# Stream Algorithms

**Question 1**: We wish to estimate the surprise number (2nd moment) of a data stream, using the method of AMS. It happens that our stream consists of ten different values, which we'll call 1, 2,..., 10, that cycle repeatedly. That is, at timestamps 1 through 10, the element of the stream equals the timestamp, at timestamps 11 through 20, the element is the timestamp minus 10, and so on. It is now timestamp 75, and a 5 has just been read from the stream. As a start, you should calculate the surprise number for this time.

For our estimate of the surprise number, we shall choose three timestamps at random, and estimate the surprise number from each, using the AMS approach (length of the stream times 2*m*-1, where *m* is the number of occurrences of the element of the stream at that timestamp, considering all times from that timestamp on, to the current time). Then, our estimate will be the median of the three resulting values.

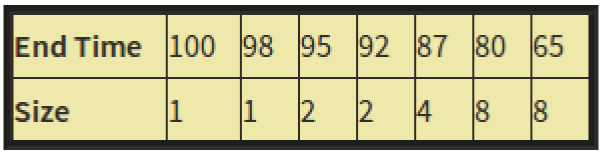
You should discover the simple rules that determine the estimate derived from any given timestamp and from any set of three timestamps. Then, take any 4 examples of the set of three "random" timestamps, find out the closest estimate among the 4 examples.

n = stream length

a = value of stream at any timestamp (random time) to start

X = n * (2 * (number of a’s in the stream till that timestamp) - 1)

**Question 2**: Suppose we are using the DGIM algorithm of Section 4.6.2 to estimate the number of 1's in suffixes of a sliding window of length 40. The current timestamp is 100, and we have the following buckets stored:



| **End Time** | 100 | 98 | 95 | 92 | 87 | 80 | 65 |
|---|---|---|---|---|---|---|---|
| **Size** | 1 | 1 | 2 | 2 | 4 | 8 | 8 |

Note: we are showing timestamps as absolute values, rather than modulo the window size, as DGIM would do.

Suppose that at times 101 through 105, 1's appear in the stream. Compute the set of buckets that would exist in the system at time 105. Buckets are represented by pairs (end-time, size).

②

| | | | | | | |
|---|---|---|---|---|---|---|
| 100 | 98 | 95 | 92 | 87 | 80 | 65 |
| 1 | 1 | 2 | 2 | 4 | 8 | 8 |

| |
|---|
| 101 |
| 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 100 | 98 | 95 | 92 | 87 | 80 | 65 |
| 1 | 1 | 2 | 2 | 4 | 8 | 8 |

| | | | | |
|---|---|---|---|---|
| 100 | 95 | 87 | 80 | 65 |
| 2 | 4 | 4 | 8 | 8 |

| |
|---|
| 102 |
| 1 |

| | | | | | |
|---|---|---|---|---|---|
| 101 | 100 | 95 | 87 | 80 | 65 |
| 1 | 2 | 4 | 4 | 8 | 8 |

| |
|---|
| 103 |
| 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 102 | 101 | 100 | 95 | 87 | 80 | 65 |
| 1 | 1 | 2 | 4 | 4 | 8 | 8 |

| | | | | | |
|---|---|---|---|---|---|
| 102 | 100 | 95 | 87 | 80 | 65 |
| 2 | 2 | 4 | 4 | 8 | 8 |

| |
|---|
| 104 |
| 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 103 | 102 | 100 | 95 | 87 | 80 | 65 |
| 1 | 2 | 2 | 4 | 4 | 8 | 8 |

| |
|---|
| 105 |
| 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 104 | 103 | 102 | 100 | 95 | 87 | 80 | 65 |
| 1 | 1 | 2 | 2 | 4 | 4 | 8 | 8 |

| | | | |
|---|---|---|---|
| 104 | 102 | 95 | 80 |
| 2 | 4 | 8 | 16 |

final bucket at timestamp 105

| | | | | |
|---|---|---|---|---|
| 105 | 104 | 102 | 95 | 80 |
| 1 | 2 | 4 | 8 | 16 |

⇒ 65 is discarded as bucket size is 40 $\boxed{\therefore\ 105-65 \geq 40}$

**Question 3**: We wish to use the Flagolet-Martin algorithm of Section 4.4 to count the number of distinct elements in a stream. Suppose that there are ten possible elements, 1, 2,..., 10, that could appear in the stream, but only four of them have actually appeared. To make our estimate of the count of distinct elements, we hash each element to a 4-bit binary number. The element *x* is hashed to 3*x* + 7 (modulo 11). For example, element 8 hashes to 3*8+7 = 31, which is 9 modulo 11 (i.e., the remainder of 31/11 is 9). Thus, the 4-bit string for element 8 is 1001.

A set of four of the elements 1 through 10 could give an estimate that is exact (if the estimate is 4), or too high, or too low. You should figure out under what circumstances a set of four elements falls into each of those categories. Then, take any 4 examples of the set of four elements, find out the exactly correct estimate among 4 examples.

