Adv Data Struct & Algorithm: Homework 3

Zimian Li

May 5, 2018

1. Suppose you have a biased source of random bits that in $O(1)$ time returns 1 with probability p, and 0 with probability $1-p$. However, you do not know the value of p.

   (a) Design an efficient algorithm that uses the defective source to return 1 with probability 1/2 and 0 with probability 1/2.

Algorithm 1 Unbiased Random

```
1: procedure UnbiasedRandom
2:     while true do
3:         a = BiasedRandom()
4:         b = BiasedRandom()
5:         if a ≠ b then
6:             return a
7:         end if
8:     end while
9: end procedure
```

I called BiasedRandom() twice, there are 4 kinds of result:
1. $\{a = 0, b = 0\}$, with probability $(1-p)^2$;
2. $\{a = 0, b = 1\}$, with probability $p(1-p)$;
3. $\{a = 1, b = 0\}$, with probability $p(1-p)$;
4. $\{a = 1, b = 1\}$, with probability p^2;

And events $\{a = 0, b = 1\}$ and $\{a = 1, b = 0\}$ have the same probability, thus I decide to return a, it has 1/2 probability to return 0, and 1/2 probability to return 1.

(b) As a function of p what is the expected running time of your algorithm.

The probability of event $\{(a = 0, b = 1), (a = 1, b = 0)\}$ is $2p(1-p)$, so the expected running times of the while-loop is $1/(2p(1-p))$. And all other operations including getting a biased random bit is in $O(1)$ time. So the expected running time is $O(1/(2p(1-p)))$.

(c) Discuss how to modify your algorithm to return a random value from an arbitrary bernoulli trial, i.e., a random variable that returns 1 with probability q and 0 with probability $1-q$. (You may assume that q is a rational number with a finite binary expansion.)

q is a rational number, so it can be written as a fraction. Suppose $q = \frac{x}{y}, 0 \leq x \leq y, y \neq 0$.
Let's get back to the original algorithm of (a) above. I ran BiasedRandom() twice there, but this time let's suppose I modified it to ran BiasedRandom() n times, then I can get a string of 0s and 1s with length n. Think about all possible outputs of this string, it's obvious that there are $\binom{n}{i}$ of them with $i \cdot 1$ and $(n-i) \cdot 0$, $0 \leq i \leq n$. And all of them have the same probability $p^i(1-p)^{n-i}$.
Ok, now I can construct an event with probability $q = \frac{x}{y}$:
Step 1: Choose a right pair of n and i that can meet $\binom{n}{i} = y$. The easist way is let $n = y, i = 1$.
Step 2: Construct the sample space, all 01 strings with length y and only one "1".
Step 3: Construct the event, select x arbitrary strings from the sample space.

Algorithm 2 Unbiased Random with probability $q = \frac{x}{y}$

```
1: procedure UnbiasedRandomQ(x, y)
2:     Sample ← all 01 strings with length y and only one "1"
3:     Event ← Sample[0:x]
4:     while true do
5:         str = [], count = 0
6:         for i ← 0 to y-1 do
7:             a = BiasedRandom()
8:             str.append(a)
9:             if a == 1 then
10:                count += 1
11:            end if
12:            if count > 1 then
13:                break
14:            end if
15:        end for
16:        if count == 1 and str is in Event then
17:            return 1
18:        else
19:            if count == 1 and str is not in Event then
20:                return 0
21:            end if
22:        end if
23:    end while
24: end procedure
```

2. Consider a regular binary search tree T build on the elements of a set S. Without loss of generality, we may assume that $S = \{1, 2, ..., n\}$. In general, there are many different binary search trees for S. The exact shape of one of these trees depends on the order in which the elements of S were inserted into T. Assume that the elements are inserted in random order. For $x \in S$ and fixed T, let A_x denote the set of ancestors of x in T. Clearly, the worst case of $d(x)$ is $n-1$. However, we are interested in $E[d(x)]$. Let $x_{\leq} = \{1, 2, ..., x\}$ and $x_{\geq} = \{x, x+1, ..., n\}$. Let Y and Z be random variables defined as follows:
$Y = |x_{\leq} \cap A_x|, Z = |x_{\geq} \cap A_x|$

You are asked to analyze the expected values of Y and Z.

