Homework-7

Sources: Class notes and wikipedia
Collaboration:Nischal and Abhilash.

Problem 1:

Part a:

- $\binom{n}{k} \left(\frac{1}{n}\right)^k \left(1-\frac{1}{n}\right)^{n-k}$

Part b:

- $Y_i = X_{i1} + X_{i2} + \ldots + X_{in}$
- $E[Y_i] = E[X_{i1}] + E[X_{i2}] + \ldots + E[X_{in}]$
- $E[X_{ij}] = 1.P(X_{ij}=1) + 0.P(X_{ij} = 0) = 1/n$
- $E[Y_i] = 1/n + 1/n + \ldots + 1/n$(repeated n times)
- $E[Y_i] = n/n = 1$

Part c:

$n/\delta \geq 2^6$,can also be written as $\delta/n \geq 2^{-6}$

$\log_2(n/\delta) \geq 6$,we know $t \geq 6$ E[Y]. From part b $E[Y_i] = 1$

$t = \log_2(n/\delta)$

Y_i is a binomial distribution and $t = t = \log_2(n/\delta)$

From Chernoff bound we know $Pr[Y \geq t] \leq 2^{-t}$

Hence, $Pr[Y_i \geq \log_2(n/\delta)] \leq \delta/n$

Part d:

- Let E_1 be the even that bin i has at least $\log_2(n/\delta)$, $E_2 = Y_2 \geq \log_2(n/\delta)$ and $E_n = Y_n \geq \log_2(n/\delta)$
- According to union bound $Pr[E_1 \text{ or } E_2 \text{ or..or } E_n] \leq \Sigma_{j=1}^{n} Pr[E_j]$ (slightly modified for convenience, don't cut marks)
- In our case these events are nothing but the probability that j'th bin has at least $\log_2(n/\delta)$ balls.
- $Pr[E_1 \text{ or } E_2 \text{ or..or } E_n] \leq Pr[E_1] + Pr[E_2] + .. + Pr[E_n]$
- From Chernoff bound $Pr[Y_i \geq \log_2(n/\delta)] \leq \delta/n$
- $Pr[E_1 \text{ or } E_2 \text{ or..or } E_n] \leq \delta/n + \delta/n + \delta/n + ... + \delta/n$ (repeats n times)
- $Pr[E_1 \text{ or } E_2 \text{ or..or } E_n] \leq (\delta/n)*n \leq \delta$

Problem 2:

Part a:

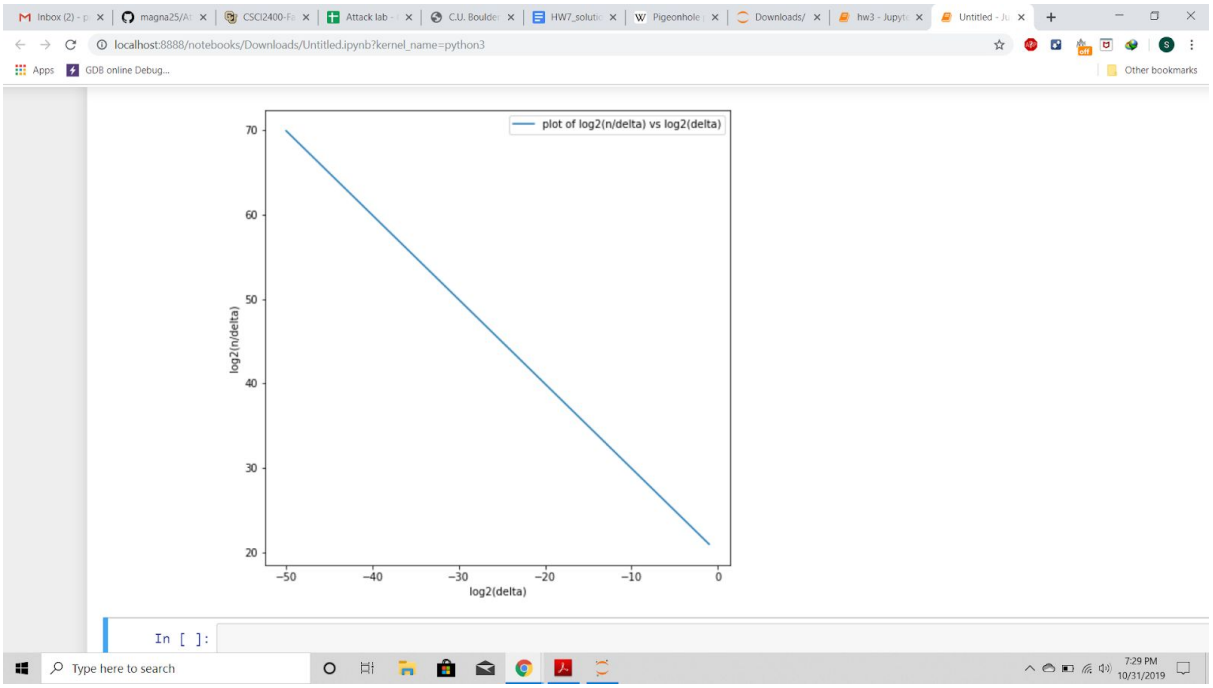
- Pigeon hole principle states that ,"if m items are put into n containers,such that $m \geq n$. Then at least one container must contain more than 1 items"
- In our case hash has n slots and universe of elements being hashed is
  $m \geq n^2$.
- Then by the pigeonhole principle at least one slot must contain more than n elements.
- Initially after running for n*n elements each slot will have at least n elements. So adversary knows these inputs filling up all the slots in just n inputs.
- Next time she inputs n elements such that they map to the same slot and that's how she will hurt our performance.

