CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

Reminders: Homework 6 due April 23 at 11:59 PM

Course Evaluation: Your feedback
is valued! Live until April 28th at 11:59PM
http://www.purdue.edu/idp/courseevaluations/CE_Students.html

13.4 MAX 3-SAT

Maximum 3-Satisfiability

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

exactly 3 distinct literals per clause

$$
\begin{aligned}
C_1 &= x_2 \vee \overline{x_3} \vee \overline{x_4} \\
C_2 &= x_2 \vee x_3 \vee \overline{x_4} \\
C_3 &= \overline{x_1} \vee x_2 \vee x_4 \\
C_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\
C_5 &= x_1 \vee \overline{x_2} \vee \overline{x_4}
\end{aligned}
$$

Remark. NP-hard search problem.

Simple idea. Flip a coin, and set each variable true with probability ½, independently for each variable.

3

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 $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$

- Let $Z = \sum_{j \le k} Z_j$ be the weight of clauses satisfied by assignment.

linearity of expectation

$$
\begin{aligned}
\mathbf{E}[Z] &= \sum_{j=1}^{k} \mathbf{E}[Z_j] \\
&= \sum_{j=1}^{k} \Pr[\text{Clause } C_j \text{ is satisfied}] = \frac{7}{8}k
\end{aligned}
$$

4

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!

5

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 ≥ 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 ≥ 7k/8 clauses are satisfied.

$$
\begin{aligned}
\frac{7}{8}k = \mathbf{E}[Z] &= \sum_{j \ge 0} j \cdot p_j = \sum_{j < \frac{7}{8}k} j \cdot p_j + \sum_{j \ge \frac{7}{8}k} j \cdot p_j \\
&\le \left(\frac{7k}{8} - \frac{1}{8}\right) \sum_{j < \frac{7}{8}k} p_j + k \sum_{j \ge \frac{7}{8}k} p_j \le \left(\frac{7k}{8} - \frac{1}{8}\right) \cdot 1 + kp
\end{aligned}
$$

Rearranging terms yields p ≥ 1 / (8k). ▪

6

Maximum 3-Satisfiability: Analysis

Johnson's algorithm. Repeatedly generate random truth assignments until one of them satisfies ≥ 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. ▪

7

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 ρ-approximation algorithm for MAX-3SAT (and hence MAX-SAT) for any ρ > 7/8.

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

8

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.

9

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 ≥ ½.

Can decrease probability of false negative to 2^{-100} by 100 independent repetitions

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

running time can be unbounded, but on average it is fast

Theorem. P ⊆ ZPP ⊆ RP ⊆ NP.

Fundamental open questions. To what extent does randomization help? Does P = ZPP? Does ZPP = RP? Does RP = NP?

10

Polynomial Identity Testing

Given a polynomial $p(x_1, \ldots, x_n)$ we want to know if $p(x_1, \ldots, x_n) = 0$
- Example 1: $p(x, y) = (x + y)(x - y) - x^2 + y^2$
  - **Answer:** YES! After expanding and canceling…
- Example 2: $p(x, y) = (x + y)(x + y) - x^2 - y^2$
  - **Answer:** NO! After expanding we get $p(x, y) = 2xy$
- Example 3: $p(x, y, z) = (x + 2y)(3y - z) - 3xy - 6y^2 + xz + 2yz$
  - **Answer:** YES! But checking is getting more complicated

Approach 1: Expand and cancel
- Takes up to $\binom{n+d}{d}$ steps for degree d polynomial (exponential in d)

Approach 2: Randomize!

Theorem [Schwartz-Zippel]: Suppose $p(x_1, \ldots, x_n)$ is not identically zero and has degree d. Then given any finite set $S \subseteq \mathbb{R}$ picking $y_1, \ldots, y_n \sim S$ uniformly at random we have

$$Pr[p(y_1, \ldots, y_n) = 0] \leq \frac{d}{|S|}$$

11

Polynomial Identity Testing

Approach 1: Expand and cancel
- Takes up to $\binom{n+d}{d}$ steps for degree d polynomial (exponential in d)

Approach 2: Randomize!

