Report of Sketches Experiment

Zhendong Liu A0159369L

September 25, 2016

1 Introduction

This report makes conclusion of the experimenting result of two different sketch algorithms towards the streaming problem on problem set 1.1, where we use a set of 1-sparse samplers to count the appearance times of an item in a stream.

In this experiment, we tested the two algorithms on both uniform distribution input stream and exponential distribution input stream. By design a *correctRate* standard on measuring the accuracy of algorithms and specifying many possible value of the parameters M, N, A, B, we see the overall performance of algorithms under different conditions.

We find that whether algorithm2 is better than algorithm1 on accuracy is depend on the value of N and B, but have no relevance with the distribution of the items in stream or the different items in stream. For detailed conclusions and discussion please refer to section 5.

The source code wrote by myself is on Github:
`https://github.com/lzddzh/cs5234/blob/master/pset4/countItemFrequncy.py`

2 Implementation

We choose Python as the programming language for this project since it is easy to write and we concern about the algorithms accuracy but not their speed (In general, Python program is not fast). In the code, a $CounterSampler$ class has been implemented, with member variables stores the value of A, B, P(the prime number in hash functions), etc., and with several member functions such as $push(num), getResult(num), hash(num, a, b, P), etc.$ We will not include the whole class here but only some key data structures will be introduced below.

2.1 Counters

It is easy to think about using a 2D array with unsigned int data type to implement the counters, since the counters count only positive values and the number of rows or columns of the counters won't change. Here what we use is a A * B sized numpy matrix with 0 initialled:

- $counters = numpy.zeros((A, B), dtype =' uint64')$

Then access the ith row and jth column counter $C(i, j)$ by using $counters[i][j]$

2.2 Hash functions

Its fool's effort to directly store a big number of hash functions in a class. Instead we implement the hash functions by using a single member function and a member variable of our class:

- $hash(num, a, b)$ A member function that takes num as its input and returns $(ka * num + kb)\%Prime$. This hash function can be set with different paramaters ka and kb. And the $Prime$ we choose is $Prime = min\{P|P > 100 * B,\ P\ is\ prime\}$, where B is the number of columns of counters.
- $parametersList = ((a_1, b_1), (a_2, b_2), ...)$ An array of tuples for storing the A hash functions parameters. In this array, we have A pairs of tuple (a,b), whose value are $1 \leq a, b \leq Prime$

To use the different hash functions, we just specify different pairs of (a,b) for the member function hash(num, a, b), $h_i(num) = hash(num, a_i, b_i)$, so by storing A pairs of (a,b) in the member variable of $CounterSampler$ class, we can use any of the hash functions as we need.

2.3 Two Algorithms

Since the two algorithms are only different in their way of getting result from counters, we just use two different member functions $getResult1(num)$ and $getResult2(num)$ in the class $CounterSampler$ to get result from Algorithm1 and Algorithm2.

- $CounterSampler.getResult1(num)$ returns the minimum value among $C(i, h_i(num)$ for all $0 \leq i \leq A-1$
- $CounterSampler.getResult2(num)$ returns the median value of $Estimate(x, i) = C(i, h_i(num) - neighbor(C(i, h_i(num)))$ for all $0 \leq i \leq A-1$

3 Test Design

3.1 Random Numbers for Test Data

- $UniformlyDistribution$: We use the build-in Python function $random.randint(a, b)$ which generate a uniformly random number r that $a \leq r \leq b$, and we just repeat this functions N times so we have N random numbers.
- $ExponentialDistribution$: We use the build-in Python function $numpy.random.exponential(\beta, size)$ to generate a list of random numbers that follows the exponential distribution $f(x) = \frac{1}{\beta} exp^{-\frac{x}{\beta}}$. Here we choose $size = N, \beta = \frac{M}{11} + 1$, which let the output of random numbers are very likely to with in the range $0 \leq x \leq M$(if we are very not lucky and x exceeds M, we just re-generate it again). And since we need integers but not float numbers, so we going to just use the integers part of the generated numbers.

