Project – 2 : Bloom Filters

The following project contains a zip file, which consists of 3 source code files(java) which contains the logic for implementing bloom filter, counting bloom filter and coded bloom filter algorithms and corresponding output text files.

Below are the steps which guides to run the project on command line to the functionality of the algorithms mentioned above.
1) Launch the terminal application and go to the "src" folder of the project.
2) Begin by executing the following commands: "javac P2_BloomFilter.java" and "java P2_BloomFilter ", "javac P2_CountingBloomFilter.java" and "java P2_CountingBloomFilter ", "javac P2_CodedBloomFilter.java" and "java P2_CodedBloomFilter ",
3) Executing these commands will result in the creation of output files in the "src" directory and, if those files already exist, will replace them.

1. **bloomFilter.java:**

   A data structure known as a Bloom filter is one that employs hashing and is predicated on probabilities. It is often used to either add items to an existing set or check to see whether an item is already part of an existing set. It is highly space efficient. However, you do not add the individual pieces to the set individually. Instead, a hash of the items is added to the set where they originally belonged. In the same way that a hash table employs a hash function to link a key to a bucket, a bloom filter does the same thing to connect a key to a bucket. On the other hand, it will not place the key in that bucket. Instead, it will only indicate that it is full when it is done. Therefore, there is a possibility that many keys would match up with the same full bucket, which would be an example of a false positive.

2. **countingBloomFilter.java:**
   Bloom Filters and Counting Bloom Filters produce the same effects, although the former uses counters instead of numbers (0 or 1). Bloom Filters don't enable us to delete sections of the data structure.
   Counting bloom filters need m slots, each of which has a counter. All slots in an empty data structure are set to 0. Counting Bloom Filter begins with a preset-size array (m). It utilizes the same space every time (1). As it grows, it will need more room.

3. **codedBloomFilter.java:**
   Code Bloom Filters are used for performing the Bloom filters for the sets of elements which employs the logarithmic number of implementations of the Bloom filters to hash and store the values.

This project is tested out on both IntelliJ and command line for execution, and below are the screenshots for the outputs. Projec starts when above commands is run and contains the calls to the methods to execute bloomFilter, countingBloomFilter and codedBloomFilter.

Output Screenshots:

1. **Bloom Filter:**

```
(base) jaswanth@jaswanths-MacBook-Air Project2 % javac P2_BloomFilter.java
(base) jaswanth@jaswanths-MacBook-Air Project2 % java P2_BloomFilter
Array A: Value of the Lookup Count : 1000
Array B: Value of the Lookup Count : 8
```

```
(base) jaswanth@jaswanths-MacBook-Air Project2 % cat P2_BloomFilterOutput.txt
Array A: Value of the Lookup Count : 1000
Array B: Value of the Lookup Count : 8
```

2. Counting Bloom Filter:

```
(base) jaswanth@jaswanths-MacBook-Air Project2 % javac P2_CountingBloomFilter.java
(base) jaswanth@jaswanths-MacBook-Air Project2 % java P2_CountingBloomFilter
Value of Lookup Count for the Array: 506
```

```
(base) jaswanth@jaswanths-MacBook-Air Project2 % cat P2_CountingBloomFilterOutput.txt
Value of Lookup Count for the Array: 506
(base) jaswanth@jaswanths-MacBook-Air Project2 %
```

3. Coded Bloom Filter:

```
(base) jaswanth@jaswanths-MacBook-Air Project2 % javac P2_CodedBloomFilter.java
(base) jaswanth@jaswanths-MacBook-Air Project2 % java P2_CodedBloomFilter
Value of Lookup Count for the Array: 6852
(base) jaswanth@jaswanths-MacBook-Air Project2 % cat P2_CodedBloomFilterOutput.txt
Value of Lookup Count for the Array: 6852
(base) jaswanth@jaswanths-MacBook-Air Project2 %
```