# DS 702 – Assignment 3

**Muhammad Umar Salman**

1. Sampling Data in a Stream

   a) For each university, estimate the average number of students in a course

   We need to stream 1/20th of the original stream data and answer the query based on the sample. To create a representative sample of the streaming data we will use a hash function to generate random integers between 0 and 19 (inclusive).

   Since we have to estimate average number of students in each course, we will identify the tuple by using a composite <u>key of (University, courseID)</u>. We will then hash the incoming tuples based on this composite key to random integers in 20 buckets. We will then only accept those hash values which are 0 to store 1/20th of the sample and discard the others. Then for each university and for each course we can calculate the average number of students in a course.

   b) Estimate the fraction of students who have a GPA of 3.5 or more

   We need to stream 1/20th of the original stream data and answer the query based on the sample. To create a representative sample of the streaming data we will use a hash function to generate random integers between 0 and 19 (inclusive).

   Since we have to estimate students with a GPA of 3.5 or more, we will identify the tuple by using a composite <u>key of (University, studentID)</u>. We will then hash the incoming tuples based on this composite key to random integers in 20 buckets. We will then only accept those hash values which are 0 to store 1/20th of the sample and discard the others. Then for students we can calculate the estimate of the fraction of students with a GPA of 3.5 or higher.

   c) Estimate the fraction of courses where at least half the students got "A."

   We need to stream 1/20th of the original stream data and answer the query based on the sample. To create a representative sample of the streaming data we will use a hash function to generate random integers between 0 and 19 (inclusive).

   Since we have to estimate the fraction of courses where at least half the students got an A grade, we will identify the tuple by using a composite <u>key of (University, courseID)</u>, we won't go with a composite key of (University, courseID, studentID) because the question clearly states that different universities assign same studentID's to different students but since we have University as our key we won't face that issue. We will then hash the incoming tuples based on this composite key to random integers in 20 buckets. We will then only accept those hash values which are 0 to store 1/20th of the sample and discard the others. Then for courses we can calculate the estimate of the fraction of courses where half the students got an A.

2. Filtering Streams

Exercise 1)

For the situation of the running example in the book (8 billion bits, 1 billion members of the set S) [Section 4.3.1][1], calculate the false-positive rate if we use three hash functions? What if we use four hash functions?

Ans) With 3 hash functions

$(1 – e^{-km/n})^{k}$

$(1 – e^{-3.1/8})^{3} = 0.0305$

With 4 hash functions

$(1 – e^{-4.1/8})^{4} = 0.0239$

Exercise 2)

As a function of n, the number of bits and m the number of members in the set S, what number of hash functions minimizes the false positive rate?

Derivate $(1 – e^{-km/n})^{k}$ with respect to k and equate to 0.
Let k be the number of hash functions
K = n/m. ln (2)

3. Distinct Elements

Exercise 3.1: Suppose our stream consists of the integers 3, 1, 4, 1, 5, 9, 2, 6, 5. Our hash functions will all be of the form h(x) = ax+ b mod 32 for some a and b. You should treat the result as a 5-bit binary integer. Determine the tail length for each stream element and the resulting estimate of the number of distinct elements if the hash function is:

(a) h(x) = 2x + 1 mod 32.

(b) h(x) = 3x + 7 mod 32.

(c) h(x) = 4x mod 32.

## HASH FUNCTION (A)

**h(x) = 2x + 1 mod 32**

| **Element** | **Hashed Value** | **Binary** | **Tail Length (R)** | **# of Distinct Elements (2^R)** |
|---|---|---|---|---|
| 3 | 7 | 00111 | 0 | 1 |
| 1 | 3 | 00011 | 0 | 1 |
| 4 | 9 | 01001 | 0 | 1 |
| 1 | 3 | 00011 | 0 | 1 |
| 5 | 11 | 01011 | 0 | 1 |
| 9 | 19 | 10011 | 0 | 1 |
| 2 | 5 | 00101 | 0 | 1 |
| 6 | 13 | 01101 | 0 | 1 |
| 5 | 11 | 01011 | 0 | 1 |
| | | | | Average: 1 |
| | | | | Max: 1 |

## HASH FUNCTION (B)

**h(x) = 3x + 7 mod 32**

| **Element** | **Hashed Value** | **Binary** | **Tail Length (R)** | **# of Distinct Elements (2^R)** |
|---|---|---|---|---|
| 3 | 16 | 10000 | 4 | 16 |
| 1 | 10 | 01010 | 1 | 2 |
| 4 | 19 | 10011 | 0 | 1 |
| 1 | 10 | 01010 | 1 | 2 |
| 5 | 22 | 10110 | 1 | 2 |
| 9 | 2 | 00010 | 1 | 2 |
| 2 | 13 | 01101 | 0 | 1 |
| 6 | 25 | 11001 | 0 | 1 |
| 5 | 22 | 10110 | 1 | 2 |
| | | | | Average: 3.222222222 |
| | | | | Max: 16 |

