Report on

"Bloom Filters in C"

"project id - 30"

Submitted in partial fulfillment of the requirements for ***Sem VI***

Design Patterns

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Rohit Vishwakarma   PES1201800152

Under the guidance of

Prof . Nagbhusan Satya Kumar
Professor
PES University, Bengaluru

January – May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

Introduction to Bloom Filters

The Bloom filter data structure tells whether an element may be in a set, or definitely isn't. The only possible errors are false positives: a search for a nonexistent element can give an incorrect answer. With more elements in the filter, the error rate increases.

Bloom filters are both fast and space-efficient. However in general elements can only be added into the Bloom Filters and are not removed *(They can be by making a set bit unset but not done in practice).* Content distribution networks use them to avoid caching one-hit wonders, files that are seen only once. Web browsers use them to check for potentially harmful URLs. I am discussing bloom filters as a Design Pattern in the up coming pages .

## Pattern Description

Design of probabilistic data structure for the Filtering Design pattern can be called as Bloom Filter or put the other way Bloom Filtering in Design Pattern is a part of filtering pattern in general .

#Pattern Description

Bloom filters are simply filters of data which evaluate each data record separately and makes the desicion on some property or rule (given condition) so to make sure whether the current record (input to the bloom filter as record) will it pass through the filter or not .

Incase of the bloom filters we use a tuple of hashing function on each of the records and then decide the result . The hashing fuction can be any function which hashes to the index in the range of [0, n) where n is number of bits used in the implementation.

Bloom filters provide much faster and compact way of getting a quick presence check or responses rather than storing the items in a set . These bloom filters are also called as SISMEMBERS.

## Intent of the pattern

#Intent of the pattern

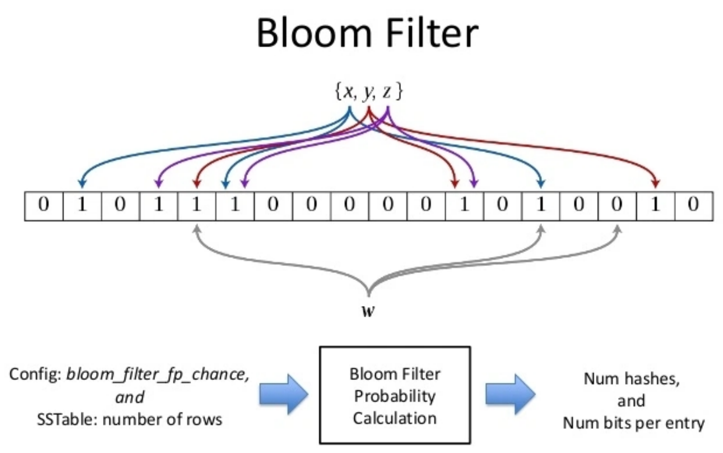
Bloom filter tries to answer the question of set membership by the looking for the fixed number of hot values (set bits) this fixed number is basically the number of hashing function . The job of these is to extract features from a given input record and generate hot values in a bit array. This way it tries to answer the set membership. The queries of the bloom filters may become little inaccurate and we call this as false positivity.

Motivation of the Pattern

#Motivation

Have you ever found yourself in a situation where you needed to keep track of whether or not you’ve seen an item but the number of items you have to keep track of is either gigantic or completely unbounded .Now if you make a hash map for such a gigantic data a lot of memory is going to be used and it would seem impractical Eg. Keeping track large Google Search Engine . Bloom filter comes to your rescue and helps you do this task with an extremely space efficient method with remarkable accuracy .

Working of the Pattern


Bloom Filter
{x, y, z}
0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0
w
Config: bloom_filter_fp_chance, and SSTable: number of rows
Bloom Filter Probability Calculation
Num hashes, and Num bits per entry

#Working of the Pattern

We create a probabilistic data structure which is called as bloom filter . It consists of a bit vector of length (n) based on the expected number of inputs (m) . It first calculates the optimum number of hashing functions (K) to be used per input record based on expected false positivity rate (p) of the bloom filter implementation. Now on a given input string we calculate K hashes values from K hash function where each would output a number in the range [0, m) and these set of indices in the bit array would be set up. Hashing functions will always generate the same output index tuple for a particular string . This way a string is added to the bloom filter . To do a look up in the bloom filter for a string is always done by looking at the hashed indices in the bit array and if all are not hot in the tuple of indices then we can perfectly say that particular element has not been seen yet . If all bits are up then we can say that element has been seen before with false positivity probability = "p".

