CS 124 Problem Set 5

Author: 30943147. I collaborated with 50996336.

March 31, 2017

Problem 1

a

We are hashing $k \geq c_1\sqrt{n}$ people into n bins. The probability that no two have the same hash is equal to

$$\prod_{i=0}^{k}(1-\frac{i}{n}) \leq \prod_{i=0}^{k}(e^{-i}) = e^{-\sum_{i=0}^{k}\frac{i}{n}} = e^{\frac{k(k+1)}{2n}} \leq e^{\frac{c_1\sqrt{n}(c_1\sqrt{n}+1)}{2n}} \leq e^{\frac{-c_1^2}{2}}$$

The last step above follows from $n \geq 1$. Choosing $c_1 \geq \sqrt{2}$ then gives the probability that no two have the same hash as $\leq e^{-1}$.

b

Let P be the probability that no two people share a birthday. Using the identity $1 - x \geq e^{-x-x^2}$ (which is valid since $x \leq \frac{c_2\sqrt{n}}{n} \leq \frac{1}{2}$ since we are working with "sufficiently large n"), we get

$$P \geq \prod_{i=0}^{k} e^{\frac{-i}{n}-\frac{i^2}{n^2}} = e^{-\frac{k(k+1)}{2n}-\frac{k(k+1)(2k+1)}{6n^2}} \geq e^{-\frac{c_2\sqrt{n}(c_2\sqrt{n}+1)}{2n}-\frac{c_2\sqrt{n}(c_2\sqrt{n}+1)(2c_2\sqrt{n}+1)}{6n^2}}$$

We want to choose c_2 such that

$$e^{-\frac{c_2\sqrt{n}(c_2\sqrt{n}+1)}{2n}-\frac{c_2\sqrt{n}(c_2\sqrt{n}+1)(2c_2\sqrt{n}+1)}{6n^2}} \geq \frac{1}{2}$$

We take the log and simplify.

$$\frac{c_2\sqrt{n}(c_2\sqrt{n}+1)}{n} + \frac{c_2\sqrt{n}(c_2\sqrt{n}+1)(2c_2\sqrt{n}+1)}{3n^2} \leq 2\ln(2)$$

Since n is "sufficiently large", all terms with a negative power of n are negligable compared to the constant terms, and we reduce this expression to

$$c_2^2 \leq 2\ln 2$$

$$c_2 \leq \sqrt{2\ln 2}$$

Problem 2

a

We are hashing n elements into k hashtables, each of size $\frac{m}{k}$. Each slot in the table has b bits. Therefore, hashing $\geq 2^b$ elements into that slot will produce overflow. First, we find the probability that an arbitrarily chosen slot doesn't overflow (all slots are identical for this purpose). Call that probability X. X is equal to the probability that the slot has $< 2^b$ elements. The probability that the slot has i elements is equal to

$$(\frac{k}{m})^i(\frac{m-k}{m})^{n-i}\binom{n}{i}$$

Therefore, X_i is equal to

$$X = \sum_{i=0}^{2^{b-1}}(\frac{k}{m})^i(\frac{m-k}{m})^{n-i}\binom{n}{i}$$

The probability that our arbitrarily chosen slot overflows is then equal to

$$P = 1 - X = 1 - (\sum_{i=0}^{2^{b-1}}(\frac{k}{m})^i(\frac{m-k}{m})^{n-i}\binom{n}{i})$$

b

Plugging in the given values into our equation and evaluating with Mathematica gives the following probabilities of overflow.

b	P(a specific counter overflows)
3	$5.93 * 10^{-8}$
4	$3.37 * 10^{-18}$
5	$4.49 * 10^{-42}$

I would choose to use a 4-bit counter. Even though the odds of a specific counter overflowing are order 10^{-8}, there are 10^6 counters. The probability that any counter will overflow seems subtle to calculate since whether two counters overflow is not linearly independent, but it seems like 3-bit counters will give a nontrivial probability that one of the 10^6 counters will overflow. 4-bit counters bring the odds of a specific counter overflowing down to order 10^{-18}, which is still tiny even when one considers 10^6 counters.