## HASH FUNCTION (C)

**h(x) = 4x mod 32**

| **Element** | **Hashed Value** | **Binary** | **Tail Length (R)** | **# of Distinct Elements (2^R)** |
|---|---|---|---|---|
| 3 | 12 | 01100 | 2 | 4 |
| 1 | 4 | 00100 | 2 | 4 |
| 4 | 16 | 10000 | 4 | 16 |
| 1 | 4 | 00100 | 2 | 4 |
| 5 | 20 | 10100 | 2 | 4 |
| 9 | 4 | 00100 | 2 | 4 |
| 2 | 8 | 01000 | 3 | 8 |
| 6 | 24 | 11000 | 3 | 8 |
| 5 | 20 | 10100 | 2 | 4 |
| | | | | Average: 6.222222222 |
| | | | | Max: 16 |

| | **Max of 2^R** | **Average of 2^R** |
|---|---|---|
| **Hash Func A** | **1** | **1** |
| **Hash Func B** | **16** | **3.2222** |
| **Hash Func C** | **16** | **6.2222** |
| | | |
| **Median Value** | **16** | **3.2222 ~ 3** |

Here we have found two different methods of calculating the estimate of the number of distinct elements

1. In the first method, we take different hash functions and run them on the streaming values. The hashed values are then converted to binary values. For each binary value we look at the trailing zeros count or the tail length (r(x)). For all counts we then take the estimate count of distinct elements which is 2^R. In method 1, we take the average across all these 2^R values for each hash group. Then we sort these averages and take the median average value which in our example comes out to be 3.222 or rounded off to 3.

2. In the second method, we take different hash functions and run them on the streaming values. The hashed values are then converted to binary values. For each binary value we look at the trailing zeros count or the tail length (r(x)). For all counts we then take the estimate count of distinct elements which is 2^R. In method 2, instead of taking the average of all the 2^R values we take the max of all tail lengths for each hash group. We then again sort them and take the median of the max 2^R of each hash group and that will be our estimate which <u>in our case is 16.</u>

Exercise 3.2: Do you see any problems with the choice of hash functions in Exercise 3.1? What advice could you give someone who was going to use a hash function of the form h(x) = ax + b mod 2^k?

While calculating the estimate of the count of distinct elements we can see that our algorithm depends on the hash function parameters we use. So for someone who is using the form h(x) = ax + b mod $2^k$ firstly, the length of the hash-bit string must be sufficient that there are more possible hash function results than are the elements of the universal set. Secondly, the parameter a should be an odd number if we are using $2^k$ as our modulus values so that we avoid collisions. We can see this in our example h(x) = 4x mod 32 that elements 1 and 9 both hash to value 4 (shown in figure above) causing a collision. So, for good hash results with varying trailing lengths we would also prefer a to be odd regardless of the value of b.

## 4. Counting Ones in Windows

Exercise 4.1: Suppose the window is as shown in Fig. 1. Estimate the number of 1's the last k positions, for k = (a) 5 (b) 15. In each case, how far off the correct value is your estimate?

a) For K = 5:

The bucket that overlaps with k=5 is the 3rd bucket from the right. Thus, the estimate would be the sum of all the previous buckets plus half of the overlapping bucket. That is:

Estimate = 1 + 1 + (0.5*2)
Estimate = 3

The actual number of 1's for the last 5 positions is also 3 so we are exactly on spot.

b) For K = 15:

The bucket that overlaps with k=15 is the 5th bucket from the right. Thus, the estimate would be the sum of all the previous buckets plus half of the overlapping bucket. That is:

Estimate = 1 + 1 + 2 + 4 + (0.5*4)
Estimate = 10

(1/9) * 100 = 11%

The actual number of 1's for the last 5 positions is 9 and we estimated 10. Thus, we are only 11% far off from the correct value.

Exercise 4.2: There are several ways that the bit-stream 1001011011101 could be partitioned into buckets. Find all of them.

There are 2 ways that the bit stream can be partitioned into buckets:

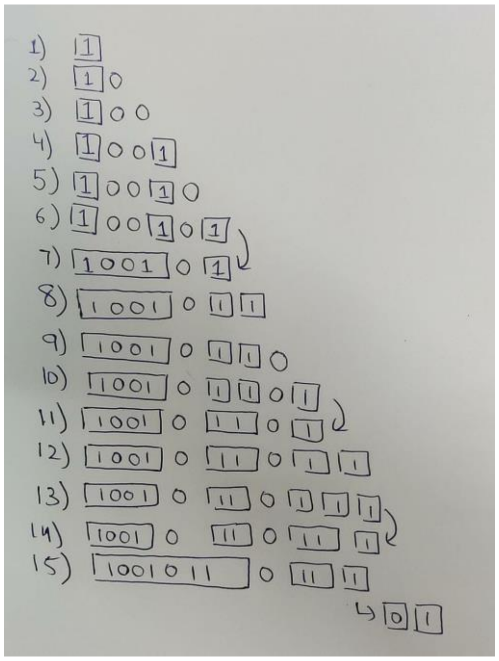
Exponential Windows

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

DGIM Method

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

DGIM



Exponential