Part b:

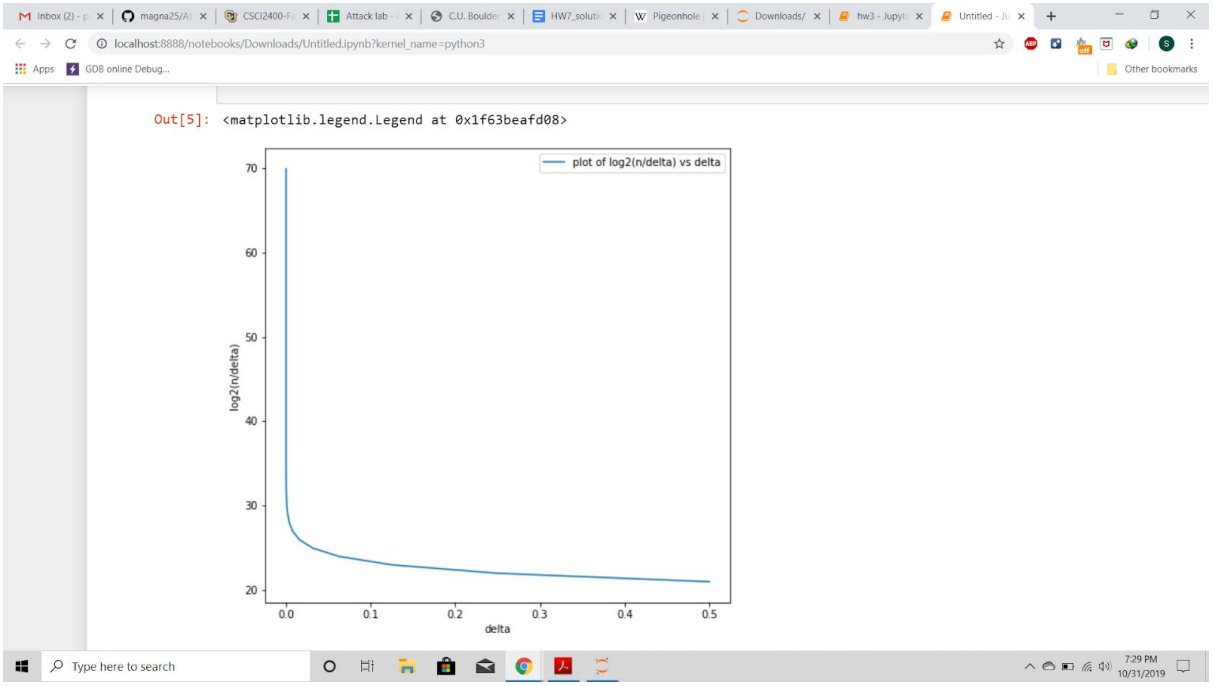
- E_1 = OR of all Events e where (Event e is such that atleast one bin has $\log_2(n/\delta)$ balls). E_2 = OR of all Events e where (Event e is such that atmost one bin has $\log_2(n/\delta)$ balls). $E_1 \cap E_2$ = OR of all Events e where (Event e is that bins have exactly $\log_2(n/\delta)$ balls).
- $Pr[E_1] + Pr[E_2] - P[E_1 \cap E_2] = 1$
- From 1.d we have proved that $pr[E_1] \leq \delta$
- $Pr[E_2] - P[E_1 \cap E_2] \geq 1-\delta$
- We know that $P[E_1 \cap E_2] \geq 0$, Hence, $Pr[E_2] \geq 1-\delta$