Problem 3

a

Generally, we have the probability of matching using a single sketch as

$$p(r) = \sum_{k}^{n} \binom{n}{k} r^k (1-r)^{n-k}$$

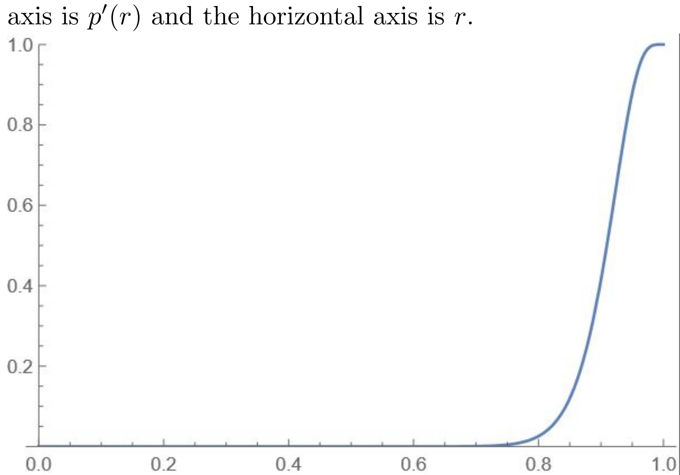
So, for our initial hashing, we have

$$p(r) = \sum_{k}^{84} \binom{84}{k} r^k (1-r)^{84-k}$$

For the rehashing, we have $k = 2$, $n = 6$, and $r' = r^{14}$, which is the probability that all 14 values input into a given hash match. For now, we are assuming no collisions. This is a reasonable assumption because we are hashing 14 values into a hashtable of size 2^{64}, so the probability of collisions will be tiny. Regardless, a small number of false positives is not a problem–it will only result in a few websites being marked as duplicate when they shouldn't be. The probability $p'(r)$ of a match using this rehashing is

$$p'(r) = \sum_{k=2}^{6} \binom{6}{k} r^{14k} (1-r^{14})^{6-k}$$

Graphing $p'(r)$ with Mathematica gives the following graph, where the verticle axis is $p'(r)$ and the horizontal axis is r.



Evaluating $p'(r)$ for select values of r gives the following results:

r	$p'(r)$
0.70	0.00068
0.75	0.0045
0.80	0.026
0.85	0.12
0.90	0.42
0.95	0.88
0.98	0.996
0.99	0.9998

We can see that documents with resemblance ≤ 0.75 have only a tiny chance of being marked as duplicate, and documents with resemblance ≥ 0.98 have onlly a tiny chance of being marked as not duplicate.

b

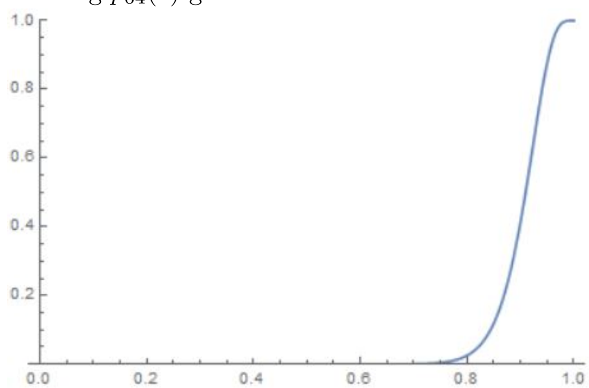
Now we turn our attention to the possibility of collisions. We must modify our equation

$$p'(r) = \sum_{k=2}^{6} \binom{6}{k} r^{14k}(1-r^{14})^{6-k}$$

by replacing r^{14} with $r^14+(1-r^{14})\frac{1}{H}$, where H is the size of our hashtable. r^14 is the probability that the group matches, and $(1-r^{14})\frac{1}{H}$ is the probability that the group does not match but they happen to hash the same regardless. We must also replace the probability they don't agree, ie $(1-r^{14})$, with $(1-r^{14}-\frac{1}{H}(1-r^{14})$, subtracting out the probability that they randomly hash the same despite being different. Making the correction for the 64-bit hash, ie $H = 2^{64}$, results in negligible changes

$$p_{64}(r) = \sum_{k=2}^{6} \binom{6}{k} (r^{14} + (1-r^{14})\frac{1}{2^{64}})^k (1 - r^{14} - \frac{1}{2^{64}}(1-r^{14}))^{6-k}$$