(a) Express $d(x)$ in terms of random variables Y and Z.

I'm not sure if the ancestors of a node should contain itself or not. Let's just suppose $x \notin A_x$ here.
$\because x_{\leq} \cup x_{\geq} = S, x_{\leq} \cap x_{\geq} = \{x\}, x \notin A_x$
$\therefore (x_{\leq} \cap A_x) \cup (x_{\geq} \cap A_x) = A_x, (x_{\leq} \cap A_x) \cap (x_{\geq} \cap A_x) = \emptyset$
$\therefore d(x) = |A_x| = |(x_{\leq} \cap A_x) \cup (x_{\geq} \cap A_x)| = |x_{\leq} \cap A_x| + |x_{\geq} \cap A_x| = Y + Z$

(b) Let $y \in x_{\leq}$. Provide necessary and sufficient conditions for $y \in A_x$.

y was inserted before x.

(c) For $y \in x_{\leq}$, let $I_y = I\{y \in A_x\}$. What is $E[I_y]$?

$E[I_y] = Pr[I_y] = Pr[y \in A_x] = Pr[y \text{ was inserted before } x] = 0 \cdot \frac{1}{n} + \frac{1}{n-1} \cdot \frac{1}{n} + \frac{2}{n-1} \cdot \frac{1}{n} + ... + \frac{n-1}{n-1} \cdot \frac{1}{n} = \frac{1+2+...+n-1}{n-1} \cdot \frac{1}{n} = \frac{n}{2} \cdot \frac{1}{n} = \frac{1}{2}$

(d) Express Y as a sum of indicator variables and compute $E[Y]$.

$E[Y] = \sum^{y \in x_{\leq}} E[I_y] = x \cdot \frac{1}{2} = \frac{x}{2}$

(e) Proceed similarly to compute $E[Z]$.

Similarly, $E[I_z] = \frac{1}{2}$, and $E[Z] = \sum^{z \in x_{\geq}} E[I_z] = (n - x + 1) \cdot \frac{1}{2} = \frac{n-x+1}{2}$

(f) Finally, use the results above to estimate $E[d(x)]$.

$E[d(x)] = E[Y] + E[Z] = \frac{x}{2} + \frac{n-x+1}{2} = \frac{n+1}{2}$

3. Suppose that two remote computers M_1 and M_2 store databases DB_1 and DB_2, respectively. The two databases need to be synchronized and need to store exactly the same contents. Let $x = x_1x_2...x_n$ denote the binary contents of DB_1 and $y = y_1y_2...y_n$ the contents of DB_2. You want to determine if $x = y$. Since a deterministic algorithm would have to transfer n bits to accomplish your goals, you decide, instead, to use the following randomized algorithm:

Step 1 Machine M_1 chooses a prime $p \leq n^2$ at random with uniform probability and computes s = x mod p.

Step 2 Machine M_1 sends s and p to M_2.

Step 3 Machine M_2 receives s and p and computes t = y mod p.

Step 4 Machine M_2 compares s with t and concludes that $x = y$ if $s = t$; otherwise, it concludes that $x \neq y$.

Suppose. for example, that $x = 01111$, and $y = 10110$. M_1 chooses $p = 5 \leq 5^2$ from $\{2, 3, 5, 7, 11, 13, 17, 19, 23\}$. Since x mod 5 = 15 mod 5 = 0, M_1 sends 0 and 5 to M_2. Upon computing y mod 5 = 22 mod 5 = 2, M_2 concludes that $x \neq y$, so the databases are not synchronized.

(a) Clearly, this randomized algorithm is of the Monte Carlo variety. Describe the circumstances under which it makes a wrong decision.

