CS38
Introduction to Algorithms

Lecture 19

June 3, 2014

Outline

- randomness in algorithms
  - max-3-sat approximation algorithm
  - universal hashing
  - load balancing
- Course summary and review

Randomization

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

in practice, access to a pseudo-random number generator

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.

Max-3-SAT approximation algorithm

Expectation

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

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

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?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j\,\underbrace{(1-p)^{j-1}}_{j-1 \text{ tails}}\,\underbrace{p}_{1 \text{ head}} = \frac{p}{1-p}\sum_{j=0}^{\infty} j\,(1-p)^j = \frac{p}{1-p}\cdot\frac{1-p}{p^2} = \frac{1}{p}$$

Expectation: two properties

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

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

not necessarily independent

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

Benefit. 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 = number of correct guesses = $X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1 / n + \ldots + 1 / n = 1.$ ▪

linearity of expectation

Guessing cards

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

Guessing with memory. Guess a card uniformly at random from cards not yet seen.

Claim. The expected number of correct guesses is $\Theta(\log n)$.

Pf.

- Let $X_i = 1$ if i^{th} prediction is correct and 0 otherwise.
- Let X = number of correct guesses = $X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - i - 1)$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1 / n + \ldots + 1 / 2 + 1 / 1 = H(n).$ ▪

linearity of expectation

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

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 ≥ 1 coupon of each type?

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

Pf.

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

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

prob of success = (n – j) / n

⇒ expected waiting time = n / (n – j)

Maximum 3-satisfiability

exactly 3 distinct literals per clause

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

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

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

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

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 = weight of clauses satisfied by assignment Z_j.

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

linearity of expectation

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. [Paul Erdös] Prove the existence of a non-obvious property 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?
A. Yes (but 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.

$$\begin{aligned} \tfrac{7}{8}k \;=\; E[Z] \;&=\; \sum_{j \geq 0} j\, p_j \\ &=\; \sum_{j < 7k/8} j\, p_j \;+\; \sum_{j \geq 7k/8} j\, p_j \\ &\leq\; \left(\tfrac{7k}{8} - \tfrac{1}{8}\right) \sum_{j < 7k/8} p_j \;+\; k \sum_{j \geq 7k/8} p_j \\ &\leq\; \left(\tfrac{7}{8}k - \tfrac{1}{8}\right) \cdot 1 \;+\; k\, p \end{aligned}$$

Rearranging terms yields $p \geq 1 / (8k)$. ▪

13

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 $\geq 1 / (8k)$. By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most $8k$. ▪

14

Universal hashing

June 3, 2014 CS38 Lecture 19 15

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 $S = \varphi$.
- `insert(u)`: add element $u \in U$ to S.
- `delete(u)`: delete u from S (if u is currently in S).
- `lookup(u)`: is u 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.

16

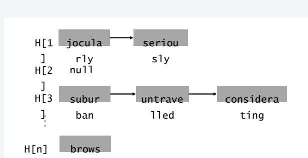
Hashing

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.
- Separate chaining: $H[i]$ stores linked list of elements u with $h(u) = i$.