Plotting $p_{64}(r)$ gives



Evaluating $p_{64}(r)$ at select values gives the exact same table of values (at least to the level of precision we have decided to use)

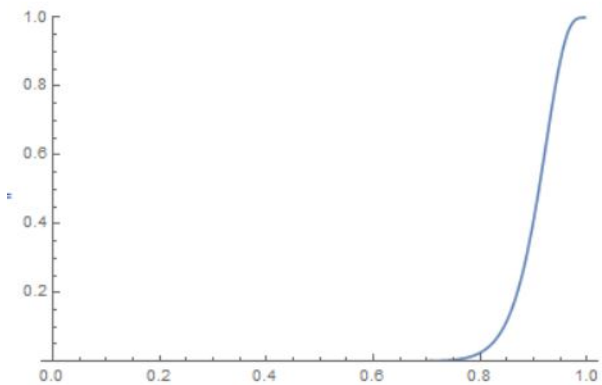
r	$p_{64}(r)$
0.70	0.00068
0.75	0.0045
0.80	0.026
0.85	0.12
0.90	0.42
0.95	0.88
0.98	0.996
0.99	0.9998

This confirms our assumption in part a that the possibility of a collision on 64-bit hash values is negligible.

Now consider 16-bit hashtables

$$p_{16}(r) = \sum_{k=2}^{6} \binom{6}{k} (r^{14} + (1-r^{14})\frac{1}{2^{16}})^k (1 - r^{14} - (1-r^{14})\frac{1}{2^{16}})^{6-k}$$

Plotting p_{16} gives

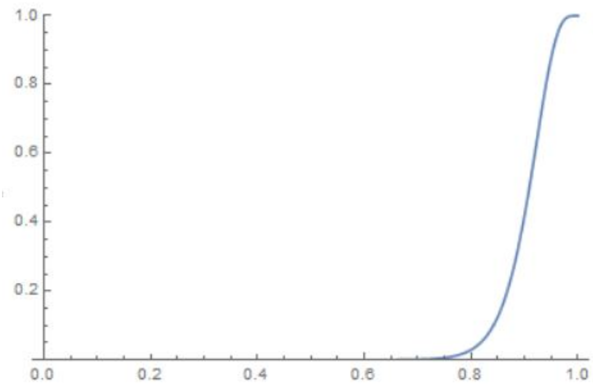


r	$p_{16}(r)$
0.70	0.00068
0.75	0.0045
0.80	0.026
0.85	0.12
0.90	0.42
0.95	0.88
0.98	0.996
0.99	0.9998

The differences do not show up in our 2-significant figure tables, but they are visible when looking at the longer forms of the decimals that Mathematica spits out.

Now consider 8-bit hashtables

$$p_8(r) = \sum_{k=2}^{6} \binom{6}{k} (r^{14} + (1 - r^{14})\frac{1}{2^8})^k (1 - r^{14} - (1 - r^{14})\frac{1}{2^8})^{6-k}$$



r	$p_{16}(r)$
0.70	0.0017
0.75	0.0066
0.80	0.030
0.85	0.13
0.90	0.42
0.95	0.88
0.98	0.996
0.99	0.9998

8-bit hashing produces noticeable but still small differences compared to the 64 or 16-bit hashing. I would probably still choose the 8-bit hashes to save space, because from looking at this data I doubt the errors resulting from collisions would be significant compared to the general probabilistic behaviour of the algorithm. Also, false positives are not a big deal in the AltaVista example.

Problem 4

This algorithm will only work if all patterns P_i are of the same length, but a piazza post said that we could make that assumption.

We use the fingerprinting algorithm described in lecture, with the following modification. We move through the document calculating

$$N' = 10 \cdot (N - 10^{|P|-1} \cdot a) + b$$

where a is the leftmost digit and b is the rightmost digit of our old number N. After each time we compute N', instead of merely comparing N' to a single pattern P, we compare it to all of our P_i and return true if any match.