#Mathematical Survey of bloom filters

Assume that a Hash function selects each array position with equal probability. If m is the number of bits in the array,the probability that a certain bit is not set to 1 by a certain hash function during the insertion of an element is

$$1-\frac{1}{m}.$$

If *k* is the number of hash functions and each has no significant correlation between each other, then the probability that the bit is not set to 1 by any of the hash functions is

$$\left(1-\frac{1}{m}\right)^k.$$

We can use the well-known identity for e^{-1} .

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m=\frac{1}{e}$$

to conclude that, for large *m*,

$$\left(1-\frac{1}{m}\right)^k=\left(\left(1-\frac{1}{m}\right)^m\right)^{k/m}\approx e^{-k/m}.$$

Now to check the membership of an item we need to find the probability that the value is setbit = 1

$$1-\left(1-\frac{1}{m}\right)^{kn}\approx 1-e^{-kn/m}.$$

continued below

The number of hash functions *k* must be a positive integer. Putting this constraint aside, for a given *m* and *n*, the value of *k* that minimizes the false positive probability is k .

$$\varepsilon = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

Now we find the optimum number of hash in this equation by tweaking in the value $k = \frac{m}{n} \ln 2.$ (1) this when put in the above equation looks like

$$\varepsilon = \left(1 - e^{-(\frac{m}{n}\ln 2)\frac{n}{m}}\right)^{\frac{m}{n}\ln 2}$$

(2)

which on further simplification yields (4) $\ln \varepsilon = -\frac{m}{n}\left(\ln 2\right)^2.$

taking advantage of this equation I use it
to calculate the optimum number of bits to be alloted in the bloom filter only by knowing the number elements as the estimated input.

On a given bloom filter of size n and positivity k estimating the number of elements in the bloom filter is by the below .

$m = -\frac{n \ln \varepsilon}{(\ln 2)^2}$ (3)

I have used these equations extensively to code in the Project .

Coming to operations on bloom filters with same n .

The union and intersection of bloom filters .
Bloom filters are a way of compactly representing a set of items. It is common to try to compute the size of the intersection or union between two sets. Bloom filters can be used to approximate the size of the intersection and union of two sets.

For finding the union of two bloom filter of same size n and m with same p.

The union is = bitwise OR of the bloom filter array .

The intersection = bitwise AND of the bloom filter array .

Jaccard Index of two bloom filters

Jaccard Index = number of # elements in intersection / number of elements in union if Jaccard Index = 1 then identical bloom filters and 0 if nothing common .

Applicability and Consequences

Applicability :

We can use bloom filter where the some false positive result are okay .

Consequences :

When a trained bloom filter is exposed to a set of strings then it produces an output which has a subset of the input set which passed the membership test . But the bloom filter can also output the strings which were not added because of false positivity.

Known uses and Resemblances

##Known uses

Sometimes, checking whether some value is a member of a set is going to be expensive. For example, you might have to hit a webservice or an external database to check whether that value is in the set. The situations in which this may be the case are far and few between, but they do crop up in larger organizations. Instead of dumping this list periodically to your cluster, you can instead have the originating system produce a Bloom filter and ship that instead. Once you have the Bloom filter in place and filter out most of the data, you can do a second pass on the records that make it through to double check against the authoritative source. If the Bloom filter is able to remove over 95% of the data, you'll see the external resource hit only 5% as much as before! With this approach, you'll eventually have 100% accuracy but didn't have to hammer the external resource with tons of queries.

Resemblances

Bloom Filter are hot in the fields of big data and Data analytics . Present technology gets benefit from the data being big is exploitable in Pig and SQL as they can use bloom filter .

Performance

Performances

The performances of the bloom filter are very great as they are faster , extremely space efficient and can be implemented in a file too !. The hash functions als are a more or less have cheap time. The look up operation is also very cheap in bloom filters O(1) time.

Implementation

The following is the way bloom filter is implemented by me in C . #functions prototypes are show below

1.)

Function for initialization of bloom filter

```
int bloom_filter_init(bloom_filter * bloomFilter, uint64_t approximate_elements,
                      float false_positivity, bloom_hash hash_func);
```

Initialize a bloom filter with approximate number of elements , false positivity and your desired hashing function

2.)

Function to show the current health of the bloom filter

```
void stats_bloom_filter(bloom_filter * bloomFilter);
```

3.)

