

Randomized Algorithms Project

Pooya Davoodi

March 6, 2009

1 Multiple Choice Test

1.1 Subproblem

We know that $Pr[X_i = 1] = \frac{1}{4}$ because the challenged student guesses the solution for these m questions uniformly.

$$E[X] = E\left[\sum_{i=1}^m X_i\right] = \sum_{i=1}^m E[X_i] = \sum_{i=1}^m Pr[X_i = 1] = \sum_{i=1}^m \frac{1}{4} = \frac{m}{4}$$

$$m = \frac{3}{5}n \Rightarrow E[X] = \frac{3n}{20}$$

1.2 Subproblem

We know that a student who gets a score in the test, passes if $score \geq \frac{n}{2}$. Also we know that the student answers too m questions randomly and X of these answers are correct.

$score = \text{score of random answers} + \text{score of known answers} = (X - \frac{1}{3}(m - X)) + \frac{2}{5}n \Rightarrow$
($n = \frac{5}{3}m$ and $score \geq \frac{n}{2}$) \Rightarrow

$$\frac{4}{3}X \geq \frac{1}{2}m \Rightarrow X \geq \frac{3}{8}m$$

($E[X] = \frac{m}{4}$) \Rightarrow

$$X \geq \frac{3}{2}E[X]$$

1.3 Subproblem

Challenged student answers to the questions independently, therefore all the X_i 's are independent variables and we can use the Chernoff bound for them which is

$$Pr[X > (1 + \delta)\mu] < \left[\frac{e^\delta}{(1 + \delta)^{(1 + \delta)}} \right]^\mu$$

In subproblem 1.2 we showed that a challenged student only passes if $X \geq \frac{3}{2}\mu$, therefore $(1 + \delta) = \frac{3}{2}$ and

$$\delta = \frac{1}{2}$$

and

$$\left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}} \right]^\mu = \left[\frac{e^{\frac{1}{2}}}{\left(\frac{3}{2}\right)^{\frac{3}{2}}} \right]^{\frac{3n}{20}}$$

We need to find a size n of the test such that $Pr[X > \frac{3}{2}\mu] < 0.05$. Therefore $\left[\frac{e^{\frac{1}{2}}}{\left(\frac{3}{2}\right)^{\frac{3}{2}}} \right]^{\frac{3n}{20}} \leq 0.05$

$$\frac{3n}{20} \ln \left[\frac{e^{\frac{1}{2}}}{\left(\frac{3}{2}\right)^{\frac{3}{2}}} \right] \leq \ln 0.05$$

$$\frac{3n}{20} \left(\frac{1}{2} - \frac{3}{2} \ln \frac{3}{2} \right) \leq \ln 0.05$$

$$\left(\frac{1}{2} - \frac{3}{2} \ln \frac{3}{2} \right) < 0 \implies$$

$$n \geq 184.5839$$

or

$$m \geq 110.750351$$

Consequently $n = 185$ and $m = 110$ are the smallest numbers to be sure that a challenged student passes with probability at most 0.05 (Indeed this is an upper bound based on the Chernoff bound).

1.4 Subproblem

$$E[Y_n] = E \left[\sum_{i=1}^R Y_{n,i} \right] = \sum_{i=1}^R E[Y_{n,i}] = \sum_{i=1}^R Pr[Y_{n,i} = 1] = \sum_{i=1}^R p_n = p_n R$$

1.5 Subproblem

The test is not good $\implies p_n \leq 0.05$,

but is accepted as a good test $\implies Y_n \leq 0.04R$.

We know that the challenged students answer to the questions independently from each other. Therefore $Y_{n,i}$'s are independent variables and we can use the following Chernoff bound to solve the problem

$$Pr \left[Y_n < (1 - \delta)\mu \right] \leq e^{-\frac{\mu\delta^2}{2}}$$

$$(\mu = p_n R) \implies$$

$$Pr \left[Y_n < (1 - \delta)p_n R \right] \leq e^{-\frac{p_n R \delta^2}{2}}$$

By using this bound, we need to find R such that $Pr[Y_n < 0.04R] \leq 0.05$.

Because of $(p_n R \delta^2) \geq 0$, when we increase the value of p_n and δ , the value of $e^{-\frac{p_n R \delta^2}{2}}$ is decreased. Therefore $e^{-\frac{p_n R \delta^2}{2}}$ is maximum when p_n and δ are minimum.

$$\min(p_n) = 0.05$$

$$(1 - \delta)p_n = 0.04 \Rightarrow \min(\delta) = \frac{1}{5}$$

$$Pr[Y_n < 0.04R] \leq e^{\frac{-0.05R(\frac{1}{5})^2}{2}} = e^{\frac{-R}{1000}}$$

$$(Pr[Y_n < 0.04R] \leq 0.05) \Rightarrow$$

$$e^{\frac{-R}{1000}} \leq 0.05$$

$$R \geq -1000 \cdot \ln 0.05 \Rightarrow R \geq 2995.732$$

Consequently the smallest number of challenged students to ensure that a non-test good is accepted as good with probability at most 0.05 is $R = 2996$.

