Lab 17 - Antiplagiarism poem system

Sepehr Alidousti Shahraki
S289456
Prof. Paolo Giaccone
Computer-aided simulations lab

I. Preliminary study

The inputs of the software are:

- a set of sentences each sentence with the size of 6 words
- a bit array (which can be used for Bloom Filter and Bit String Array)
- the parameters of number of bits for bit array
- number of hash functions (for Bloom Filter)

The outputs of the software are:

- The probability of false positives both in theory and in simulation
- A comparison of the theoretical and simulated false positive probabilities in plots
- (Optional) A plot of the difference between the actual number of sentences and the number of distinct elements in the Bloom filter as more sentences are added to it
- A plot of the memory occupation of the Bloom filter and Bit string array in simulation.

The number of sentences with the size of 6 is equal to 96227. And the corresponding memory occupancy is almost equal to 12907 KB. Which makes the average size of each sentence almost equal to 134 bytes.

II. Fingerprint set

To find the minimum value of bits in simulation (Bexp) such that no collisions are experienced when storing all the sentences in a fingerprint set. This is done by iterating over different values of b, generating fingerprints for each sentence in the set using a specific value of b, and checking if there are any collisions by comparing the length of the fingerprint list with the length of the set of fingerprints. If there are no collisions, Bexp is set to that value of b and the loop is broken.

Then to calculate analytically the number of bits necessary (Bteo) to get a probability of 0.5 of fingerprint collision when storing all the sentences in a fingerprint set. This is done by using the formula (log2(m/1.17))^2. Where m is the number of sentences.

Then my code calculates the probability of false positive (Pr_FP_Bexp) for a fingerprint set in which the number of bits is Bexp, as computed in the first step. This is done by using the formula 1 - (1 - 1 /n) ^m, where n=2^Bexp.

III. Bit string array

For this part I created the class BITSTRINGARRAY which has different functions in it, I only describe some of the important ones:

1. bit_array_size(self): This method returns the size of the bit array in bytes
2. hash_function(self, element): This method takes an element as input and returns its hash value. The method uses the md5 hash function to compute the hash value.
3. add(self, element: str): This method takes an element as input, computes its hash value using the hash_function method, and sets the corresponding bit in the bit array to 1.
4. count_ones(self) -> int: This method counts the number of 1's in the bit array and returns it as an integer.
5. **contains**(self, element: str) -> bool: This method is used to check if an element is in the set or not. It takes an element as input, computes its hash value using the hash_function method, and checks if the corresponding bit in the bit array is set to 1. If the bit is set to 1, it returns True, otherwise it returns False.

The first part of the simulation calculates the false positive probability and memory occupation of the BSA for each value of "b". For each value of "b", a new BSA object is created with a size of 2^b. The sentences in the "sentence_set" are then added to the BSA using the "add()" method. The false positive probability is calculated by counting the number of ones in the BSA and dividing by number of bits. This value is appended to "prob_FP_sim". The memory occupation is calculated using the formula 0.000125 * m / prob_FP and appended to "memory_simulation".

The Figure 1 is the plot related to "The Comparison of false positive probability in Simulation and Theory".

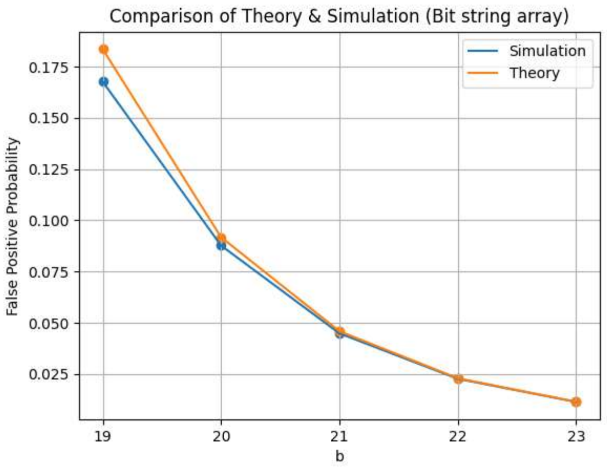

Figure 1

If I want to make a comparison between using 'Bexp' and using Bitstring array, I can state that from the point of view of memory occupancy and false positive probability the bit string array is much more efficient in the memory wise but using 'Bexp' in fingerprint set generates a real lower false positive probability.

