

Randomized Algorithms, Spring 2010: Project

version 1 – January 24, 2010

This project consists of two problems that each have both theoretical and practical aspects. The subproblems outlines a possible approach. If you follow the suggested approach, you must answer all subproblems to get full credit. However, you may choose a different approach, but then you should make a similarly thorough analysis and documentation to get full credit.

You are allowed to work in groups (of at most 3 students) and make a single hand-in per group. The report should be made using LaTeX or similar. Handwriting is not acceptable.

The evaluation of your report will put weight on simple short precise answers/documentation. For the programming subproblems, documentation includes

- a precise description of the purpose of the experiment
- a well motivated design of the experiment
- a clear description of the outcome of the experiment, and
- a careful analysis and discussion of the outcome compared to theoretical predictions.

For the experiments you will need a source of random / pseudo-random numbers. In your report, you must explain carefully what source you have used. You may use the links on the course web page.

The hand-in should include program source code files in addition to the report in pdf format. All files are packaged in a zip-file and submitted via the course web page.

Deadline for hand-in: Friday March 6th at 12 noon.

Problem 1

This problem considers robustness of a score (grade) given to students based on a multiple choice test. The problem set-up is based on the paper Frandsen and Schwartzbach, *A singular choice for multiple choice*, [<http://doi.acm.org/10.1145/1189136.1189164>].

Assume a multiple choice test consists of n questions, each having 4 choices. For each question precisely one choice is correct. Students are allowed to make 0 or 1 “check” (cross) for each question. The score for a question is 1 if the student has checked the correct choice, $-\frac{1}{3}$ if the student has checked a wrong choice and 0 if no choices are checked. The score for the test is computed as the sum of the scores for all questions. The maximal score is therefore n . We assume that the test is used only to decide pass/fail, and the threshold for passing is a 50% score, i.e. a score $\geq \frac{n}{2}$.

Define a *challenged* student to be a student that knows the answers to at most 40% of the questions.

Clearly, a challenged student has nothing to lose by guessing the answers to the questions he does not know. Let us assume that he accordingly puts down checks at random choices (one per question), so he leaves no questions unanswered.

Define a multiple choice test to be *good*, if the probability that a challenged student passes is at most 5%.

A teacher has to make a test, and naturally he wants it to be *good*. He suspects that if he has enough questions in the test then it will be good. This is indeed correct.

In this problem you are required to find a (small) size n of the test that ensures it is *good*. You are required to do this both theoretically using Chernoff bounds (see Motwani and Raghavan, ch.4) and experimentally with high confidence.

You may find the following notation helpful: Assume the challenged student guesses $m = \frac{3}{5}n$ questions, define

$$X_i = \begin{cases} 1, & \text{if the } i\text{th guess is correct} \\ 0, & \text{otherwise} \end{cases}$$

Define $X = \sum_{i=1}^m X_i$

1.1 Subproblem Determine $E[X]$.

1.2 Subproblem Show that the challenged student only passes if $X \geq \frac{3}{2}E[X]$.

1.3 Subproblem Determine a size n of the test for which the challenged student only passes with probability at most .05 using the Chernoff bound technique.

The teacher suspects that a much smaller n than the one resulting from the Chernoff bound really suffices to make the test good. He decides to run an experiment, where he simulates challenged students making guesses. For a candidate n , he simulates R challenged students. Define the following notation

$$Y_{n,i} = \begin{cases} 1, & \text{if the } i\text{th simulated challenged student passes the test} \\ & \text{with } n \text{ questions} \\ 0, & \text{otherwise} \end{cases}$$

Let $p_n = \Pr(Y_{n,i} = 1)$, i.e. the test is good precisely when $p_n \leq .05$. Let $Y_n = \sum Y_{n,i}$.

1.4 Subproblem Determine $E[Y_n]$.

The teacher decides to accept a test as good, if the outcome $Y_n \leq 0.04R$.

