6.046
PROBLEM SET 4

JOHN WANG

Collaborators:

1. Problem A

Problem 1.1. *Define $f_p : \{0,1\}^m \to \mathrm{Z}$ as $f_p(X) := g(X) \pmod p$, where p is a prime and $g : \{0,1\}^m \to \mathrm{Z}$ is a function that converts an m bit binary string to a corresponding base 2 integer. Take the set $P = \{p_1, p_2, \ldots, p_t\}$, where p_i are all primes less than some large integer K. Suppose we choose a prime p uniformly at random from the set P and take m bit strings X and Y such that $X \neq Y$. Prove that we can bound the probability of a false positive as $P(f_p(X) = f_p(Y)) \leq m/t$.*

Solution We know that $f_p(X) = f_p(Y)$ if and only if $g(X) \equiv g(Y) \pmod p$ for some prime p from the set $P = \{p_1, \ldots, p_t\}$. This is equivalent to $|g(X) - g(Y)| \equiv 0 \pmod p$. Moreover, since $g(X)$ and $g(Y)$ are both integers, we know that $|g(X) - g(Y)|$ is also an integer. By the Fundamental Theorem of Arithmetic, it can be factored into primes: $|g(X) - g(Y)| = q_1^{e_1} q_2^{e_2} \ldots q_r^{e_r}$ where q_i are prime. However, we know that $r \leq m$ because there are at most m bits in each string.

Moreover, we know that each prime q_i has a $1/t$ chance of being equal to p. Since selecting any of the q_i are independent, we see that the probability of $g(X) = g(Y) \pmod p$ is just the probability that any of the q_i are equal to p. Since there are at most m such q_i, each with a probaiblity of $1/t$ of being selected, we see that $P(f_p(X) = f_p(Y)) = m(1/t) = m/t$. □

2. Problem B

Problem 2.1. *Design a randomized algorithm that determines if there is a match between a pattern and target text for offset j, where $j \in \{1, 2, \ldots, n - m + 1\}$.*

Solution Choose a random prime from the set $P = \{p_1, p_2, \ldots, p_t\}$ where P consists of all primes less than some large integer K. Define $f_p(X)$ as above, and define λ as the pattern and $\gamma(j)$ as the target text starting at offset $j \in \{1, 2, \ldots, n - m + 1\}$ of length m bits. Now, compute $f_p(\lambda)$, and compute $f_p(\gamma(j))$. If $f_p(\lambda) = f_p(\gamma(j))$, then check to make sure that that $\lambda = \gamma(j)$ by comparing the strings bit by bit. Note that to pick p, we don't actually enumerate $P = \{p_1, \ldots, p_t\}$. Instead, we pick a random number less than K, and check if it is prime using a primality testing algorithm, and loop until we find a prime.

To examine the runtime of this algorithm, we first note that there is an algorithm that runs in polynomial time for primality testing. In other words, we can test whether an b bit number is prime in $poly(b)$ time. Since the number of primes less than k is equal to $k/\log k$, the expected number of numbers we have to check is the density of primes, or $k/(k/\log k) = \log k$. The expected running time to find p will then be $\log(k) * poly(\log k) = poly(\log k)$.

In the worst case, we will have $f_p(\gamma(j)) = f_p(\lambda)$ and the algorithm will have to compare the two strings, which will take $O(m)$ time. The expected worst case run time is just the probability of a false positive, times the time it takes to evaluate the false positive, which is $O(m^2/t)$. One can choose t to be as large as necessary, so this run-time is very small. Notice that if $t = m^2$, then the runtime becomes $O(1)$–this observation will be used in later algorithms. □

3. Problem C

Problem 3.1. *Design a formula that given $g(X(j))$ computes $g(X(j+1))$ where $X(j)$ is a length m substring of the target test that starts at position j, where $j \in \{1, 2, \ldots, n - m + 1\}$. Use it to compute $f_p(X(j+1))$ from $f_p(X(j))$.*

Solution If we expand out $g(X(j))$, denoting x_i as the ith element in the target string, we obtain:

$$g(X(j)) = x_j 2^m + x_{j+1} 2^{m-1} + x_{j+2} 2^{m-2} + \ldots + x_{j+m} \tag{3.1}$$

Therefore, we know that we can obtain $g(X(j+1))$ by using the following manipulations:

$$\begin{aligned}
g(X(j+1)) &= x_{j+1}2^m + x_{j+2}2^{m-1} + \ldots + x_{j+m+1} && (3.2)\\
&= \left((x_j 2^m + x_{j+1}2^{m-1} + \ldots + x_{j+m} - x_j 2^m\right) 2 + x_{j+m+1} && (3.3)\\
&= \left(g(X(j)) - x_j 2^m\right) 2 + x_{j+m+1} && (3.4)\\
&= 2g(X(j)) - x_j 2^{m+1} + x_{j+m+1} && (3.5)
\end{aligned}$$

