Announcements: Homework 6 deadline extended to April 24th at 11:59 PM

Course Evaluation Survey: Live until 4/29/2018 at 11:59PM. Your feedback is valued!
13.4 MAX 3-SAT
Recap: Maximum 3-Satisfiability

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

\[ C_1 = x_2 \lor \overline{x}_3 \lor \overline{x}_4 \]
\[ C_2 = x_2 \lor x_3 \lor \overline{x}_4 \]
\[ C_3 = \overline{x}_1 \lor x_2 \lor x_4 \]
\[ C_4 = \overline{x}_1 \lor \overline{x}_2 \lor x_3 \]
\[ C_5 = x_1 \lor x_2 \lor x_4 \]

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

Observation. Random assignment satisfies \( \frac{7k}{8} \) of the k clauses in expectation (proof: linearity of expectation)
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.

$$
\frac{7}{8} k = E[Z] = \sum_{j \geq 0} j \cdot p_j = \sum_{j < \frac{7}{8}k} j \cdot p_j + \sum_{j \geq \frac{7}{8}k} j \cdot p_j
$$

$$
\leq \left(\frac{7k}{8} - \frac{1}{8}\right) \sum_{j < \frac{7}{8}k} p_j + k \sum_{j \geq \frac{7}{8}k} p_j \leq \left(\frac{7k}{8} - \frac{1}{8}\right) \cdot 1 + kp
$$

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 $8k$. □
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 $\rho > 7/8$.

very unlikely to improve over simple randomized algorithm for MAX-3SAT
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.

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.

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

13.5 Randomized Divide-and-Conquer
QuickSort

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

```c
RandomizedQuicksort(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.

\[ O(\log n) \text{ extra space} \]
Quicksort

Running time.

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

first splitter, chosen uniformly at random
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 $x_7$ are not compared if their lca = $x_3$ or $x_4$ or $x_5$ or $x_6$.

Claim. $\Pr[x_i \text{ and } x_j \text{ are compared}] = \frac{2}{|j-i+1|}$. 
Quicksort: Expected Number of Comparisons

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

Pf.

$$\sum_{1 \leq i < j \leq n} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n} \sum_{j=2}^{i} \frac{1}{j} \leq 2n \sum_{j=1}^{n} \frac{1}{j} \approx 2n \int_{x=1}^{n} \frac{1}{x} \, dx = 2n \ln n$$

↑

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. $\Pr[|X - \mu| \geq k\delta] \leq 1 / k^2$. 
13.9 Chernoff Bounds
Chernoff Bounds (above mean)

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

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

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

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

  $f(x) = e^{tx}$ is monotone in $x$  
  Markov's inequality: $\Pr[X > a] \leq E[X] / a$

- Now
  $$E[e^{tX}] = E[e^{t\sum_i X_i}] = \prod_i E[e^{tX_i}]$$

  definition of $X$  
  independence
Chernoff Bounds (above mean)

**Pf.** (cont)

- Let $p_i = \Pr[X_i = 1]$. Then,

$$E[e^{tX_i}] = p_i e^t + (1 - p_i) e^0 = 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)}$$

for any $\alpha \geq 0$, $1 + \alpha \leq e^\alpha$

- Combining everything:

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

previous slide

inequality above

$$\sum_i p_i = E[X] \leq \mu \leq e^{-t(1+\delta)\mu} e^{\mu(e^t - 1)}$$

- 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

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

**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?
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 $Pr[X_i > c] < \frac{e^{c-1}}{c^c}$

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

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

- 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.
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
  \[
  \Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16n\ln n} < \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n^2}
  \]

  \[
  \Pr[X_i < \frac{1}{2}\mu] < e^{-\frac{1}{2}(\frac{1}{2})^2(16n\ln n)} = \frac{1}{n^2}
  \]

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


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 $S = \phi$.
- **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. \( h : U \rightarrow \{ 0, 1, \ldots, n-1 \} \).

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;
}
```

**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 = \( \alpha = \frac{m}{n} \).
- 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. \( U = \{ a, b, c, d, e, f \} \), \( n = 2 \).

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\hline
\text{h}_1(x) & 0 & 1 & 0 & 1 & 0 & 1 \\
\text{h}_2(x) & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

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

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\hline
\text{h}_1(x) & 0 & 1 & 0 & 1 & 0 & 1 \\
\text{h}_2(x) & 0 & 0 & 0 & 1 & 1 & 1 \\
\text{h}_3(x) & 0 & 0 & 1 & 0 & 1 & 1 \\
\text{h}_4(x) & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

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

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 $u$.

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

\[\uparrow \quad \uparrow \quad \uparrow\]

linearity of expectation \hspace{1cm} X_s \text{ is a 0-1 random variable} \hspace{1cm} \text{universal (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 $p \approx n$. ➡️ 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.** $H = \{ h_a : a \in A \}$. 

32
Designing a Universal Class of Hash Functions

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

**Pf.** Let \( x = (x_1, x_2, \ldots, x_r) \) 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
  
  \[
  a_j (y_j - x_j) \equiv \sum_{i \neq j} a_i (x_i - y_i) \mod p
  \]

  - 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 \( a_i \) 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. \( \text{see lemma on next slide} \)
  - 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.