

CS 255 Homework 1

1. Suppose we have access to a C function with prototype:

```
bool p_random ();
```

which returns 1 with probability p and 0 with probability $1-p$. Write a C function

```
int dice_random ();
```

which is allowed to call the function *p_random* that returns a number between 1 to 6 at random each with equal likelihood. *dice_random* should be your only source of randomness in your code.

Solution:

```
int dice_random ()
{
    bool a, b;
    int bits [3], diceValue;

    for (int i = 0; i < 3; i++) {
        a = p_random ();
        b = p_random ();
        if (a + b == 1) {
            bits[i] = a;
        } else {
            return dice_random ();
        }
    }

    diceValue = 4*bits [0] + 2*bits [1] + bits [2] + 1;
    if (diceValue > 6) {
        return dice_random ();
    }
    return diceValue;
}
```

- Consider the variation of the Hiring problem where we have two employees rather than one. As we interview candidates, if we find a candidate that is better than either of our two current employees, we fire the weaker of the current employees and hire the candidate. Determine how many candidates we would hire on average in this situation.

Solution:

Let X_i be an indicator random variable which represents if i^{th} candidate is hired.

$E[X_i] = \Pr(X_i = 1)$ as X_i is indicator random variable expectation is same as probability.

Probability of i^{th} candidate getting hired

So, we can think of this probability in the following method.

If we sort the first i candidates based on their score of best (say descending order) then i^{th} candidate gets hired if he is in either 1st or 2nd position.

$$\text{So } \Pr(X_i = 1) = \frac{2 \cdot (i-1)!}{i!} \\ = \frac{2}{i}$$

Using Linearity of expectation

$$\begin{aligned} E\left[\sum_{i=1}^n X_i\right] &= \sum_{i=1}^n E[X_i] \\ &= \sum_{i=1}^n \Pr(X_i = 1) \\ &= \sum_{i=1}^n \frac{2}{i} \\ &= 2\log(n) + \text{constant (using integral bound)} \end{aligned}$$

In average we need to hire **$O(\log(n))$** candidates

- Suppose we toss balls into one of n bins. Assume each bin is equally likely. Calculate with work the expected number of balls you would need to toss until there are two bins with at least two balls.

Solution:

Let X represent a random variable for number of ball tosses for at least two balls in two bins.

X_{ij} represent an indicator random variable for at least two balls in bins (i, j) .

$$X = \sum_{i=1}^n X_{ij}$$

$$E[X_{ij}] = \Pr(X_{ij} = 1)$$

Probability that we toss a ball in a bin (say bin i) is $\frac{1}{n}$

The probability that a bin has at least two balls = $1 - \Pr(\text{bin has either 1 or 0 balls})$

Probability that a bin has exactly k balls $= \binom{m}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{m-k}$; with m being the number of balls tossed.

$$\begin{aligned}
 &= 1 - \binom{m}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{m-1} - \binom{m}{0} \left(1 - \frac{1}{n}\right)^m \\
 &= 1 - \frac{m}{n} \left(1 - \frac{1}{n}\right)^{m-1} - \left(1 - \frac{1}{n}\right)^m \\
 &\geq 1 - \frac{m}{n} \left(1 - \frac{1}{n}\right)^m - \left(1 - \frac{1}{n}\right)^m \\
 &\geq 1 - \left(1 + \frac{m}{n}\right) \left(1 - \frac{1}{n}\right)^m \\
 &\geq 1 - \left(1 + \frac{m}{n}\right) \left(1 - \frac{m}{n}\right) \text{ (assumption } n \gg m) \\
 &\geq 1 - \left(1 - \frac{m^2}{n^2}\right) = \frac{m^2}{n^2}
 \end{aligned}$$

A_i = Event that bin i has at least 2 balls

$\Pr(A_i \cap A_j) = \Pr(A_i) \cap \Pr(A_j)$ (since a ball toss to bin i is independent to a ball tossed to bin j)

So $\Pr(X_{ij} = 1) = \Pr(A_i \cap A_j)$

$$\begin{aligned}
 &= \frac{m^2}{n^2} * \frac{m^2}{n^2} \\
 &= \frac{m^4}{n^4}
 \end{aligned}$$

So, we have $\binom{n}{2}$ ways to choose two bins from n bins and using linearity of expectation

$$\begin{aligned}
 \text{The expected number of tosses } E[X] &= \binom{n}{2} E[X_{ij}] \\
 &= \binom{n}{2} \left(\frac{m}{n}\right)^4 \\
 &= \frac{m^4}{2n^2}
 \end{aligned}$$

The expected number of tosses $m = (2n^2)^{\frac{1}{3}}$

Programming Assignment – Plot

Explanation for $\log(N)$ bits:

We are generating random numbers for assigning IDs to computers. For each round the random numbers are generated in the range 1 to round^3 . Using the proof of unique numbers generation for Permute_By_Sorting, we can say that we have more probability of getting unique numbers when the range is 1 to n^3 .

So, we can assure that we generate unique IDs with high probability at most at round n (n – no of computers).

The max ID can be n^3 and requires $\log(n^3) = 3 * \log(n)$ bits.

| Number of Computers ▼ | Number of Bits ▼ |
|-----------------------|------------------|
| 0 | 1 |
| 5 | 5 |
| 10 | 7 |
| 20 | 9 |
| 50 | 10 |
| 100 | 12 |
| 500 | 18 |
| 1000 | 18 |
| 2000 | 21 |
| 5000 | 23 |
| 20000 | 26 |
| 50000 | 29 |
| 100000 | 31 |