1.6 Subproblem

We implemented EXPERIMENT- Y_n which computes Y_n as the number of challenged students who pass the test.

EXPERIMENT- $Y_n(n, R)$

```

1   $m \leftarrow \frac{3}{5}n$ 
2   $Y_n \leftarrow 0$ 
3  for  $i \leftarrow 1$  to  $R$ 
4      do  $score \leftarrow 0$ 
5          for  $j \leftarrow 1$  to  $m$ 
6              do  $answer \leftarrow$  a random number in  $\{1, 2, 3, 4\}$ 
7                  if  $answer$  is correct
8                      then  $score \leftarrow score + 3$ 
9                      else  $score \leftarrow score - 1$ 
10              $score \leftarrow \frac{score + 3 * \frac{2}{5}n}{3}$ 
11             if  $score \geq \frac{n}{2}$ 
12                 then  $Y_n \leftarrow Y_n + 1$ 
13 return  $Y_n$ 

```

Time Analysis: The loop in line 3 has R iterations and the loop in line 5 has m iterations. All the operations in the algorithm take constant time. Therefore the time complexity of the algorithm is $O(m.R) = O(n.R)$

1.7 Subproblem

EXPERIMENT- $n(R)$

```

1   $n \leftarrow 1$ 
2  while true
3      do make a test with  $n$  questions
4           $Y_n \leftarrow$  EXPERIMENT- $Y_n(n, R)$ 
5          if  $\frac{Y_n}{R} \leq 0.04$ 
6              then return  $n$ 
7           $n \leftarrow n + 1$ 
8  return  $n$ 

```

Time Analysis: This algorithm has a loop in lines 2-7 which has an unknown number of iterations T that depends on R and Y_n which is specified randomly. Making the test in line 3 takes $O(n)$ time to set the correct answers of all the questions. Computing Y_n in line 4 has $O(n.R)$ complexity (as mentioned in subproblem 1.6) and all the other operations are performed in constant time. Therefore the time complexity of EXPERIMENT-N is $O(T.n.R)$.

1.8 Subproblem

We show that the error probability of the algorithm mentioned in subproblem 1.7 is at most p_e if $R \geq -1000 \cdot \ln \left(1 - (1 - p_e)^{\frac{1}{184}} \right)$,

$$Pr[\text{Error of Algorithm}] = Pr[\text{Algorithm returns a non-good } n]$$

We know that the number of questions for a good test has a minimum threshold t such that every test of $n \geq t$ questions is a good test and every test of $n < t$ questions is a non-good test. It is clear that $t \leq 185$ since in subproblem 1.3 we showed that if $n \geq 185$ then we are sure that the test is a good one. Now we calculate $Pr[n < t]$ (error probability) such that n is the returned number by the algorithm,

$$\begin{aligned} Pr[n < t] &= 1 - Pr[n \geq t] = 1 - \left(\prod_{i=1}^{t-1} 1 - Pr[n = i] \right) \leq \\ &1 - \left(\prod_{i=1}^{t-1} 1 - Pr[n < t] \right) \leq 1 - \left(1 - Pr[n < t] \right)^{t-1} \leq \\ &1 - \left(1 - Pr[n < t] \right)^{185-1} \end{aligned}$$

since we need $Pr[n < t] \leq p_e$ we get a smaller bound,

$$1 - \left(1 - Pr[n < t] \right)^{184} \leq p_e$$

\Rightarrow

$$Pr[n < t] \leq 1 - (1 - p_e)^{\frac{1}{184}}$$

In subproblem 1.5 we proved that $Pr[\text{accepting a non-good test}]$ equals

$$Pr[n < t] = Pr[Y_n < 0.04R] \leq e^{\frac{-R}{1000}}$$

\Rightarrow

$$e^{\frac{-R}{1000}} \leq 1 - (1 - p_e)^{\frac{1}{184}}$$

\Rightarrow

$$R \geq -1000 \cdot \ln \left(1 - (1 - p_e)^{\frac{1}{184}} \right)$$

1.9 Subproblem

We used Random Integer Generator in RANDOM.ORG to make a file of 50000 lines such that each line contains an integer in $\{1, 2, 3, 4\}$. Each time that we read the last number in the file, we start again from the beginning. We tested the program with $R = 8186$ to have the at most 0.05 error probability

$$R \geq -1000 \cdot \ln \left(1 - (1 - 0.05)^{\frac{1}{184}} \right) \approx 8185.27$$