Sometimes $s = t$ which means by mod p, x and y get the same remainder, but $x \neq y$.
For example, $x = 01111$, and $y = 01010$. M_1 chooses $p = 5 \leq 5^2$ from $\{2, 3, 5, 7, 11, 13, 17, 19, 23\}$. Since x mod 5 = 15 mod 5 = 0, M_1 sends 0 and 5 to M_2. Then M_2 computes y mod 5 = 10 mod 5 = 0, M_2 concludes that $x = y$, but actually $x \neq y$.

(b) What is the probability that the algorithm makes a mistake for each of the following pairs of "databases"?

Set	x	y
1	01111	10110
2	1000011110100001	0010011001011000
3	01001101110010011011010000 1101	10010001110011001100 1011001110

If x mod $p = y$ mod p, then $x = i \cdot p + r, y = j \cdot p + r$, assume $x >= y$, thus $x - y = (i - j)p$.
So if $|x - y|$ is some times of p, it'll make mistakes.
For set 1, the prime number set is $\{2, 3, 5, 7, 11, 13, 17, 19, 23\}$, $|15 - 22| = 7$, hence, the probability is $\frac{1}{9}$.
For set 2, $|34721 - 9816| = 24905 = 5 \times 17 \times 293$, and I need to choose p from $\leq 16^2 = 256$, there are 54 prime numbers. Hence, the probability is $\frac{2}{54} = \frac{1}{27}$.
For set 3, $|326266125 - 611529422| = 285263297 = 11 \times 1999 \times 12973 \times 11 \times 1999 \times 12973$, and need to choose p from $\leq 30^2 = 900$, and there are 154 prime numbers. The probability is $\frac{1}{154}$.

(c) How many bits does the randomized algorithm need to transmit in general? How much faster than the deterministic algorithm is the randomized algorithm when comparing databases of 100Mb.

Suppose $x = x_1 x_2 ... x_n$. Therefore $p \leq n^2$, $s = x$ mod $p < p \leq n^2$. In this randomized algorithm, I need to transmit s and p. The size should be $O(2 \cdot (\lfloor log(n^2) \rfloor + 1))$.
If the size of database is 100Mb, the deterministic algorithm should transmit 100Mb data, however the randomized algorithm should transmit no more than $2 \cdot (\lfloor log((100Mb)^2) \rfloor + 1) \approx 26.6$Mb data.
Hence, the randomized algorithm is more than 3.8 times faster than the deterministic algorithm.

(d) Show that the number of distinct primes that divide an n-bit integer is less than n.

Proof: Obviously, the largest n-bit integer is $2^n - 1$. Suppose a positive integer x and $x <= 2^n - 1$, assume x can be can divided by $m (m \geq n)$ distinct prime numbers.

Thus, $x = c \cdot p_1 \cdot p_2 \cdot ... \cdot p_m$, $p_1, p_2, ..., p_m$ are distinct prime numbers, c is a positive integer. As we all know, 2 is the smallest prime number, so $x = c \cdot p_1 \cdot p_2 \cdot ... \cdot p_m > c \cdot 2^m$. Because $c \geq 1$ and $m \geq n$, therefore $x > c \cdot 2^m \geq 2^n$. But $x <= 2^n - 1$, it's a contradiction! Hence, the number of distinct primes that divide an n-bit integer is less than n.

(e) Compute the probability that the randomized algorithm makes a mistake for a database of n bits.

The subtraction of two n-bit databases is a n-bit integer, and per the conclusion of (d), the number of distinct prime numbers that divide this integer is less than n.
According to the prime counting function $\pi(x) \sim \frac{x}{ln(x)}$. The number of prime numbers that at most n^2 is around $\frac{n^2}{2ln(n)}$.
Considering the prime number that can divide the n-bit integer could be more than n^2, the probability that the randomized algorithm makes a mistake for a database of n bits is $< n/(\frac{n^2}{2ln(n)}) = \frac{2ln(n)}{n}$.

