Assignment 3 Report

Yunheng Han

May 2023

1 Task 1

I implemented the test program in Python using one of the existing pacakges (`https://pypi.org/project/BloomFilter`). To generarte data for testing, I randomly selected 10^6 unique 32-bit integers and built the bloom filter on half of them, i.e., the set K in the assignment description. After that, the query set K' is created by replacing some proportion of K with true negatives, i.e., ones from the other half. Both K and K' have a size of 5×10^5. The implementation is not difficult since an existing bloom filter package can be used.

The proportion of true nagetives in K' varies from 20% to 80%, while the bloom filter is built given the expected false positive rates 2^{-7}, 2^{-8}, and 2^{-10}. The perforamce of the bloom filter is shown in Table 1. As shown in the table, the false negative rate is always 0%, while the difference between actual false positive rates and expected rates is small—usually within 0.02%. The average time to query K' increases when the expected error rate decreases, and when proportion of true negatives increases. For the test query set of size 5×10^5, the average time cost is within 1 second. Finally, the space cost of the bloom filter, in terms of the number of bits for each element, matches the theoretical value well.

2 Task 2

In the second task, I implement the test program by using the package for BBHash (`https://pypi.org/project/bbhash/`). The test data set is the same as the one generated in the first task. The installation of the BBHash package was not successful at the beginning but went smoothly eventually after I updated Python to the latest version. Except this, the implementation was also easy.

The performace of the BBHash is shown by the rows in Table 2 whose fingerprint size is 0. We observe that the false positive rate of BBHash is very high (up to 99.9%). I tried to print the hash values of true negative elements in the query set, and found that almost all of them has a valid hash value no greater than $|K|$. Consequently, many true negatives cannot be detected. The execution time of BBHash, compared with the bloom filter, is smaller. It only needs 30% of the running time of the bloom filter and the time does not increase

TN	Error	Time	Space/Expected	FN	FP/Expected
20%	2^{-7}	0.63s	10.10 / 10.10 bits	0%	0.809% / 0.781%
	2^{-8}	0.63s	11.54 / 11.54 bits	0%	0.410% / 0.391%
	2^{-10}	0.68s	14.43 / 14.43 bits	0%	0.111% / 0.098%
40%	2^{-7}	0.70s	10.10 / 10.10 bits	0%	0.803% / 0.781%
	2^{-8}	0.73s	11.54 / 11.54 bits	0%	0.407% / 0.391%
	2^{-10}	0.78s	14.43 / 14.43 bits	0%	0.118% / 0.098%
60%	2^{-7}	0.78s	10.10 / 10.10 bits	0%	0.784% / 0.781%
	2^{-8}	0.80s	11.54 / 11.54 bits	0%	0.406% / 0.391%
	2^{-10}	0.90s	14.43 / 14.43 bits	0%	0.125% / 0.098%
80%	2^{-7}	0.85s	10.10 / 10.10 bits	0%	0.806% / 0.781%
	2^{-8}	0.93s	11.54 / 11.54 bits	0%	0.441% / 0.391%
	2^{-10}	1.02s	14.43 / 14.43 bits	0%	0.120% / 0.098%

Table 1: Bloom filter perforamce

as the true negative rate increases. Finally, the size of the BBHash is 3.08 bits per element.

3 Task 3

For the last task, I implemented the finger print array using an integer list in Python for simplicity. The calculation of the space cost is according to the ideal number of bits though. The test data set is the same as previous parts. This task could be more difficult if a precise implementation of the fingerprint array has to be used.

The performance of the BBHash with fingerprint is shown in Table 2. We observe that the false positive rate drops a lot with 7 bits of fingerprint (below 1%), and becomes a half (an eighth) when fingerprints of 8 bits (10 bits) are used. Furthermore, the false positive rates of BBHash using fingerprints of b bits are close to the rates of the bloom filter corresponding to an error rate of 2^{-b}. The maximum difference between two methods is within 0.1% according to the results. The running time of BBHash increases a little to check the fingerprints, but is still much smaller than the bloom filter (at most 37%). The extra space cost is completely from the fingerprints of b bits so the total cost is $3.08 + b$ bits per element.

TN	Fingerprint	Time	Space/Expected	FN	FP
20%	0 bits	0.19s	3.08 / 3.10 bits	0%	99.91%
	7 bits	0.24s	10.08 / 11.10 bits	0%	0.794%
	8 bits	0.23s	11.08 / 11.10 bits	0%	0.404%
	10 bits	0.27s	13.08 / 13.10 bits	0%	0.107%
40%	0 bits	0.18s	3.08 / 3.10 bits	0%	99.92%
	7 bits	0.27s	10.08 / 11.10 bits	0%	0.908%
	8 bits	0.23s	11.08 / 11.10 bits	0%	0.410%
	10 bits	0.26s	13.08 / 13.10 bits	0%	0.111%
60%	0 bits	0.18s	3.08 / 3.10 bits	0%	99.91%
	7 bits	0.24s	10.08 / 11.10 bits	0%	0.798%
	8 bits	0.23s	11.08 / 11.10 bits	0%	0.419%
	10 bits	0.29s	13.08 / 13.10 bits	0%	0.108%
80%	0 bits	0.18s	3.08 / 3.10 bits	0%	99.92%
	7 bits	0.24s	10.08 / 11.10 bits	0%	0.772%
	8 bits	0.22s	11.08 / 11.10 bits	0%	0.390%
	10 bits	0.25s	13.08 / 13.10 bits	0%	0.100%

Table 2: BBHash perforamce