Bloom Filters

Project Due: Tuesday, November 22 (by 11:59pm), via Gradescope

In this assignment y Bloom filter and then analyze the false positive rate that your imple versus the theoretical claim.
You need to turn in grade will depend partially on the quality of your report. This is a sub vill try to give some suggestions below to help you with the report, but you. The top 5 reports will get 110% for the project, outstanding reports will get 100% (roughly top 35%) and good reports will receive 90%. The report should be at most 2-3 pages (including figures).

The idea is that we are considering using Bloom Filters, let's say for a project at our company. I want you to tell me what design choices there are and what are the advantages/disadvantages for the different choices. In addition how do Bloom Filters perform, and how do they do compared to the theoretical results we discussed in class. Provide plots/graphs where possible to illustrate your results.

Tasks

Your goal is to implement a Bloom filter that allows us to check efficiently whether an element is part of a given set or not, allowing for some false positives.

1. Design a suitable hash function

Let N denote the size of the universe, and assume that the elements will be non-negative integers from the set $U = \{0, 1, \ldots, N-1\}$ where N is some large positive integer (choose a HUGE N). A Bloom filter internally uses a table or array to mark set membership. Let n be the size of this table, where $n \ll N$.

Your first task is to implement a hash function h that is appropriate for your Bloom filter. It must satisfy the following criteria:

- Maps any number from $U = \{0, 1, \ldots, N-1\}$ to a random number in the range $\{0,1,\ldots, n-1\}$ with uniform probability. The hash output will serve as an index into the Bloom filter's table.
- Each run should be random and independent of previous runs (so every time you run your program it should have different results, which are random and independent of previous runs).

  Try the following two types of hash function:

1. For each of your k hash functions, pick a seed s_i and then fix these k seeds at the start of your program as global variable. Let r(x) be the built-in random number generator seeded at x. Then the i-th hash function is:

   $h_i(x) = r(s_i + x)$

   (i.e. it's the ran[illegible] seeded at $s_i + x$)

2. Choose a prim[illegible]. For each of your k hash functions, pick two random numbers a_i, b_i [illegible] in {1,2,...,p−1}and b_i is uniform in{0,1,...,p−1}. Then the hash function i[illegible]

   $h_i(x)$=($(a_i * x + b_i) mod$ p) mod m

Note: for this hash function to have the desired properties, p must be prime, and we need $p \geq N$.   You can lookup online about Mersenne primes for an easy way to get a suitable prime p.

Test out these 2 hash functions (and, optionally, other choices you might find) with random data. Try to compare these approaches; for example, you might look at a scatter plot of the values mapped to, or compare the max/min loads (or other statistics) of the bins. Compare how they perform using data drawn randomly from the universe to how well they perform with data having some correlation (for example, data in a sequence, like "first 2m even numbers entered"). Feel free to Google whether there are some input values for which the built-in random number generator has some correlations (honestly, I don't know the answer to this).

***Report**: Did one of the functions work better than the other? Did data choice matter? Why or why not? Justify your answer with relevant plots (1 or at most 2 figures) and discussion. More important is the later analysis of how do these hash functions compare when utilized in your implementation of a Bloom filter.*

2. Implement a Bloom filter

Let n be the number of items in the subset S that your Bloom filter needs to represent. Let $m = cn$ denote the size of your hash table T, where $c>1$.   Let N denote the size of your universe (this should be enormous). Finally, let k denote the number of hash functions used.

Now implement the two operations of the Bloom filter:

- ● add(x): Adds the element x to the set.
- ● contains(x): Returns true if x is possibly in the set, false if it is definitely not.

  ***Report**: Discuss the details of your Bloom filter implementation in part 2 of your report; specify any design choices you made.*

3. Analyze False Positive Rate for different values of k

Given a universe of size N (choose a huge N) and subset S where $|S|$ = n, compute the false positive rate for your Bloom filter for different choices of c and k, and compare its performance with theoretically derived values.

How to analyze the fals[illegible]value of c and k, and fix n (maybe 10,000 or 100,000) and N (should be huge); note [illegible]ed now. Insert n elements into the Bloom filter (but you need to keep track of th[illegible]ueries on random elements of U and see how many false positives you get. How[illegible] ones you insert? You could insert the first n elements of U, or you could do som[illegible]ted like choosing a random seed and then inserting the next n random numbers (but[illegible] this list for every positive query).

To compute the false positive rate you should perform multiple trials with your Bloom filter for each value of c and k (# of hash functions), rebuilding the Bloom filter every time and deriving new values for the hash function coefficients, and take the median of the false positive rate across all the trials.

Performing 10 trials for each value of c and k should be sufficient to provide reasonable results. Fix c first to be 10, and then 15, and sweep k from .4 * c to 1.0 * c. So, for example, you should perform 10 trials with c = 10, k = 4, regenerating coefficients a_i and b_i for each trial, then 10 trials with c = 10, k = 5, etc. You must perform this with both types of hash function described in Task 1.

Make a figure where the x-axis on each figure is *k (# of hash functions)* and the y-axis is the median *false positive rate*. Fix a c value and a type of hash function then draw the curve. You can put multiple curves/plots in the same figure, it's up to you how best to present the results in a clear and concise manner, but use at most 4 figures. On each plot also indicate the theoretical curve of false positives vs. k using the equation given in class and the notes, and indicate the theoretically predicted optimal k value.

Report*: Include the plots in part 3 of your report (be concise, at most 4 figures). Then discuss the results of your experiments. Briefly describe how you derived your theoretical values. How do your results compare to the theoretical false positive rate which is a function of k and c? Do your experiments agree with the theoretical result for the optimal choice of k, do the two different types of hash functions have different results? Any other conclusions, insights, or observations?*

Instructions

- You must complete this assignment on your own, not in a group.
- You can use any of the following programming languages: C, C++, Java or Python.
- You need to turn in a report (PDF), source code, and the Makefile (for C/C++ programs) via Gradescope.