Function to free the bloom filter from memory

```
int free_bloom_filter(bloom_filter * bloomFilter);
```

4.)

Function to reset the bloom filter to make health better again.

```
int free_bloom_filter(bloom_filter * bloomFilter);
```

5.)

Add String into the bloom filter

```
int add_string(bloom_filter * bloomFilter, const char * string);
```

6.)

Look up in the bloom filter for a string

```
int is_present_bloom_filter(bloom_filter * bloomFilter, const char * str);
```

7.)

Look at the current false positivity of the bloom filter .

```
float get_current_positivity_rate(bloom_filter * bloomFilter);
```

8.)
Gets the number of hot bits
```
uint64_t count_bits_set(bloom_filter * bloomFilter);
```

9.)
Estimates the number of elements added.
```
uint64_t estimating_elements(bloom_filter * bloomFilter);
```

10.)
Find union of two bloom filters with same n and hash function
```
bloom_filter * union_bloom_filters(bloom_filter * source1, bloom_filter * source2);
```

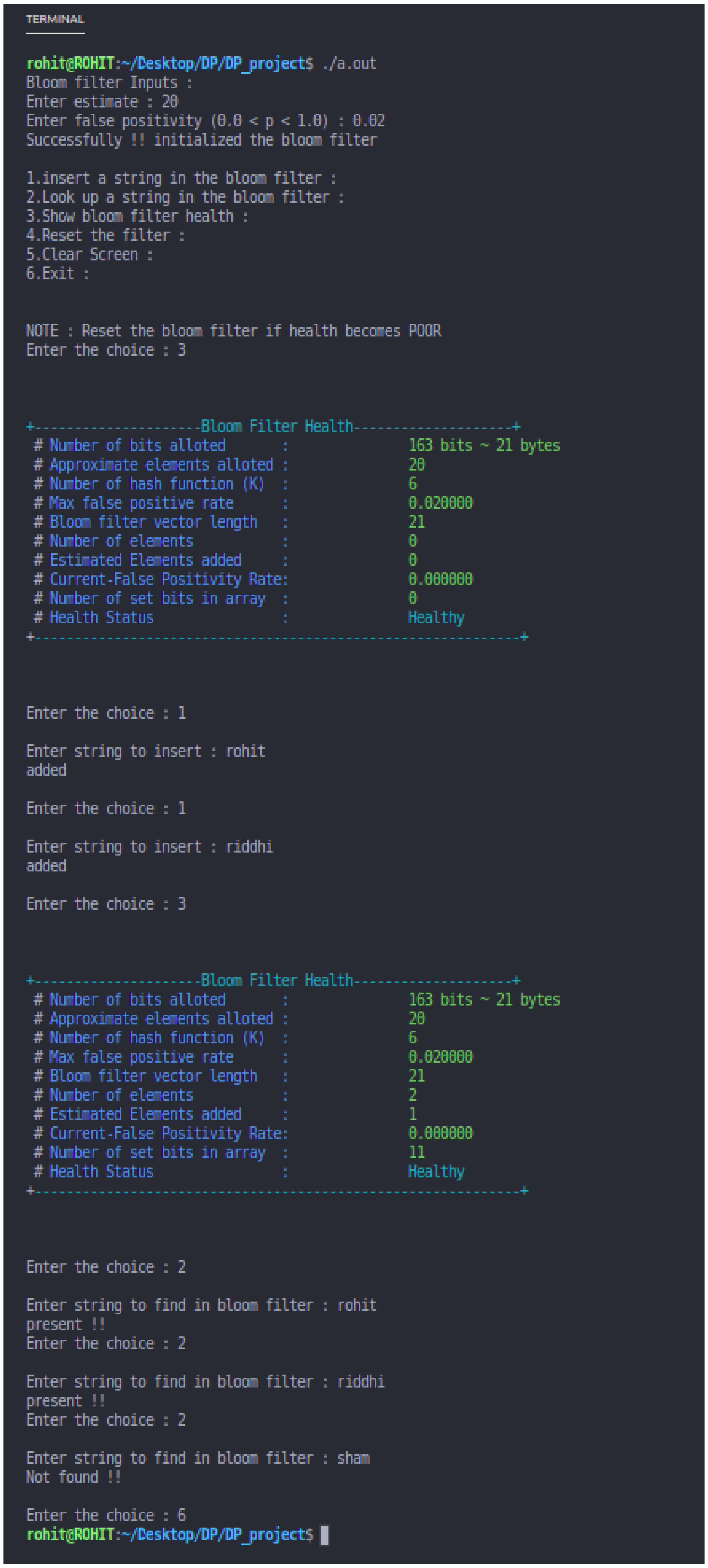
11.)
Find intersection of two bloom filters with same hash function

```
bloom_filter * intersection_bloom_filters(bloom_filter * source1, bloom_filter * source2);
```