Now we can compute $f_p(X(j+1))$ by simplying computing $g(X(j+1))$ and modding it by p, since addition and multiplication are preserved under the modulo:

$$f_p(X(j+1)) = 2f_p(X(j)) - x_j 2^{m+1} + x_{j+m+1} \pmod p \tag{3.6}$$

This completes the formula for deriving $f_p(X(j+1))$ from $f_p(X(j))$. □

4. Problem D

Problem 4.1. *Suppose that $X(j)$ and Y differ at every string position. What is the expected number of positions such that $f_p(X(j)) = f_p(Y)$?*

Solution We use the probability bound we derived in part a, namely that if $X \neq Y$, then $P(f_p(X) = f_p(Y)) \leq m/t$ and define M as a random variable of the number of positions such that $f_p(X(j)) = f_p(Y)$ for $j \in \{1, 2, \ldots, n-m+1\}$. Also define M_i as indicator random variables for whether or not there are i positions $\{j_1, j_2, \ldots, j_i\}$ such that $f_p(X(j_i)) = f_p(Y)$. We can derive a worst case expected number of positions as:

$$\begin{aligned}
\mathbb{E}[M] &= \sum_{i=0}^{n-m} M_i P(M_i = 1) && (4.1)\\
&= \sum_{i=0}^{n-m} M_i P(f_p(X(j_i)) = f_p(Y)) && (4.2)\\
&\leq \sum_{i=0}^{n-m} \frac{m}{t} && (4.3)\\
&= \frac{m(n-m+1)}{t} && (4.4)
\end{aligned}$$

We see that $\mathbb{E}[M] = O\left(\frac{m(n-m)}{t}\right) = O\left(\frac{mn}{t}\right)$. □

5. Problem E

Problem 5.1. *Using parts above, design a randomized algorithm that determines if there is a match between a pattern and a target text in $O(n+m)$ expected running time. The algorithm should always return the correct answer.*

Solution Consider the following algorithm. First, all primes in $P = \{p_1, \ldots, p_t\}$ are less than some integer k, which will be chosen such that $m^2 = \frac{k}{\log k}$. Note that we can compute k in $O(\log m^2) = O(2\log m) = O(\log m)$ time using Newton's Method. To use Newton's method, we define $f(k) = m^2 - k/\log(k)$ so that we can find a root of $f(k)$ iteratively using Newton's method of $k_{i+1} = k_i - f(k_i)/f'(k_i)$. We have the following recursive formula to compute k:

$$k_{i+1} = k_i - \frac{(m^2 \log k - k)\log k}{\log k - 1} \tag{5.1}$$

This works and has quadratic convergence as long as $k \neq 0$, which should not happen since $n > 0$. Quadratic convergence follows because $f(k)$ is continuously differentiable for $k > 0$, $f'(\alpha) \neq 0$ at the root α, and $f''(\alpha)$ exists.[1] Since the convergence is quadratic in n, we know that we can find k in $O(\log n)$ time.

Next, we randomly select $p \in \{p_1, p_2, \ldots, p_t\}$ such that for each p_j, we have $p_j < k$. This can be done in $O(1)$ time as in the directions of the Pset. Now, we compute $f_p(X(j))$ for all $j \in \{1, 2, \ldots, n-m+1\}$. We can do this by first computing $f_p(X(0))$ by using Horner's method to compute $t_1 2^m + t_2 2^{m-1} + \ldots + t_m$ using:

$$t_1 2^m + t_2 2^{m-1} + \ldots + t_m = t_m + 2(t_{m-1} + 2(t_{m-2} + \ldots + 2(t_2 + 2t_1)\ldots)) \tag{5.2}$$

[1]This follows from a theorem on Wikipedia, whose proof is given here: `http://en.wikipedia.org/wiki/Newton's_method#Proof_of_quadratic_convergence_for_Newton.27s_iterative_method`.

Next, we can compute $f_p(X(1))$ by applying the transformation from problem C. We use this recursively to compute $f_p(X(j+1))$ from $f_p(X(j))$ for all the j. Next, we compute $f_p(Y)$ using Horners method to compute $p_1 2^m + \ldots + p_m$ in the same way we computed $f_p(X(0))$. Next, we iterate through $f_p(X(j))$ for all $j \in \{1, 2, \ldots, n-m+1\}$. If $f_p(X(j)) = f_p(Y)$, then check whether the bits t_j through t_{j+m} match the bits in the pattern string Y. If the bits match, then append this j to the output. If the bits do not match, do nothing. Continue looping through all j, and return all the matches that have been found.

