Chapter 5

By Y. MENG   gmail : y(dot)meng201011

Section 1

exe. 5.1.1 Show that the assumption that we are always able to determine which candidate is best, in line 4 of precedure HIRE-ASSISTANT, implies that we know a total order of the ranks of the candidates.

A total order is a partial order that is a total relation of ($\forall a, b \in A : aRb\, or\, bRa$). And a relation is a partial order if it is reflexive, asymmetric, and transive.

In this case, assume that the relation is "**is as good as or better than**":

- Reflexive: Every candidate is as good as or better than himself/herself.
- Transive: If A is better than B and B is better than C, then A is obviously better than C.
- Asymmetric: If A is better than B, there is no way that B is better than A.

So far we have a partial order. And since we assume that we can compare any two candidates, and comparison must be a total relation and thus we have a total order.

---

exe. 5.1.2 Describe an implementation of the procedure RANDOM(a, b) that only makes calls to RANDOM(0, 1). What is the expected running time that your procedure of a generator of distinct numbers in range[a, b].

First thought that hit me.
RANDOM(a, b)

```
  $r = a$
  $for\, i \leftarrow 0\, to\, (b - a)$
      $r = r + RANDOM(0, 1)$
  $return\, r$
```

It doesn't return every number equally likely.

Procedure RANDOM(0, 1) is similar to coin flipping. It returns 0 and 1 at equally probability of $\frac{1}{2}$. And for procedure RANDOM(a, b), the problem will be checking the result of flipping a coin for $(b - a)$ times. The result space will be all the permutations composed of n 0s and 1s, where $2^n = (b - a)$. Thus, we can represent this result with a sequence of 0s and 1s, which is a BINARY number of $(1 + \lg(b - a))$ digits.

So, if we run RANDOM(0, 1) for $(1 + \lg(b - a))$ times and convert the final permutation to a decimal value plus a, that's the random value we want. And we need to be careful since (b - a) might not be exact power of 2, so we need to run RANDOM(0, 1) for ceiling of $(1 + \lg(b - a))$ times. But a serial of 0s and 1s of this length might represent a number extends (b - a), we can abandon the value and find the next permutation till it is no greater than (b - a).

RANDOM(a, b)

```
  $decimal\_rand = b$
  $times = CEIL(1 + lg(b - a))$
  $do$
      $for\, i \leftarrow 0\, to\, times$
          $binary\_perm = binary\_perm + RANDOM(0, 1)$
      $decimal\_rand = TO\_DECIMAL(binary\_perm)$
      $binary\_perm = EMPTY\_STRING$
      $while\, (decimal\_rand \geq (b - a))$
  $return\, (decimal\_rand + a)$
```

In English:
Let n = b - a.
1. We find smallest integer c such that $2^c \geq n\, (i.e., c = CEIL(lgn))$
2. We call RANDOM(0, 1) c times to get a c-digit binary number r.
3. If r $>$ n, we go back to step 2.
4. Otherwise, we return (a + r).

This procedure can generate a uniformly random number in range [a, b]. However, there is a possibility of repeating step 2. And the chance of not having to repeat step 2 is $p = \frac{n}{2^c}$. The geometric distribution suggests that on average it takes $\frac{1}{p}$ trials before we get such a number, that is $\frac{2^c}{n}$ trials. Since we call RANDOM(0, 1) c times on each trial, so the expected running time is

$$O(\frac{2^c}{n}c) = O(\frac{\lg(b-a)2^{\lg(b-a)}}{(b-a)}) = O(\lg(b-a))$$

5.1.3 Suppose that you want to output 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. At your disposal is a procedure BIASED_RANDOM that outputs either 0 or 1. It outputs 1 with some probability p and 0 with probability $(1-p)$, where $0 < p < 1$, but you don't know what p is. Give an alg that uses BIASED_RANDOM as a subroutine, and returns an unbiased answer, returning 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. What is the expected running time of your alg?

UNBIASED_RANDOM(0, 1)

```
do
    x = BIASED_RANDOM(0, 1)
    y = BIASED_RANDOM(0, 1)
while(x == y)
return x
```

- The chance of getting [0, 1] is the same as that of getting [1, 0].
- The likelihood of that happening is 2pq. Thus, the expected number of trials is $\frac{1}{2pq}$, making the running time $O(\frac{1}{p(1-p)})$

Section 2

exe. 5.2.1 In *HIRE-ASSISTANT*, assuming that the candidates are presented in a random order, what is the probability that you hire exactly one time? What is the probability that you hire exactly n times?