We ran the program 100 times and calculated the average of n . Also we compared the result with the case of using `rand()` function of C Library. The following table shows the result

RANDOM.ORG	rand()
64	54

2 Equality Test of Multisets

2.1 Subproblem

We know that $h(X) = h(Y)$ leads to a permutation π such that $\forall x_i \exists y_{\pi(i)} : h(x_i) = h(y_{\pi(i)})$. Therefore

$$\Pr[h(X) = h(Y) | X \neq Y] = \Pr \left[\bigcup_{i=1}^n h(x_i) = h(y_{\pi(i)}) | X \neq Y \right]$$

Also it is clear that

$$\Pr \left[\bigcup_{i=1}^n h(x_i) = h(y_{\pi(i)}) | X \neq Y \right] \leq \sum_{i=1}^n \Pr[h(x_i) = h(y_{\pi(i)}) | X \neq Y]$$

Based on the definition of 2-universal hash functions we have

$\forall i \in \{1, \dots, n\}$ and $\forall x_i, y_{\pi(i)} \in \{0, 1\}^b$ such that $x_i \neq y_{\pi(i)}$ and for h chosen uniformly at random from $H : \{0, 1\}^b \rightarrow F_p$

$$\Pr[h(x_i) = h(y_{\pi(i)}) | X \neq Y] \leq \frac{1}{p}$$

Consequently

$$\Pr[h(X) = h(Y) | X \neq Y] \leq \sum_{i=1}^n \Pr[h(x_i) = h(y_{\pi(i)}) | X \neq Y] \leq \frac{n}{p}$$

2.2 Subproblem

Suppose that for any $z \in F_p$, $Q(z) = f_{h(X)}(z) - f_{h(Y)}(z)$ is a univariate polynomial of degree n . Since $f_{h(X)}(z) \neq f_{h(Y)}(z)$, then $Q(z) \neq 0$ and we can use the Schwartz-Zippel Theorem to show the upper bound of $\Pr[Q(w) = 0 | Q(z) \neq 0]$ for any w chosen independently and uniformly at random from F_p

$$\Pr[Q(w) = 0 | Q(z) \neq 0] \leq \frac{n}{p}$$

2.3 Subproblem

We know that the family $H_s : \{0, 1\}^b \rightarrow F_p$ is said to be 2-universal if, for all $x = (x_1, \dots, x_s), y = (y_1, \dots, y_s) \in \{0, 1\}^b$ such that $x \neq y$, and for h chosen uniformly at random from H ,

$$\Pr[h_{a_1, \dots, a_s}(x_1, \dots, x_s) = h_{a_1, \dots, a_s}(y_1, \dots, y_s)] \leq \frac{1}{p}$$

Based on the definition of h , we have

$$\Pr[h_{a_1, \dots, a_s}(x_1, \dots, x_s) = h_{a_1, \dots, a_s}(y_1, \dots, y_s)] = \Pr\left[\sum_{i=1}^s a_i x_i = \sum_{i=1}^s a_i y_i\right] = \Pr\left[\left(\sum_{i=1}^s a_i (x_i - y_i)\right) = 0\right]$$

we define $d = (d_1, \dots, d_s) = x - y \neq 0$ and hence we need to show that

$$\Pr\left[\left(\sum_{i=1}^s a_i d_i\right) = 0\right] \leq \frac{1}{p}$$

Without loss of generality we can assume that $d_1 \neq 0$ (there must be at least one non-zero element in d because $d \neq 0$),

$$\Pr\left[\left(\sum_{i=1}^s a_i d_i\right) = 0\right] = \Pr\left[\left(a_1 = \frac{\sum_{i=2}^s a_i d_i}{d_1}\right) = 0\right]$$

Now we invoke the Principle of Deferred Decisions and assume that d_i for $(i \geq 2)$ are chosen before d_1 . Then $\frac{\sum_{i=2}^s a_i d_i}{d_1}$ is fixed at some value $v \in F_p$ and since d_1 is uniformly distributed over F_p ,

$$\Pr\left[\left(a_1 = \frac{\sum_{i=2}^s a_i d_i}{d_1}\right) = 0\right] = \Pr[d_1 = v] \leq \frac{1}{p}$$

2.4 Subproblem

To construct the function h , we need to generate s random numbers a_1, \dots, a_s and then calculate $h_{a_1, \dots, a_s}(x_1, \dots, x_s) = \sum_{i=1}^s a_i x_i$. All the a_i and x_i are in F_p and we do all the calculations in arithmetic modulo p .

