POLITECNICO DI TORINO

Computer aided simulations and performance evaluation

Academic year 2020/21

VASSALLO Maurizio, s276961

Contents

Laboratory #3

Introduction

These experiments involve the use of the English dictionary to check the performance of different data structures when dealing with the membership problem, so answer the question: ''is an element present?". This is known as set membership problem, actually here it will be considered the approximated set membership problem, since the presence of false positive is accepted.

1.1 Fingerprinting

1.1.1 Number of words in the file

The English word dictionary contains 370103 words.

1.1.2 Minimum value of bits b^{exp} for no collisions

The minimum number of bits necessary to store all the words in a fingerprint table without conflicts is: $b^{exp} = 39$ bits.
To find this value, for each one of the words, the fingerprint is calculated:

> Each word is encoded using the UTF-8 character encoding, then the MD5 hash is calculated. This MD5 hash returns a 128 bits value but since it is required to have fingerprint length is b^{exp} bits, the MD5 hash value is converted to integer and only the last b^{exp} bits are taken; this is done with the module operation:
>
> $$fingerprint\_value = word\_hash\_int \,\%\, 2^{b^{exp}}$$

This operation is repeated for each word and candidate b^{exp} value until the minimum value possible is found. In particular this search is performed using a Binary Search algorithm; to write is a bit more complex than other algorithms but it is faster: for example is faster than an algorithm that checks all numbers between 1 and infinite and stops as soon it find a value of b^{exp} such that any collision is found.

1.1.3 Input parameters

The input parameters of the simulation are:

- The file containing the English words;
- The seed value used to initialize the pseudorandom number generator;
- The confidence level used to calculate the confidence interval;
- The number of runs: the number of times that we run our simulation. This is done in order to have more accurate;
- Number words used for checking the probability of false positive.

1.1.4 Output metrics

The output metrics of the simulation are:

- The value of b^{exp};
- The theoretical value of b: b^{teo};
- The storage memories required for the data structures and the theoretical storage memories;
- The probability of false positive using a b^{exp} fingerprint set.

1.1.5 Simulator main data structure

There are 2 main data structures used:

- A python set where to store all the English words;
- A python set where to store the fingerprint of each English word.

1.1.6 Theoretical number of bit b^{teo}

It is possible to analytically compute the minimum number of bits given a specified probability p using the following formula:

$$b^{teo} = \log\left(-\frac{num\_words^2}{2 \times \ln(1-p)}\right)$$

This formula is obtained solving the probability of conflict equation of the Birthday Paradox in function of n (the number of days). In this case the number of days is substituted by the possible length of the fingerprint table ($n = 2^{bits}$).

1.1.7 Relation b^{exp} and b^{teo}

The theoretical value and the simulated value are really close:

$$b^{exp} = 39;\ b^{teo} = 37$$

(the decimal value is 36.52), ratio = 39/37 = 1.05

1.1.8 Theoretical amount of memory

The theoretical memory, in MB, required to store all the words using the two structures, fingerprint table and python set, is:

$$th\_size\_fp\_table = num\_words \times \frac{b^{exp}}{8} \times \frac{1}{1024^2} = 1.72 \text{ MB}$$

$$th\_size\_python\_set = num\_words \times 4.79 \times \frac{1}{1024^2} = 1.69 \text{ MB}$$

where 4.79 is the average length words of the English dictionary. (1 character = 8 bits) (reference).

1.1.9 Actual amount of memory

To calculate the memory used by an object in python it is possible to use the *asizeof(obj)* from the *pympler* library. This returns the memory in bytes of a given object.

Memory required to store: the fingerprint table: 16.47 MB, for the set of words: 21.46 MB

1.1.10 Probability false positive

There are different ways to calculate the probability of false positive, simulation is one of this.
This 'simulation' is performed creating some 'fake' words, these are not real word but are integer numbers between [0,n). This is done to simulate the behaviour of the hash function. Given these 'fake' word a check is performed to understand if the 'word hash' is present in the fingerprint set, if it is, then there is a case of false positive.

