Bloom filter

September 17, 2012

Any search engine, like google, keeps on crawling the web for fresh and more recent information. Before inserting a crawled page in the database it is first checked that if the current database (which is of huge size) already has a copy of this page or not. If the same page (with same freshness) exists in the database then storing it again must be avoided. Abstractly, this problem corresponds to checking for the membership of an element s_i in a set $S = \{s_1, s_2, \cdots, s_n\}$. In this assignment we implement a hash based data structure called **Bloom filter** which is used to answer set-membership queries with **high probability of success**.

Bloom filter A bloom filter is parameterized by following parameters,

- **n:** Size of the set which is represented by this bloom filter.
- **m:** A bit array of size m where each cell can contain bit 0 or bit 1.
- **k:** A set of k hash functions $f_1, \cdots, f_k$ each one having a range from 0 to $m - 1$.

A bloom filter consists of a bit array of size m where each cell can contain either 0 or 1 and a set of k hash functions $f_1, \cdots, f_k$ each having range from 0 to $m - 1$. Let S be the set, of size n, that we want to encode using bloom filter so that any membership query to this set can be efficiently handled. Following are the steps to create a bloom filter for this set S.

- Initialize all elements of bit array to 0.
- For each element $s \in S$, calculate $f_1(s)$ and get a number r in the range of 0 to $m - 1$.
- Set r^{th} index of bit array to 1.
- Repeat above two steps for each of the k different hash functions $f_1, \cdots, f_k$.

At the end of this encoding we have a bit array with 0 and 1 at different indices. Now we are ready to fire set-membership queries on this bloom filter. Checking for the set membership of any element s works as following,

- Find out the values $r_1 = f_1(s)$, $r_2 = f_2(s)$, $\cdots$, $r_k = f_k(s)$ by applying all k hash functions on s.
- Check if the bits corresponding to the locations $r_1, r_2, \cdots, r_k$ are set to 1 or 0 in the bit array.

- If at least one of these positions have 0 then the element s does not belong to this set.
- If all these positions have 1 then the element s belongs to this set **with high probability**. It is worth noting that having 1 at all these positions $r_1, \cdots, r_k$ does not necessary imply the membership of s in this set as different element could have mapped to same indices by hash functions. Any such s which is not in the set but declared as present using bloom filter is called as **False positive**.

1	1	1	1	0

Figure 1: Bit array corresponding to set $S = \{20, 23\}$, $m = 5$ and $k = 2$ such that $f_1 : x\%5$ and $f_2 : (7 * x + 11)\%5$

Figure 1 shows the bit array for representing the set $S = \{20, 23\}$ with $m = 5$ and $k = 2$ such that $f_1 : x\%5$ and $f_2 : (7 * x + 11)\%5$ where % represents modulo operator. Each of these functions f_1 and f_2 takes an integer and maps it to another integer in the range of $0 \cdots 4$. As $f_1(20) = 0, f_2(20) = 1$, $f_1(23) = 3$ and $f_2(23) = 2$, indices 0,1,2 and 3 are set to 1. Now let us query for the element 24 in this set. As $f_1(24) = 4$ and $f_2(24) = 4$ but the index 4 of array in figure 1 is 0. Therefore we conclude that $24 \notin S$ which is true. Now query for the element 25 in this set. As $f_1(25) = 0$ and $f_2(25) = 1$ and indices 0 and 1 in figure 1 are set to 1 therefore we conclude that $25 \in S$ which is wrong. This is an example of **False positive**. If p different membership queries are fired to bloom filter and out of these q queries resulted in false positives (element was not in the set but bloom filter found 1 in all corresponding positions) then the **rate of false positive** is given by q/p.

Having described the functionality of bloom filter we are ready to define the problem statement.

Problem statement In this assignment you have to implement a bloom filter and study the impact of parameters k (number of hash functions) and m (size of bit array) on the rate of false positives. For this you will be given two non-zero positive numbers L and n such that $n < L$. You have to generate a set S of n **different** numbers, chose randomly in the range of $1 \cdots L$, and encode it in a bloom filter (as shown in figure 1). Let us call the set of these numbers as S. Now randomly pick any element from 1 to L and check for its set membership in S. Execute a large number of such random queries and find out the rate of false positives. Finding out if a query resulted in a false positive or not should be easy because you already know the randomly created set S for which the bloom filter was created.

One important question is what should be the value of m and k for your implementation of bloom filter? For this assignment you are supposed to experiment with different values of m and k and report the impact of these parameters on false positive rates. Choices of m are $5 * n$, $10 * n$, $15 * n$, $20 * n$, $25 * n$, $30 * n$ and $35 * n$. For number of hash functions k you will experiment with

$k = 1 \cdots 8$ different hash functions. $k = 4$ means that you will take 4 of these hash functions and apply them all for insertion and set-membership checking.

The hash functions $(f_1, f_2, f_3, ..., f_k)$ are to be generated in the following manner - for each hash function f, you should generate random numbers a and b each distributed uniformly between 0 and m^2, so that f_i can then be taken as

$$(a * i + b) mod(m) \tag{1}$$

To make k hash functions you have to generate and store k such pairs.

Note: The random numbers are integers everywhere in the problem.

Sample input 1000 400
First number (1000) denotes the value L and second number (400) denotes the value n.

Sample output

```
##k(hash functions) m=5*n, m=10*n, m=15*n, m=20*n,m=25*n,m=30*n,m=35*n
1                   0.6    0.15    0.5     0.14   0.4    0.17   0.3
2                   0.16   0.8     0.13    0.3    0.15   0.4    0.17
3                   0.8    0.11    0.5     0.15   0.6    0.14   0.3
4                   0.19   0.6     0.12    0.5    0.18   0.7    0.15
5                   0.19   0.7     0.14    0.6    0.16   0.6    0.16
6                   0.7    0.17    0.5     0.19   0.7    0.16   0.4
7                   0.20   0.5     0.19    0.6    0.17   0.5    0.15
```

First column represents number of hash functions used in bloom filter. Column 2 to 8 represents the value of m(size of bit array) as a function of input n (size of the set encoded using bloom filter). One advantage of this output is that we can directly use an open source tool called `gnuplot` to plot the graph corresponding to this data. For this purpose first save your output in a file named "test.dat". Now save the file "test1.pg" (uploaded with this assignment) in the same directory where "test.dat" is stored. Execute the following commands on command line (linux).

- chmod +X test1.pg
- ./test1.pg > output.png

After executing second command a new file named "output.png" will be created in your current directory. Open it in any image viewer and you will see the plot associated with your data. For further information about `gnuplot` you can also refer to the following link **http://www.gnuplot.info/**.