**IDS Project 4**

**Bitmap**

Here bitmap is being used to check the spread of each flow in a single flow spread.

When we receive a flow, we hash its value to an index in the bitmap and set it to 1. Ideally the number of 1’s would be the number of unique elements in the flow, which is nothing but the spread. But there are always chance of hash collisions so we take several parameters into consideration to formulate the flow spreads estimated value.

*Implementation:*

We have used a very simple approach to deal with the bitmap.

* In the class bitmap we have declared the size m of bitmap globally and the bitmap itself as an array of size m, and an array flow which holds all the elements related to one flow. We also have our random object to call generate random numbers.
* When we take a flow of given spread size, we invoke the constructor with that size and then create the flow array of that size. We fill it with unique random positive values and then perform Hashing and record the elements by encoding bitmap B[index] to 1.
* We make sure that the bitmap is filled with zeroes by using inbuilt method arrays.fill() and set them to zeroes before hashing
* We use a method fillRandomElements() which takes in an array as argument to generate unique positive values.
* For querying we use estimatedSpread() which first calculates the number of zeroes in bitmap and then substitute in the formula discussed in class for the estimated spread value.
* **We have to keep in mind the case of saturated bitmaps where all the bits are set to 1 which means that the number of zeroes will be 0, that is our u=0, there by leading our log value to infinity.**
* **In these cases, I am returning the estimated count to the Maximum value an Integer can have that is 2147483647** **because when we perform the substitution it will automatically assign this value to our variable because it is an integer and that is the maximum value it can store.**
* We store the output of this progam in a text file BitmapOutput.txt

For ease of flow, we have clubbed several operations but we can also split the methods for recording and querying.

Throughout our project we used Math.abs to make sure the random numbers generated are made positive only.

**ProbabilisticBitmap**

Here bitmap is being used to check the spread of each flow in a single flow spread.

When we receive a flow, we hash its value and then store it with a sampling probability p given and then hash again to an index in the bitmap and set it to 1. Ideally the number of 1’s would be the number of unique elements in the flow, which is nothing but the spread. But there are always chance of hash collisions so we take several parameters into consideration to formulate the flow spreads estimated value.

*Implementation:*

* In the class ProbabilisticBitmap we have declared the size m of bitmap globally and the bitmap itself as an array of size m, the sampling probability p given, and an array flow which holds all the elements related to one flow. We also have our random object to call generate random numbers.
* When we take a flow of given spread size, we initialize the constructor with that size and then create the flow array of that size. We fill it with unique random positive values. We take two different hashes, and an integer x which is the maximum hash value, for sake of sampling. We make sure that the bitmap is filled with zeroes by using inbuilt method arrays.fill() and set them to zeroes before hashing
* To perform recording we first hash the element H’(e) and check with the sampling probability formula and then sample it using it (H’(e)<x\*p) and it is satisfied we hash element e H(e) and record the elements by encoding bitmap B[index] to 1.
* We use a method fillRandomElements() which takes in an array as argument to generate unique positive values.
* For querying we use estimatedSpread() which first calculates the number of zeroes in bitmap and then substitute in the formula discussed in class for the estimated spread value.
* **We have to keep in mind the case of saturated bitmaps where all the bits are set to 1 which means that the number of zeroes will be 0, that is our u=0, there by leading our log value to infinity.**
* **In these cases, I am returning the estimated count to the Maximum value an Integer can have that is 2147483647** **because when we perform the substitution it will automatically assign this value to our variable because it is an integer and that is the maximum value it can store.**
* We store the output of this progam in a text file ProbabilisticBitmapOutput.txt

For ease of flow, we have clubbed several operations but we can also split the methods for recording and querying.

**HyperLogLog**

Here bitmap with registers is being used to check the spread of each flow in a single flow spread.

When we receive a flow, we hash its value and the perform geometric hashing whose value is the number of leading zeroes in hash value generated and then decide which register to store it in first. The recoding is slightly different as we store the bitmap value itself as maximum of uniform hash and geometric hash. But there are always chance of hash collisions so we take several parameters into consideration to formulate the flow spreads estimated value.

*Implementation:*

* In the class Hyper Log Log sketch, we have declared the size m of bitmap globally and the bitmap itself as an array of size m, and an array flow which holds all the elements related to one flow. We also have our random object to call generate random numbers.
* When we take a flow of given spread size, we initialize the constructor with that size and then create the flow array of that size. We fill it with unique random positive values. We generate hash and we make sure that the bitmap is filled with zeroes by using inbuilt method arrays. Fill() and set them to zeroes before hashing.
* To perform recording we first hash the element H(e) and get its value and also find the geometric hash G’(e) and record the elements by encoding bitmap B[index] to the maximum{B[index], G’(e)}.
* We use a method fill Random Elements() which takes in an array as argument to generate unique positive values.
* We use method G which takes in integer and returns its number of leading zeroes in binary form of given number.
* For querying we use estimated Spread() which first calculates the constant alpha from given formula and then , the harmonic sum of all the given bitmap values and the substitute in the formula discussed in class for the estimated spread value.
* **Keep in mind that Hyper Log log value is not accurate when the flow spread is small.**
* We store the output of this program in a text file HLLOutput.txt

For ease of flow, we have clubbed several operations but we can also split the methods for recording and querying.