability \( \geq 1 - 1/n \) no processor receives more than \( e \gamma(n) = \Theta(\log n / \log \log n) \) jobs.

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

Fact: this bound is asymptotically tight: with high probability, some processor receives \( \Theta(\log n / \log \log n) \) jobs.

Extra Slides
13.5 Randomized Divide-and-Conquer

Quicksort

Sorting. Given a set of $n$ distinct elements $S$, rearrange them in ascending order.

Quicksort($S$) {
    if $|S| = 0$ return
    choose a splitter $a_i \in S$ uniformly at random
    foreach ($a \in S$) {
        if ($a < a_i$) put $a$ in $S^-$
        else if ($a > a_i$) put $a$ in $S^+$
    }
    RandomizedQuicksort($S^-$)
    output $a_i$
    RandomizedQuicksort($S^+$)
}

Remark. Can implement in-place.

$O(| \log n |)$ extra space

Quicksort

Running time.
- [Best case.] Select the median element as the splitter:
  quicksort makes $\Omega(|n \log n |)$ comparisons.
- [Worst case.] Select the smallest element as the splitter:
  quicksort makes $\Omega(|n^2 |)$ comparisons.

Randomize. Protect against worst case by choosing splitter at random.

Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $\Omega(|n \log n |)$ comparisons.

Notation. Label elements so that $x_1 < x_2 \ldots < x_n$.

Quicksort: BST Representation of Splitters

BST representation. Draw recursive BST of splitters.

$S^-$ $x_8$ $x_7$ $x_6$ $x_5$ $x_4$ $x_3$ $x_2$ $x_1$

$S^+$

First splitter, chosen uniformly at random

Copyright 2000, Kevin Wayne
Quicksort: BST Representation of Splitters

Observation. Element only compared with its ancestors and descendants.
- \( x_2 \) and \( x_7 \) are compared if their lca = \( x_2 \) or \( x_7 \).
- \( x_2 \) and \( x_8 \) are not compared if their lca = \( x_3 \) or \( x_4 \) or \( x_5 \) or \( x_6 \).

Claim. \( \Pr[ x_i \text{ and } x_j \text{ are compared}] = \frac{2}{|j - i + 1|}. \)

Theorem. Expected # of comparisons is \( O(n \log n) \).

Pf. 

\[
\sum_{i \geq j \geq k \geq x} \frac{2}{j - i + 1} = \frac{2}{4} \sum_{j \geq 2} \frac{1}{j} \leq \frac{2n}{\sum_{j \geq 2} j^{-1}} \approx 2n \int_{x=1}^{x=n} \frac{1}{x \log x} \, dx = 2n \ln n
\]

probability that \( i \) and \( j \) are compared

Theorem. [Knuth 1973] Stddev of number of comparisons is \( \sim 0.65N \).

Ex. If \( n = 1 \) million, the probability that randomized quicksort takes less than \( 4n \ln n \) comparisons is at least 99.94%.

Chebyshev’s inequality. \( \Pr[|X - \mu| \geq k\delta] \leq \frac{1}{k^2}. \)