CS 124: Problem Set 5

Chao Cheng
Collaborators: Hao Wang

February 4, 2021

Problem 1. Suppose there are k people in the room. Then the probability of no two people having the same hash is

$$1\big(1-\frac{1}{n}\big)\left(1-\frac{2}{n}\right)\ldots\left(1-\frac{k-1}{n}\right)$$

Using the fact that $1-x \le e^{-x}$, we have

$$\begin{aligned}1\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\ldots\left(1-\frac{k-1}{n}\right) &\le 1\big(e^{-1/n}\big)\big(e^{-2/n}\big)\ldots\big(e^{-(k-1)/n}\big)\\ &\le e^{\left(-\frac{1}{n}-\frac{2}{n}-\ldots-\frac{k-1}{n}\right)}\\ &\le e^{-\frac{k(k-1)}{2n}}\end{aligned}$$

Since we want this probability to be at most e^{-1}, our inequality becomes

$$\begin{aligned}e^{-\frac{k(k-1)}{2n}} &\le e^{-1}\\ -\frac{k(k-1)}{2n} &\le -1\\ k(k-1) &\ge 2n\\ k^2-k-2n &\ge 0\end{aligned}$$

From the quadratic equation, we get $k \ge \frac{1+\sqrt{1+8n}}{2}$. As n grows, $\lim_{n\to\infty} k = \sqrt{2n}$. Since $k = c_1\sqrt{n}$, we have (for large values of n) $c_1 = \sqrt{2}$.

For the second part, we go back to the earlier expression for probability of no two peopling having the same hash. We want this probability to be at least $\frac{1}{2}$, so we have

$$\begin{aligned}e^{-\frac{k(k-1)}{2n}} &\ge \frac{1}{2} = e^{-\ln 2}\\ \frac{k(k-1)}{2n} &\le \ln 2\\ k(k-1) &\le 2n\ln 2\\ k^2-k-2n\ln 2 &\le 0\end{aligned}$$

Again, from the quadratic equation, we get $k \le \frac{1+\sqrt{1+8n\ln 2}}{2}$. As n grows, $\lim_{n\to\infty} k = \sqrt{2n\ln 2}$. Since $k = c_2\sqrt{n}$, we have (for large values of n) $c_2 = \sqrt{2\ln 2}$.

Problem 2. Since there are k has functions and m total locations, the probability of each element being hashed to a particular location is $\frac{k}{m}$. Thus, the probability of q elements out of n total elements being hashed to a particular location is $\binom{n}{q}\left(\frac{k}{m}\right)^q\left(1-\frac{k}{m}\right)^{n-q}$. Since a b-bit counter overflows if more than 2^b-1 elements are hashed to it, the probability of a particular location overflowing is

$$1-\sum_{q=0}^{2^b-1}\binom{n}{q}\left(\frac{k}{m}\right)^q\left(1-\frac{k}{m}\right)^{n-q}$$

or in other words, the probability of overflowing is 1 minus the probability of having at most 2^b-1 elements hashed to it.

For $n=10^5$, $m=10^6$, and $k=\lceil\frac{m}{n}\cdot\ln 2\rceil=7$, we have that for $b=3$, $P_{\text{overflow}}\approx 7.7\times10^{-7}$, for $b=4$, $P_{\text{overflow}}\approx 8.2\times10^{-17}$, and for $b=5$, $P_{\text{overflow}}\approx 2.1\times10^{-41}$. Thus the ideal counter size would be 4-bit or 5-bit, as the probability of overflowing at that point would be extremely low.

Problem 3. For each of the 6 sketches of the overall super-sketch, the two documents will agree on hashed value only if they agree on all 14 permutations of the sketch. The probability that two documents with resemblance r agree on each permutation is r, so the probability that they agree on any group of 14 permutations is r^{14}. Let m be the number of sketches that match. Then the probability of m sketches matching is

$$P(m) = \binom{6}{m}\left(r^{14}\right)^{m}\left(1-r^{14}\right)^{6-m}$$

and the probability of two or more sketches matching is

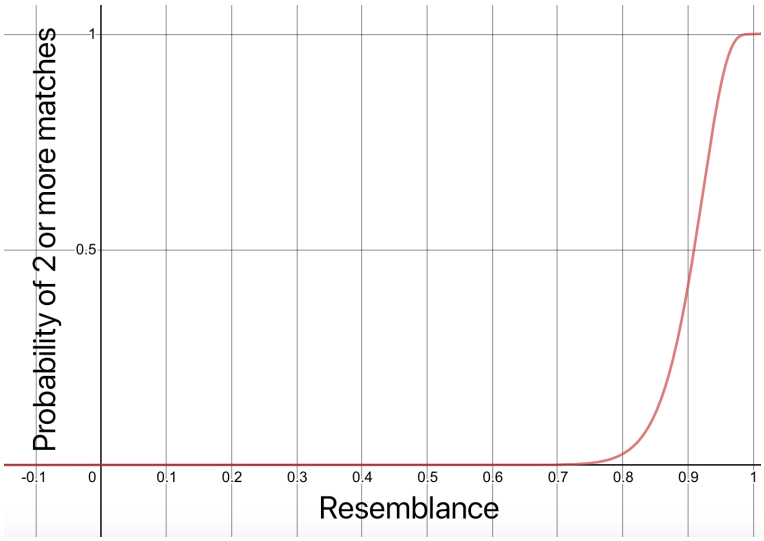
$$\begin{aligned} P(m \geq 2) &= 1 - P(1) - P(0) \\ &= 1 - \binom{6}{1}\left(r^{14}\right)^{1}\left(1-r^{14}\right)^{5} - \binom{6}{0}\left(r^{14}\right)^{0}\left(1-r^{14}\right)^{6} \\ &= 1 - 6r^{14}\left(1-r^{14}\right)^{5} - \left(1-r^{14}\right)^{6} \end{aligned}$$

