Lecture 1: Old friends and new beginnings CSE 525, Fall 2016
Instructor: James R. Lee

1 A little probability of error goes a long way

Suppose Bob has just spent hours downloading the latest episode of Game of Thrones—a 4 GB file. He wants to check that he has the same data as Alice, but obviously they don't want to transfer all 4 GB of data again. Of course, the solution is for Alice to send Bob a *hash* of her data.

Let's suppose that Alice and Bob have two n-bits strings denoted A and B. Let T denote a parameter we'll pick later. Alice chooses a uniformly random prime number $p \in \{2, 3, \ldots, T\}$ and sends to Bob the hash

$$H_p(A) = A \bmod p\,.$$

Note that the length of this message is at most $\log_2 T$ bits.

Bob also computes a hash of his string $H_p(B)$. If $H_p(A) = H_p(B)$, then Bob ACCEPTS the fact that he and Alice probably have the same file. Otherwise, Bob REJECTS (and is really annoyed, because a major character is probably dead and he doesn't even know it).

One-sided error. This is a randomized protocol with *one-sided error*. If $A = B$, then Bob always accepts. On the other hand, if $A \neq B$, then we might still have $H_p(A) = H_p(B)$, meaning that Bob could incorrectly accept sometimes even if the files are different.

Probability technique: Counting. The key (we hope) is that if $A \neq B$, then Bob *almost always* rejects. We make the following claim.

Claim 1.1. If $A \neq B$, then

$$\mathbb{P}\left[H_p(A) \neq H_p(B)\right] \leqslant 1.26 \frac{n \ln T}{T \ln n}\,.$$

Before we prove the claim, let's see what it means. With concrete numbers, if $n = 2^{32}$ (approx. 4 GB), and $T = 2^{64}$ (meaning that the hash is only 64 bits), then Claim 1.1 says that

$$\mathbb{P}[\text{Bob incorrectly accepts}] \leqslant 1.26 \frac{2^{32}}{2^{64}} \frac{64}{32} \leqslant 0.000000006\,.$$

More generally, if we put $T = cn$ for some constant $c > 0$, then

$$\mathbb{P}[\text{error}] \leqslant \frac{1.26}{c}\left(1 + \frac{\ln c}{\ln n}\right).$$

Setting, e.g., $c = n$, we get an $O(\log n)$-bit hash with error probability $O(1/n)$.

Proof of Claim 1.1. Let $\pi(x)$ denote the number of primes $\leqslant x$. Suppose that $A \neq B$. If $H_p(A) = H_p(B)$, then $A \equiv B \pmod p$, hence $p \mid |A - B|$. So we have

$$\mathbb{P}[H_p(A) = H_p(B)] \leqslant \frac{\#\text{ primes dividing } |A - B|}{\pi(T)} \leqslant \frac{n}{\pi(T)}\,,$$

where we have used the fact that at most n distinct primes can divide $|A - B| \leqslant 2^n$ because every prime is at least 2. We can analyze this using the Prime Number Theorem:

Theorem 1.2. *As $x \to \infty$, $\pi(x) \sim \frac{x}{\ln x}$. In fact, for $x \geqslant 17$, we have*

$$\frac{x}{\ln x} \leqslant \pi(x) \leqslant 1.26 \frac{x}{\ln x}\,. \tag{1.1}$$

In fact, it is true that the number of primes dividing a number N is at most $\pi(\log_2 N)$, so we can achieve a slightly better bound:

$$\mathbb{P}[H_p(A) = H_p(B)] \leqslant \frac{\pi(n)}{\pi(T)} \leqslant 1.26 \frac{n}{T} \frac{\ln T}{\ln n}\,,$$

concluding the proof. □

One thing we didn't see is how to choose a random prime $p \in \{2, 3, \ldots, T\}$. This is generally done by choosing a uniformly random number $k \in \{2, 3, \ldots, T\}$ and then checking whether k is prime. Primality testing is a long-studied topic, but the algorithms of choice for this task are again randomized. The earliest example is the Miller-Rabin primality test (look on Wikipedia).

2 Hashing for pattern matching

We can use this simple hashing trick to do something more substantial. We now describe the Karp-Rabin algorithm for string matching.

Suppose we are given two strings $X = x_1 x_2 \cdots x_n$ and $Y = y_1 y_2 \cdots y_m$ with $m < n$. Our goal is to check whether $Y = X(j)$ for some $j \in \{1, \ldots, n - m + 1\}$, where $X(j) = x_j x_{j+1} \cdots x_{j+m-1}$ is the substring of length m at position j in X. A naïve algorithm would simply scan every possible substring, consuming $O(mn)$ time. There are fancier deterministic algorithms that can solve this task in $O(m + n)$ time, but we now describe a very elegant randomized algorithm. While this algorithm is not used much for string matching, it is widely used when one wants to check for more than one string, i.e. when we have substrings $Y_1, Y_2, \ldots, Y_k$ and we would like to know if any Y_i occurs as a substring of X.

Karp-Rabin algorithm.

