Homework 7
Colorado CSCI 5454

Sai Siddhi Akhilesh Appala

November 30, 2022

People I studied with for this homework: None

Problem 1

Solution: From the class lectures, We all know that even we have l number of bins, i.e we will be having 37% which is approximately $\frac{1}{3}$ will remain empty in the complete bins.

- As given we have l number of bins, we need to find the expected number of bins, so we can go in reverse order by saying whats the probability of keeping a ball into a bin
- As we have l number of bins, to place a ball into one of the bin, the probability is $\frac{1}{l}$.

$$Prob_{success} + prob_{failure} = 1$$

$$\frac{1}{l} + prob_{failure} = 1$$

$$prob_{failure} = 1 - \frac{1}{l}$$

The above is a case for one ball not to be pushed into any of the bin. If we repeat and require the same result for l balls. then the probability of failure for l balls not to push into a bin is

$$prob_{failure-for-n} = (1 - \frac{1}{l})^n$$

Now we are considering the expected value of the above scenario

$$E[X_i] = \sum_{n=1}^{l} Prob_{failure}$$

$$E[X_i] = (1 - \frac{1}{l})^n$$

$$E[X_i] = (1 - \frac{1}{l})^n$$

There is an approximation that is proved when we have an equation with $\forall x \in \mathbb{R}$ (1+x) can be written as e^x

$$(1 + x) \leq e^x$$

When we substitute $-x$ in place of x, even then the below should be accurate, so the equation will look like

$$1 - x \leq e^{-x}, \forall x \in R$$

From the above we can conclude that

$$E[X_i] \leq \sum_{n=1}^{l} (e^{\frac{-1}{l}})^n$$

$$E[X_i] \leq \sum_{n=1}^{l} (e^{\frac{-n}{l}})$$

When n = l , this gives a 1, $\frac{1}{e}$ approximately 0.367 probability of being empty. By linearity of expectation, we expect below $\frac{l}{e}$ empty bins

$$E[X_i] \leq l * (e^{\frac{-n}{l}})$$

Substitute n =1 in the above equation

$$= l * (e^{-}1)$$

$$= \frac{l}{e}$$

Problem 2

Solution:

- As given we need to l bins, n is the number of items being hashed, and the universe size of n*l items, some bin has at least n items.
- For proving the above, we will use a pigeon hole which says that or l+1 or more pigeons are distributed among l pigeonholes, then at least one pigeonhole contains two or more pigeons. So the average number of pigeons in each pigeonhole can be said to be $\frac{l+1}{l} > 1$.
- Using the above here, we can say that for l bins such that X1...Xl, and n items being hashed, we have the number of items as $\sum_{n=1}^{l} X_i = n$
- Assume that the total number of items n is to be placed in l bins, and $n > l$. Let there be no bins with at least n items, i.e., each bin has fewer than n items

Number of items in each bin $< n/l$

$$\sum_{n=1}^{l} X_i < l * \frac{n}{l}$$

$$\sum_{n=1}^{l} X_i < n$$

From above we can conclude that number of items is strictly equal to n, our assumption that each bin holds less than n items is violated. As a result, there we can say that there exists a choice of n items that all hash to the same bin.

Problem 3

part a

Below are the data structures that will be the best suited for the given scenarios. please find them below respectively.

1. For the 1st task, we will be using **bloom filters**.
    - As it is quick to check whether a user is a VIP or not. and As mentioned it's okay to get a few false positives.
    - So bloom filters is the apt data structure for this scenario and for the 1st step. which is not a deep check.
    - Algo for Bloom filter Suppose we try this: using our array A of m bits, we implement Add(x) by setting A[h(x)] = 1, and we implement CheckIfAdded(x) by returning true if A[h(x] = 1, false otherwise.

2. For the 2nd task, **hashtable** are the best because

   - Once the above check returns true, As it is clearly mentioned there is no chance of getting a wrong answer.
   - So the hash table is the apt data structure for this scenario and for the 1st step. which is a deep check and there is no way to get a wrong output as we directly store the details of the user.

3. For the 3nd task, **Count-min** sketch is the best because

   - Once the above checks are done, we need to keep the track of top VIP visitors, and any VIP who makes more than 1% of the total visits should be shown a special thanks page.
   - So count-min is the apt data structure for this scenario for the count of the VIP user visits. Once after applying the hash function to each and everyone, we will be having the count of each user in different hash tables. Now we need to pick the min count mentioned in the table. which is the total number of visits for a VIP user.
   - As clearly mentioned your data structure can be approximate and it may over-count user's visits a little (but must not under-count them), there is no chance for the count-min data structure to get the under-count if we take the min count of all the hash functions. so it's the best suit.
   - once we check whether it's 1% of the total visits, we can return the thanks page for them.

part B

For task 1 - Bloom Filters

As given we have seven million users per day, 2000 of whom are VIPs, let's compute the required parameters for each of the proposed techniques for the above optimal solutions.

