Asmt 1: Hash Functions and PAC Algorithms

Yulong Liang (u1143816)

January 24, 2018

1 Birthday Paradox (30 points)

A In the first simulation, it took 68 trials to reach a collision, i.e., $k = 68$.

B The cumulative density plot is shown below:

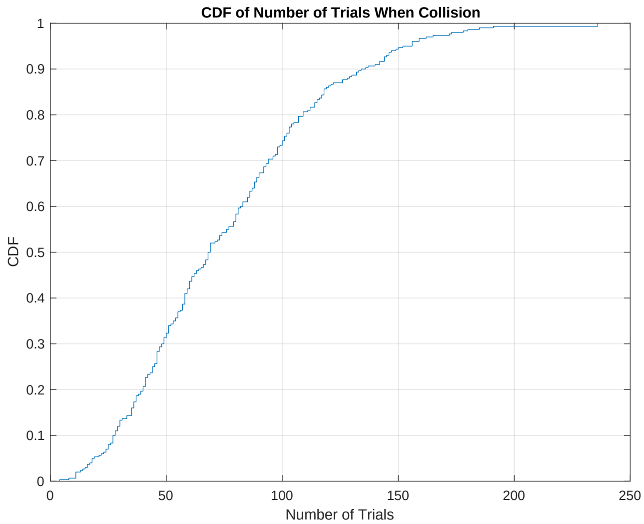

Figure 1: Cumulative Density Plot of Number of Trials When Collision

C The empirical expected number is $\mathbb{E} = 75.9600$.

D First I implemented the experiment with **Matlab**, using `containers.Map` and a `vector` of size n, i.e., a normal array respectively. Then I switched to **Python** using a `python list`, a `numpy array` and a `python set` respectively to compare the speed between the two languages. For the experiment of $n = 4000$ and $m = 300$, the runtime is as follows,

Language	Data Type	Run Time
Matlab	Map	0.4528
Matlab	Vector	0.0430
Python	Numpy Array	0.0591
Python	List	0.0491
Python	Set	0.0432

Below is a plot of the run time of the implementation with `Matlab Vector` for $m = 300$, $m = 2000$, $m = 10000$

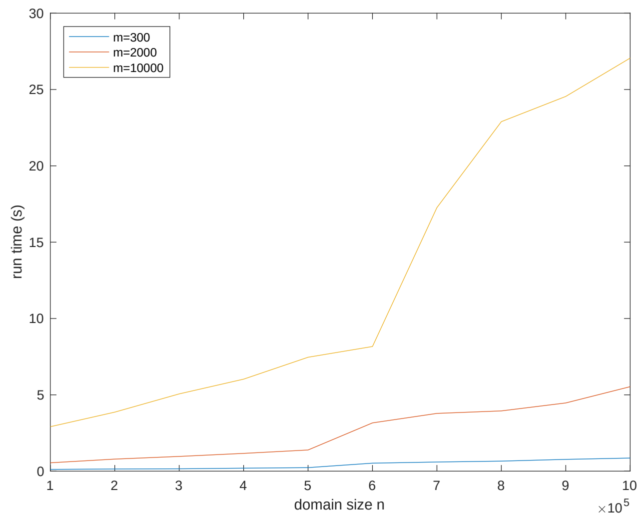

Figure 2: Run Time of the Implementation with `Matlab Vector`

The codes are as follows,

```
function k = birthdaySimulatora2(n)
    map = zeros(1,n);
    k = 0;
    while 1
        x = randi(n);
        if map(x) == 1
            return;
        else
            map(x) = 1;
            k = k + 1;
        end
    end
end
```

2 Coupon Collectors (30 points)

A In the first simulation, it took 1367 trials to cover all the numbers, i.e., $k = 1367$.

B The cumulative density plot is shown below:

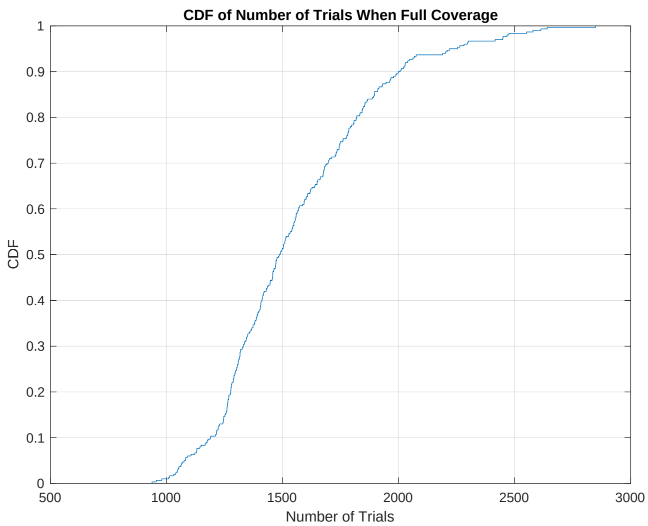

Figure 3: Cumulative Density Plot of Number of Trials When Full Coverage

C The empirical expected number is $\mathbb{E} = 1555.7233$.