Above, we assumed that the probability of hashing collisions when comparing the hashes of the sketches was negligible, since the chance of a collision would be $\frac{1}{2^64}$ for a 64-bit hash function (that is, the only time the hashes will match is when the 14 permutations of the sketch agree). However, for 16-bit and 8-bit hashes, the chance of a false match is much greater, so we need to factor it into the probability equation. For a b-bit hash function, the chance of collision is $\frac{1}{2^b}$, so the probability that two documents matching on a sketch is now $r^{14} + \frac{1}{2^b}(1-r^{14})$. Thus, the probability of m sketches matching would then be

$$P(m) = \binom{6}{m}\left(r^{14} + \frac{1}{2^b}\left(1-r^{14}\right)\right)^{m}\left(\left(1-r^{14}\right)\left(1-\frac{1}{2^b}\right)\right)^{6-m}$$

for 16-bit and 8-bit hashes.

The graph of $P(\geq 2)$ as function of r, for 64-bit hashes, is shown below. As desired, the probability that there are more than two matches is very low until roughly between 0.8 and 0.9, and then very high after 0.9. Thus, the documents won't agree unless the resemblance is very high already, which is what we want.



Problem 4. (a) 2 is a witness for 636127, since $2^{636127-1} \equiv 195504 \not\equiv 1 \pmod{636127}$.

(b) Fermat's little theorem will not work for Carmichael numbers since their property is that for all a where $\gcd(a, n) = 1$, we will have that $a^{n-1} \equiv 1 \pmod{n}$. Therefore, we need to use the Rabin-Miller method. We have that 124410 is a witness for 294409, since $n - 1 = 294408 = 2^3 \cdot 36801$, and

$$\begin{aligned} 124410^{36801} &\equiv 207830 \pmod{294409} \\ (124410^{36801})^2 &\equiv 207830^2 \equiv 270101 \pmod{294409} \\ ((124410^{36801})^2)^2 &\equiv 270101^2 \equiv 1 \pmod{294409} \end{aligned}$$

Since we have found a non-trivial square root for 1 (mod 294409), 294409 is composite.

(c) "Give me an A" converted into binary using ASCII codes is

$$01000111011010010111011001100101001000000011011 01$$

$$01100101001000000110000101101110001000000100000 1$$

which is 221009320130778009770266542 73 in decimal, so we have

$$\begin{aligned} n &= 469478487497204305296287390 81 \\ e &= 372674862636792350620645369 73 \\ x &= 221009320130778009770266542 73 \end{aligned}$$

and encoding,

$$x^e \equiv 270167643401181923957124923 78 \pmod{n}$$

Thus, the encoded message in decimal is 270167643401181923957124923 78.

Problem 5. 1. The code for this problem is located at the end of this document. To simulate a trial, we keep track of the bin loads in an array `binCounts` of size 100 (we can safely limit the size to 100 since it is very, very unlikely that any one bin surpasses that number of balls). Each entry in the array corresponds to the number of bins with *index* balls in it, so we set `binCounts` $= 10^9$ initially. Then we generate a random number n between 1 and 1 billion, and add a ball to the corresponding bin. The corresponding bin is the n-th bin of `binCounts`, where we have arranged the bin in order of increasing load size. We repeat this 1 billion times for 1 billion balls, and record the maximum load size.

The results of running 100 trials of throwing 1 billion balls into 1 billion bins are summarized in the table below.

Maximum Load Frequencies	
Max Load	*Frequency*
11	49
12	49
13	2

2. To simulate the modified experiment, instead of generating only one random number between 1 and 1 billion, we generate two and add a ball to the smaller one since it corresponds to the bin with less balls.

In this experiment, each trial resulted in a maximum bin load of 4.

Maximum Load Frequencies	
Max Load	*Frequency*
4	100

For both experiments, it seems like the distribution of maximum loads is very narrow. Naturally, the second experiment resulted in a much more even spread of balls across all loads, and it was interesting to see that every trial resulted in a maximum load of 4 — even with such a large quantity of balls, the modification of choosing the smaller of two bins made the distribution of the balls much more equal.

Problem 5 Code

```
public class Problem5 {
        public static int runSim(int n) {
                int[] binCounts = new int[100];
                binCounts[0] = n;
                Random rand = new Random(System.currentTimeMillis());

                for (int a = 0; a < n; a++) {
                        // Generate two random numbers and use the smaller one
                        int b = rand.nextInt(n) + 1;
                        int c = rand.nextInt(n) + 1;
                        if (c < b) {
                                b = c;
                        }
                        // Find the corresponding bin and
                        // increment its number of balls
                        int i = 0;
                        while(true) {
                                if (b <= binCounts[i]) {
                                        binCounts[i]--;
                                        binCounts[i+1]++;
                                        break;
                                } else {
                                        b -= binCounts[i];
                                        i++;
                                }
                        }
                }
                int maxLoad = 0;
                while (binCounts[maxLoad + 1] != 0) {
                        maxLoad++;
                }
                return maxLoad;
        }
        public static void main(String[] args) {
                int n = (int) Math.pow(10, 9);
                int [] maxLoadFreq = new int[100];
                for (int k = 0; k < 100; k++) {
                        System.out.println("=====Trial " + k + "=====");
                        int maxLoad = runSim(n);
                        maxLoadFreq[maxLoad]++;
                }
                System.out.println(Arrays.toString(maxLoadFreq));
        }
}
```