Introduction to Algorithms October 10, 2005
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik D. Demaine and Charles E. Leiserson Handout 13

Problem Set 3 Solutions

Problem 3-1. Pattern Matching

Principal Skinner has a problem: he is absolutely *sure* that Bart Simpson has plagiarized some text on a recent book report. One of Bart's sentences sounds oddly familiar, but Skinner can't quite figure out where it came from. Skinner decides to see if some smart-alec MIT student can help him out.

Skinner gives you a DVD containing the full text of the Springfield public library. The data is stored in a ***binary string*** $T[1], T[2], \ldots, T[n]$, which we view as an array $T[1\,..\,n]$, where each $T[i]$ is either 0 or 1. Skinner also gives you the quote from Bart Simpson's book report, a shorter binary string $P[1\,..\,m]$, again where each $P[i]$ is either 0 or 1, and where $m < n$. For a binary string $A[1\,..\,k]$ and for integers i, j with $1 \le i \le j \le k$, we use the notation $A[i\,..\,j]$ to refer to the binary string $A[i], A[i+1], \ldots, A[j]$, called a ***substring*** of A. The goal of this problem is to determine whether P is a substring of T, i.e., whether $P = A[i\,..\,j]$ for some i, j with $1 \le i \le j \le n$.

For the purpose of this problem, assume that you can manipulate $O(\log n)$-bit integers in constant time. For example, if $x \le n^7$ and $y \le n^5$, then you can calculate $x + y$ in constant time. On the other hand, you may not assume that m-bit integers can be manipulated in constant time, because m may be too large. For example, if $m = \Theta(\log^2 n)$ and x and y are each m-bit integers, you *cannot* calculate $x + y$ in constant time. (In general, it is reasonable to assume that you can manipulate integers of length logarithmic in the input size in constant time, but larger integers require proportionally more time.)

(a) Assume that you have a hash function $h(x)$ that computes a hash value of the m-bit binary string $x = A[i\,..\,(i+m-1)]$, for some binary string $A[1\,..\,k]$ and some $1 \le i \le k-m+1$. Moreover, assume that the hash function is perfect: if $x \ne y$, then $h(x) \ne h(y)$. Assume that you can calculate the hash function in $O(m)$ time. Show how to determine whether P is a substring of T in $O(mn)$ time.

Solution: We compute the hash of the pattern string, and compare it to the hash of all possible length-m substrings of A, i.e., compare $h(P)$ to $h(A[i\,..\,(i+m-1)])$, for $1 \le i < n-m+1$. Since the hash function is perfect, $h(P) = h(A[i\,..\,(i+m-1)])$ if and only if $P = A[i,..\,(i+m-1)]$. There are $O(n)$ hash functions to compute, $O(n)$ comparisons of hash values, and each computation and comparison requires $O(m)$ time, for a total running time of $O(mn)$.

Note that because calculation of the hash function takes $O(m)$ time, this algorithm is not asymptotically any better than simply comparing the substrings directly. This part is designed as motivation for the rest of the problem.

(b) Consider the following family of hash functions h_p, parameterized by a prime number p in the range $[2, cn^4]$ for some constant $c > 0$:

$$h_p(x) = x \pmod{p} .$$

Assume that p is chosen uniformly at random among all prime numbers in the range $[2, cn^4]$. Fix some i with $1 \leq i \leq n - m + 1$, and let $x = T[i\,..\,(i + m - 1)]$. Show that, for an appropriate choice of c, and if $x \neq P$, then

$$\Pr_p \left\{ h_p(x) = h_p(P) \right\} \leq \frac{1}{n} .$$

Hint: Recall the following two number-theoretic facts: (1) an integer x has at most $\lg x$ prime factors; (2) the Prime Number Theorem: there are $\Theta(x/\lg x)$ prime numbers in the range $[2, x]$.

Solution: Both x and P have the same hash value only if $(x - P) = 0 \pmod{p}$, i.e., if p is a factor of $x - P$. Since x and P are both m-bit numbers, $(x - P)$ has at most $m + 1 \leq n$ bits. Since $(x - P) \leq 2^n$, $(x - P)$ has at most $\lg 2^n = n$ prime factors.

By the Prime Number Theorem, there are at least $\Omega\left(\frac{cn^4}{\lg cn^4}\right)$ primes in the interval $[2, cn^4]$.

$$\Pr\{h_p(x) = h_p(P)\} \leq \frac{n}{\Omega\left(\frac{cn^4}{\lg cn^4}\right)} = O\left(\frac{n(\lg c + 4\lg n)}{cn^4}\right).$$

For suitably chosen constants, we can show that this probability is $O(1/n)$.

A more formal argument proceeds as follows: There exists constants n' and c' such that for all $n > n'$, the number of primes in the interval $[2, cn^4]$ is at least $c'\,(cn^4/\lg cn^4)$. Therefore, for all $n > n'$, the probability of choosing a p that divides $(x - P)$ is at most

$$\Pr\{h_p(x) = h_p(P)\} \leq \frac{n}{c'\left(\frac{cn^4}{\lg cn^4}\right)} = \frac{n(\lg c + 4\lg n)}{c'cn^4}.$$

It is straightforward to verify that $\lg c + 4\lg n < n$ if $n \geq \max\{2\lg c, 64\}$.