$n = 7 * 10^6$ and M is at least 2000

From lecture notes, we know that, for a false positive rate δ

$$K = \log_2 \frac{1}{\delta} \tag{1}$$

$$m = \frac{nk}{\ln 2}$$

By substituting the m and n values, we get the below solution

$$2000 = \frac{7 * 10^6 * k}{\ln 2}$$

From above we can get K as

$$k = 0.000198$$

From the above equation (1) of K, we can get δ value as

$$\delta = 2^{-k}$$

By substituting the values

$$\delta = 2^{-0.000198}$$

$$\delta = 0.9993$$

From above by calculating we can say that false positive rate is 0.9993

For task 2 - Hash Table with chaining

From the class we know that the

$$\alpha = \frac{n}{l}$$

we have $n = 7 * 10^6$ and as we require minimum as 2000 bins, we can take length as 2000

$$\alpha = \frac{7 * 10^6}{2000} = 3500$$

if we say that the elements are evenly distributed, then maximum load will become $d = \alpha$. From above we can say that length of list d = 3500.

Total size = 4 * 3500 = 14000Bytes = 14KB

For task 3 - Count min

Since the array is 2 dimentional of w × d, d = number of hash functions ,height = w
From the lecture notes, Number of rows in count-min array (w) = $\lceil \frac{e}{\epsilon} \rceil$

Assume that $\epsilon = 1\% = 0.01$ and $\delta = e^{-10} \approx 0.00005$

$\implies$ w = $\lceil 100e \rceil$ = 272

$\therefore$ number of rows = 272

Number of columns in count-min array d = $\lceil \ln \frac{1}{\delta} \rceil$

$\implies$ d = $\lceil \ln \frac{1}{e^{-10}} \rceil$ =$\lceil \ln e^{10} \rceil$ = 10

Total elements = w * d = 272 * 10 = 2720

If we consider each element in the array as integer, total size = 2720 x sizeof(int) = 2720*4 = 10880 Bytes $\approx$ 11KB.

Problem 4

Part a

Given information
Pairwise distance between all keys and the maximum distance between consecutive keypresses.

Assumptions

So, let's consider the edge between keys whose distance is less than or equal to the maximum possible distance between consecutive keypresses.
Lets consider adjacency matrix A_G where:

$$A_G(i,j) = \begin{cases} 1 & if\ \ there;\ exists\ \ an\ \ edge \\ 0 & \text{otherwise} \end{cases}$$

Algorithm

Output : Returns count of passwords of length **t**

1. Initialize A_G
2. Find A_G^{t-1}

3. Return $Number\ of\ passwords = \sum_{i=1}^{k} \sum_{j=1}^{k} A_G^{t-1}(i,j)$, where k represents the number of keys.

Proof of Correctness

- As there is no restriction on passwords we can repeat each pair of keys any number of times, so according to the theorem 1 from notes A_G^{t-1} gives the number of paths from I to j in the graph G, with repeated vertices allowed, of length exactly t.

- From above, the number of paths from each pair of vertices, therefore, the total number of paths that can be formed is the sum of the results of the number of paths from each pair of vertices.

Part b

Algorithm

Algorithm 1 Algorithm for finding A_G^{t-1}

```
Input: Given A_G, t
Set output = I – Identity matrix
P = t − 1, Base = A_G
while P > 0 do
    if P%2 == 1 then
        output = output * Base
    end if
    Power = P/2
    Base = Base * Base
end while
Return output
```

Proof of Correctness:

The task is to find A_G^{t-1}.

1st iteration

- Dividing (t-1) as the sum of powers of 2 We can find A_G^{t-1} as powers of 2. Like consider t-1 = 5 can be written as 1 + 4 as the binary is 101

2nd iteration:

- Finding the value of 2 powers Lets consider A_G^k
- if k is even we can recursively solve it as $A_G^k = A_G^{\frac{k}{2}} * A_G^{\frac{k}{2}}$
- if k is odd $A_G^k = A_G * A_G^{\frac{k-1}{2}} * A_G^{\frac{k-1}{2}}$

$\therefore$ Time taken to find value of both the iterations are $A_G^t = \mathrm{O}(\log t)$

total time = $2(\log t) = \mathrm{O}(\log t)$