IV. Bloom filters

To implement the Bloom Filter structure in python I made a class, which has different functions in it. Most of them are like the 'Bit String Array' class, but what makes it different is the number of hash functions and how to have more than 1 hash function.

To make multiple value for the hash function I added an integer number as string in the end of the sentence. This integer number is in the range of [1, k]; which k is the hash functions that we have for the Bloom Filter. In this way the code K different values for the same sentence.

To find the optimal k for different number of bits I started to iterate over the list of k in range [0,200]. At the worst case scenario, we found if b is equal to 23 the optimal k is 60.

In order to find the optimal k for each b it iterates over different value of k and computes the probability of False Positive for each one. We find the min of the probabilities and the assigned k for that.

The probability is computed with this formula: (#1/n) ^k

Figure 2 is the result of iterating over k in range [0,200] for different values of b and computing the Probability of False Positive.

b value	optimal k
19	4
20	8
21	15
22	30
23	60

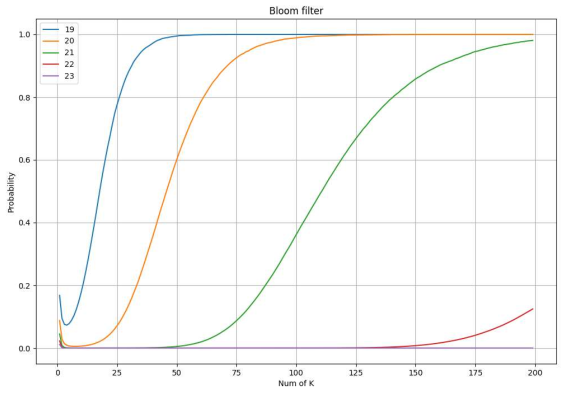

Figure 2

We can compare the results of the probability of false positive in theory and simulation for the optimal number of hash functions in Figure 3.

As it's obvious in the plot the difference between simulation and theory is not that much and the when the number of bits grows then the results become more similar.

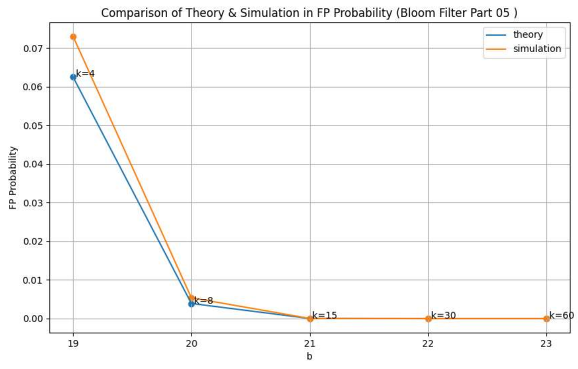

Figure 3

For the optional part since it wasn't mentioned in the task I used the optimal number of hash finctions for each value of b. And inorder to show the accuracy of the formula I plotted the difference between the actual number of sentences in Bloom Filter and what formula returns as the value of the the number of distinct elements stored in a bloom filter.

In Figure 4 we can see the result by checking the value of the formula after every 5000 sentence to make the plot more clear.

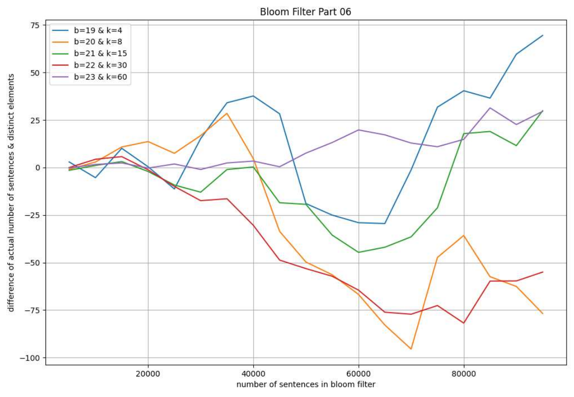

Figure 4

But I also plotted the real result after inserting of every sentence, which we can see the result in figure 5.

