Randomized Algorithms - Project 2

Mathies Boile Christensen, Lukas Walther, Christian Budde Christensen

April 2014

Introduction

This project considers equality tests of multisets. A multiset $X = [x_1, \ldots, x_n]$ is equal to another multiset $Y = [y_1, \ldots, y_n]$ if they represent the same multiset of lines. In order to determine equality of multisets we represent them as polynomials. These polynomials can then be evaluated and compared deterministically or by randomization using the Schwartz-Zippel lemma. Since it is very inefficient to evaluate polynomials on a random number, we introduce fingerprinting by doing arithmetic modulo a large prime p. Unfortunately, this is not sufficient in order to achieve a good error bound. The result is allowed to be wrong with a probability of at most 2^{-12} when the multisets contains at most 2^{24} lines. This issue is resolved by introducing 2-universal hashing in the algorithm.

Part 1

The first part of the report contains a thorough examination of designing an efficient randomized fingerprint algorithm for multiset equality. In order to implement the algorithm we first need to reason about 2-universal hash functions which are defined by

Definition 1. *2-universal hash function*

$$x \neq y \Rightarrow \Pr[h(x) = h(y)] \leq \frac{1}{m}$$

where m is the size of the hash table and x and y are elements from the universe. This way of fingerprinting can then be used together with the Scwartz-Zippel lemma for equality testing of multisets.

Subproblem 1

We want to prove that given two distinct multisets $X = [x_1, \cdots, x_n]$ and $Y = [y_1, \cdots, y_n]$ then the probability that the multisets $h(X) = [h(x_1), \cdots, h(x_n)]$ and $h(Y) = [h(y_1), \cdots, h(y_n)]$, where h is uniformly random selected from H, are identical is bounded by

$$\Pr[h(X) = h(Y)] \leq \frac{n}{p}.$$

If two multisets X and Y are to be identical, then there exists a one-to-one mapping between the two sets. For $h(X)$ and $h(Y)$ to be identical, every element $h(x_i)$ has to be identical to some element in $h(Y)$. Because h is a randomly chosen 2-universal hash function, the probability that $h(x_i)$ is identical to a specific element in $h(Y)$ is bounded by $\frac{1}{p}$. The probability that $h(x_i)$ is identical to at least one element in $h(Y)$ must be

$$\Pr[h(x_i) \in h(Y)] = \frac{1}{p} + \frac{1}{p} + \cdots \leq \frac{n}{p}.$$

The probability that $h(X)$ is identical to $h(Y)$ is $\Pr[h(x_i) \in h(Y)]$ times the probability that the rest of the elements also have a match, given that $h(x_i)$ has a match. This probability has to be smaller or equal to $\frac{n}{p}$, because $\Pr[h(x_1) \in h(Y)]$ is multiplied by some probability, which is always smaller or equal than one.

Subproblem 2

We want to prove that if $f_{h(X)}$ and $f_{h(Y)}$ are distinct polynomials and $w \in F_p$ is chosen uniformly random then

$$\Pr[f_{h(X)}(w) = f_{h(Y)}(w)] \leq \frac{n}{p}.$$

Because all operations are done in an unique factorization domain we know that

$$f_{h(X)} = f_{h(Y)} \Leftrightarrow h(X) = h(Y)$$
$$f_{h(X)} \neq f_{h(Y)} \Leftrightarrow h(X) \neq h(Y).$$

By definition of the universal hash function we also know that

$$X = Y \Rightarrow h(X) = h(Y)$$

which by contraposition gives us

$$h(X) \neq h(Y) \Rightarrow X \neq Y.$$

Combining the assumption $f_{h(X)} \neq f_{h(Y)}$ and the above results gives us that $X \neq Y$. Thus we can conclude that

$$\Pr[f_{h(X)}(w) = f_{h(Y)}(w)] = \Pr[h(X) = h(Y)] \leq \frac{n}{p}$$

by applying the statement proved in subproblem 1.