---

exe. 5.2.2 In *HIRE-ASSISTANT*, assuming that the candidatas are presented in a random order, what is the probability that you hire exactly twice?

---

exe. 5.2.3 Use indicator random variables to compute the expected value of the sum of n dice.

---

exe. 5.2.4 Use indicator random variables to solve the following problem, which is known as the hat-check problem. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

---

exe. 5.2.5 Let A[1..n] be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A. (See Problem 2-4 for more on inversions.) Suppose that the elements of A form a uniform random permutation of $< 1, 2, \cdots, n >$. Use indicator random variables to compute the expected number of inversions.

Section 3

5.3.1 Professor Marceau objects to the loop invariant used in the proof of Lemma 5.5. He questions whether it is true prior ot the first iteration. He reasons that we could just as easily declare that an empty subarray contains no 0-permutation should be 0, thus invalidating the loop invariat prior to the first iteration. Rewrite the procedure *RANDOMIZE_IN_PLACE* so that its associtated loop

invariant applies to a nonempty subarray prior to the first iteration, and modify the proof of Lemma 5.5 for your procedure.

5.3.2 Professor Kelp decides to write a procedure that produces at random at permutation besides the identity permutation. he proposes the following procedure. Does this code do what Prof. Kelp intends?

PERMUTE_WITHOUT_INDENTITY(A)

```
n = A.length
for i = 1 to (n - 1)
    swap A[i] with A[RANDOM(i + 1, n)]
```

5.3.3 Suppose that instead of swapping element A[i] with a random element from the subarray A[i..n], we swapped it with a random element from anywhere in the array. Does the code below produce a uniform random permutation? Why or why not?

PERMUTE_WITH_ALL(A)

```
n = A.length
for i = 1 to (n - 1)
    swap A[i] with A[RANDOM(1, n)]
```

5.3.4 Professor Armstrong suggests the following procedure for generating a uniform random permutation. Show that each element A[i] has a $\frac{1}{n}$ probability of winding up in any particular position in B. Then show that Prof. Armstrong is mistaken by showing that the resulting permutation is not uniformly random.

5.3.5* Prove that in the array P in procedure *PERMUTE_BY_SORTING*, the probability that all elements are unique is at least $(1 - \frac{1}{n})$.

5.3.6 Explain how to implement the algorithm *PERMUTE_BY_SORTING* to handle the case in which two or more pririties are identical. That is, your algorithm should produce a uniform random permutation, even if two or more priorities are identical.

5.3.7 Suppose we want to create a <u>random sample</u> of the set {1, 2, 3, ..., n}, that is, an m-element subset S, where $0 \le m \le n$, such that each m-subset is equally likely to be created. One way would be to set A[i] = i for i = 1, 2, 3, ..., n, call *RANDOMIZE_IN_PLACE(A)*, and then take just the first m array elements. This method would make n calls to the *RANDOM* procedure. If n is much larger than m, we can create a random sample with fewer calls to *RANDOM*. Show that the follwing recursive procedure returns a random m-subset S of {1, 2, 3, ..., n}, in which each m-subset is equally likely, while making only m calls to *RANDOM*:

RANDOM_SAMPLE(m, n)

```
if m == 0
    return $\emptyset$
S = RANDOM_SAMPLE(m - 1, n - 1)
    i = RANDOM(1, n)
    if i $\in$ S
        S = S $\cup$ {n}
    else S = S $\cup$ {i}
    return S
```

Section 4

5.4.1 How many people must there be in a room before the probability that someone has the same birthday as you do is at least $\frac{1}{2}$? How many people must there be before the probability that at least two people have a birthday on July 4 is greater than $\frac{1}{2}$?

5.4.2 Suppose that we toss balls in to b bins until some bin contains two balls. Each toss is indepent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?

5.4.3* For the analysis of the birthday paradox, is it important that the birthdays be mutually independent, or is pairwise independence sufficient? Justify your answer.

5.4.4* How many people should be invited to a party in order to make it likely that there are three people with the same birthday?

5.4.5* What is the probability that a k-string over a set of size n forms a k-permutation? How does this question relate to the birthday paradox?

5.4.6* Suppose that n balls are tossed into n bins, where each toss is independent and the ball is equally likely to end up in any bin. What is the expected number of empty bins? What is the expected number of bins with exactly one ball?

5.4.7* Sharpen the lower bound on streak length by showing that in n flips of a fair coin, the probability is less than $\frac{1}{n}$ that no streak long than $(\lg n - \lg\lg n)$ consecutive heads o

---

```
In [ ]:
```