CSEP 521, Winter 2021: Homework 4

Krishan Subudhi (ksubudhi@uw.edu) - 2040900

February 4, 2021

Problem 1

Q: Show that the number $1048579 = 2^{20} + 3$ *is not prime without factoring the number*

Fermat's theorm :

$$a^{p-1} \mod p = 1, \text{if } p \text{ is prime}, \quad \text{for all } a, 1 \le a \le p$$

We need to find an a where Fermat's theorm fails to prove that the number 1048579 is composite.

$number = 1048579 = 2^{20} + 3$

Let's take $a = 2$. Now if we can prove $2^{2^{20}+2} \mod 1048579$ is not 1, then p is not a prime.

$$\begin{aligned}
2^{2^1} \mod 1048579 &= (2^{2^0} \mod 1048579 * 2^{2^0} \mod 1048579) \mod 1048579 &= 4\\
2^{2^2} \mod 1048579 &= (2^{2^1} \mod 1048579 * 2^{2^1} \mod 1048579) \mod 1048579 &= 16\\
2^{2^3} \mod 1048579 &= (2^{2^2} \mod 1048579 * 2^{2^2} \mod 1048579) \mod 1048579 &= 256\\
2^{2^4} \mod 1048579 &= (2^{2^3} \mod 1048579 * 2^{2^3} \mod 1048579) \mod 1048579 &= 65536\\
2^{2^5} \mod 1048579 &= (2^{2^4} \mod 1048579 * 2^{2^4} \mod 1048579) \mod 1048579 &= 1036291\\
2^{2^6} \mod 1048579 &= (2^{2^5} \mod 1048579 * 2^{2^5} \mod 1048579) \mod 1048579 &= 1048147\\
2^{2^7} \mod 1048579 &= (2^{2^6} \mod 1048579 * 2^{2^6} \mod 1048579) \mod 1048579 &= 186624\\
&\cdots\\
&\cdots\\
2^{2^{19}} \mod 1048579 &= (2^{2^{18}} \mod 1048579 * 2^{2^{18}} \mod 1048579) \mod 1048579 &= 870510\\
2^{2^{20}} \mod 1048579 &= (2^{2^{19}} \mod 1048579 * 2^{2^{19}} \mod 1048579) \mod 1048579 &= 588380
\end{aligned}$$

Hence ,

$$2^{2^{20}+2} \mod 1048579 = (2^{2^{20}} \mod 1048579 * 2^2 \mod 1048579) \mod 1048579 = 256362$$

Since $256362 \neq 1$, the number 1048579 is not a prime.

Problem 2

A hash table of size m is used to store n items, with $n \leq m/2$, so the load factor is at most $\frac{1}{2}$

Open addressing is used for collision resolution.

a) Assuming uniform hashing, show that for $i = 1, 2, ..., n$*, the probability that the i-th insertion requires strictly more than k probes is at most* 2^{-k}

P (i-th insertion requires strictly more than k probes is at most 2^{-k})

= P (first k hashes in i-th iteration end up in collision)

$$\begin{aligned} P &= \left(\frac{i-1}{m}\right)^k \\ &\leq \left(\frac{n}{m}\right)^k \\ &\leq \left(\frac{m/2}{m}\right)^k \\ &\leq 2^{-k} \end{aligned}$$

b) Show that for $i = 1, 2, ..., n$*, the probability that the i-th insertion requires more than* $2 \log n$ *probes is at most* $1/n^2$. We previously showed in (a) that ,

$$P(i - \text{th insertion requires strictly more than k probes}) \leq 2^{-k}$$

Hence for $k = 2 \log n$,

$$\begin{aligned} P(i - \text{th insertion requires strictly more than } 2\log n \text{ probes}) &\leq 2^{-2\log n} \\ &\leq \frac{1}{n^2} \end{aligned}$$

Hence, the probability that the i-th insertion requires more than $2 \log n$ probes is at most $1/n^2$

c,d Definitions:

X_i = the number of probes required by the i-th insertion.