12.)
Index which calculates how similar are two bloom filters .

```
long double jaccard_index_bloom_filters(bloom_filter * source1, bloom_filter * source2);
```

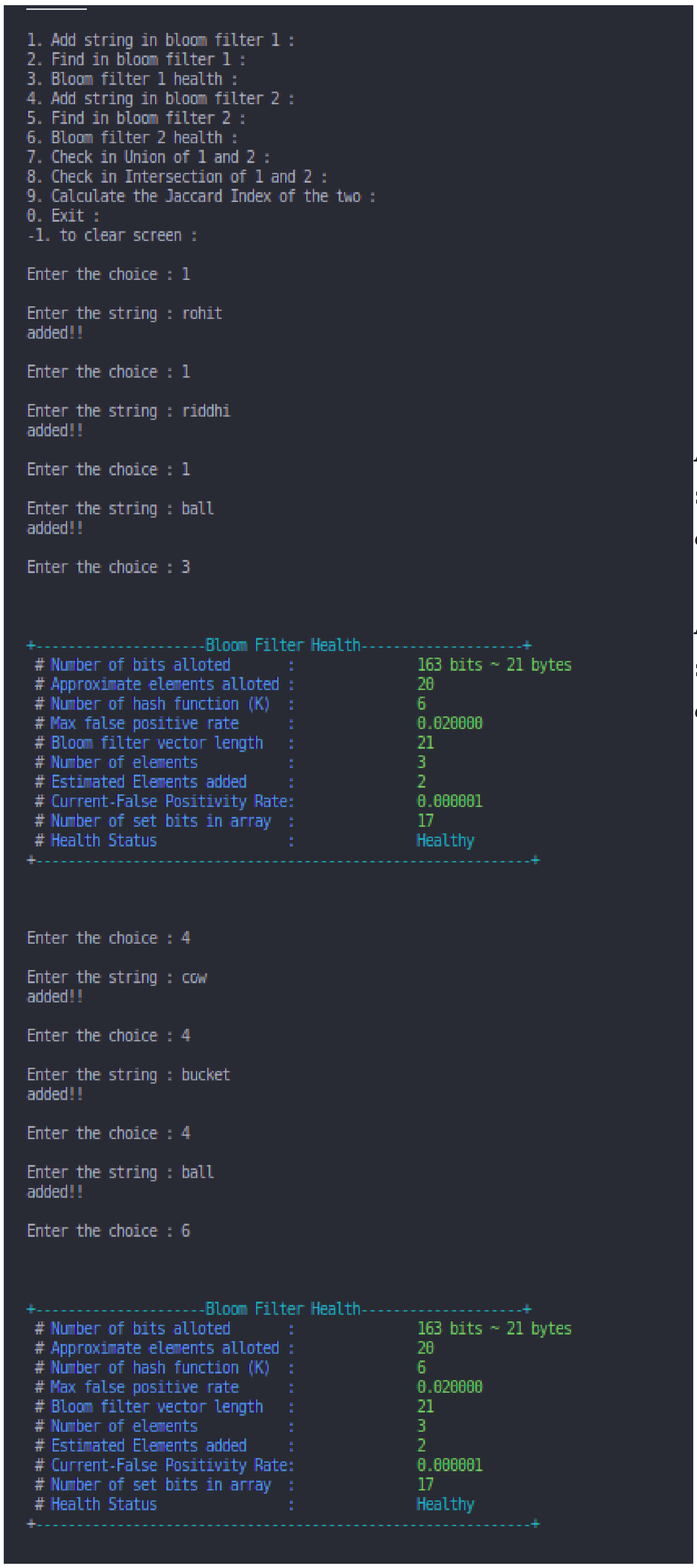
Screenshots



TERMINAL

```
rohit@ROHIT:~/Desktop/DP/DP_project$ ./a.out
Bloom filter Inputs :
Enter estimate : 20
Enter false positivity (0.0 < p < 1.0) : 0.02
Successfully !! initialized the bloom filter

1.insert a string in the bloom filter :
2.Look up a string in the bloom filter :
3.Show bloom filter health :
4.Reset the filter :
5.Clear Screen :
6.Exit :


NOTE : Reset the bloom filter if health becomes POOR
Enter the choice : 3



+--------------------Bloom Filter Health-------------------+
 # Number of bits alloted       :               163 bits ~ 21 bytes
 # Approximate elements alloted :               20
 # Number of hash function (K)  :               6
 # Max false positive rate      :               0.020000
 # Bloom filter vector length   :               21
 # Number of elements           :               0
 # Estimated Elements added     :               0
 # Current-False Positivity Rate:               0.000000
 # Number of set bits in array  :               0
 # Health Status                :               Healthy
+-------------------------------------------------------------+



Enter the choice : 1

Enter string to insert : rohit
added

Enter the choice : 1

Enter string to insert : riddhi
added

Enter the choice : 3



+--------------------Bloom Filter Health-------------------+
 # Number of bits alloted       :               163 bits ~ 21 bytes
 # Approximate elements alloted :               20
 # Number of hash function (K)  :               6
 # Max false positive rate      :               0.020000
 # Bloom filter vector length   :               21
 # Number of elements           :               2
 # Estimated Elements added     :               1
 # Current-False Positivity Rate:               0.000000
 # Number of set bits in array  :               11
 # Health Status                :               Healthy
+-------------------------------------------------------------+



Enter the choice : 2

Enter string to find in bloom filter : rohit
present !!
Enter the choice : 2

Enter string to find in bloom filter : riddhi
present !!
Enter the choice : 2

Enter string to find in bloom filter : sham
Not found !!

Enter the choice : 6
rohit@ROHIT:~/Desktop/DP/DP_project$
```

Screenshots (14)