Theorem [Schwartz-Zippel]: Suppose $p(x_1, \ldots, x_n) \neq 0$ is not identically zero and has degree d. Then given any finite set $S \subseteq \mathbb{R}$ picking $y_1, \ldots, y_n \sim S$ uniformly at random we have

$$Pr[p(y_1, \ldots, y_n) = 0] \leq \frac{d}{|S|}$$

Example: if $S = \{1, \ldots, 2d\}$ then $\Pr[p(y_1, \ldots, y_n) = 0] \leq \frac{1}{2}$
- Repeat k times if $p(x_1, \ldots, x_n) \neq 0$ → $\Pr[\text{Output } 0] \leq \frac{1}{2^k}$
- One Sided Error: Polynomial Identity testing in RP
- No known deterministic/polynomial time algorithm!

Remark: Schwartz-Zippel also holds for other fields $\mathbb{F}$

12

Polynomial Identity Testing and Perfect Matchings

Example 4: Given a bipartite graph G with nodes (V,U) and let

$$A[u,v] = \begin{cases} 0 & \text{otherwise} \\ x_{u,v} & \text{if } (u,v) \in E(G) \end{cases}$$

Be the Edmonds Matrix then $\det(A)$ is a polynomial of degree n

$$\det(A) = \sum_{\pi} c(\pi) \prod_{u \in U} A\,[u, \pi(u)\,]$$

Theorem: G has a perfect matching if and only if $\det(A)$ is identically 0.

Implication: Randomized algorithm to test if G has a perfect matching (and find one if it exists) in time $O(n^{\omega})$

- Remark 1: Similar Approach works for Non-Bipartite Graphs [using determinant of Tutte Matrix]
- Remark 2: Improves on best known deterministic algorithm for dense graphs

Recall: $\omega \le 2.373$ for fastest matrix multiplication algorithms

13

Randomized Primality Test

Input: n
Output: PRIME or COMPOSITE

Theorem[Fermat]: If n is a prime then $[x^{n-1}\ mod\ n] = 1$ for any x.

Example: n=5, x=2 → $[2^4\ mod\ 5] = [16\ mod\ 5] = 1$

Attempt 1: Pick random $x < n$ and check if $[x^{n-1}\ mod\ n] = 1$

Carmichael Number: Non-prime numbers that satisfy $[x^{n-1}\ mod\ n] = 1$ for any x.

14

Randomized Primality Test

Input: n
Output: PRIME or COMPOSITE

Theorem[Fermat]: If n is a prime then $[x^{n-1}\ mod\ n] = 1$ for any x.

Example: n=5, x=2 → $[2^4\ mod\ 5] = [16\ mod\ 5] = 1$

Attempt 1: Pick random $x < n$ and check if $[x^{n-1}\ mod\ n] = 1$

Carmichael Number: Non-prime numbers that satisfy $[x^{n-1}\ mod\ n] = 1$ for any x.

Theorem: If $n \ge 3$ is a prime then $n-1$ is even and can be written as $n-1 = 2^s d$ for any x it holds that either

- $[x^d\ mod\ n] = 1$, or
- $[x^{2^r d}\ mod\ n] = n-1$ for some $0 \le r < s$

15

Randomized Primality Test

Input: n
Output: PRIME or COMPOSITE
Theorem[Fermat]: If n is a prime then $[x^{n-1}\ mod\ n] = 1$ for any x.

Theorem: If $n \ge 3$ is a prime then $n-1$ is even and can be written as $n-1 = 2^s d$ for any x it holds that either

- $[x^d\ mod\ n] = 1$, or
- $[x^{2^r d}\ mod\ n] = n-1$ for some $0 \le r < s$

Witness of Non-Primality: $x < n$ such that $[x^d\ mod\ n] \ne 1$ **and** $[x^{2^r d} mod\ n] \ne n-1$ for all $0 \le r < s$
Theorem: If $n \ge 3$ is not a prime and $x < n$ is randomly picked then

$$\Pr[\text{x is witness of non} - \text{primality for n}] \ge \frac{3}{4}$$

16

Miller-Rabin Primality Test

Witness of Non-Primality: $x < n$ such that $[x^d\ mod\ n] \ne 1$ **and** $[x^d\ mod\ n] \ne n-1$ for all $0 \le r < s$

