Randomized Algorithms (RA-MIRI): Assignment #3A

The goal of this assignment is that you carry out a detailed empirical study of the performance of Bloom filters (BF). More specifically, of their probability of false positives f_{fp}, as a function of the bitvector size M, of the number of items n and of the number of hash functions k used in the BF.

You will need first to program BFs. C++ is the preferred choice, but other imperative programming languages are also fine. Make sure that the user can control all relevant parameters of the BFs, namely, the size M of the bitvector and the number k of hash functions.

Each instantiation of a BF must use a different set of hash functions. You can use some base hash function (e.g., Jenkins or MurmurHash), then combine it with two random numbers to define as many new hash functions as necessary (see the code below and the references at the end of this document).

```
template <class T>
class HashFunction {
   long a_, b_;
   int M_;
public:
   HashFunction(int M) {
      a_ = a random positive integer;
      b_ = another random positive integer
      M_ = M;
   }
int operator()(const T& x) const {
   // base_hash_function is a fixed good hash function
   convert x to a sequence of bytes y
   return (a_ * base_hash_function(y) + b_) % M_
};
...
template <class T>
BloomFilter::BloomFilter(int M, int k) {
    F = bitvector(M, 0);
    for (int i = 0; i < k; ++i) // each new object of the
                                // class HashFunction<T> has
                                // different a_ and b_
       h.push_back(HashFunction<T>(M));
}
```

For the experimental study you should use a large set of n distinct elements to build a BF, then perform a large number of searches in the BF and count the proportion of times that the BF gives a wrong answer. A useful trick is to create the BF inserting odd integers ($2i + 1$ for some random i), then search for even integers ($2j$ for some random j), since any search reporting success

is a false positive. Thus if we make U (e.g., $U = 5000$ or more) unsuccessful searches and the BF says the item belongs to the filter in F of them, we have an estimation of the false positive probability $\hat{f_{\text{fp}}} = F/U$. Be aware that the approximate analysis of Bloom filters often given in textbooks (or my slides) does not give the probability that the filter reports x as a positive **given** that x was never inserted in the filter. It gives a good approximation of the probability that x is reported as in the filter **and** x is not actually in the filter. Not of the conditional probability mentioned before. By making sure that all x that we query do not belong to F, we have

$$\Pr\{\text{filter returns } x \text{ is in } F \wedge x \notin F\} = \Pr\{\text{filter returns } x \text{ is in } F\}.$$

To get precise experimental estimations of f_{fp} (the probability of false positives), the experiment must be repeated B times creating a new BF (each one will have its own set of hash functions) for the same set of elements, then averaging for the B experiments.

Each run of B experiments will give you an estimation of f_{fp}, for fixed n, M and k. To study how f_{fp} depends on the parameters, the same steps must be followed for different values of n, M and k.

Once the full suite of experiments has been executed and data has been gathered, you have to prepare a report:

1. Describe briefly your implementation of BFs and the program to execute the experiments.

2. Describe briefly the experimental setup, how many different combinations of the parameters have you studied, how many runs have you performed for a particular M and k, what was the size of the dataset stored in the BFs, etc.

3. Provide tables and plots summarizing the results of the experiments. In particular, you should give plots showing how f_{fp} varies with the load factor n/M, with n, with M and how f_{fp} varies with k. Show empirically the best number k^* of hash functions (the one minimizing f_{fp}). Plot f_{fp} against k for different values of α; depict also the location of k^* against α. Avoid 3-D plots.

4. Compare the experimental results with the theoretical predictions. Plots combining the theoretical values and the experimental results are useful, but it is also important to quantify the difference between the theoretically predicted values and the empirical estimations.

5. Write down your conclusions.

Instructions to deliver your work

Submit your work using the FIB-Racó. The deadline for submission is December 3rd, 2021 at 23:59. It must consist of a zip or tar file containing all

your source code, auxiliary files and your report in PDF format. Include a README file that briefly describes the contents of the zip/tar file and gives instructions on how to produce the executable program(s) used and how to reproduce the experiments. The PDF file with your report must be called `YourLastName_YourFirstName-3A.pdf`,

N.B. I encourage you to use LaTeX to prepare your report. For the plots you can use any of the multiple packages that LaTeX has (in particular, the bundle TikZ+PGF) or use independent software such as matplotlib and then include the images/PDF plots thus generated into your document.

Some references

Bloom filters are described in many textbooks and in many places in the Web.

Jérémie Lumbroso from Princeton University has developed a simple and practical family of random hash functions in Java, his code can be easily adapted to C++ or other languages, or used to inspire your own variants.
random-hash

For information about Jenkins hash or MurmurHash functions start your exploration in Wikipedia (search "Jenkins hash function", "MurmurHash" or, more generally, "Non-cryptographic hash functions").