birthday paradox



17

Ad-hoc hash function

Ad hoc hash function.

```
int hash(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?

18

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.

19

Hashing performance

Ideal hash function. Maps m elements uniformly at random to m 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

20

Universal hashing

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

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

chosen uniformly at random

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

21

Universal hashing: analysis

Proposition. Let H be a universal family 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.

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

linearity of expectation — X_s is a 0-1 random variable — universal (assumes $u \notin S$)

Q. OK, but how do we design a universal class of hash functions?

22

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 $0 \le a_i < p$, define

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

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

23

Designing a universal family of hash functions

Theorem. $H = \{ h_a : a \in A \}$ is a universal family 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 $x \neq y$, there exists an integer j such that $x_j \neq y_j$.
- We have $h_a(x) = h_a(y)$ iff

$$\underbrace{a_j(y_j - x_j)}_{z} = \underbrace{\textstyle\sum_{i \text{ not equal } j} a_i(x_i - y_i)}_{m} \bmod p$$

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

24

Load balancing

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 ⌠ ε E[X] and for any ™ > 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 $t > 0$,

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

f(x) = e^{tX} is monotone in x   Markov's inequality: Pr[X > a] δ 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. [continued]

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

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

for any 〈 ε 0, 1+〈 δ e〈

- Combining everything:

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

previous slide   inequality above   □ᵢ pᵢ = E[X] δ ⌠

- 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 ⌠ ≤ $E[X]$ 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 $\delta < 1$.

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 ® with probability ≥ $1 - 1/n$ no processor receives more than $e\,\gamma(n) = \Theta(\log n / \log\log n)$ jobs.

Bonus fact: with high probability,
some processor receives Θ(logn / log log n) jobs

Load balancing: many jobs

Theorem. Suppose the number of jobs $m = 16\, n \ln n$. 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

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

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

Course summary and review

Algorithmic design paradigms

- Greedy (see: matroids)
- Divide and Conquer
- Dynamic Programming
- Flows, cuts and matchings
- Linear Programming

more sophisticated/general as go down list

Algorithmic design paradigms

- Many problems are NP-complete
- Unlikely to have solutions in P
- Coping with intractibility
  - special cases
  - fixed-parameter algorithms/analysis
  - approximation algorithms

Fundamental algorithms

- Graph traversals in $O(n + m)$ time
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
- applications:
  - BSF yields shortest paths in undirected graph
  - DFS used for topological sort and strongly connected components

Fundamental algorithms

- Single source shortest paths
  - with non-negative edge weights

    Dijkstra's algorithm $O(n + m \log n)$
  - with negative edge weights

    Bellman-Ford $O(nm)$ to detect negative cycles and find shortest paths if no negative cycles

Fundamental algorithms

- All-pairs shortest paths
  - Floyd-Warshall $O(n^3)$
- Minimum cost spanning tree
  - Kruskal $O(m \log m)$
  - Prim $O(m + n \log n)$
- compression via variable length coding
  - Huffman codes $O(n \log n)$

Data structures

- binary min-heap
  - INSERT $O(\log n)$
  - EXTRACT-MIN $O(\log n)$
  - DECREASE-KEY $O(\log n)$
- Fibonacci heap (amortized analysis)
  - INSERT $O(1)$
  - EXTRACT-MIN $O(\log n)$
  - DECREASE-KEY $O(1)$

Data structures

- Union-Find data structure
  - path compression
  - union-by-rank
  - amortized analysis:
    m find and n union operations in $O(m \log^* n)$

Fundamental algorithms

- Sorting in $O(n \log n)$ time
  - heapsort
  - mergesort
  - quicksort with random pivot (expected time)
  - lower bound for comparison-based
- selection in $O(n)$ time
  - randomized and deterministic

Fundamental algorithms

- closest pair of points in plane $O(n \log n)$
- integer multiplication $O(n^{\log_2 3})$
- matrix multiplication $O(n^{\log_2 7})$
- FFT $O(n \log n)$
  - polynomial multiplication and division with remainder $O(n \log n)$

Fundamental algorithms

- two strings of length n, m:
  - edit distance $O(nm)$
  - longest-common-subsequence $O(nm)$

Fundamental algorithms

- max-flow in a network (= min-cut)
  - Ford-Fulkerson method $O(m\ nC)$
  - capacity-scaling $O(m^2 \log C)$
  - shortest augmenting path $O(m^2\ n)$
  - blocking-flow implementation $O(mn^2)$
  - unit-capacity simple graphs $O(mn^{1/2})$
  - use last for bipartite matching in same time
  - min/max weight perfect matching $O(n^3)$

June 3, 2014 CS38 Lecture 19 43

Fundamental algorithms

- Linear programming
  - primal/dual and strong duality theorem
  - simplex algorithm (worst case exponential)
  - ellipsoid algorithm (in P)
  - in practice: interior points methods

June 3, 2014 CS38 Lecture 19 44

Fundamental algorithms

- Coping with intractibility
  - e.g.: hard graph problems easy on trees
  - e.g.: fixed parameter algorithms for VC
- approximation algorithms
  - knapsack $(1 + \epsilon)$
  - VC and weighted VC 2 (via LP relaxation)
  - set cover $\ln m + 1$
  - TSP 1.5
  - center selection 2

June 3, 2014 CS38 Lecture 19 45

Fundamental algorithms

- randomized algorithm for global min-cut
- 8/7 approximation for max-3-sat
- other applications:
  - contention resolution
  - hashing
  - load-balancing
  - …

June 3, 2014 CS38 Lecture 19 46