**Project 4**

**Venkata Sai Karthik Metlapalli**

**UFID: 65764476**

**Description**

* This project involves java-based implementation of Bitmap, Probabilistic bitmap (Bitmap with sampling), and Hyper log log sketches.
* The zipped folder includes the source codes and the output text files generated for each of the algorithms

1. **Bitmap:**

* This Bitmap is implemented in JAVA and the class file name is Bitmap.java.
* To run this file, go to the IDS-4/src directory, then type the following commands in command prompt/ terminal
  + - javac Bitmap.java
    - java Bitmap <bitMapSize>

This program accepts the size of the bitmap array as input parameters through command line arguments. For the demo the values of command line arguments passed is 10000.

In case of wrong input parameters, the program throws valid exception messages to the user suggesting to correct the input.

* The Bitmap class consists of a 1D bitMap, bitmapSize and maps to store the flowId and its flow elements, actual spread of every flow, calculated estimated spread after recording and querying. These are stored in the flowValues, actualSize, estimatedSize hashMaps respectively.
* After passing the inputs, the parametrized constructor initializes the bitmap of the passed size whose initial values are all zeros, and unique random numbers of given spread sizes are generated and placed in the flowValues using generateUniqueValues() function which makes use of random() function in java.
* Next in the recordAndQuery() function, we generate a random number and for each flowID, we xor each unique element with this number. Then take modulo wrt bitmapSize and set the bit at this location to 1 i.e., Bitmap[randomNo ^ element] % bitmapSize = 1.
* In the query part of each flow, we calculate the no of places in the bitmap that are still zeros. Then, the spread is estimated using -1 \* bitMapSize \* ln(no of zeros/ bitmapSize) and is stored in estimatedSizeMap.
* Then all the bits in bitmap are reset to zeros and recording and query is done for next flows.
* All the actualSize and estimatedSize values are outputted in the newOutputBitmap.txt file.
* **NOTE:** For larger spread values, the entire bitmap values are set to 1 and hence the fraction of zeros = 0; which causes ln value to be undefined since ln(0) is undefined and hence the program outputs INTEGER.MAX\_VALUE for such cases.

1. **Probabilistic Bitmap:**

* This ProbabilisticBitmap is implemented in JAVA and the class file name is ProbabilisticBitmap.java.
* To run this file, go to the IDS-4/src directory, then type the following commands in command prompt/ terminal
  + - javac ProbabilisticBitmap.java
    - java ProbabilisticBitmap <bitMapSize>

This program accepts the size of the bitmap array and sampling probability as input parameters through command line arguments. For the demo the values of command line arguments passed are 10000 and 0.1.

In case of wrong input parameters, the program throws valid exception messages to the user suggesting to correct the input.

* The ProbabilisticBitmap class consists of a 1D bitMap, bitmapSize, sampling probability and maps to store the flowId and its flow elements, actual spread of every flow, calculated estimated spread after recording and querying. These are stored in the flowValues, actualSize, estimatedSize hashMaps respectively.
* After passing the inputs, the parametrized constructor initializes the bitmap of the passed size whose initial values are all zeros, and unique random numbers of given spread sizes are generated and placed in the flowValues using generateUniqueValues() function which makes use of random() function in java.
* Next in the recordAndQuery() function, we generate a random number. For each flowID, we find the maximum hashvalue generated. Then we xor each unique element of the flow with the above generated random number. If the xored value is less than sampling probability \* maxhashvalue, then we record it in bitmap by using another hash function. This is generated by xor of element with another random number and setting corresponding bit in hashmap. So, we are recording with some probability and set Bitmap[randomNo2 ^ element] % bitmapSize = 1
* In the query part of each flow, we calculate the no of places in the bitmap that are still zeros. Then, the spread is estimated using -1 \* bitMapSize \* ln(no of zeros/ bitmapSize) and is stored in estimatedSizeMap.
* Then all the bits in bitmap are reset to zeros and recording and query is done for next flows.
* All the actualSize and estimatedSize values are outputted in the newOutputProbabilisticBitmap.txt file.

**NOTE:** For larger spread values, the entire bitmap values are set to 1 and hence the fraction of zeros = 0; which causes ln value to be undefined since ln(0) is undefined and hence the program outputs INTEGER.MAX\_VALUE for such cases. This estimates spread values better than normal bitmap for larger values since we don’t record all elements rather do based on probability, hence we set lesser bits to 1 reducing the occurance of ln(0) (all ones in bitmap).

1. **Hyper Log Log sketch:**

* This Hyperloglog sketch is implemented in JAVA and the class file name is HyperLogLogSketch.java.
* To run this file, go to the IDS-4/src directory, then type the following commands in command prompt/ terminal
  + - javac HyperLogLogSketch.java
    - java HyperLogLogSketch <numberOfRegisters>

This program accepts the number of the registers arrays as input parameters through command line arguments. For the demo the values of command line arguments passed are 256 and each register size is 5 bits.

In case of wrong input parameters, the program throws valid exception messages to the user suggesting to correct the input.

* The HyperLogLogSketch class consists of a registerSize, numberOfRegisters(m) and maps to store the flowId and its flow elements, actual spread of every flow, calculated estimated spread after recording and querying. These are stored in the flowValues, actualSize, estimatedSize hashMaps respectively.
* After passing the inputs, the parametrized constructor initializes the m registers of the given size whose initial values are all zeros, and unique random numbers of given spread sizes are generated and placed in the flowValues using generateUniqueValues() function which makes use of random() function in java.
* Next in the recordAndQuery() function, we generate a random number. For each flowID, we we xor each unique element of the flow with the above generated random number. We also calculate G’(element) which is a geometric hash by finding the number of leading zeros in the element’s binary representation and adding 1 to it. We store in bitmap[H(e)], the greater value among G’(e) and bitmap[H(e)].
* In the query part of each flow, we calculate the alpha value as 0.7213/ (1 + 1.079 / numberOfRegisters). And final estimatedSize of flow = (alpha \* numberOf registers^2 \* (1 / estimatedVal); where estimatesVal = sigma (1/ 2 ^ bitmap[i]) for all 0<= i < No.of bitmaps
* Then all the bits in bitmap are reset to zeros and recording and query is done for next flows.
* All the actualSize and estimatedSize values are outputted in the newHyperLogLogSketch.txt file.