Comp 480/580: Assignment #1

Rice University — Due Date: Thursday, 1/31/2019

1 A Twist on Chaining — 30 Points

a. Chaining with two different hash functions — **15 points**

We have two independent hash functions $h_1(.)$ and $h_2(.)$ taking n possible values, i.e., range of h is $0, 1, ..., n-1$. Assume both h functions are completely random so $Pr(h(x_1) = h(x_2) = ...h(x_k) = j) = \frac{1}{n^k}$ for any k and j. We still use collision resolution using chaining. So linked list for all collisions. There is a small twist, instead of just inserting element x at $h(x)$ and chaining in case of collision, we choose to add x at either $h_1(x)$ or $h_2(x)$ depending on whichever location has a smaller chain. We just gave an extra random choice to reduce the blowup.

Given that we inserted n elements. Prove that with probability less than $1 - \frac{1}{n}$, the longest chain has length $O(\log \log n)$. Contrast it with the bound shown in the class for standard chaining.

b. Implementation — **15 points**

Compare, chaining and the variants shown above. Fix a range of 1000. Now generate k different random integers between [2000 - 50,000] with seed i. Fix the hash function as $h_1(x) = x \ mod \ 1000$ and $h_2(x) = (x^2 + 362437 * x + 23427) \ mod \ 1000$. Add k numbers into the hash function and simply record the number of elements colliding in each bucket. Note the largest number L_i (also the length of the longest chain) for both normal chaining (for normal chaining just use h_1) and the variant above.

For a fixed k, change the random seed 100 times $i = 1, 2, ..., 100$ and report the average $A_k = \frac{1}{100} \sum_i L_i$.

Vary $k = 10, 100, 250, 500, 1000, 1500, 2000, 5000$ and get A_k for each.

You will submit:

1. Values of A_k (mean) and its standard deviation for both the methods as a function of k in a table format.
2. A plot of A_k for both the variants as a function of k and comment on the difference. Include standard deviation in the plot.
3. Your code and instructions on how to run and reproduce your results.

2 Inequalities — 20 points

Extend the proof in the class for 5-independent hash functions. Prove that linear probing has O(1) expected cost for searching with load factor $\alpha = 1/3$.

Hint 1: Use 4^{th} Moment Bound:

$$Pr(|B_s - E[B_s]| \geq a) \leq \frac{4thMoment(B_s)}{a^4},$$

where

$$4thMoment(B_s) = E[(B_s - E(B_s))^4]$$

Hint 2: 5-Independence implies, with X_i as defined in class, that $E[X_i X_j X_k X_l] = E[X_i]E[X_j]E[X_k]E[X_l]$ for any quadruples $[X_i, \ X_j, \ X_k, \ X_l]$

3 Implement and Test Bloom Filters 35 points

Build your own Bloom Filter

The instructions are given in python for demonstration. If using any other language, feel free to choose equivalent alternatives. Never hesitate to ask question, when in doubt

In class we saw the basic Bloom filter, where we used k independent random hash-functions $h_1, h_2, ..., h_k$ to hash a set S of N elements into an array A of R bits. Recall that the formulas for computing the best k to use for a given key set size M and maximum false positive rate at given range,

$$k = \frac{R}{N} ln2 \tag{1}$$

with false positive rate

$$fp = 0.618^{\frac{R}{N}} \tag{2}$$

3.1 Warmup Tasks

- 1. Implement a simple hash function factory. Given an argument m, the desired hash table size, the factory should return a hash function that maps integers into a table of that length. (Or use murmurhash from sklearn.utils import murmurhash3_32)
- 2. Implement a Bloom filter as a class. A Bloom filter's primary constructor receives two arguments: the desired false positive rate, c, and the expected number of keys to store, n.
- 3. You will generate two dataset: a membership set: a list of 10,000 unique integers selected randomly from the range $[10000..99999]$, and a test set: a list of 10,000 unique integers not in the membership set, also selected randomly from the range $[10000..99999]$. Demonstrate your Bloom filter for each of these false positive rates:
  0.01
  0.001
  0.0001
  In all cases, insert the items in the membership set into a Bloom filter using its primary constructor. Then look up all the items in the test and compute the false positive rate.

3.1.1 Deliverables

BF.py

```
def hashfunc(m):
...

class BloomFilter():
        def __init__(self, n, fp_rate):
        ...

    def insert(self, key):
        ...

    def test(self, key):
        ...
```

Results.txt

```
Theoretical FP                              Real FP
0.01
0.001
0.0001
```

3.2 Extended Practices

- Download Aol dataset which includes anonymized user ids and click data. (url)
- Data prepossessing: get the unique urls from the data file.

```
import csv
import pandas as pd

data = pd.read_csv('user-ct-test-collection-01.txt', sep="\t")
urllist = data.ClickURL.dropna().unique()
```

- Treat urllist as the membership set and add them to your bloom filter object.
- Sample 1000 urls from urllist as the test set. Report the false positive rate and memory usage of your design of bloom filter. Plot the false positive rate with memory by varying R. (use k=0.7R/N, N=377871 in this dataset).(memory usage of a object can be check with sys.getsizeof()).
- Insert the membership set to a python hashtable and compare the memory usage of it and that of the bloom filter.

4 For Graduate Students (COMP 580) 10 points

Read the paper "Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream" by Mitzenmacher and Vadhan. Write one page summary of the paper.