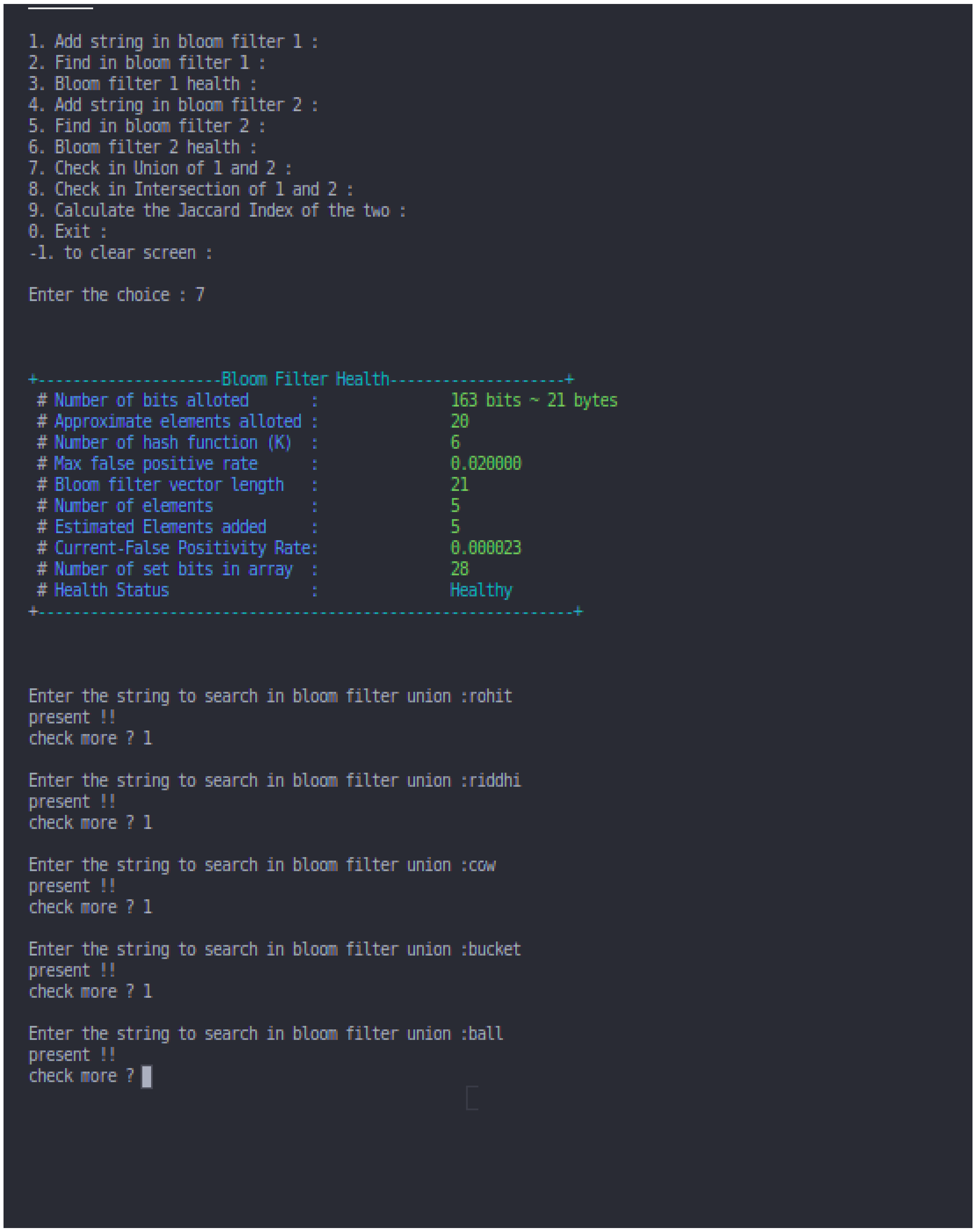


```
1. Add string in bloom filter 1 :
2. Find in bloom filter 1 :
3. Bloom filter 1 health :
4. Add string in bloom filter 2 :
5. Find in bloom filter 2 :
6. Bloom filter 2 health :
7. Check in Union of 1 and 2 :
8. Check in Intersection of 1 and 2 :
9. Calculate the Jaccard Index of the two :
0. Exit :
-1. to clear screen :

Enter the choice : 1

Enter the string : rohit
added!!

Enter the choice : 1

Enter the string : riddhi
added!!

Enter the choice : 1

Enter the string : ball
added!!

Enter the choice : 3



+--------------------Bloom Filter Health--------------------+
 # Number of bits alloted        :              163 bits ~ 21 bytes
 # Approximate elements alloted  :              20
 # Number of hash function (K)   :              6
 # Max false positive rate       :              0.020000
 # Bloom filter vector length    :              21
 # Number of elements            :              3
 # Estimated Elements added      :              2
 # Current-False Positivity Rate:               0.000001
 # Number of set bits in array   :              17
 # Health Status                 :              Healthy
+-------------------------------------------------------------+



Enter the choice : 4

Enter the string : cow
added!!

Enter the choice : 4

Enter the string : bucket
added!!

Enter the choice : 4

Enter the string : ball
added!!

Enter the choice : 6



+--------------------Bloom Filter Health--------------------+
 # Number of bits alloted        :              163 bits ~ 21 bytes
 # Approximate elements alloted  :              20
 # Number of hash function (K)   :              6
 # Max false positive rate       :              0.020000
 # Bloom filter vector length    :              21
 # Number of elements            :              3
 # Estimated Elements added      :              2
 # Current-False Positivity Rate:               0.000001
 # Number of set bits in array   :              17
 # Health Status                 :              Healthy
+-------------------------------------------------------------+
```