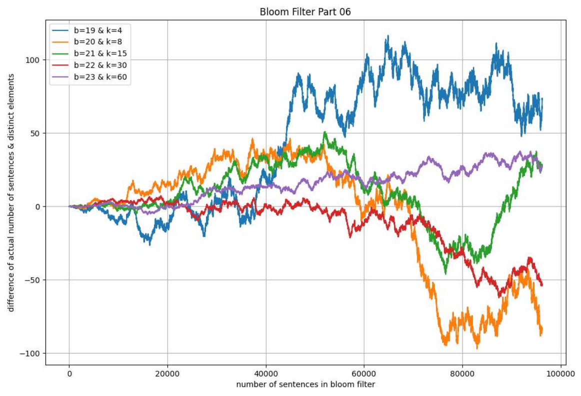

Figure 5

V. Comparing the Results

As it was requested in the end of the task, we can see the different tradeoff between memory and performance achieved each data structure, based on the results above summarized in the following table.
For the cases of Bloom filter and Bit String Array instead of showing the result in table, they are shown in the related Figures.

Data structure	**Memory [kB]**	**Prob. false positive**
Set of sentences	12907.688	0.0002285738033641908
Fingerprint set (X=Bexp)	3880.248	0.00000070014332820278
Bit string array (X varying)	Figure 6	Figure 1
Bloom filter (X varying)	Figure 7	Figure 3

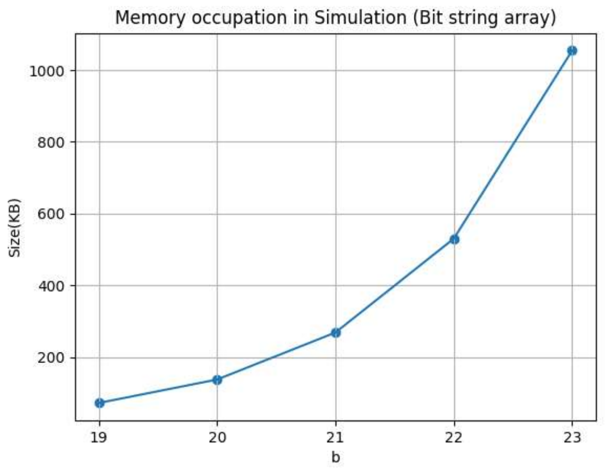

Figure 6

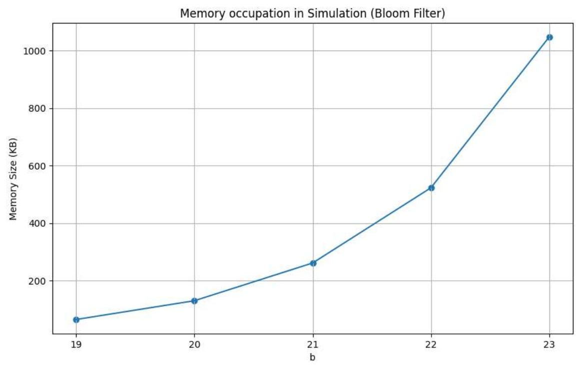

Figure 7

Also, in Figure 8 we can see the difference of Memory Occupancy for Bloom Filter and Bit String Array (BF_Memory – BSA_Memory).

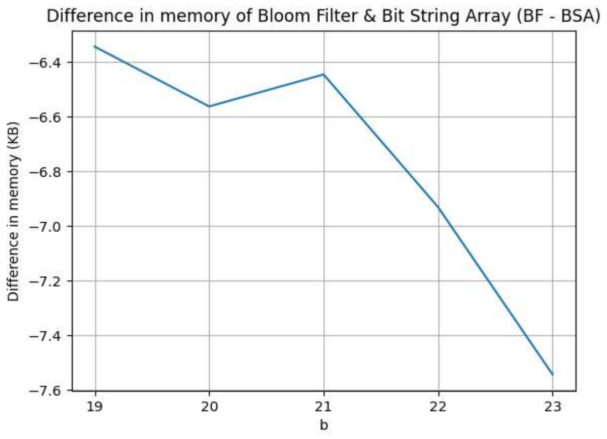
Difference in memory of Bloom Filter & Bit String Array (BF - BSA)
Difference in memory (KB)
−6.4
−6.6
−6.8
−7.0
−7.2
−7.4
−7.6
19
20
21
22
23
b

Figure 8