(f) Given a maximum tolerance for error $0 < \varepsilon \leq 1$, adapt the proposed algorithm so that the probability of reaching a wrong conclusion is no more than ε.

We need to repeat this algorithm more times, if all rounds of result return $x = y$, then return $x = y$, else if one of them return $x \neq y$, return $x \neq y$.
And the repeating number of times i should meet $(\frac{2ln(n)}{n})^i \leq \varepsilon$, so $i \geq \log_{\frac{2ln(n)}{n}} \varepsilon$.

4. You are given a set of n items $a_1, ..., a_n$ with a_i stored in location ℓ_i of a linked list A. To search for a_i you scan the list from left to right. Thus, the time to find a_i is proportional to ℓ_i. Let p_i be the probability that the item searched for is a_i. An arrangement, i.e., a choice of ℓ_i's, is optimal if it minimizes the average search cost.

(a) What is the average search cost?

Average search cost $= \ell_1 \cdot p_1 + \ell_2 \cdot p_2 + ... + \ell_n \cdot p_n = \sum_1^n (\ell_i \cdot p_i)$.

(b) Suppose that you are allowed to rearrange the items as you please and that you know the access probabilities $p_1, ..., p_n$. What list arrangement minimizes the average search cost? Prove your claim.

Put the item with the highest access probability to the first location, then the second highest to the second location, and so forth, until put the item with smallest access probability to the last location.
Suppose $p_1 \geq p_2 \geq ... \geq p_n$, $p_1 + p_2 + ... + p_n = 1$, the minimal average search cost should be $p_1 + 2p_2 + ... + np_n$.
Proof: Choose two arbitrary items of $p_1 + 2p_2 + ... + np_n$, suppose them be ip_i and jp_j, and $i < j$, $p_i \geq p_j$.
Swap i and j, then get jp_i and ip_j. Compare them with the original ones. $(jp_i + ip_j) -$

$(ip_i + jp_j) = (j - i)(p_i - p_j) \geq 0$.
Therefore, every swapping of factors of p_i and p_j will increase the whole value. So the current value is the minimal.

(c) We assume now that there is an underlying access probability distribution that this is not known to you. To obtain a reasonable search time you start with a randomly arranged list and decide that every time an element is accessed you will move it to the head of the list, leaving the rest if the items in the same relative ordering.

i. Suppose that a_i and a_j were accessed in the past. What is the probability that $\ell_i < \ell_j$?

Suppose c_i is the number of a_i were accessed, c_j is the a_j were accessed.
$\Pr(\ell_i < \ell_j) = c_i/(c_i + c_j)$

ii. Define decision variables $X_{ij} = I(a_i$ appears before $a_j)$. Express ℓ_i in terms of these decision variables and find $E(\ell_i)$.

$\ell_i = E(X_{1i}) + E(X_{2i}) + ... + E(X_{ni}) = \sum_{j=1, j\neq i}^{n} E(X_{ji})$
Suppose c be the total access number.
$E(\ell_i) = \sum_{i=1}^{n} c_i \ell_i / c$

iii. Assume that the algorithm has been running for a while. What is the expected search cost? (It is ok to analyze the performance in terms of a probability distribution that is not known to the algorithm).

$E(\ell_i) = \sum_{i=1}^{n} c_i \ell_i / c$

iv. Compare your answer to that of part (b) and explain why you have, in fact, designed a good approximation algorithm.

$\ell_i = E(X_{1i}) + E(X_{2i}) + ... + E(X_{ni}) = \frac{c_1}{c_i + c_1} + \frac{c_2}{c_i + c_2} + ... + \frac{c_n}{c_i + c_n}$.
It's obvious that ℓ_i is inversely proportional to c_i.
$E(\ell_i) = \sum_{i=1}^{n} (c_i/c) \cdot \ell_i$
When (c_i/c) gets larger, ℓ_i gets smaller. Actually, it's similar with (b) part. That means the expected search cost of this algorithm is minimal.