Applied Algorithms Assignment 4

Fei Fan (Peter) Chen

January 28, 2021

Exercise 1 (10 pts)

$$1048579 = 2(2^1 9 + 1) + 1 = 2 \cdot 524289 + 1$$

Using Miller-Rabin test with $r = 1$ and $d = 524289$, we choose a random $a = 3$. Using fast modulo exponentiation where:

$$a = c \mod N$$
$$a^2 = (nN + c)^2 = (n^2 N^2 + 2dnN + c^2) = c^2 \mod N$$
$$3^{524289} \mod 1048579 = 27320$$

And since $r = 1$, we find that this number is a composite.

Exercise 2 (10 pts)

a) The probability of a hash collision on the k probe for item i is $\frac{i-1}{m}^k$. For $i < n < m/2$, $\frac{i-1}{m}^k < \frac{1}{2}^k$.

b) The probability of a hash collision on $2log(n)$ probe is $\frac{i-1}{m}^{2log(n)}$. For $i < n < m/2$, $\frac{i-1}{m}^{2log(n)} \leq 2^{-2log(n)} = \frac{1}{n^2}$.

c)

$$\begin{aligned} Pr\{X > 2log(n)\} &= Pr\{max_{1 \leq < i < n} X_i > 2log(n)\} \\ &= Pr\{X_1 > 2log(n)\} + Pr\{X_2 > 2log(n)\} + ... + Pr\{X_n > 2log(n)\} \\ &\leq n \cdot \frac{1}{n^2} = \frac{1}{n} \end{aligned}$$

d) Assume $t = log(n)$

$$\begin{aligned}
E[X] &= \sum_{j=1}^{\infty} jPr(X = j) \\
&= \sum_{j=1}^{t} jPr(X = j) + \sum_{j=t+1}^{\infty} jPr(X = j) \\
&\leq t \sum_{j=1}^{t} Pr(X = j) + \sum_{j=t+1}^{\infty} (j - t)Pr(X = j) \\
&= t + \sum_{j=t+1}^{\infty} (j - t)Pr(X = j) \\
&= log(n) + \sum_{j=t+1}^{\infty} (j - t)Pr(X = j)
\end{aligned}$$

We use a summation range change to re-express the second term above:

$$E[X] \leq log(n) + \sum_{j=t+1}^{\infty} \sum_{k=t}^{j-1} Pr(X = j)$$

Notice that the last summation can be re-ordered and once re-ordered is $1 - CDF(X)$:

$$\begin{aligned}
E[X] &\leq log(n) + \sum_{k=t}^{\infty} \sum_{j=k+1}^{\infty} Pr(X = j) \\
&= log(n) + \sum_{k=t}^{\infty} Pr(X > k)
\end{aligned}$$

We use result from a) and c) where $Pr(X > k) \leq \frac{n}{2^k}$:

$$\begin{aligned}
E[X] &\leq log(n) + \sum_{k=t}^{\infty} \frac{n}{2^k} \\
&= log(n) + \frac{n}{2^t} \sum_{j=0}^{\infty} 2^{-j} \\
&= log(n) + n2^{-log(n)} \times 2 \\
&= log(n) + 2 \\
&= O(log(n))
\end{aligned}$$

NOTE: I found a lot of help with this solution online.

Exercise 3 (10 pts)

You can logical OR the two halves of the bloom filter together. Using the same k hashes, the bit location can then be applied with modulo $n/2$. This has the effect of increasing the false positive rate from $f_1 = (1 - e^{-\frac{kn}{m}})^k$ to $f2 = ((1 - e^{-\frac{2kn}{m}})^k$.

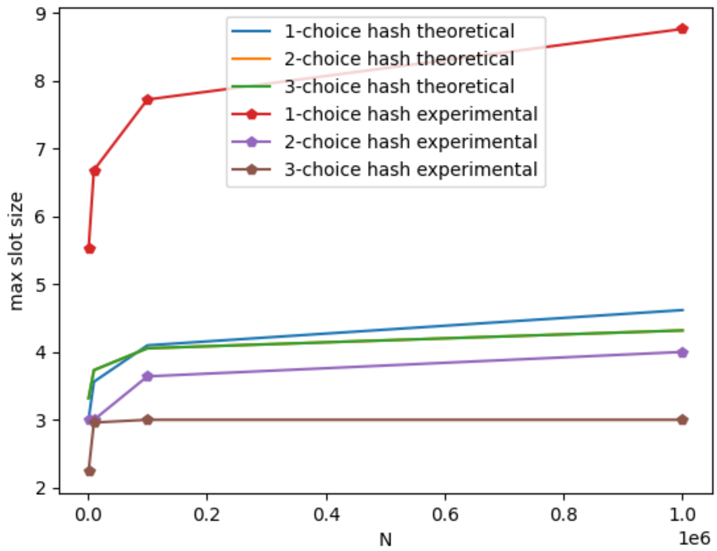

Figure 1: [1,2,3]-choice hashing and max items per slot

Exercise 4 (10 pts)

The results of 2-choice and 3 choice hashing seem to be quite a bit better than 1-choice hashing. Although even for upwards of a million items, 1-choice hashing seems to be not too bad.

Code

```
#!/usr/bin/python
import math
import random
import matplotlib.pyplot as plt

class HashSimulator():
  def __init__(self, N, hashes):
    self._N = N
    self._hashes = hashes

  def simulate(self, runs=1):
    maxSlotSizeAvg = 0
    for _ in range(runs):
      maxSlotSize = 0
      hash_table = [ 0 for i in range(self._N) ]
      for _ in range(self._N):
        indices = [ int(random.uniform(0, self._N))
            for i in range(self._hashes) ]
        index_sizes = [ (i, hash_table[i]) for i in indices ]
        index = min( index_sizes, key=lambda x: x[1] )[0]
        hash_table[index] += 1
        if hash_table[index] > maxSlotSize:
          maxSlotSize = hash_table[index]
      maxSlotSizeAvg += maxSlotSize
    return maxSlotSizeAvg / runs

if __name__ == "__main__":
  random.seed()
  N = [1000, 10000, 100000, 1000000]
  hashes = [1, 2, 3]
  runs = 25
  experimental_results = []
  theoretical_results = []
  for h in hashes:
    experimental_results.append([])
    theoretical_results.append([])
    for n in N:
      if h == 1:
        theoretical_results[-1].append(
            math.log2(n)/math.log2(math.log2(n)))
      else:
        theoretical_results[-1].append(math.log2(math.log2(n)))
      hs = HashSimulator(n,h)
      res = hs.simulate(runs)
      print(f"hashes: {h} size: {n} avg: {res}")
      experimental_results[-1].append(res)
```

```
print("all experimental results:", experimental_results)
print("all theoretical results:", theoretical_results)
plt.figure()
plt.plot(N, theoretical_results[0], "-",
  label="1-choice hash theoretical")
plt.plot(N, theoretical_results[1], "-",
  label="2-choice hash theoretical")
plt.plot(N, theoretical_results[2], "-",
  label="3-choice hash theoretical")
plt.plot(N, experimental_results[0], "-p",
  label="1-choice hash experimental")
plt.plot(N, experimental_results[1], "-p",
  label="2-choice hash experimental")
plt.plot(N, experimental_results[2], "-p",
  label="3-choice hash experimental")
plt.ylabel("max slot size")
plt.xlabel("N")
plt.legend()
plt.show()
```