1.1.11 Summary previous results

Storage	Bits per fingerprint	Prob. false positive	Min theo. memory (MB)	Actual memory (MB)
Word set	N.A.	0	1.69	21.46
Fingerprint set	39	2.66×10^{-7}	1.72	16.47

It is possible to see that the memory required for the python set is much more than the memory required to store the fingerprint table. This difference is more evident if the elements to be stored are different in sizes, for example: to store audio files, the memory required for the python set would be very high instead the one for the fingerprint table would not change (as long as we keep the same value of number of bits).
The probabilities of false positive is calculated using 5×10^6 number of testing words (words not in the English dictionary) but the probability of false positive for the fingerprint table in some runs were still 0.
Actually, this is a rare event and would require even an higher

number of test words but this would take more time and computational power so in this case it is better to calculate analytically with the following formula:

$$\text{prob(false\_pos)} = \text{number\_words} \;/\; 2^{39} = 6.732 \times 10^{-7}$$

1.2 Bit String Array

1.2.1 Input Parameters

The input parameters of the simulation are:

- The file containing the English words;
- A list with the possible values of the number of bits: [19, 20, 21, 22, 23, 24];
- An integer flag to indicate that we want to use Bit String Array or Bloom Filter. (0: Bit String Array).

1.2.2 Output Metrics

The output metrics of the simulation are:

- Number of bits;
- The probability of false positive;
- The memory occupancy of the bit string array;
- The theoretical memory occupancy.

All these value are stored in a file and each field is tab separated.

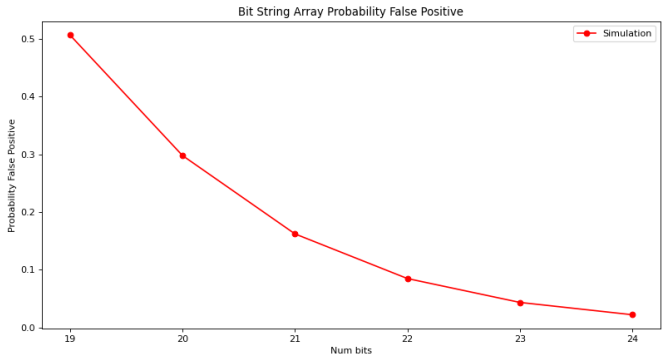
1.2.3 Main data structures

There are 3 main data structures used:

- A python set to store all the English words;
- A bit string array, from the *bitarray* library, of length $2^{\#bits}$;
- A numpy array, of length #runs, where to store the probability of false positive at each run.

The bit string array allows to determine if an element is already stored or not with a constant access time of $\mathcal{O}(1)$, since it share the same property of a simple array.
This implementation allows to reduce the storage memory.

1.2.4 Probability of false positive in function of bits per fingerprint



The probability of false positive is calculated as:

$$prob(false\_pos) = \frac{\#1s}{total\_length} \tag{1.1}$$

where $\#1s$ is the number of 1s in the bit string array and $total\_length$ is the total size of the bit string array.

1.2.5 Summary previous results

Storage	Bits fingerprint	Prob. false positive	Min theo. memory (MB)	Actual memory (MB)
Bit String Array	19	0.506	0.044	0.066
	20	0.297		0.125
	21	0.162		0.25
	22	0.084		0.5
	23	0.043		1.0
	24	0.022		2.0

The theoretical probability is calculated as the memory required to store the number of words using a 1 bit (possible values: True, False) so it does not depend on the number of bit used. Formula:

$$theoretical\_memory = num\_words \times \frac{1}{8} \times \frac{1}{1024^2} = 0.044 \text{ MB}$$

It is possible to notice that probability of false positive is not as low as the one of fingerprint table but the used memory is much lower.

1.3 Bloom Filters

The Bloom Filters are just an extension of the Bit String Array so they share lot of properties and code.