Therefore, if we choose $c > 1/c'$ and $n \geq \max\{n', 2\lg c, 64\}$, we have

$$\Pr\{h_p(x) = h_p(P)\} \leq \frac{1}{c'cn^2} \leq \frac{1}{n^2} \leq \frac{1}{n}.$$

(c) How long does it take to calculate $h_p(x)$, as defined in part (b)? *Hint:* Notice that x is an m-bit integer, and hence cannot be manipulated in constant time.

Solution: Because $p \leq cn^4$, we know that p has $O(\lg n)$ bits. Assuming that we can manipulate $O(\lg n)$-bit integers in constant time, it is possible to compute $w \pmod p$ in constant time if w also has $O(\lg n)$ bits.

The computation becomes slightly more complicated, however, if x is an m-bit number and $m = \omega(\lg n)$. Instead, we compute $h_p(x)$ in $O(m)$ time by incrementally computing the hash functions of prefixes of x.

Let y be the k most-significant bits of the string $x = A[1\,..\,m]$, and suppose we have already computed $h_p(y) = y \pmod p$. Then, with a constant number of operations, we can compute the hash function of z, the $(k+1)$ most significant bits of x.

Interpreting the string $z = A[1\,..\,(k+1)]$ as a number, we have that $z = 2y + A[k+1]$. Therefore, $h_p(z)$ is just

$$h_p(z) = (2y + A[k+1]) \pmod p = (2h_p(y) + A[k+1]) \pmod p.$$

Given $h_p(y)$, computing $h_p(z)$ requires only three additional integer operations: a single left-shift, addition, and a division. Thus, we perform a constant amount of work for every bit of x.

The purpose of this part is to realize that $h_p(x)$ can not be computed in constant time, and therefore the algorithm in part (d) represents an asymptotic improvement. An answer of $O(m)$ is acceptable. Note that we could do this computation faster by dividing x into $m/(c\lg n)$ digits, each $c\lg n$ bits long. The previous algorithm could then be modified to calculate $h_p(x)$ in $O(m/\lg n)$ time.

(d) For $1 \leq i \leq n - m$, show how to calculate $h_p(A[(i+1)\,..\,(i+m)])$ in constant time if you already know the value of $h_p(A[i\,..\,(i+m-1)])$, as defined in part (b)?

Solution: Interpreting the strings as binary numbers, we have that

$$A[(i+1)\,..\,(i+m)] = 2\left(A[i\,..\,(i+m-1)] - 2^{m-1}A[i]\right) + A[i+m].$$

Taking both sides of this equation modulo p, we get

$$h_p(A[(i+1)\,..\,(i+m)]) = \left(2\left(h_p(A[i\,..\,(i+m-1)]) - 2^{m-1}A[i]\right) + A[i+m]\right) \pmod p.$$

Assume that we have already pre-computed the value $2^{m-1} \pmod p$ and stored this value in some variable α. To compute $h_p(A[(i+1)\,..\,(i+m)])$, we calculate $\alpha A[i] \pmod p$, and subtract this value from $h_p(A[i\,..\,(i+m-1)])$. Then we left-shift by 1 bit to multiply by 2, add in the bit $A[i+m]$, and compute the remainder modulo p. Since our hash values are $O(\lg n)$-bit integers, all these operations can be done in constant time.

(e) Using the family of hash functions from part (b), devise an algorithm to determine whether P is a substring of T in $O(n)$ expected time.

Solution: We can use the same algorithm as in part (a), of comparing the hash of P with the hash functions of all length-m substrings of A until we find a match or until we have exhausted the text A. We use a hash function h_p, selected from the hash family described in part (b), and we use the method from part (d) to incrementally compute the hash functions. Since h_p is not a perfect hash function, if we discover that the hash values match, we then compare the two strings to see if they are equal.

To analyze the runtime, we analyze two separate costs: the cost to compute the hash function for each substring, and the cost of comparing the substrings when the hash values match. We show that both these expected runtimes are $O(n)$, giving us the desired result.

Computing the first hash function, $h_p(A[1\,..\,m-1])$ requires $O(m)$ time. Using the method from part (d), however, computing the hash values of all length-m strings requires a total of $O(m+n) = O(n)$ time.

If the hash values match, the cost of comparing two length-m substrings is $O(m)$. Let X_i be an indicator random variable that is 1 if we have a false positive at index i, i.e., if $h_p(P) = h_p(A[i\,..\,(i+m-1)])$, but $P \neq A[i,..\,(i+m-1)]$. By part (b), $E[X_i] < 1/n$. Then the expected total cost of comparing substrings for false positives is

$$\sum_{i=1}^{n-m+1} E[X_i]O(m) = O(m) = O(n).$$

The cost of comparing substrings for a match is $O(m) = O(n)$, because the algorithm stops after finding one match.