Subproblem 3

We want to prove that the class of hash functions

$$H_s = \{h_{a_1,\ldots,a_s} \mid a_1, \ldots, a_s \in F_p\}$$

is 2-universal where $s = {}^{b}/_{16}$ represents the length of the line in unicode characters. This can be proven by showing that

$$\Pr[h(x) = h(y)] \leq \frac{1}{p}$$

where x and y are distinct elements from the universe $\{0,1\}^b$ and p is the size of the hash table. We can rewrite the probability as the following

$$\begin{aligned}\Pr[h(x) = h(y)] &= \Pr\left[\sum_{i=1}^{s} a_i x_i = \sum_{i=1}^{s} a_i y_i\right] \\ &= \Pr\left[\sum_{i=1}^{s} a_i x_i - \sum_{i=1}^{s} a_i y_i = 0\right].\end{aligned}$$

We now assume without loss of generality that $x_1 \neq y_1$. If that is not the case, we can choose some other x_i and y_i such that this holds, following associativity of sums and the fact that x and y are distinct. We rewrite the probability further

$$\begin{aligned}\Pr[h(x) = h(y)] &= \Pr\left[a_1 x_1 - a_1 y_1 = -\sum_{i=2}^{s} a_i x_i + \sum_{i=2}^{s} a_i y_i\right] \\ &= \Pr\left[a_1(x_1 - y_1) = -\sum_{i=2}^{s} a_i x_i + \sum_{i=2}^{s} a_i y_i\right].\end{aligned}$$

We now use the principle of deferred decisions by fixing $a_2, ..., a_s$ which enables us to define the constants

$$\begin{aligned}q &= x_1 - y_1 \\ k &= -\sum_{i=2}^{s} a_i x_i + \sum_{i=2}^{s} a_i y_i\end{aligned}$$

where $q \neq 0$ because x and y are distinct. By substituting we have that

$$\Pr[a_1 q = k] \leq \frac{1}{p}$$

since there exists at most one element a_1 in F_p that leads to equality. Hence H_s is 2-universal when a_i are chosen uniformly random from F_p.

Subproblem 4

Given the 2-universal hash function

$$h_{a_1,\ldots,a_s}(x_1,\ldots,x_s) = \sum_{i=1}^{s} a_i x_i$$

and the equivalent representation of multisets as polynomials

$$f_X(z) = (z - x_1)\cdots(z - x_n)$$

then we can design an efficient randomized fingerprint algorithm for multiset equality. The pseudo code is given below and is designed such that the multisets only need to be traversed once.

Data: multiset X and Y and a 31 bit prime p
choose $a_1, \ldots, a_{80}$ uniformly random from F_p;
choose r uniformly random from F_p;
$p = 1$;
$q = 1$;
for *line x in X* **do**
  split x into $x_1, \ldots, x_{80}$;
  $p = p \cdot (r - \sum_{i=1}^{80} a_i \cdot x_i)$;
end
for *line y in Y* **do**
  split y into $y_1, \ldots, y_{80}$;
  $q = q \cdot (r - \sum_{i=1}^{80} a_i \cdot y_i)$;
end
if $p - q = 0$ **then**
  return true;
else
  return false;
end

Algorithm 1: MS-EQ

This is a false-biased Monte Carlo algorithm and the error probability is controlled by the error probability of the 2-universal hash function, which has been calculated earlier, and of the Schwartz-Zippel lemma, which is bounded by

$$\frac{d}{|S|}$$

where d is the degree of the polynomium and S is the set from which the random input parameter is chosen from. We know that $d = n$ and $|S| = p$ which means that the error probability of the multiset equality algorithm is bounded by

$$\Pr[\text{MS-EQ}(X, Y, p) = \text{true} \mid X \neq Y] \leq \frac{n}{p} + \frac{n}{p} = \frac{2n}{p}.$$

Given that the multisets contains at most 2^{24} lines and that p is at 31 bit number then we get an error probability of

$$\Pr[\text{MS-EQ}(X, Y, p) = \text{true} \mid X \neq Y] \leq \frac{2 \cdot 2^{24}}{2^{31}} = 2^{25} \cdot 2^{-31} = 2^{-6}.$$

In order to achieve a lower error probability we can calculate two extra polynomials t and s, using fresh values for $a_1, \ldots, a_{80}$ and r, and return the conjunction $p - q = 0 \wedge t - s = 0$ which will yield an error probability of

$$2^{-6} \cdot 2^{-6} = 2^{-12}.$$

This probability can also be adjusted by choosing the prime p with more bits because that will make the denominator of the error probability larger.