1.5 Subproblem Determine a value of R that is sufficiently large to ensure that a non-good test is accepted as good with probability at most 0.05. You should use the Chernoff bound technique.

Now comes the programming part.

1.6 Subproblem You should implement a method that given n, R uses a random/pseudorandom source to determine an experimental value for Y_n .

1.7 Subproblem Using that method you should construct and implement

an algorithm that determines a (small) n that ensures a good multiple choice test.

1.8 Subproblem Analyze what confidence you can have in the outcome from running the algorithm.

1.9 Subproblem Run the algorithm and compare the outcome of the experiment with the earlier theoretically determined minimal size of a good test.

Problem 2

This problem considers equality test of multisets. This can be used to check that a sorting algorithm works correctly. The problem set-up is based on the paper Blum and Kannan, *Designing programs that check their work*, [<http://doi.acm.org/10.1145/200836.200880>].

Consider the following example, where each line describes a single element of a multiset

Multiset A:

```
[Holger Uggerhøj,movies]
[Egon Hausgaard,soccer]
[Vera Nicolaisen,poker]
[Niels Vestergaard,riding]
[Silius Thur,archery]
[Egon Hausgaard,soccer]
[Winona Møller,gardening]
```

Multiset B:

```
[Winona Møller,gardening]
[Holger Uggerhøj,movies]
[Vera Nicolaisen,poker]
[Egon Hausgaard,soccer]
[Vera Nicolaisen,poker]
[Niels Vestergaard,riding]
```

[Silius Thur, archery]

In the example multisets A and B are distinct; one difference being that A has two occurrences of the element [Egon Hausgaard, soccer] while B has only a single occurrence of that element.

One may sort the multisets and check bitwise equality, but a potentially faster approach consists in taking randomized finger prints of each multiset and compare the fingerprints only (Motwani and Raghavan, ch.7 and 8). We expect such an approach to be faster, since a fingerprint may be computed in a single sweep through the multiset.

We formalize the problem:

Multiset Equality

Input: $X = [x_1, \dots, x_n], Y = [y_1, \dots, y_n]$, two multisets of lines
 k , a positive integer.

Output: Yes, if X and Y represent the same multiset of lines.
No, otherwise.

Assumption: A line consists of at most 80 characters, and each character is represented by at most 16 bits, so each element in the multiset is represented by at most $b = 80 \cdot 16 = 1280$ bits.

The output is allowed to be wrong with small probability. In particular we seek an algorithm that guarantees the error probability is at most 2^{-12} when we assume a multiset contains at most $n \leq 2^{24}$ lines. These bounds are selected to include the test data provided in

<http://www.cs.au.dk/~gudmund/randomF10/multisets/>

Here the files $test^i a$ and $test^i b$ each contains 10^i lines for $i = 1, \dots, 7$.

In this problem you are required to design, implement and test an efficient randomized fingerprint algorithm for multiset equality test. Your algorithm must be able to compute a fingerprint from a single sweep through the multiset (streaming algorithm). You must test your algorithm on the provided test data, and you must compare its performance with the performance of a (best available) deterministic algorithm (programme your own sorting based algorithm or use UNIX built-in sort and difference test or similar)

You are free to use your creativity in combination with the tools from the course with one restriction. Your solution must not be significantly less efficient than the suggested approach below, and it should preferably be faster than sorting. The error and time analysis of your approach should also be as rigorous as the analysis in the suggested approach.

Suggested approach

Define polynomials $f_X(z) = (z - x_1) \cdots (z - x_n)$ and $f_Y(z) = (z - y_1) \cdots (z - y_n)$. Here we have interpreted each line x_i or y_i as a b -bit integer.

Because of unique factorization in polynomial rings the polynomials f_X and f_Y are identical if and only if $X = [x_1, \dots, x_n]$ and $Y = [y_1, \dots, y_n]$ are identical when regarded as multisets.

