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 2m-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.

Ans:

```
def surprise(s):
    timestamp = {} #stores the count of each element till the timestamp 75
    for n in s:
        timestamp[n] = timestamp.get(n, 0) + 1  #n: is a key to search in timestamp, if it doesn't exists return value
    return sum([v*v for k,v in timestamp.items()]) #compute the surprise number = sum(count^2), for every element
```

```
s = [(i % 10) + 1 for i in range(0, 75)] #generate elements 1 through 10 repeatedly till the timestamp 75
print(surprise(s))
```

```
565
```

```
timestamps_random = [[20, 49, 53],
                     [17, 43, 51],
                     [25, 34, 47],
                     [37, 46, 55]]
```

```
[20, 49, 53] => 375
[17, 43, 51] => 525
[25, 34, 47] => 675
[37, 46, 55] => 375
```

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).

Ans:

End Time	101	100	95	87	80	65
Size	1	2	4	4	8	8

End Time	102	101	100	95	87	80	65
Size	1	1	2	4	4	8	8

End Time	103	102	100	95	87	80	65
Size	1	2	2	4	4	8	8

End Time	104	103	102	100	95	87	80	65
Size	1	1	2	2	4	4	8	8

End Time	105	104	102	95	80
Size	1	2	4	8	16

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 *3x + 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.

Ans:

```
import json

elements = [[3, 4, 8, 10],
            [1, 2, 3, 9],
            [4, 5, 6, 7],
            [3, 7, 8, 10]]
```

```
#compute the hash function for each element
def modulo(v):
    return (3*v + 7) % 11
```

```
#compute the number of trailing zeroes in h(a)
def bits(x):
    m = [0xf, 0x7, 0x3, 0x1]
    for i in range(0, 4):
        if m[i] & x == 0:
            return 4-i
    return 0
```

```
for e in elements:
    R = max([bits(modulo(v)) for v in e]) #r stores the maximum number of trailing zeroes
    print('%s = %d' % (json.dumps(e), R*R)) #r*r: gives the number of distinct elements
```

```
[3, 4, 8, 10] = 9
[1, 2, 3, 9] = 1
[4, 5, 6, 7] = 16
[3, 7, 8, 10] = 4
```

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?

Ans:

Given a number of buckets = $10^6(0 - 999999)$

Let n be the number of emails in the stream

therefore, probability of each email hashed to each bucket =$n/10^6$

Total space required per bucket = $(n/10^6) * 100bytes$

Threshold $t = t + 1$

$(n/10^6) * 100 * (t + 1)$ <= 10^{10}

$n/10^4 * (t + 1) <= 10^{10}$

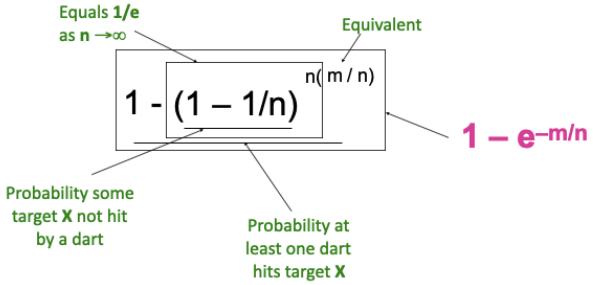
t+1 <= $10^{14}/n$

t <= (10^{14}/n) - 1

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.

Ans:

- We have ***m*** darts, ***n*** targets
- **What is the probability that a target gets at least one dart?**



n = denotes bit array and m = denotes elements

$m = 23, n = 100$

Fractions of 0's in the array = $(1 - 1/100)^{23} = e^{-}m/n = e^{-}23/100$

Fraction of 1's in the array = $1 - e^{-}23/100$