1.3.1 Input parameters

The input parameters of the simulation are:

- The file containing the English words;
- A list with the possible values of the number of bits: [19, 20, 21, 22, 23, 24];
- An integer flag to indicate that we want to use Bit String Array or Bloom Filter. (1: Bloom Filter).
- The seed value used to initialize the pseudorandom number generator;
- The confidence level used to calculate the confidence interval;
- The number of runs: the number of times that we run our simulation. This is done in order to have more accurate.

1.3.2 Output metrics

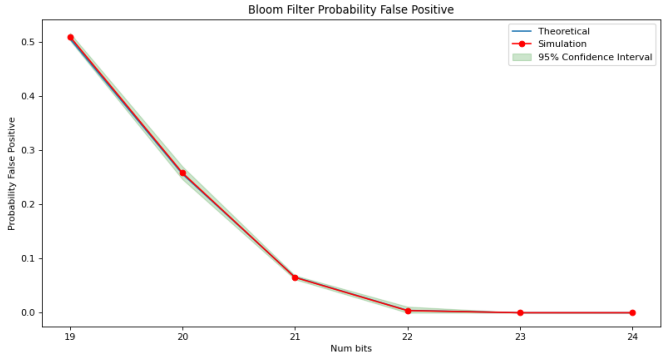
The output metrics of the simulation are:

- Number of bits;
- The number of hashes used.
- The lower confidence interval value;
- The average probability of false positive;
- The upper confidence interval value;
- The relative error;
- The theoretical probability of false positive;
- The memory occupancy of the bit string array;
- The theoretical memory occupancy.

1.3.3 Main data structures

The same as the bit string array.
For Bloom Filters, the time needed to determine if an element is present or not is constant and it has a worst case access time of $\mathcal{O}(k)$, where k is the number of hash functions used.

1.3.4 Probability of false positive



The simulation is run using 30000 'fake' words and even with such a lower number the simulated results are quite accurate. The theoretical value is calculated as:

$$prob(false\_pos) = \left(1 - e^{\frac{k_{opt} \times num\_words}{total\_length}}\right)^{k_{opt}} \quad (1.2)$$

where $total\_length$ is the total size of the bloom filter, k_{opt} is the optimal number of hashes to use and is calculated as:

$$k_{opt} = \frac{2^{\#bits}}{number\_words} \times \ln(2) \quad (1.3)$$

Given this value, one should check both upper and lower integer (ceil() and floor()) and pick the best one but usually round to the nearest integer works fine.

1.3.5 Summary previous results

Storage	Bits fingerprint	Prob. false positive	Min theo. memory (MB)	Actual memory (MB)
Bloom Filter	19	0.51	0.044	0.066
	20	0.26		0.132
	21	0.065		0.265
	22	0.004		0.531
	23	1.66×10^{-5}		1.062
	24	0*		2.125

In general it is possible to see that the performance are better than the bit string array's one: since the memory used is almost the same while the probability of false positive is much lower.
In general the performance follow the relations:

$$bloom\ filters > bit\ string\ arrays > fingerprint\ tables$$
$$(with\ k_{opt} > 1)$$

*This result is obtained with 30000 words and 2 runs. Since this is a rare event it would require a lot more words.

1.3.6 Performances given 1 MB of memory

Theoretically, given 1 MB of memory the performance using the different data structures are:

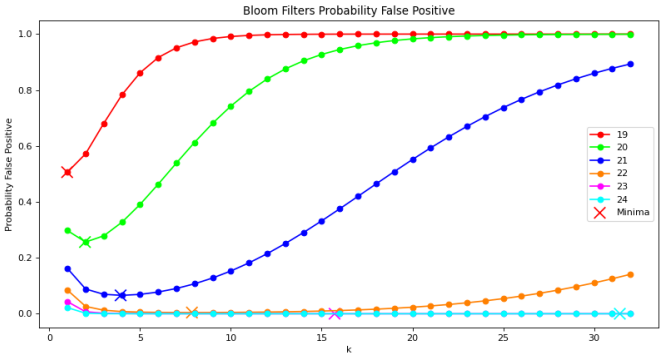
- Word set: the storage is not possible since we may only store, $\frac{1MB}{1.69MB} = 0.59$, 59% of the total words;