Now, the chance of a false positive at a given step (with k patterns) is

$$1 - (1 - \frac{\log_2 10^{|P|}}{\pi(z)})^k$$

where $\frac{\log_2 10^{|P|}}{\pi(z)}$ is the probability of a false positive at a single step in the original algorithm. $\pi(z)$ is the number of primes $< z$, where z is the randomly chosen prime we are using as a modulus. Using the union bound, this gives the probability of a false positive over the entire document as

$$|D|(1 - (1 - \frac{\log_2 10^{|P|}}{\pi(z)})^k)$$

where $|D|$ is the length of the document.

Problem 5

The code used in problem 5 is submitted as q5.java. Code is mostly copied from rsa.java.

a

We show that $n = 636127$ is composite using Fermat's little theorem. We adapt our code from problem 6 (which I did first) to calculate

$$2^{n-1} \mod n = 2^{318063} \mod 636127 = 469435 \neq 1$$

$n = 636127$ fails Fermat's little theorem for $a = 2$, and therefore cannot be prime.

b

Fermat's little theorem is an if, not an if and only if. This means that there are composite numbers n that have $a^{n-1} \mod n = 1$. These are the Carmichael numbers, and $n = 294409$ is one. This means we have to use the Rabin-Miller primality test to show that n is composite.

We show that $n = 294409$ is composite using the Rabin-Miller primality test with $a = 2$. First, we decompose $n = 2^t u$ for $t = 3$ and $u = 36801$. Now, we use our power mod code in q5.java to calculate the following values (which I then checked with Mathematica's power mod function).

$2^u \mod n$	512
$2^{2u} \mod n$	262144
$2^{4u} \mod n$	1
$2^{8u} \mod n = 2^{n-1} \mod n$	1

We have found that 2^{2u} is a non-trivial square root of 2^{4u}. Therefore, by the Rabin-Miller primality test, n is composite.

Problem 6

The code for this section is submitted as rsa.java

The message m, when encoded into ascii then converted to decimal, is 221009320130778009770266654273. The corresponding rsa encryption is equal to $m^e \mod n$, where (n, e) is the recipient's public key. The encrypted message is 270167643401181923957124923778.

Code

rsa.java calculates $m^e \mod n$ efficiently by using the repeated squaring algorithm, and by taking the $\mod n$ after every multiplication to keep the sizes of the numbers down. Even so, I had to use java's BigInteger class (which supports arbitrarily large numbers) because m and e are larger than the size of max long on my machine. Java has most built-in arithmetic methods for BigInteger, however, it is missing a method for logarithm. I got around this by using the bit length of the BigInteger in question, which is the ceiling of the log. Since all I needed was the floor of the logarithm (for decomposing e into powers of 2 for repeated squaring), this was enough information.

For repeated squaring, I used the same strategy that I used several psets ago for a repeated squaring problem. I decompose the exponent e into powers of 2, then save m^{2^0}, m^{2^1}, ... $m^{2^{\lfloor \log_2 e \rfloor}}$ in an array mpows, then construct m^e by multiplying the correct elements of the array together.

I checked my answers with Mathematica's built-in PowerMod function.

Problem 7

Results are averaged over 10 trials. In each row, Bins is the number of bins with n balls in it.

a

n	Bins
0	367878378
1	367882968
2	183935495
3	61315803
4	15327350
5	3065557
6	511157
7	73057
8	9111
9	1005
10	104
11	8
12	0

b

n	Bins
0	238403309
1	532094246
2	220607587
3	8888847
4	6008
5	0

As we would expect, using method b to assign buckets to bins greatly reduces the number of bins with more than a few balls in them. I noticed that method b results in fewer bins with 0 balls in them than method a. This makes sense becuase when method b chooses 2 bins at random, it is likely that at least one of them will have 0 balls in it. Then, method b will put a ball in that bin, so that it now has 1 ball. Method a, in contrast, has only 1 chance per ball to choose a bin with 0 balls already in it.

When writing this code, the first time I ran method b I accidentally had it set to choose the bin with more balls rather than the bin with fewer balls. This resulted in bins with as many as 16 balls in them.

I used the strategy suggested in the hint for reducing the memory usage of the algorithm. Instead of creating 1 billion bins and keeping track of how many