Theorem: If $n \ge 3$ is not a prime and $x < n$ is randomly picked then

$$\Pr[\text{x is witness of non} - \text{primality for n}] \ge \frac{3}{4}$$

Miller-Rabin test runs in time $O(kn^3)$ and mistakenly identifies a composite as prime with probability at most 4^{-k}

FFT-Multiplication: Reduces running time to $\tilde{O}(kn^2)$

There is a polynomial time algorithm to test if a n-bit number is prime…
…but the running time is $O(n^8)$

Miller-Rabin is used in practice in crypto libraries

17

13.5 Randomized Divide-and-Conquer

Quicksort

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

```
RandomizedQuicksort(S) {
   if |S| = 0 return

   choose a splitter a_i ∈ S uniformly at random
   foreach (a ∈ 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

19

Quicksort

Running time.
- [Best case.] Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons. $T(n) = 2T\left(\frac{n}{2}\right) + n$
- [Worst case.] Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons. $T(n) = T(n-1) + n$

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

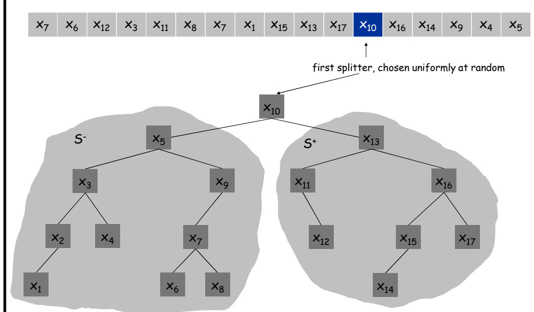
Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $\Theta(n \log n)$ comparisons.

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

20

Quicksort: BST Representation of Splitters

BST representation. Draw recursive BST of splitters.

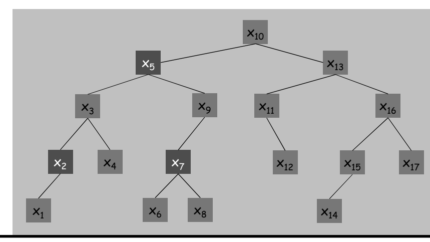


21

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_7 are not compared if their lca = x_3 or x_4 or x_5 or x_6.

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



22

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_7 are not compared if their lca = x_3 or x_4 or x_5 or x_6.

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

Random Variable.

$$y_{i,j} = \begin{cases} 1 & if\, x_i \text{ and } x_j \text{ are compared} \\ 0 & otherwise \end{cases}$$

Expected Value $\mathbf{E}[y_{i,j}] = \frac{2}{|j-i+1|}$

23

Quicksort: BST Representation of Splitters

Random Variable.

$$y_{i,j} = \begin{cases} 1 & if\, x_i \text{ and } x_j \text{ are compared} \\ 0 & otherwise \end{cases}$$

Expected Value: $\mathbf{E}[y_{i,j}] = \frac{2}{|j-i+1|}$

Total Comparisons:

$$Y = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} y_{i,j}$$

24

Quicksort: Expected Number of Comparisons

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

$$
\begin{aligned}
\mathbf{E}[Y] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbf{E}[y_{i,j}] \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&\le 2n \sum_{j=1}^{n} \frac{1}{j} \\
&= 2n \times H(n) \\
&\le 2n + 2\,n \ln n
\end{aligned}
$$

ln(n+1) < H(n) < 1 + ln n

25

Quicksort: Expected Number of Comparisons

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

Theorem. [Knuth 1973] Stddev of number of comparisons is ~ 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. $\Pr[|X - \mu| \ge k\delta] \le 1 / k^2$.

26

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 $\mu \ge E[X]$ 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

Example Quick Sort Comparisons: $\mu = 2n + 2n\ \ln n \ge E[Y]$ set δ=1

$$\Pr[Y > 2\mu] < \left[\frac{e}{4}\right]^{2+2\,n\ln n} \le e^{-n}$$

What is the flaw in the above argument?

Answer: the random variable $y_{i,j}$ are not all independent!

28

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 $\mu \ge E[X]$ 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,

$$\Pr[X > (1+\delta)\mu] \underset{\text{f(x) = }e^{tX}\text{ is monotone in x}}{=} \Pr\left[e^{tX} > e^{t(1+\delta)\mu}\right] \underset{\text{Markov's inequality: Pr[X > a]} \le \text{E[X] / a}}{\le} e^{-t(1+\delta)\mu} \cdot E[e^{tX}]$$

- Now

$$E[e^{tX}] \underset{\text{definition of X}}{=} E[e^{t\sum_i X_i}] \underset{\text{independence}}{=} \prod_i E[e^{tX_i}]$$

29

Chernoff Bounds (above mean)

Pf. (cont)

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

$$E[e^{tX_i}] = p_i e^t + (1-p_i)e^0 = 1 + p_i(e^t - 1) \underset{\text{for any } \alpha \ge 0,\ 1+\alpha \le e^{\alpha}}{\le} e^{p_i(e^t-1)}$$

- Combining everything:

$$\Pr[X > (1+\delta)\mu] \underset{\text{previous slide}}{\le} e^{-t(1+\delta)\mu} \prod_i E[e^{tX_i}] \underset{\text{inequality above}}{\le} e^{-t(1+\delta)\mu} \prod_i e^{p_i(e^t-1)} \underset{\sum_i p_i = E[X] \le \mu}{\le} e^{-t(1+\delta)\mu}\, e^{\mu(e^t-1)}$$

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

30

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

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

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

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

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.

- 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 $\Theta(\log n / \log \log n)$

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

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

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

- 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.

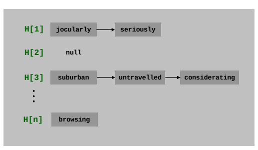
38

Hashing

Hash function. $h : U \to \{ 0, 1, \ldots, n-1 \}$.

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.



39

Ad Hoc Hash Function

Ad hoc hash function.

