



Vidyavardhini's College of Engineering &
Technology

Department of Computer Engineering

Experiment No. 5
Data Stream Algorithms: Implement Bloom filter algorithm using
any programming language
Date of Performance: 14/09/2023
Date of Submission: 05/10/2023



Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Aim:

Data Stream Algorithms: Implement bloom filter algorithm using any programming language

Theory:

Bloom filter algorithm approximates the number of unique objects in a stream or a database in one pass. If the stream contains n elements with m of them unique, this algorithm runs in $O(n)$ time and needs $O(\log(m))$ memory.

Algorithm:

1. Create a bit vector (bit array) of sufficient length L , such that $2L > n$, the number of elements in the stream. Usually a 64-bit vector is sufficient since 264 is quite large for most purposes.
2. The i -th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in $0i$. So initialize each bit to
3. The i -th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in $0i$. So initialize each bit to
4. The i -th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in $0i$. So initialize each bit to
5. Once input is exhausted, get the index of the first 0 in the bit array (call this R). By the way, this is just the number of consecutive 1s (i.e. we have seen $0, 00, \dots, 0R-1$ as the output of the hash function) plus one.
6. Calculate the number of unique words as $2R/\phi$, where ϕ is 0.77351. A proof for this can be found in the original paper listed in the reference section.
7. The standard deviation of R is a constant: $\sigma(R) = 1.12$. (In other words, R can be off by about 1 for $1 - 0.68 = 32\%$ of the observations, off by 2 for about $1 - 0.95 = 5\%$ of the observations, off by 3 for $1 - 0.997 = 0.3\%$ of the observations using the Empirical rule of statistics). This implies that our count can be off by a factor of 2 for 32% of the observations, off by a factor of 4 for 5% of the observations, off by a factor of 8 for 0.3% of the observations and so on.



Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Code:

```
n = 20 #no of items to add
p = 0.05 #false positive probability

bloomf = BloomFilter(n,p)
print("Size of bit array: {}".format(bloomf.size))
print("False positive Probability: {}".format(bloomf.fp_prob))
print("Number of hash functions: {}".format(bloomf.hash_count))
# words to be added
word_present = ['abound','abounds','abundance','abundant','accessible',
                'bloom','blossom','bolster','bonny','bonus','bonuses',
                'coherent','cohesive','colorful','comely','comfort',
                'gems','generosity','generous','generously','genial']

# word not added
word_absent = ['bluff','cheater','hate','war','humanity',
               'racism','hurt','nuke','gloomy','facebook',
               'geeksforgeeks','twitter']

for item in word_present:
    bloomf.add(item)

shuffle(word_present) shuffle(word_absent)
test_words = word_present[:10] + word_absent
shuffle(test_words) for word in test_words:
    if bloomf.check(word):
        if word in word_absent:
            print("{} is a false positive!".format(word)) else:
            print("{} is probably present!".format(word))
    else:
        print("{} is definitely not present!".format(word))
```



Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Output:

```
ubuntu@ubuntu-HP-Elite-Tower-600-G9-Desktop-PC:~/bloomfilter$ python3 bloom_test.py
Size of bit array:124
False positive Probability:0.05
Number of hash functions:4
'gloomy' is definitely not present!
'cohesive' is probably present!
'geeksforgeeks' is definitely not present!
'hate' is definitely not present!
'bluff' is definitely not present!
'abundant' is probably present!
'nuke' is definitely not present!
'twitter' is a false positive!
'cheater' is definitely not present!
'accessible' is probably present!
'bonus' is probably present!
'generosity' is probably present!
'comely' is probably present!
'genial' is probably present!
'humanity' is a false positive!
'comfort' is probably present!
'war' is definitely not present!
'generous' is probably present!
'bolster' is probably present!
'facebook' is definitely not present!
'hurt' is definitely not present!
'racism' is definitely not present!
ubuntu@ubuntu-HP-Elite-Tower-600-G9-Desktop-PC:~/bloomfilter$
```

Conclusion:

The execution of this experiment proved the successful implementation of the Bloom filter algorithm through a programming language, illustrating its efficiency in estimating the count of distinct items in a data stream. The Bloom filter's mechanism relies on a combination of a bit array and hash functions to probabilistically determine the presence or absence of an element within the stream. The provided code demonstrated the creation of a Bloom filter while specifying a desired false positive probability. Subsequently, it was tested with a list of words, some of which were confirmed to be present while others were known to be absent. This experiment underscores the practical utility of Bloom filters in scenarios necessitating swift membership assessments and approximations of item cardinality, while also recognizing the occasional occurrence of false positives, which stem from the probabilistic nature of the algorithm.