Choose a prime $p \in \{2, 3, \ldots, T\}$ uniformly at random.
Compute $H_p(Y) = Y \bmod p$.
For $j = 1, 2, \ldots, n - m + 1$,
  Compute $H_p(X(j))$
  Check $H_p(X(j)) \stackrel{?}{=} H_p(Y)$
  If yes, output "matched at position j (?)"
Output "no match found."

Observe that this algorithm again has one-sided error: If there *is* a match, then we will always see it. On the other hand, we can produce false-positives. This algorithm can be converted into a *zero-error* algorithm: Whenever we find a potential match, we can actually check whether there is really a match at position j. (Of course, if we accidentally output lots of false-positives, this could make the running time blow up.)

We should also note that a naïve implementation of this algorithm again has the poor running time $O(mn)$. But we can do something more clever: We can compute the hash of $X(j+1)$ from the hash of $X(j)$ by shifting:

$$H_p(X(j+1)) = 2(H_p(X(j)) - 2^{j-1} x_j) + x_{j+m-1} \bmod p\,.$$

If the prime p fits in a machine word, then this update only takes $O(1)$ time. So our whole algorithm runs in time $O(m+n)$.

Correctness of the algorithm. Clearly if there is a match at position j, the algorithm will output that match. On the other hand, the algorithm may sometimes output false-positives. Let's bound the probability of this happening.

Note that, as in our analysis in the preceding section, for the algorithm to make such an error, it must be that $p \mid |Y - X(j)|$ for some position $j \in \{1, 2, \ldots, n-m+1\}$. Hence, it must be that

$$p \mid \prod_{j=1}^{n-m+1} |Y - X(j)| \,.$$

Observe that since each factor $|Y - X(j)|$ is an m-bit number, the product is an mn-bit number. So as in our previous analysis, we have

$$\mathbb{P}[\text{error}] \leqslant \frac{\pi(mn)}{\pi(T)} \leqslant 1.26 \frac{mn \ln T}{T \ln(mn)} \,.$$

Thus if we choose $T = cmn$, we an error probability of at most $O(1/c)$ using only a $O(\log c + \log n)$-bit hash.

Remark 2.1. Note that in real applications, we often care about *noisy* matches. This is the case when Y is only *close* to a substring of X. The hash functions H_p are pretty lousy for this purpose since they can map two strings Y and Y' that differ in only one bit to very different numbers modulo p. Later we will study *locality sensitive* hash functions that do much better in this respect.

3 When the answer is everywhere

Finally, let's consider a task known as *program checking*. Suppose we are given three $n \times n$ real matrices A, B, C, and someone claims that $A = BC$. We want to check this. Of course, we could do the multiplication BC ourself, but this is an expensive operation. The best asymptotic bounds for this task requires roughly $O(n^{2.373})$ time. But it turns out that we can be sure whether $A = BC$ with high confidence by doing only a few *matrix-vector* multiplications (which are much cheaper than *matrix-matrix multiplications*).

We will choose a vector $v = (v_1, v_2, \ldots, v_n) \in \{-1, 1\}^n$ uniformly at random and check whether $Av = (BC)v$. Note that since $(BC)v = B(Cv)$, this only requires three matrix-vector multiplications, and this can be done in $O(n^2)$ time.

Claim 3.1. If $A \neq BC$, then $\mathbb{P}[Av = BCv] \leqslant \frac{1}{2}$.

Proof. Let $D = A - BC$. Since $A \neq BC$, we know that $D \neq 0$. Let i, j be indices such that $D_{ij} \neq 0$. Then,

$$\mathbb{P}[Av = BCv] = \mathbb{P}[Dv = 0] \leqslant \mathbb{P}[(Dv)_i = 0] \,, \tag{3.1}$$

where $(Dv)_i = \sum_{k=1}^{n} D_{ik} v_k$ is the ith coordinate of Dv.

Now we write

$$(Dv)_i = D_{ij} v_j + \underbrace{\sum_{k \neq j} D_{ik} v_k}_{(*)} \,.$$

We can now employ a powerful principle (sometimes called the "principle of deferred decisions"): Let us fix all the values $\{v_k : k \neq j\}$. This fixes the sum (*). Since v_j is independent of $\{v_k : k \neq j\}$, even conditioned on those values, v_j takes the values -1 and 1 each with probability $1/2$. Since $D_{ij} \neq 0$ (by assumption), at most one of the two possible values of v_j can make the total sum equal to zero. Thus $\mathbb{P}[(Dv)_i = 0] \leqslant \frac{1}{2}$. Combined with (3.1), this completes the proof. □

Probability technique: Amplification using independence. An error probability of $\frac{1}{2}$ might not seem very good, but note that we can simply choose k *indepenent* random vectors $v^{(1)}, v^{(2)}, \ldots, v^{(k)} \in \{-1, 1\}^n$ and check that $Av^{(i)} = BCv^{(i)}$ for every $i = 1, 2, \ldots, k$. Now if $A \neq BC$, then by independence

$$\mathbb{P}[Av^{(i)} = BCv^{(i)} \,\forall i] \leqslant \left(\frac{1}{2}\right)^k .$$

Thus the error probability goes down exponentially with the number of samples.