First, we start with proof of correctness. We know that if $X(j) = Y$, then we must have $f_p(X(j)) = f_p(Y)$ for any $p \in \{p_1, p_2, \ldots, p_t\}$. This follows directly from problem a. Next, we know that if we check the bits of $X(j)$ with Y, then we can be sure whether $X(j)$ is the same as Y with certainty. Therefore, if $X(j) = Y$ for any j, then the algorithm will correctly identify that $f_p(X(j)) = f_p(Y)$ and then that $X(j) = Y$ by checking the bits. Next, we need to prove that the algorithm will not return a false positive. Comparing the bits in $X(j) = Y$ returns a positive output if and only if $X(j) = Y$. Therefore, the algorithm will never return a false positive since we will always check the bits. Therefore, the algorithm must return a correct output.

Next, we will analyze the runtime. We can compute k in $O(\log m)$ time, as mentioned in the presentation of the algorithm. We can compute $f_p(X(0))$ using Horner's rule in $O(m)$ time, since $X(0)$ is a polynomial of degree m. Computing all of the $f_p(X(j))$ for $j \in \{1, 2, \ldots n-m+1\}$ requires $O(n-m)$ time since computing $f_p(X(j+1))$ from $f_p(X(j))$ requires $O(1)$ arithmetic operations, each costing $O(1)$ time. Therefore, the total time for computing $f_p(X(j))$ for $j \in \{0, 1, \ldots, n-m+1\}$ takes $O(n+m)$ time. Each comparison of $f_p(X(j))$ with $f_p(Y)$ requires $O(1)$ time, so it takes $O(n-m) = O(n)$ to compare all the $f_p(X(j))$. If M is the number of matches where $f_p(X(j)) = f_p(Y)$, and therefore the number of comparisons required. The rest of the algorithm requires $O(Mm)$ time, since each comparison of two m bit strings requires $O(m)$ time. Thus, the running time is $T(n,m) = O(m) + O(\log m) + O(n-m) + O(n+m) + O(n) + O(Mm) = O(n+m) + O(Mm)$. To find the expected worst case running time, we invoke the results from part d about $\mathbb{E}[M]$, the expected number of positions such that $f_p(X(j)) = f_p(Y)$:

$$\mathbb{E}[T(n,m)] = \mathbb{E}[O(n+m) + O(Mm)] \tag{5.3}$$

$$= O(n+m) + \mathbb{E}[M]O(m) \tag{5.4}$$

$$= O(n+m) + O\left(\frac{m^2(n-m)}{t}\right) \tag{5.5}$$

Where we have substituted $E[M] = O\left(\frac{m(n-m)}{t}\right)$ from part d. Next, we know that $t \sim \frac{k}{\log k}$. Since we have set k such that $m^2 = \frac{k}{\log k}$, we see that $t \sim m^2$. This means that our expected runtime becomes:

$$\mathbb{E}[T(n,m)] = O(n+m) + O\left(\frac{m^2(n-m)}{m^2}\right) \tag{5.6}$$

$$= O(n+m) + O(n) \tag{5.7}$$

$$= O(n+m) \tag{5.8}$$

Now, we will go through an example of the algorithm. Let us have the pattern $Y = 2 = 10_2$ and the text string $X = 5 = 101_2$. First, we note that $m = 2$ and $n = 2$. We pick k such that $m^2 = 4 = k/\log k$. A close approximation is $k = 9$, which would have been computed using Newton's method. Now, we pick a random prime less than 9. Let us pick $p = 3$. Now, we compute $f_p(X(0))$ and $f_p(X(1))$. There are $f_p(X(0)) = 2 \pmod 3$ and $f_p(X(1)) = 1 \pmod 3$. We also compute $f_p(Y) = 2 \pmod 3$. We see that $f_p(X(0))$ matches $f_p(Y)$, and $f_p(X(1))$ does not. So we check to see if the bits of $X(0)$ match the bits of Y. Indeed, they are both 10, so the algorithm returns $j = 0$ as the only output. We can see by inspection that this is the correct output. □

6. Problem f

Problem 6.1. *Provide a bound for the probability that the running time is more than 100 times the expected running time.*

Solution Markov's inequality states that $P[X > c\mathbb{E}[X]] \leq \frac{1}{c}$ where c is a constant and X is a non-negative random variable. Since $T(n,m)$ is a non-negative random varialbe (the runing time can never be negative, we can apply Markov's inequality to $\mathbb{E}[T(n,m)]$. We observe:

$$P\left[T(n,m) > 100\mathbb{E}[T(n,m)]\right] \leq \frac{1}{100} \tag{6.1}$$

Thus, the probability that the running time is more than 100 times the expected running time is less than 1/100. □