D First I implemented the experiment with **Matlab**, using `containers.Map` and a `vector` of size n respectively. Then I switched to **Python** using a `python list`, a `numpy array` and a `python set` respectively to compare the speed between the two languages. For the experiment of $n = 250$ and $m = 300$, the run time is as follows,

Language	Data Type	Run Time
Matlab	Map	3.3359
Matlab	Vector	0.9666
Python	Numpy Array	0.6793
Python	List	0.5535
Python	Set	0.5349

Below is a plot of the run time of the implementation with `Matlab Vector` for $m = 300$, $m = 1000$, $m = 5000$,

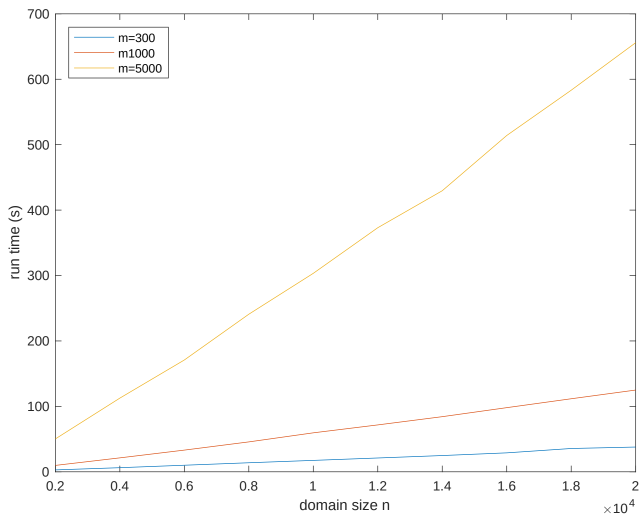

Figure 4: Run Time of the Implementation with `Matlab Vector`

The codes are as follows,

```
function k = couponSimulatora2(n)
    map = ones(1,n);
    k = 0;
    length = n;
    while length > 0
        k = k + 1;
        x = randi(n);
        if map(x) == 0
            continue;
        else
            map(x) = 0;
            length = length - 1;
        end
    end
end
```

3 Comparing Experiments to Analysis (24 points)

A The analytical expected number is $\mathbb{E} = 75$, which is very near to the empirical expected number $\mathbb{E} = 75.9600$.

The formula I used is:

$$\frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \dots \cdot \frac{n-(k-1)}{n} = \prod_{i=1}^{k-1} \frac{n-i}{n} \leq 0.5$$

The solution to the formula is:

$$k \geq 75$$

The strategy of coding is using a for loop to calculate the product from k = 1 when the product is 1 until the product is less than or equal to 0.5. Then return the value of k.

B The analytical expected number is $\mathbb{E} = 1525.1688$, which is very near to the empirical expected number $\mathbb{E} = 1555.7233$.
The formula I used is:

$$T = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} = 250 \cdot \sum_{i=1}^{250} \frac{1}{i} \approx 1525.1688$$

The strategy of coding is using a for loop to calculate the summation.

4 Random Numbers (16 points)

A Call `rand-bit()` function for 10 times, which will provide 10 random binary numbers. The conbination of the 10 binary numbers will form a random integer a between 0 and 1023. Let $a+1$ be the final result.

B Using the method of 4A, the algorithm will generate random numbers in [1024]. If the number is in [1000], the algorithm will finish and return the number it generated. If the number is in [1001, 1024], the algorithm will generate another random number in [1024]. The algorithm will run iteratively until it generates a number in [1000].

For each generation, the probability of success is $\frac{1000}{1024} = 0.9765625$, the probability of failure is $\frac{24}{1024} = 0.0234375$. The expected number of generations is,

$$\mathbb{E} = 1 \cdot p + 2 \cdot pq + 3 \cdot pq^2 \cdot i \cdot qp^{i-1} \cdots = \frac{1}{p} = 1.024$$

C The expected number of calls of `rand-bit()` made for a random number in domain $[n]$ is,

$$\frac{2^{\lceil \log_2 n \rceil}}{n} \cdot \lceil \log_2 n \rceil$$

5 BONUS : PAC Bounds (2 points)

From the context, μ and $\frac{1}{n}$ are the maximum and expectation value of the frequency of value i respectively.

$$\mu = max(\frac{f_i}{k}) \tag{1}$$

$$\frac{1}{n} = \mathbb{E}(\frac{f_i}{k}) \tag{2}$$

Assume we have a set of k iid random variables $\{X_1, X_2, \cdots, X_k\}$, $X_i \in \{0, 1\}$ for each $i \in [k]$. Then,

$$A = \frac{1}{k} \sum_{i=1}^{k} X_i = \frac{f_i}{k} \tag{3}$$

$$-1 \leq X_i \leq 1 \quad \Rightarrow \quad \Delta = 1 \tag{4}$$

Substituting (1)(2)(3)(4) into the Chernoff-Hoeffding inequality,

$$\textsf{\textbf{Pr}}[|A - E[A]| > \epsilon] \leq 2 \cdot \exp(\frac{-r\epsilon^2}{2\Delta^2}) \tag{5}$$

$$\textsf{\textbf{Pr}}[|\mu - \frac{1}{n}| > \epsilon] \leq 2 \cdot \exp(\frac{-k\epsilon^2}{2}) \tag{6}$$

In order that the right part of the inequality equals 0.02,

$$2 \cdot \exp(\frac{-k\epsilon^2}{2}) = 0.02 \tag{7}$$

$$k = \frac{2 \ln 100}{\epsilon^2} \tag{8}$$

In order that the right part of the inequality equals 0.002,

$$2 \cdot \exp(\frac{-k\epsilon^2}{2}) = 0.002 \tag{9}$$

$$k = \frac{2 \ln 1000}{\epsilon^2} \tag{10}$$

Thus, k need to be great than or equal to $\frac{2 \ln 100}{\epsilon^2}$ for the probability of failure to be less than or equal to 0.02 and $\frac{2 \ln 1000}{\epsilon^2}$ for the probability of failure to be less than or equal to 0.002.