Problem 3-2. 2-Universal Hashing

Let $\mathcal{H}$ be a class of hash functions in which each $h \in \mathcal{H}$ maps the universe U of keys to $\{0, 1, \ldots, m-1\}$. We say that $\mathcal{H}$ is **2-*universal*** if, for every fixed pair $\langle x, y\rangle$ of keys where $x \neq y$, and for any h chosen uniformly at random from $\mathcal{H}$, the pair $\langle h(x), h(y)\rangle$ is equally likely to be any of the m^2 pairs of elements from $\{0, 1, \ldots, m-1\}$. (The probability is taken *only* over the random choice of the hash function.)

(a) Show that, if $\mathcal{H}$ is 2-universal, then it is universal.

Solution: If $\mathcal{H}$ is 2-universal, then for every pair of distinct keys x and y, and for every $i \in \{0, 1, \ldots m-1\}$,

$$\Pr_{h\in\mathcal{H}}\left[\langle h(x), h(y)\rangle = \langle i, i\rangle\right] = \frac{1}{m^2}$$

There are exactly m possible ways for to have x and y collide, i.e., $h(x) = h(y) = i$ for $i \in \{0, 1, \ldots m-1\}$. Thus,

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \sum_{i=0}^{m-1} \left(\Pr_{h \in \mathcal{H}}[\langle h(x), h(y) \rangle = \langle i, i \rangle] \right) = \frac{m}{m^2} = \frac{1}{m}.$$

Therefore, by definition, $\mathcal{H}$ is universal.

(b) Construct a specific family $\mathcal{H}$ that is universal, but not 2-universal, and justify your answer. Write down the family as a table, with one column per key, and one row per function. Try to make m, $|\mathcal{H}|$, and $|U|$ as small as possible.

Hint: There is an example with m, $|\mathcal{H}|$, and $|U|$ all less than 4.

Solution: We can find an example $m = |\mathcal{H}| = |U| = 2$.

On a universe $U = \{x, y\}$, consider the following family $\mathcal{H}$:

	x	y
h_1	0	0
h_2	1	0

If we chose a random hash function from $\mathcal{H}$, the probability that two keys x and y collide is the probability of choosing h_1, or $1/m = 1/2$. Thus $\mathcal{H}$ is a universal hash family.

On the other hand, with a 2-universal hash family, for a randomly chosen hash function h, all the possible pairs of $\langle h(x), h(y) \rangle$, i.e., $\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle$ must be equally likely. In this example, $\langle h(x), h(y) \rangle$ never equals $\langle 0, 1 \rangle$ or $\langle 1, 1 \rangle$, so $\mathcal{H}$ is not 2-universal.

(c) Suppose that an adversary knows the hash family $\mathcal{H}$ and controls the keys we hash, and the adversary wants to force a collision. In this problem part, suppose that $\mathcal{H}$ is universal. The following scenario takes place: we choose a hash function h randomly from $\mathcal{H}$, keeping it secret from the adversary, and then the adversary chooses a key x and learns the value $h(x)$. Can the adversary now force a collision? In other words, can it find a $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$?

If so, write down a particular universal hash family in the same format as in part (b), and describe how an adversary can force a collision in this scenario. **If not**, prove that the adversary cannot force a collision with probability greater than $1/m$.

Solution: By adding one extra key to our previous example, we can construct a scenario where the adversary can force a collision.

On a universe $U = \{x, y, z\}$, consider the following family $\mathcal{H}$:

	x	y	z
h_1	0	0	1
h_2	1	0	1

$\mathcal{H}$ is still a universal hash family: x and y collide with probability 1/2, x and z collide with probability 1/2, and y and z collide with probability $0 < 1/2$.

The adversary can determine whether we have selected h_1 or h_2 by giving us x to hash. If $h(x) = 0$, then we have chosen h_1, and the adversary then gives us y. Otherwise, if $h(x) = 1$, we have chosen h_2 and the adversary gives us z.

(d) Answer the question from part (c), but supposing that $\mathcal{H}$ is 2-universal, not just universal.

Solution: With a 2-universal hash family, the adversary cannot force a collision with probability better than $1/m$. Essentially, knowing $h(x)$ gives the adversary *no information* about $h(y)$ for any other key y.

We can prove this formally using conditional probabilities. Suppose we choose a random hash function $h \in \mathcal{H}$, and then the adversary forces us to hash some key x and learns the value $h(x) = X$. Then the adversary gives us any key $y \neq x$, hoping to cause a collision. By definition of 2-universality, we have for any x and y with $x \neq y$,

$$\Pr_{h \in \mathcal{H}}[h(y) = h(x) \mid h(x) = X] = \frac{\Pr_{h \in \mathcal{H}}[h(y) = h(x) \text{ and } h(x) = X]}{\Pr_{h \in \mathcal{H}}[h(x) = X]} = \frac{1/m^2}{1/m} = \frac{1}{m}.$$

Therefore, no matter which x the adversary chooses first, and which $h(x) = X$ value it learns, the probability of any particular y colliding with x is only $1/m$.