Solution of Gradiance Homework – 3

– By Anirban haldar

Stream Data Analysis

1. **A variant of a Bloom filter uses a single hash function that generates several distinct bit numbers. We shall consider this variant in a tiny example. It uses an array of 9 bits and a hash function h that generates two distinct bit numbers. We want to test membership in a set S of three elements, so we hash each of the three elements using h, and we set to 1 the two bits that any of the three elements is hashed to by h.**

   When a new element x arrives, we compute h(x), and we say x is in the set if both these bits are 1. Assume x is not in the set S. What is the probability of a false positive; i.e., the probability of saying that x is in S. Identify this probability, correct to three decimal places, in the list below.

   a) .271 b) .257 c) .164 d) .280

   ⇨ Here, according the question, although we are using a single hash function, but the function returns two distinct hash values (instead of one as we see normally in hashing). And we have 9 hash buckets, say the numbers 0 … 8. Then we can write the hash function as,

   $$h(a) = [x, y] \qquad \text{where} \quad x, y \in [0 \dots 8]$$

   Now, as per question, in the input set S, we have 3 values say A, B & C. now for each of these values, the h() will generate two distinct numbers and set corresponding bits = 1 in the hash table. So at the end, we will get minimum 2 count of 1s (if all the inputs hash to same buckets) and maximum 6 count of 1s (if all hash to different buckets). We may get any no of ones in range [2 … 6].

   Probability that a given bucket is empty = $\left(1-\frac{1}{9}\right)^6 = 0.493$
   Probability that a given bucket has at least one entry = $1-\left(1-\frac{1}{9}\right)^6 = 0.507$
   Now, for x to be able to hash into buckets with both 1s = (0.507)*(0.507) = 0.257

   So the correct answer is **0.257**

2. **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? From the list below, identify the true statement about a value of n and its value of t.

- **a) n = 10^9; t = 999**
- **b) n = 10^{11}; t = 1000**
- **c) n = 10^{12}; t = 100**
- **d) n = 10^{11}; t = 999**

⇨ From the given question, we can collect some data as
- ❖ Number of users = 10^8
- ❖ Total storage = 10^{10} Bytes
- ❖ Data per email = 100 Bytes
- ❖ Bucket Numbers = 0 to 999,999

Now, from the given data, we can say, we have total 100,000 hash buckets, and if we consider uniform hashing over all users, then
- ❖ in each bucket, we will get 10^8 / 1000,000 = 100 users / bucket

we will only store emails generated by the users having hash bucket number <= t.
Now, we will validate each option from the question as,

➢ **Option a) n = 10^9 ; t = 999**

We have total number of emails = 10^9
So email / user = 10^9 / 10^8 = 10
Threshold t = 999. Now number of buckets having bucket number <= 999 = 1000
So number of effective users = effective buckets x user per bucket
= 1000 x 100
= 10^5

So total space requires = (email / user) x (effective users) x (email size) Bytes
= 10 x 10^5 x 100 Bytes
= 10^8 Bytes
Comment : We still have space left, so we are not utilizing full potential

➢ **Option b) n = 10^{11} ; t = 1000**

We have total number of emails = 10^{11}
So email / user = 10^{11} / 10^8 = 10^3
Threshold t = 1000. Now number of buckets having bucket number <= 1000 = 1001
So number of effective users = effective buckets x user per bucket
= 1001 x 100
= 100100

So total space requires = (email / user) x (No of effective users) x (email size) Bytes
= 10^3 x 100100 x 100 Bytes
= 1.001 x 10^{10} Bytes
Comment : We are overflowing our allocated space.

- **Option c) n = 10^{12} ; t = 100**
  We have total number of emails = 10^{12}
  So email / user = 10^{12} / 10^8 = 10^4
  Threshold t = 100. Now number of buckets having bucket number <= 100 = 101
  So number of effective users = effective buckets x user per bucket
  = 101 x 100
  = 10100

  So total space requires = (email / user) x (effective users) x (email size) Bytes
  = 10^4 x 10100 x 100 Bytes
  = 1.01 x 10^{10} Bytes
  Comment : We are overflowing our allocated space.