- Random generating: We assume that $p \leq 2^{31} - 1$ and hence we use `rand()`, the new version of pseudo-random integral number generator of C library (Cstdlib). Because this function generates numbers in the range $[0, 2^{31} - 1]$ and we can change the limits of `rand()` to specify $[0, p - 1] \subseteq [0, 2^{31} - 1]$. Also if the generated number is larger than $p - 1$, we can calculate its remainder in division by p .
- For doing the calculations in $\sum_{i=1}^s a_i x_i$ we use the `mod(p)` operation after each multiplication and addition. We know that the result of multiplication of two 31 bit numbers would be a 62 bit number which can not be stored in an integer variable. Therefore we use casting before each multiplication to force the compiler to do a 64 bit multiplication. Since we assume that all the arithmetic operations modulo p take unit time, the calculation of $\sum_{i=1}^s a_i x_i$ is carried out in $O(s)$.

2.5 Subproblem

The algorithm MULTISETSEQUALITY converts two multi sets S_1 and S_2 to corresponding polynomials and calculates the values of the polynomials FPS_1 and FPS_2 for a random number w and returns the equality of S_1 and S_2 based on the equality of FPS_1 and FPS_2 .

MULTISETSEQUALITY(S_1, S_2)

```

1   $w \leftarrow$  a random number
2   $FPS_1 \leftarrow 1$ 
3   $FPS_2 \leftarrow 1 \triangleright FPS_1$  and  $FPS_2$  are the fingerprints of  $S_1$  and  $S_2$ 
4  for each  $x \in S_1$ 
5      do  $t \leftarrow h_{a_1, \dots, a_s}(x_1, \dots, x_s)$ 
6          if  $w < t$ 
7              then  $FPS_1 \leftarrow FPS_1 * (p + w - t)$ 
8              else  $FPS_1 \leftarrow FPS_1 * (w - t)$ 
9  for each  $x \in S_2$ 
10     do  $t \leftarrow h_{a_1, \dots, a_s}(x_1, \dots, x_s)$ 
11         if  $w < t$ 
12             then  $FPS_2 \leftarrow FPS_2 * (p + w - t)$ 
13             else  $FPS_2 \leftarrow FPS_2 * (w - t)$ 
14  if  $FPS_1 = FPS_2$ 
15     then return equal
16  else return not equal

```

Time Analysis: The operations in lines 1-3 and 12-14 are done in constant time. Clearly the two loops in this algorithm have the same running time. Therefore we only consider the first loop in lines 4-8. The number of iterations of the loop is $|S_1| = n$ and the calculation of h in line 5 takes $O(s)$ (as mentioned in subproblem 2.4). All the arithmetic operations in lines 6-8 are performed in constant time and consequently the time complexity of the algorithm is $O(s.n) = O(b.n)$.

Error Analysis: In subproblem 2.1 we showed that

$$\Pr \left[h(X) = h(Y) | X \neq Y \right] \leq \frac{n}{p}$$

\Rightarrow

$$\Pr \left[h(X) \neq h(Y) | X \neq Y \right] \geq 1 - \frac{n}{p}$$

Also in subproblem 2.2 we showed that

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

and we conclude that

$$\Pr \left[f_{h(X)}(w) \neq f_{h(Y)}(w) | h(X) \neq h(Y) \right] \geq 1 - \frac{n}{p}$$

We know that $\Pr[f_{h(X)}(w) \neq f_{h(Y)}(w) | X \neq Y]$ is the probability of the success of the algorithm and

$$\Pr \left[f_{h(X)}(w) \neq f_{h(Y)}(w) | X \neq Y \right] =$$

$$\Pr \left[f_{h(X)}(w) \neq f_{h(Y)}(w) | h(X) \neq h(Y) \right] \cdot \Pr \left[h(X) \neq h(Y) | X \neq Y \right] \geq \left(1 - \frac{n}{p} \right)^2$$

Therefore the error probability of the algorithm is

$$1 - \left(1 - \frac{n}{p} \right)^2$$

$(n \leq 2^{19})$ and $(p = 2^{31} - 1) \Rightarrow$

$$1 - \left(1 - \frac{n}{p} \right)^2 \leq 1 - \left(1 - \frac{2^{19}}{2^{31} - 1} \right)^2 \leq 2^{-11} - 2^{-24} \leq 2^{-11} \leq 2^{-10}$$

Performance: Also we calculated the running time of the program and compared it with the naive solution which is sorting of all the databases. The following table shows the result of the comparison. For sorting program we used the sort() function of C++ Library. The dataset of this experiment was 9 databases from the course home page. The solution of the fingerprinting approach was exactly the same as sorting approach. The hackers No.2, 6 and 7 have the real database in a different permutation and other hackers have random databases.

Sorting	FingerPrinting
24 second	2 second