Given Hash function is h(x) = (3x + 7) % 11

If distinct element (de) == 4 □ exact

If de < 4 □ low

If de > 4 □ high

**Example 1:** 3, 4, 8, 10

| | | **h(x)** | | **binary** | | **count of trailing zeros** |
|---|---|---|---|---|---|---|
| 3 | □ | 5 | □ | 101 | □ | 0 |
| 4 | □ | 8 | □ | 1000 | □ | 3 |
| 8 | □ | 9 | □ | 1001 | □ | 0 |
| 10 | □ | 4 | □ | 100 | □ | 2 |

Max (count of trailing zeros) = 3 => r = 3

R = $2^r$ = 8 □ **high**

**Example 2**: 2, 3, 6, 9

| | | **h(x)** | | **binary** | | **count of trailing zeros** |
|---|---|---|---|---|---|---|
| 2 | □ | 2 | □ | 10 | □ | 1 |
| 3 | □ | 5 | □ | 101 | □ | 0 |
| 6 | □ | 3 | □ | 11 | □ | 0 |
| 9 | □ | 1 | □ | 01 | □ | 0 |

Max (count of trailing zeros) = 1 => r = 1

R = $2^r$ = 2 □ **low**

**<u>Example 3</u>**: 2, 6, 8, 9

|  | **<u>h(x)</u>** | **<u>binary</u>** | **<u>count of trailing zeros</u>** |
|---|---|---|---|
| 2 □ | 2 □ | 10 □ | 1 |
| 6 □ | 3 □ | 11 □ | 0 |
| 8 □ | 9 □ | 1001 □ | 0 |
| 9 □ | 1 □ | 01 □ | 0 |

Max (count of trailing zeros) = 1 => r = 1

R = $2^r$ = 2 □ **low**

**<u>Example 4</u>**: 1, 3, 9, 10

|  | **<u>h(x)</u>** | **<u>binary</u>** | **<u>count of trailing zeros</u>** |
|---|---|---|---|
| 1 □ | 10 □ | 1010 □ | 1 |
| 3 □ | 5 □ | 101 □ | 0 |
| 9 □ | 1 □ | 01 □ | 0 |
| 10 □ | 4 □ | 100 □ | 2 |

Max (count of trailing zeros) = 2 => r = 2

R = $2^r$ = 4 □ **Exact**

**Question 4**: A certain Web mail service (like Gmail, e.g.) has $10^8$ users, and wishes to create a sample of data about these users, occupying $10^{10}$ bytes. Activity at the service can be viewed as a stream of elements, each of which is an email. The element contains the ID of the sender, which must be one of the $10^8$ users of the service, and other information, e.g., the recipient(s), and contents of the message. The plan is to pick a subset of the users and collect in the $10^{10}$ bytes records of length 100 bytes about every email sent by the users in the selected set (and nothing about other users).

The method of Section 4.2.4 will be used. User ID's will be hashed to a bucket number, from 0 to 999,999. At all times, there will be a threshold t such that the 100-byte records for all the users whose ID's hash to t or less will be retained, and other users' records will not be retained. You may assume that each user generates emails at exactly the same rate as other users. As a function of n, the number of emails in the stream so far, what should the threshold t be in order that the selected records will not exceed the $10^{10}$ bytes available to store records?

Here given that there are currently ‘N’ emails in the stream and $10^6$ buckets.

we can calculate the email capacity of each bucket as $\frac{N}{10^6}$ emails.

We also know that each email need 100 bytes, therefore the total space required per bucket is $\left(\frac{N}{10^6} * 100\right) bytes = \left(N * 10^{-4}\right) bytes$

Let us consider the worst case, where all ‘N’ emails in the stream have to be retained.

Let us assume the total number of buckets we would need for this scenario is (t+1), since we started the bucket count from 0.

So, (space requirement per bucket) * (total no. of buckets) <= (Total available space)

- ⇨ $\left(N * 10^{-4}\right)(t + 1) \leq 10^{10}$
- ⇨ $t + 1 \leq \frac{10^{10}}{N*10^{-4}}$
- ⇨ $t \leq \left(\frac{10^{14}}{N}\right) - 1$

E.g. If N =$10^3$, then t <= 9

**Question 5**: Suppose we hash the elements of a set S having 23 members, to a bit array of length 100. The array is initially all-0's, and we set a bit to 1 whenever a member of S hashes to it. The hash function is random and uniform in its distribution. What is the expected fraction of 0's in the array after hashing? What is the expected fraction of 1's? You may assume that 100 is large enough that asymptotic limits are reached.

m = Length of Set = 23

n = Array Length = 100

Fraction of 1’s in the array after hashing = $1 - e^{-\frac{m}{n}} = 1 - e^{-\frac{23}{100}}$

Fraction of 0’s in the array after hashing = $1 - \left[1 - e^{-\frac{m}{n}}\right] = 1 - \left[1 - e^{-\frac{23}{100}}\right]$