- **Option d) n = 10^{11} ; t = 999**
  We have total number of emails = 10^{11}
  So email / user = 10^{11} / 10^8 = 10^3
  Threshold t = 999. Now number of buckets having bucket number <= 999 = 1000
  So number of effective users = effective buckets x user per bucket
  = 1000 x 100
  = 10^5

  So total space requires = (email / user) x (effective users) x (email size) Bytes
  = 10^3 x 10^5 x 100 Bytes
  = 10^{10} Bytes
  Comment : This combination is suitable for our allocated space.

So the correct combination of n and t is => **n = 10^{11} ; t = 999**

3. **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, identify from the list below the set of three "random" timestamps that give the closest estimate.

- **a) {4, 31, 72}**
- **b) {25, 34, 47}**
- **c) {31, 32, 44}**
- **d) {24, 44, 65}**

⇨ According to the question, our data stream till now is like 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 …. 7 8 9 10 1 2 3 4 5 – total 75 values as [1 … 10] * 7 + [1 … 5]

Now, to get estimation from any given timestamp, the logic is to count (c) that item from that timestamp to the current time and estimated_val = n x (2 x c - 1)

For this given problem, say we have a given timestamp t, then we can write

C = 7 – floor(t / 10) + (1 if t % 10 <= 5 else 0)

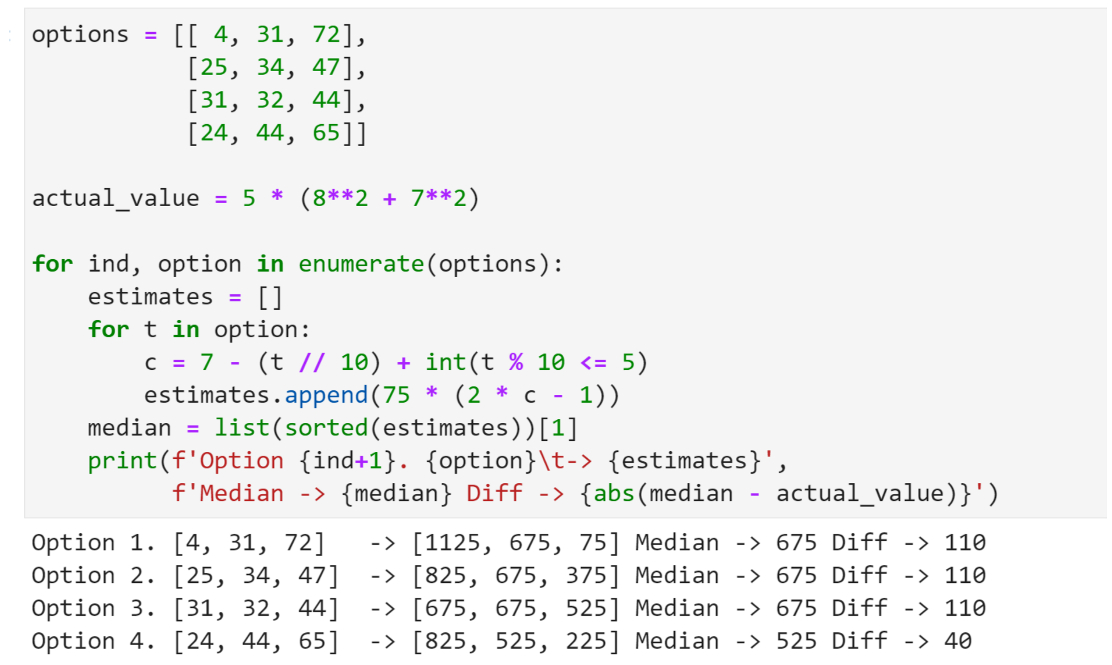
And, the actual value of 2nd moment is = $(8^2 + 8^2 + 8^2 + 8^2 + 8^2 + 7^2 + 7^2 + 7^2 + 7^2 + 7^2)$

= 5 x (64 + 49)

= 565

Now, we have to find c for each values in the three random timestamps, and calculate estimated_val using the formula, and choose the median of those values.