```
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?

40

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.

41

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

42

Universal Hashing

Universal class of hash functions. [Carter-Wegman 1980s]

- For any pair of elements u, v ∈ U, $\Pr_{h \in H}[h(u) = h(v)] \le 1/n$ ← chosen uniformly at random
- Can select random h efficiently.
- Can compute h(u) efficiently.

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

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1

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

not universal

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1
$h_3(x)$	0	0	1	0	1	1
$h_4(x)$	1	0	0	1	1	0

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

universal

43

Universal Hashing

Universal hashing property. Let H be a universal class of hash functions; let h ∈ H be chosen uniformly at random from H; and let u ∈ U. For any subset S ⊆ 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 ∈ 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.

$$E_{h \in H}[X] = E[\textstyle\sum_{s \in S} X_s] \underset{\text{linearity of expectation}}{=} \sum_{s \in S} E[X_s] \underset{X_s \text{ is a 0-1 random variable}}{=} \sum_{s \in S} \Pr[X_s = 1] \underset{\substack{\text{universal} \\ \text{(assumes } u \notin S)}}{\le} \sum_{s \in S} \frac{1}{n} = |S| \frac{1}{n} \le 1$$

44

Designing a Universal Family of Hash Functions

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

Modulus. Choose a prime number p ≈ n. ← no need for randomness here

Integer encoding. Identify each element u ∈ 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 $0 \le a_i < p$, define

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

Hash function family. $H = \{ h_a : a \in A \}$.

45

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 $\Pr[h_a(x) = h_a(y)] \le 1/n$.

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

$$a_j \underbrace{(y_j - x_j)}_{z} = \underbrace{\sum_{i \ne j} a_i (x_i - y_i)}_{m} \bmod p$$

- Can assume a was chosen uniformly at random by first selecting all coordinates a_i where i ≠ j, then selecting a_j at random. Thus, we can assume a_i is fixed for all coordinates i ≠ j.
- Since p is prime, $a_j z = m \bmod p$ has at most one solution among p possibilities. ← see lemma on next slide
- Thus $\Pr[h_a(x) = h_a(y)] = 1/p \le 1/n$. ▪

46

Number Theory Facts

Fact. Let p be prime, and let z ≠ 0 mod p. Then αz = m mod p has at most one solution 0 ≤ α < p.

Pf.

- Suppose α and β are two different solutions.
- Then (α - β)z = 0 mod p; hence (α - β)z is divisible by p.
- Since z ≠ 0 mod p, we know that z is not divisible by p; it follows that (α - β) is divisible by p.
- This implies α = β. ▪

Bonus fact. Can replace "at most one" with "exactly one" in above fact.
Pf idea. Euclid's algorithm.

47

Extra Slides