Added in bloomfilter1 = {“rohit” , “riddhi”, “ball”}

Added in bloomfilter2 = {“cow”, “bucket”, “ball”}

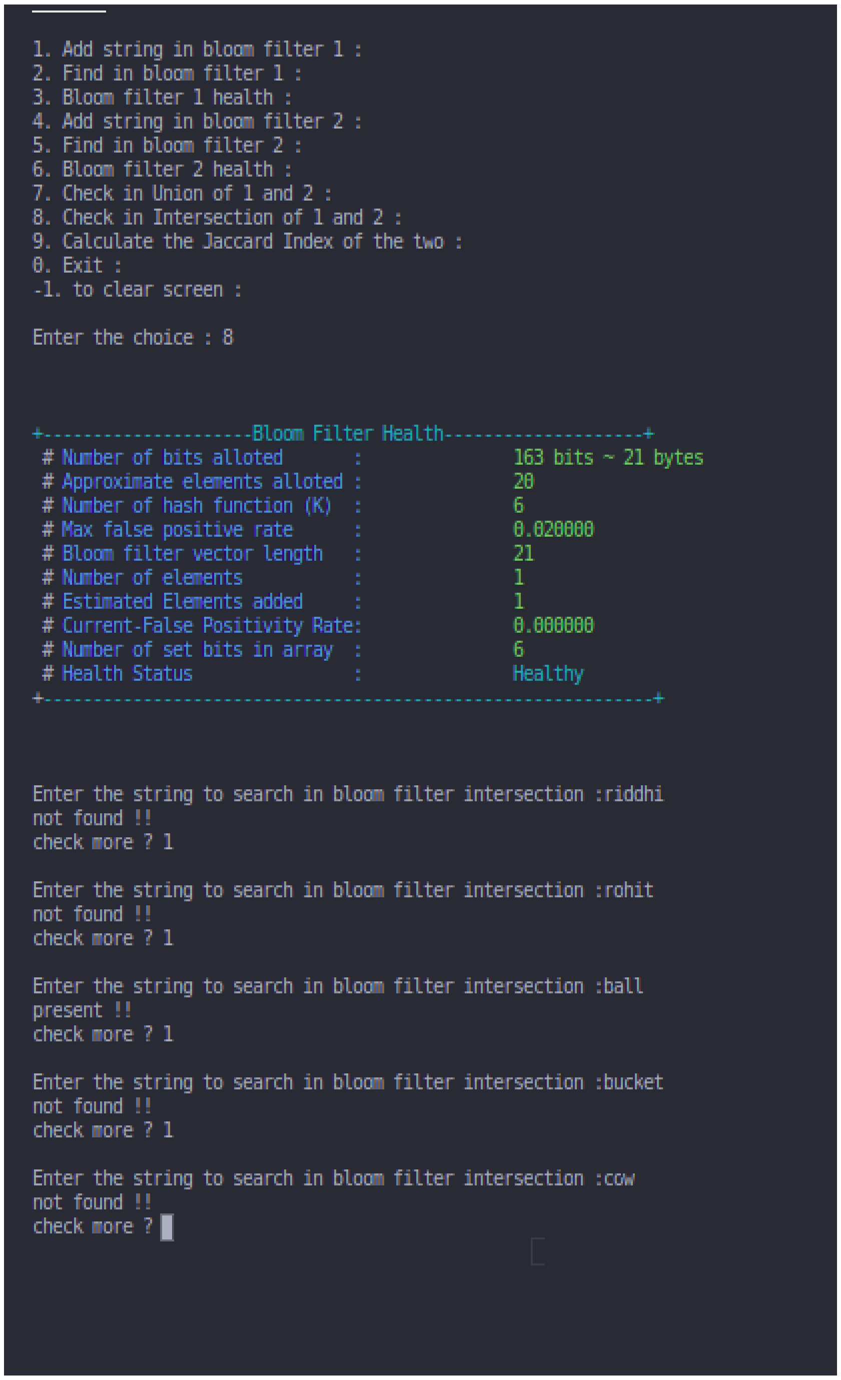
Union Screenshot (15)



```
1. Add string in bloom filter 1 :
2. Find in bloom filter 1 :
3. Bloom filter 1 health :
4. Add string in bloom filter 2 :
5. Find in bloom filter 2 :
6. Bloom filter 2 health :
7. Check in Union of 1 and 2 :
8. Check in Intersection of 1 and 2 :
9. Calculate the Jaccard Index of the two :
0. Exit :
-1. to clear screen :

Enter the choice : 7



+--------------------Bloom Filter Health-------------------+
 # Number of bits alloted        :              163 bits ~ 21 bytes
 # Approximate elements alloted :              20
 # Number of hash function (K)  :              6
 # Max false positive rate       :              0.020000
 # Bloom filter vector length    :              21
 # Number of elements            :              5
 # Estimated Elements added      :              5
 # Current-False Positivity Rate:              0.000023
 # Number of set bits in array   :              28
 # Health Status                 :              Healthy
+-------------------------------------------------------------+



Enter the string to search in bloom filter union :rohit
present !!
check more ? 1

Enter the string to search in bloom filter union :riddhi
present !!
check more ? 1

Enter the string to search in bloom filter union :cow
present !!
check more ? 1

Enter the string to search in bloom filter union :bucket
present !!
check more ? 1

Enter the string to search in bloom filter union :ball
present !!
check more ?
```

cat , ball , rohit, riddhi, cow all are present in bloomfilter union.