Now if we iterate through the options then we can find,



```python
options = [[ 4, 31, 72],
           [25, 34, 47],
           [31, 32, 44],
           [24, 44, 65]]

actual_value = 5 * (8**2 + 7**2)

for ind, option in enumerate(options):
    estimates = []
    for t in option:
        c = 7 - (t // 10) + int(t % 10 <= 5)
        estimates.append(75 * (2 * c - 1))
    median = list(sorted(estimates))[1]
    print(f'Option {ind+1}. {option}\t-> {estimates}',
          f'Median -> {median} Diff -> {abs(median - actual_value)}')
```

```
Option 1. [4, 31, 72]   -> [1125, 675, 75] Median -> 675 Diff -> 110
Option 2. [25, 34, 47]  -> [825, 675, 375] Median -> 675 Diff -> 110
Option 3. [31, 32, 44]  -> [675, 675, 525] Median -> 675 Diff -> 110
Option 4. [24, 44, 65]  -> [825, 525, 225] Median -> 525 Diff -> 40
```

So the closest estimation is {24, 44, 65} which have minimum distance = 40

4. **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**
Start	**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.

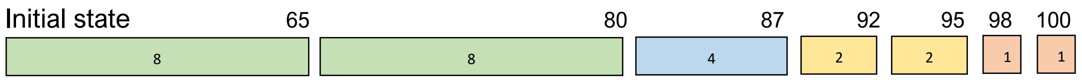
Suppose that at times 101 through 105, 1's appear in the stream. Compute the set of buckets that would exist in the system at time 105. Then identify one such bucket from the list below. Buckets are represented by pairs (end-time, size).

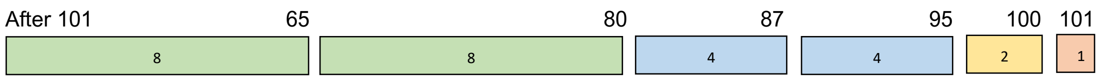
a) (103,1) **b) (95,8)** **c) (98,4)** **d) (103,2)**

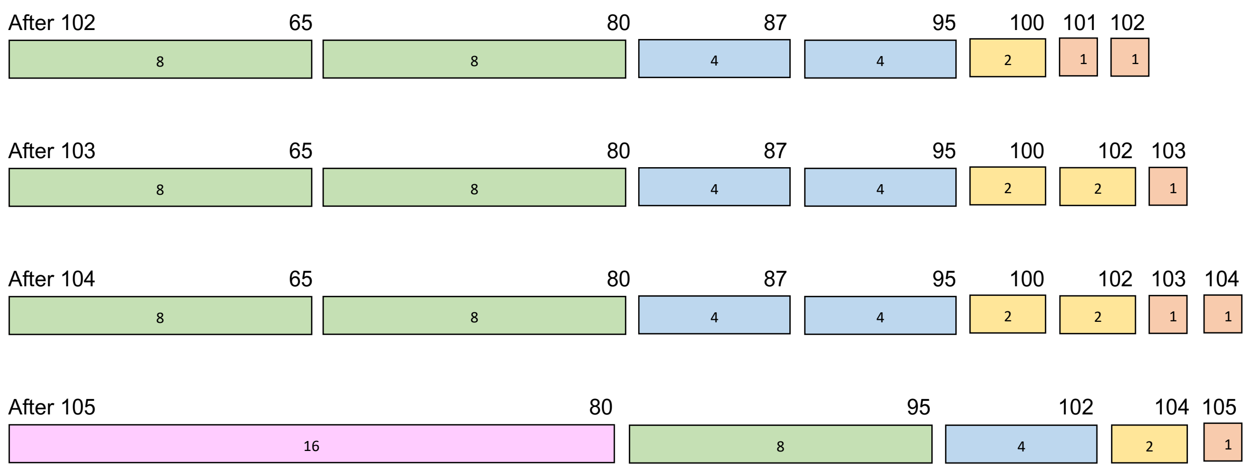
⇨ The DGIM algorithm maintains last k bit window. In this window, we maintain some bit buckets of size in power of 2 along with the timestamp of each bucket, which is basically the timestamp of the last 1 in that bucket. Now the buckets have properties