As per (b), $P(X_i > 2 \log n) \leq 1/n^2$

$X = \max_{1 \leq i \leq n} X_i$ = maximum number of probes required by any of the n insertions

c) Show that $Pr(X > 2 \log n) \leq 1/n$

$$\begin{aligned} P(X > 2\log n) &= P((X_1 > 2\log n) \cup (X_2 > 2\log n) \cup ... \cup (X_n > 2\log n)) \\ &\leq P((X_1 > 2\log n) + (X_2 > 2\log n) + ... + (X_n > 2\log n)) \\ &\leq n * \frac{1}{n^2} \\ &\leq \frac{1}{n} \end{aligned} \quad [2.1]$$

Proved

d) Show that the expected length of the longest probe sequence is $E[X] = O(\log n)$

$$
\begin{aligned}
E(X) &= \sum_{k=1}^{n} k\, P(X = k) \\
&= \sum_{k=1}^{2\log n} k\, P(X = k) + \sum_{n>2\log n}^{n} k\, P(X = k) \\
&\le 2\log n \sum_{k=1}^{2\log n} P(X = k) + n \sum_{k>2\log n}^{n} P(X = k) && [2.2]
\end{aligned}
$$

$$\sum_{k=1}^{2\log n} P(X = k) = Pr(X \le 2\log n) \le 1$$

As per 2.1, $Pr(X > 2\log n) \le 1/n$.

$$\implies \sum_{k>2\log n}^{n} P(X = k) = Pr(X > 2\log n) \le \frac{1}{n}$$

Hence equation 2.2 can be written as

$$
\begin{aligned}
E(X) &\le 2\log n * 1 + n * \frac{1}{n} \\
&\le 2\log n + 1 && [2.3]
\end{aligned}
$$

Hence as per 2.3 , $E(X) = O(\log n)$.

Problem 3

We have a bloom filter that uses a table of size n with k hash functions.

We are asked to adapt this Bloom filter to a setting where the table is of size $n/2$

Let's assume that n is a power of 2.

$$n = 2^x$$

To reduce the size of exiting boom filter but retain current information, we can halve the table and

- *OR* the two halves. The first half will be replaced with the bits from the *OR* operation. The 2nd half will be discarded.
- The hash functions needs to map bits in 2nd half to the corresponding bits in the first half.

  Since $n = 2^x$, the hash function must be using x bits for finding bit location in the bloom filter. Now if we mask the most significant bit, from the hash function output, the remaining $x - 1$ bits will map to the first half of the bloom filter. For example if n =8, and hash function output was 7(111) before, now it will map to 3 .
- The *OR* operation + masking will avoid losing existing data as the old data if mapped with the masked hash function would have set the bits now set by the OR operation.

3.1 Performance

Let's say the bloom filter had m entries before.

Previously before shrinking,

$$\begin{aligned} P(\text{a particular bit is 0}) &= \left(1 - \frac{1}{n}\right)^{km} = e^{-km/n} \\ P(\text{fasle positive}) &= (1 - P(\text{a particular bit is 0}))^k \\ &= (1 - e^{-km/n})^k = (1 - p)^k \end{aligned} \qquad [3.1]$$

Where $p = e^{-km/n}$.

After shrinking,

$$\begin{aligned} P(\text{a particular bit i is 0}) &= P(\text{bit i is 0}) * P(\text{bit n/2+i was 0}) \\ &= e^{-km/n} * e^{-km/n} = p^2 \\ P(\text{fasle positive}) &= (1 - P(\text{a particular bit i is 0}))^k \\ &= (1 - p^2)^k \end{aligned} \qquad [3.2]$$

Hence as per 3.1, and 3.2, False positive probability will increase by $(1 - p^2)^k/(1 - p)^k = (1 + p)^k$ times.

Problem 4

4.1 Description

Here I have implemented k choice hashing. n items are hashed using k hash functions (simulated). Item will be assigned to one of the buckets selected by the hash functions. The bucket with the lowest count will be chosen as the destination for each element.

4.2 Result

n	k	max_items	max_items/log logn	max_items/(log n/ log log n)	standard Deviation
1000	1	5.1	2.638871	1.426873	1.036340
	2	3.0	1.552277	0.839337	0.689928
	3	2.3	1.190079	0.643492	0.589915
10000	1	6.4	2.882459	1.542841	0.998899
	2	3.1	1.396191	0.747314	0.706682
	3	3.0	1.351152	0.723207	0.597327
100000	1	8.2	3.355883	1.740345	0.997477
	2	3.7	1.514240	0.785277	0.701313
	3	3.0	1.227762	0.636711	0.597930
1000000	1	8.7	3.313286	1.653532	0.999492
	2	4.0	1.523350	0.760245	0.703571
	3	3.0	1.142512	0.570183	0.596419

Table 1: k choice hashing for different values of n - average of 10 iterations

For k = 1, max_items grow at the rate $O(logn/loglogn)$. For k>1, max_items grows at the rate $O(loglogn)$ and constant factor decreases with increase in k . Standard deviation is also lower for higher k which means the distribution of items in buckets are more even with higher k.