If f_X and f_Y are distinct, then the Schwartz-Zippel Lemma tells us that we have a high probability of discovering this when evaluating f_X and f_Y on a randomly chosen number.

This suggests a simple probabilistic algorithm for multiset equality test: Evaluate polynomials f_X and f_Y on a random number and look for equality of results. However, this simple algorithm is highly inefficient, taking quadratic time in the bit size of the input, when using “school” arithmetic.

The main reason for the inefficiency of the simple algorithm is that the evaluation of polynomials f_X and f_Y on a b -bit number results in an enormous number containing bn or more bits. So the total evaluation takes $\Omega(n)$ arithmetic operations, some of which are applied to $\Omega(bn)$ bit numbers.

To improve efficiency we may use fingerprinting to make sure we only have to do arithmetic on numbers that are so small that they fit in a computer word.

Let $p = 2^{31} - 1$, a 31 bit prime number. Arithmetic modulo p may be done via 64 bit long arithmetic. Let F_p denote the finite field with p elements i.e. the numbers $\{0, 1, \dots, p - 1\}$ with arithmetic modulo p .

One might think that we just have to evaluate f_X and f_Y modulo p . Unfortunately this approach is too naive. We have to be a bit more sophisticated

to get a sufficiently good error bound.

Let H be a 2-universal class of hash-functions from $\{0, 1\}^b$ into F_p . (Don't worry about the construction of H yet)

2.1 Subproblem Show that if $X = [x_1, \dots, x_n]$ and $Y = [y_1, \dots, y_n]$ are distinct multisets and h is selected uniformly randomly from H then multisets $h(X) = [h(x_1), \dots, h(x_n)]$ and $h(Y) = [h(y_1), \dots, h(y_n)]$ are identical with probability at most n/p .

In the following, let $h \in H$. It is a fact that if $h(X) = [h(x_1), \dots, h(x_n)]$ and $h(Y) = [h(y_1), \dots, h(y_n)]$ are distinct multisets then $f_{h(X)}(z) = (z - h(x_1)) \cdots (z - h(x_n))$ and $f_{h(Y)}(z) = (z - h(y_1)) \cdots (z - h(y_n))$ are also distinct polynomials over the field F_p .

2.2 Subproblem Show that if $f_{h(X)}$ and $f_{h(Y)}$ are distinct and w is chosen uniformly randomly from F_p then $f_{h(X)}(w)$ and $f_{h(Y)}(w)$ are identical with probability at most n/p .

We still need to construct a class of 2-universal hash functions H . One might think that we could simply use the class from section 8.4.3 in Motwani and Raghavan. However, that construction requires us to use arithmetic modulo a prime larger than 2^b , which is inefficient when b is larger than the word size of our computer. We really only want to do arithmetic modulo p as defined earlier. However, there is a way around this obstacle.

Without loss of generality, we may assume that b is divisible by 16, and we let $s = b/16$. (if the bit string represents a line of unicode characters then s is simply the length of the line). A bit string of length b may be divided into s blocks of length 16, where each block can be interpreted as a number in F_p . This suggests defining a 2-universal class H_s of hash functions from F_p^s into F_p , and use H_s as our H .

Define $H_s = \{ h_{a_1, \dots, a_s} \mid a_1, \dots, a_s \in F_p \}$, where $h_{a_1, \dots, a_s}(x_1, \dots, x_s) = \sum_{i=1}^s a_i x_i$.

2.3 Subproblem Show that H_s is 2-universal (Hint: use Principle of Deferred Decisions, MR section 7.1).

2.4 Subproblem Formulate and analyse a randomized fingerprint algorithm for testing whether two multisets are equal (based on your answers to

the previous subproblems). Hint: you may first design an algorithm with error probability at most 2^{-6} for $n \leq 2^{24}$ and afterwards improve it to obtain error probability at most 2^{-12} .

2.5 Subproblem Implement your algorithm and compare its performance on the test data with the performance of a (best) deterministic (sorting based) algorithm.