- Sizes of buckets are 1, 2, 4, 8, 16, 32…. In this fashion.
- The buckets are always sorted in non-decreasing order of their sizes.
- When a new 1 bit occurs, we put that bit into a new bucket of size 1, and the timestamp of that bucket = the timestamp of that 1.
- If we have consecutive 3 buckets of same size, then we merge the older two buckets to form a bigger bucket with size = 2 x size of smaller buckets, and the **timestamp of new bucket = the timestamp of newer bucket. (according to the book. This will always result in overestimates.)**
  Note: In the lecture, we were taught to use the timestamp of older bucket while merging. This results in more balanced estimates (as the timestamp is roughly in the middle of the interval, sometimes we will over-estimate and sometimes we will under-estimate).
- If the timestamp of any bucket goes beyond the window size, then we will discard that bucket from our bucket list.
- The last bucket may contain partially into the window.

Now, according to the question, initially we have 7 windows with size & timestamp as below. Now we take 5 more incoming bits from 101 to 105, and the orientation of buckets after each input will be like,







So, out final buckets and their timestamps are,

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

5. In this problem, we shall simulate the "popular elements" algorithm of Section 4.7.3, which uses an exponentially decaying window. We use the unrealistically large decay parameter c=0.1, so we multiply by 0.9 at each stage. We use a threshold of 0.6, in place of the 0.5 suggested in that section. The sequence of elements arriving on the stream is:

a b c a d e a c b b

Simulate the computation of the popularity score on this stream. When computing scores, you should round to 3 decimal places after each step. Then, identify in the list below the correct popularity score for one of the elements at the end of the stream.

a) The score of d is .590
b) The score of e is .656
c) The score of c is 1.874
d) The score of a is 1.483

⇨ Here, we allocate one variable S_i to every unique item to store a sum value over a period of time. The equation to update the variable is

$$S_{t+1} = S_t(1-c) + a_{t+1}$$

Here,
c => a small constant and from the question, c = 0.1
a_{t+1} => 1 if the current element is the one for which we are calculating the sum, 0 otherwise.

If we see any item, which has no sum variable, then we initialise a new variable for that item, and initialise it as $S_i = 1$, and from next input stream, count S_i using the above equation.

Now, one more condition is, if the sum value S_i becomes less than the threshold value (according to question, threshold = 0.6), then we will drop the sum variable for that element, because it is no more popular now.

Here, in the question, we get a stream of characters consisting 5 unique characters or items as - a, b, c, d & e. Now, I have written a small piece of code to simulate the trending character which utilizes the equation & conditions described above.

```python
c = 0.1
stream = ['a', 'b', 'c', 'a', 'd', 'e', 'a', 'c', 'b', 'b']
sum_vars = dict()

for item in stream:
    all_vars = list(sum_vars.keys())
    for var in all_vars: sum_vars[var] = round(sum_vars[var] * (1-c) + int(item == var), 3)
    if item not in sum_vars: sum_vars[item] = 1
    [sum_vars.pop(var) for var in all_vars if sum_vars[var] < 0.6]

print(sum_vars)
```

```
{'a': 1.648, 'c': 1.288, 'e': 0.656, 'b': 1.9}
```

So the correct answer is => The score of e is .656