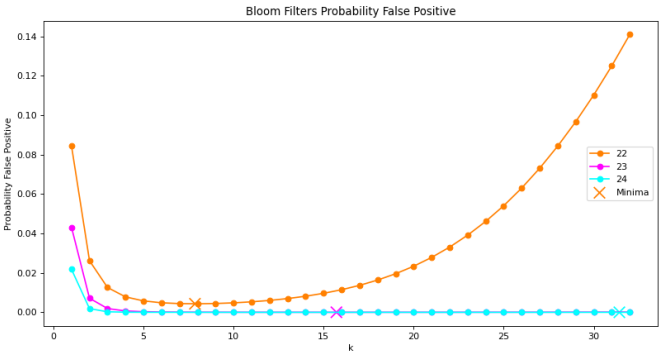
  For the other structures it is possible to use number of bits equal to $\frac{1024^2 \times 8}{\#words} = 22.66$ bits, so approximating this value, in average for each word, only 22 bits:
- Fingerprint table: the storage of the whole list of words is always possible but there is a change on the probability of false positive depending on the number of bits used. Using 22 bits the probability of false positive would be $eps = \frac{\#words}{2^{22}} = 0.0882$;
- Bit String Array: Using 22 bits the probability of false positive would be $eps = \frac{1}{22} = 0.0454$;
- Bloom filter: Using 22 bits the probability of false positive would be $eps = \frac{1}{2^{\frac{22}{1.44}}} = 2.51 \times 10^{-5}$.

Data Structure	Total Storage Formula (bit)	Max Num bits	Formula solved for Epsilon	Epsilon Value
Fingerprint Table	$m \times \log_2 \frac{m}{eps}$	22	$\frac{m}{2^{22}}$	0.0882
Bit String Array	$m \times \frac{1}{eps}$		$\frac{1}{22}$	0.0454
Bloom Filter	$m \times 1.44 \log_2 \frac{1}{eps}$		$\frac{1}{2^{\frac{22}{1.44}}}$	2.51×10^{-5}

where m is the number of words to be stored.
It is possible to see that the bloom filters would have better performances given 1 MB of space.

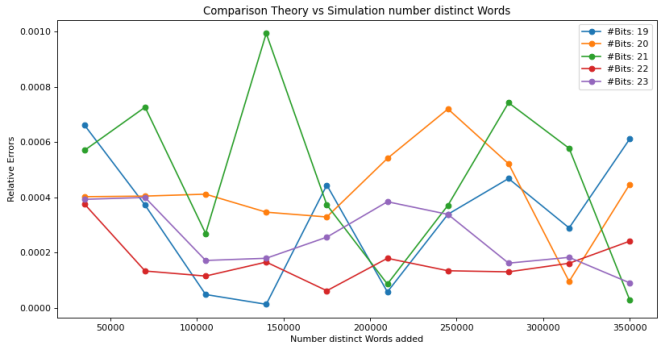
1.3.7 (optional)Optimal number of hash functions





It is possible to see that the theoretical formula is accurate since the minimum is found where the probability of false positive is lowest.

1.3.8 (optional) Estimation distinct elements in a bloom filter



Given the theoretical formula:

$$theo\_dist\_words = -\frac{n}{k} \ln\left(1 - \frac{N_1}{n}\right)$$

where: n is the bloom filter storage length, k is the optimal number of hash functions given a number of bit (1.3) and N_1 is the actual number of 1s in the bloom filter.

It is possible to see that the simulation is quite accurate with respect to the theoretical formula. This is shown using the relative errors ($rel\_err = \frac{|theo\_dist\_words - dist\_words|}{dist\_words}$) and these relative errors are low (the axis scale is quite low) so the theoretical formula is accurate. Moreover it can be noted that the relative errors with a higher number of bits seems to be more constant while for lower values of bits there is more fluctuation.