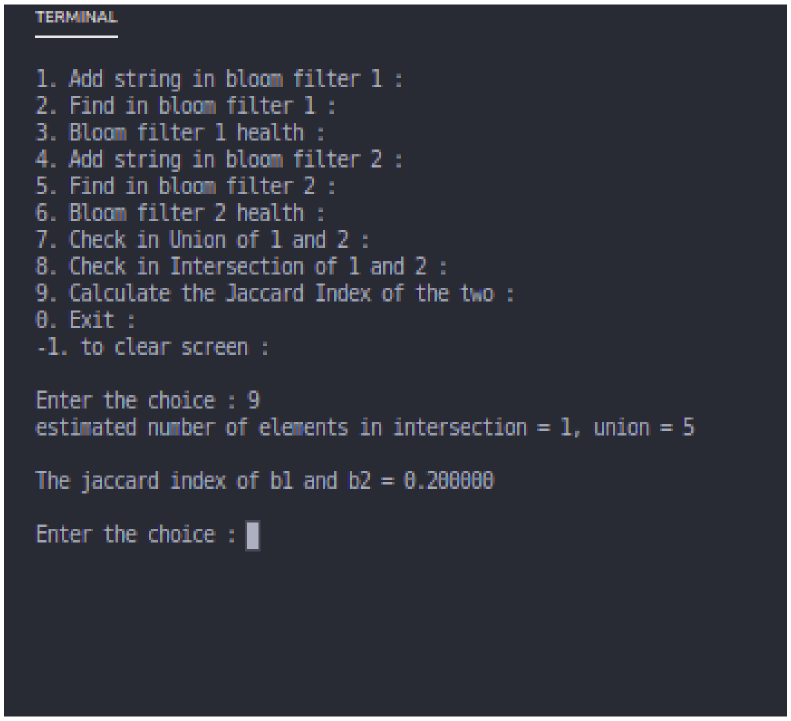
Intersection of bloom filters Screenshots (16)



```
1. Add string in bloom filter 1 :
2. Find in bloom filter 1 :
3. Bloom filter 1 health :
4. Add string in bloom filter 2 :
5. Find in bloom filter 2 :
6. Bloom filter 2 health :
7. Check in Union of 1 and 2 :
8. Check in Intersection of 1 and 2 :
9. Calculate the Jaccard Index of the two :
0. Exit :
-1. to clear screen :

Enter the choice : 8



+--------------------Bloom Filter Health-------------------+
 # Number of bits alloted       :              163 bits ~ 21 bytes
 # Approximate elements alloted :              20
 # Number of hash function (K)  :              6
 # Max false positive rate      :              0.020000
 # Bloom filter vector length   :              21
 # Number of elements           :              1
 # Estimated Elements added     :              1
 # Current-False Positivity Rate:              0.000000
 # Number of set bits in array  :              6
 # Health Status                :              Healthy
+----------------------------------------------------------+



Enter the string to search in bloom filter intersection :riddhi
not found !!
check more ? 1

Enter the string to search in bloom filter intersection :rohit
not found !!
check more ? 1

Enter the string to search in bloom filter intersection :ball
present !!
check more ? 1

Enter the string to search in bloom filter intersection :bucket
not found !!
check more ? 1

Enter the string to search in bloom filter intersection :cow
not found !!
check more ?
```

only common item ball is present in intersection of bloomfilters

Jaccard Index (17)



```
TERMINAL

1. Add string in bloom filter 1 :
2. Find in bloom filter 1 :
3. Bloom filter 1 health :
4. Add string in bloom filter 2 :
5. Find in bloom filter 2 :
6. Bloom filter 2 health :
7. Check in Union of 1 and 2 :
8. Check in Intersection of 1 and 2 :
9. Calculate the Jaccard Index of the two :
0. Exit :
-1. to clear screen :

Enter the choice : 9
estimated number of elements in intersection = 1, union = 5

The jaccard index of b1 and b2 = 0.200000

Enter the choice :
```

Shows the jaccard Index of the bloom filter (b1, b2) took initially .

REFERENCES AND BIBLIOGRAPHY

https://learning.oreilly.com/library/view/mapreduce-design-patterns/9781449341954/apa.html

https://en.wikipedia.org/wiki/Bloom_filter

https://www.hindawi.com/journals/scn/2018/5841967/

https://blog.demofox.org/2015/02/08/estimating-set-membership-with-a-bloom-filter/

https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff

https://www.youtube.com/watch?v=bgzUdBVr5tE
Gaurav Sen

https://www.youtube.com/watch?v=H1tFqoY37Wc
Ian Barland

https://web.stanford.edu/~balaji/papers/bloom.pdf