Part c:

```
In [5]: import math
        import matplotlib.pylab as plt
        %matplotlib inline

        numbers = []
        i=-1
        while i!=-51:
            numbers.append(i)
            i-=1
        deltas = []
        for i in numbers:
            deltas.append(2**i)
        k=[]
        for i in deltas:
            k.append(math.log2(1000000/i))

        logdeltas = []
        for i in deltas:
            logdeltas.append(math.log2(i))
        plt.figure(figsize=(8,8))
        l1, = plt.plot(deltas,k)
        l1.set_label("plot of log2(n/delta) vs delta")
        plt.ylabel("log2(n/delta)")
        plt.xlabel("delta")
        plt.legend()

        plt.figure(figsize=(8,8))
        l1, = plt.plot(logdeltas,k)
        l1.set_label("plot of log2(n/delta) vs log2(delta)")
        plt.ylabel("log2(n/delta)")
        plt.xlabel("log2(delta)")
        plt.legend()
```




```
In [ ]:
```


Inbox (2) - p
magna25/At
CSCI2400-Fa
Attack lab -
C.U. Boulder
HW7_solutio
Pigeonhole
Downloads/
hw3 - Jupyte
Untitled - Ju
localhost:8888/notebooks/Downloads/Untitled.ipynb?kernel_name=python3
Apps
GDB online Debug...
Other bookmarks
Out[5]: <matplotlib.legend.Legend at 0x1f63beafd08>
plot of log2(n/delta) vs delta
log2(n/delta)
delta
70
60
50
40
30
20
0.0
0.1
0.2
0.3
0.4
0.5
Type here to search
7:29 PM
10/31/2019

Problem 3:

Part a:

- The choice of data structure would be to use Bloom filters. As the website gets a lot of hits from one time users we if we use hash table we would require a lot of memory.Count-min sketch would not be a good choice because we don't have to return count of occurrence of IP address but just it's existence. Instead we could use Bloom Filters with k hash functions and when an IP address comes in we check if hash of IP address is 1 for all k functions . If any one of those is 0 we return false else true.

Part b:

- The choice of data structure would be to use count-min sketch. In count-min sketch we pass IP address through d hash functions and get the minimum from the list having counts returned from each hash function. We get an estimation of cost if not the exact count and we argue in notes that this estimate is never smaller than the actual number of occurrences of IP address neither it is an overestimate. Bloom filters just set the bits and don't store counts. So they're not a choice. Hash table could be used when we want list of elements and their count when elements are not available to be hashed again and checked for count. In this case elements are available to find count so we can use cost-min sketch.

Part c:

- The choice of data structure would be to use hash table. Bloom filters are not useful in this case because they're mainly used to check if a given IP address has visited the website before. We neither store IP address nor its count. Count-min Sketch may seem apt initially but it just gives us occurrence of IP address when supplied with one and does not store IP address itself. Hash table overcomes all of these shortcomings. When an IP address visits the website we check if has already visited and increment count or else set it to 1(visited for the first time).

Problem 4:

- Suppose m is between some 2^i and 2^{i+1} such that $2^i \leq m < 2^{i+1}$
- Total time is time taken to double the bins and rehash plus insertion.
- Insertion for element is O(1) and for the m elements is O(m).
- We double when n when $m = 2^2, 2^3, ..., 2^i$
- Total time is nothing but summation each such instance of doubling.
- $T(\text{doubling and rehashing}) = 2^2+2^3+...+2^i = 2^{i+1} - 4$
- i can be at max $\log_2(m)$ so $T(\text{doubling and rehashing}) \leq 2^{\log2(m)+1} - 4 \leq 2*m - 4$
- Ignoring lower order terms for O. So T(doubling and rehashing) = O(m)
- Total time = O(m) + O(m) = O(m)