Part 2

The second part of the project contains the performance results of the randomized algorithm compared with a deterministic algorithm. Both the efficiency and the error probability of the randomized algorithm is studied.

Subproblem 5

Given that the randomized fingerprint algorithm is false-biased, then we know that the result can only be erroneous when the algorithm claims that two multisets are equal. This is false whenever the supplied multisets are distinct, which means that the test data should consist of distinct multisets.

The level of distinctness between multisets affects the error probability. In the worst case scenario the multisets has all but one element in common. In the best case scenario all elements from the two multisets are distinct. The theoretic bound is computed by $2 \cdot 10^i / 2^{20}$. In the last row the theoretic bound exceeds one and becomes useless. In any case the theoretic bound holds. We also see that the fraction of errors increases as the multisets gets bigger. This makes sense because the polynomials will contain more terms which improves the chances that one of them will evaluate to zero, i.e. the entire polynomial will evaluate to zero. Furthermore, the data shows that the distinctness of the data indeed changes the error probability of the algorithm.

lines	iterations	best case input	worst case input	theoretic bound
10^2	10^7	$1.0 \cdot 10^{-6}$	$9.58 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$
10^3	10^6	$4.0 \cdot 10^{-6}$	$9.56 \cdot 10^{-4}$	$1.9 \cdot 10^{-3}$
10^4	10^5	$1.2 \cdot 10^{-4}$	$9.45 \cdot 10^{-3}$	$1.9 \cdot 10^{-2}$
10^5	10^4	$8.8 \cdot 10^{-3}$	$9.12 \cdot 10^{-2}$	$1.9 \cdot 10^{-1}$
10^6	10^3	$3.90 \cdot 10^{-1}$	$6.18 \cdot 10^{-1}$	1.0

Table 1: Fraction of errors using a prime of 20 bits and checking one polynomial

We can boost the error probability by checking two polynomials

lines	iterations	best case input	worst case input	theoretic bound
10^2	10^7	0.0	0.0	$3.6 \cdot 10^{-8}$
10^3	10^6	0.0	$2.0 \cdot 10^{-6}$	$3.6 \cdot 10^{-6}$
10^4	10^5	0.0	$2.0 \cdot 10^{-4}$	$3.6 \cdot 10^{-4}$
10^5	10^4	$1.0 \cdot 10^{-4}$	$7.3 \cdot 10^{-3}$	$3.6 \cdot 10^{-2}$
10^6	10^3	$1.5 \cdot 10^{-1}$	$3.33 \cdot 10^{-1}$	1.0

Table 2: Fraction of errors using a prime of 20 bits and checking two polynomials

Or we can boost the error probability by choosing a larger prime.

lines	iterations	best case input	worst case input	theoretic bound
10^2	10^7			$9.5 \cdot 10^{-5}$
10^3	10^6			$9.5 \cdot 10^{-4}$
10^4	10^5			$9.5 \cdot 10^{-3}$
10^5	10^4			$9.5 \cdot 10^{-2}$
10^6	10^3			$9.5 \cdot 10^{-1}$

Table 3: Fraction of errors using a prime of 21 bits and checking one polynomial

The running time of the randomized (double polynomial) and deterministic algorithm. When the multisets are small the overhead of the randomized algo-

rithm makes it slower than the deterministic. For larger input the randomized wins.

lines	iterations	randomized algorithm	deterministic algorithm
10^2	10^7	0.72	0.01
10^3	10^6	0.50	0.24
10^4	10^5	4.96	4.01
10^5	10^4	45.73	88.10
10^6	10^3		

Table 4: Runningtime in milliseconds

The deterministic algorithm is implemented by sorting and comparing the two multisets using the standard java library. Furthermore the theoretic bound is sound.