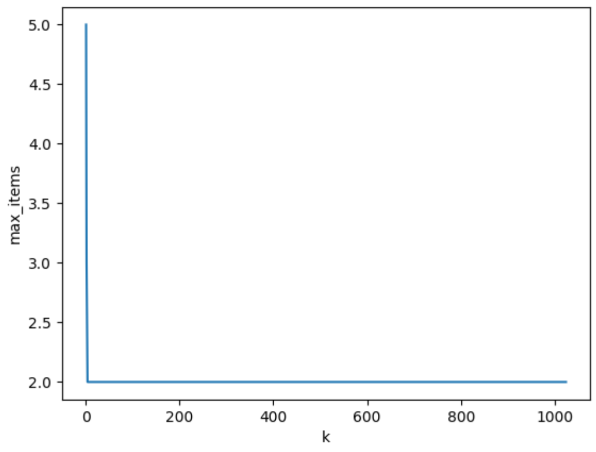

Figure 1: k-choice hashing performance as k ranges from 1 to n for n=1000

```python
import random
import numpy as np
import logging
for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)
class KchoiceHashing():
    def __init__(self, n, k):
        self.n = n
        self.k = k
        # Buckets only store counts in this simulation.
        self.buckets = [0] * self.n #nothing is assigned to the buckets yet
        self.logger = logging.getLogger(name = __name__)
        # random hash functions with different seeds.
        # Each hash function is a random number generator.
        self.h = [random.Random() for i in range(k)]

    def assign_item(self, item = None):
        '''Assigns a new item to a bucket.
        For this simulation, we don't need to assign actual items.
        We will just increment item count in the assigned bucket
        '''
        # Compute hash functions and shortlist buckets
        selected_buckets = [h.randint(0, n-1) for h in self.h]
        current_items = np.array([self.buckets[b] for b in selected_buckets])
        # select bucket with least amount of items in it
        min_bucket = selected_buckets[np.argmin(current_items)]

        self.logger.debug(f'selected bucket {min_bucket} from {selected_buckets}')

        # assign item to that bucket
        self.buckets[min_bucket] += 1

        self.logger.debug(self.buckets)

    def assign_n_items(self):
        for i in range(n):
            self.logger.debug(f'iteration {i}')
            self.assign_item(i)

    def max_items(self):
        return max(self.buckets)

    def std(self):
        return np.array(self.buckets).std()

import pandas as pd
import matplotlib.pyplot as plt
if __name__ == '__main__':

    logging.basicConfig(level=logging.INFO)
    logger = logging.getLogger(name = __name__)
    series = []
    for n in [1_000,10_000,100_000,1_000_000]:
        for k in [1,2,3]:
            max_items = 0
            total_iterations = 10
            for iteration in range(total_iterations):
                hashing = KchoiceHashing(n,k)
                hashing.assign_n_items()
                max_items += hashing.max_items()
            max_items = max_items/total_iterations
            ratio = max_items/np.log(np.log(n))
            ratio2 = max_items/(np.log(n)/np.log(np.log(n)))
            s = {
                'n':n,
                'k':k,
                'max_items':max_items,
                'max_items/log logn': ratio,
                'max_items/(log n/ log log n)': ratio2,
                'standard Deviation':hashing.std()
            }
            series.append(s)
            logger.info(f'n = {n}, k = {k}, max_items = {max_items}, ratio = {ratio:.3},
                                                         std = {hashing.std():.3}')
    df = pd.DataFrame(series)
    df = df.set_index(['n','k'])
    print(df.to_latex())

    df.to_csv('q4_out.csv')

    n = 1024
    vals = {}
    for k in range(0,int(np.log2(n))+1):
        print(k)
        k = 2**k
        print(k)
        hashing = KchoiceHashing(n,k)
        hashing.assign_n_items()
        max_items = hashing.max_items()
        vals[k]= max_items
    plt.plot(vals.keys(), vals.values())
    plt.xlabel('k')
    plt.ylabel('max_items')

    plt.show()
```