

# Comp 480/580: Assignment #1

Rice University — Due Date: Thursday, 02/13/2020

## 1 Testing Hash Functions

20 Points

**Avalanche Analysis:** A common test of hash function performance is whether or not it achieves “avalanche.” This refers to the desirable characteristic that

$$P(\text{Output bit } i \text{ changes} \mid \text{Input bit } j \text{ changes}) = 0.5 \quad \forall i, j$$

We are going to analyze three hash functions. Lets fix the range to 10 bits, and  $P = 1048573$  Pick,  $a, b, c, d$  uniformly between  $[1 - 1048573]$ , and store them. (you need to report the numbers you generated) Your hash functions  $h(x)$  are

1.  $((ax + b) \bmod P) \bmod 1024$  (2-universal)
2.  $((ax^2 + bx + c) \bmod P) \bmod 1024$  (3-universal)
3.  $((ax^3 + bx^2 + cx + d) \bmod P) \bmod 1024$  (4-universal)
4. murmurhash3 with a fixed seed (use murmurhash from sklearn.utils import murmurhash3\_32. Feel free to take extra mod to constrain it in range [0-1023])

Randomly generate 5000 positive integers 31-bit keys (values of  $x$ ) to start with. Design your experiment to estimate the probability values (you need to flip bits now).

Create a 2 dimensional heatmap : On x-axis, we have bits in the input. For every input bit  $j$ , we have to calculate 10 values, which is the probability

$$P(\text{Output bit } i \text{ changes} \mid \text{Input bit } j \text{ changes}) = 0.5 \quad \forall i, j$$

Find the most convenient way to have this heatmap. (value of 0.5 should be dark for best visualization).

Write a paragraph on the plot and your conclusion.

### 1.1 What to Submit:

Submit codes, plots, report, makefile (if any) and a main script to run the code that generate the heatmap. Submit one compressed file via Canvas. We will only type the scripts in the command line, and it should generate all the outputs. Make sure you fix the random seeds so that multiple runs of the code produces the same output.

## 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<sup>th</sup> 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

40 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  $h_1, h_2, \dots, h_k$  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,

$$k = \frac{R}{N} \ln 2 \quad (1)$$

with false positive rate

$$fp = 0.618^{\frac{R}{N}} \quad (2)$$

#### 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,  $n$ . You should not use arrays but instead use bitmaps. You can make use of an example of bitarrays in python from here [url](#) and make sure that your range is power of 2.
- 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 1000 unique integers not in the membership set and 1000 selected randomly from the membership test. 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 preprocessing: 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 and 1000 random strings as false urls. 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.

In less than a page report, describe your conclusions.

### 3.3 What to Submit:

Submit codes, plots, report, makefile (if any) and a main script to run the code that generate the plots. Submit one compressed file via Canvas. We will only type the scripts in the command line, and it should generate all the outputs. Make sure you fix the random seeds so that multiple runs of the code produces the same output.

#### **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.