3.2 Correct Rate

If we want to compare the accuracy of two algorithms, we must have a standard. Here we use a $correctRate$ to represent the accuracy of the results output by one particular algorithm.
We define a the correct Rate on a given algorithm $K \in \{1, 2\}$ and the set of numbers appeared in the input stream $Z = \{x_1, x_2, ..., x_n\}$ by

$$correctRate(K, Z) = \frac{\sum_{x \in Z} g(K, x)}{|Z|} \tag{1}$$

$$g(K, x) = \begin{cases} 1 \ (CounterSampler.getResultK(x) = n(x)) \\ 0 \ (CounterSampler.getResultK(x) \neq n(x)) \end{cases} \tag{2}$$

where $n(x)$ is the times of x appears in the stream, and $CounterSampler.getResultK(x)$ is the one in Section 2.3.

3.3 Parameter Choosing

There are four parameters in our algorithms that could influence the correct rate: M, N, A, B, separately represents the maximum number in the stream, the length of the stream, the rows of counters and the columns of counters. We want to see how algorithms works on different values of this parameters, so we choose each of these variables a range of values to test the two algorithms.

- M: we will test $M \in \{10, 100, 1000, 10000\}$
- N: we will test $N \in \{10, 100, 1000, 10000\}$
- A: we will test $A \in \{3, 4, 7, 10, 100\}$
- B: we will test $B \in \{10, 20, 40, 80, 160, 500, \}$

For each kind of test data set. First we are going to fix A and B then test the combinations of values in M and values in N, see the correct rate of Algorithm1 and Algortihm2. Next, we will fix M and N then test the combinations of values in A and values in B, see the correct rate of Algorithm1 and Algorithm2.

4 Results

4.1 Uniform Distribution, Fix A = 10 and B = 500

When we fix set $A = 10$ and $B = 500$, it means we use 5000 space in our algorithm, so the test cases that $M \leq 5000$ in fact is meaningless since in this cases the sketch costs more space then normal method. $Figure1$ shows the two algorithms comparison on different values of M and N when A and B are fixed.

```
M= 10000 N= 10  correct rate of A1 and A2:  (1.0, 0.4)
M= 10000 N= 100  correct rate of A1 and A2:  (1.0, 0.25510204081632654)
M= 10000 N= 1000  correct rate of A1 and A2:  (0.003125, 0.221875)
M= 10000 N= 10000  correct rate of A1 and A2:  (0.0, 0.10287081339712918)
M= 10000 N= 100000  correct rate of A1 and A2:  (0.0, 0.0254)
M= 100000 N= 10  correct rate of A1 and A2:  (1.0, 0.5)
M= 100000 N= 100  correct rate of A1 and A2:  (1.0, 0.21)
M= 100000 N= 1000  correct rate of A1 and A2:  (0.004012036108324975, 0.17753259779338015)
M= 100000 N= 10000  correct rate of A1 and A2:  (0.0, 0.1327582579423522)
M= 100000 N= 100000  correct rate of A1 and A2:  (0.0, 0.052614105259829576)
```

Figure 1: Uniformly Data Test A and B Fixed

In $Figure1$, there are 10 cases, in which 4 cases shows Algorithm1's correct rate is higher than Algorthm2's and 6 cases shows Algorithm2's correct rate is higher. So in cases that N is not so large, algorithm1's correct rate is higher, as N becomes very larger, we see algorithm2's correct rate is higher.

4.2 Uniform Distribution, Fix M = 10000 and N = 1000

In this group of test, we filter out the ones that $A * B >= M$ since it use too much spaces. Then we have $Figure2$ that shows the test output when we fix $M = 10000$ and $N = 100$.

```
A= 3 B= 10  correct rate of A1 and A2:  (0.0, 0.0)
A= 3 B= 20  correct rate of A1 and A2:  (0.0, 0.02)
A= 3 B= 40  correct rate of A1 and A2:  (0.0, 0.03)
A= 3 B= 80  correct rate of A1 and A2:  (0.58, 0.02)
A= 3 B= 160  correct rate of A1 and A2:  (0.73, 0.27)
A= 3 B= 500  correct rate of A1 and A2:  (0.92, 0.56)
A= 4 B= 10  correct rate of A1 and A2:  (0.0, 0.02)
A= 4 B= 20  correct rate of A1 and A2:  (0.0, 0.01)
A= 4 B= 40  correct rate of A1 and A2:  (0.0, 0.02)
A= 4 B= 80  correct rate of A1 and A2:  (0.27, 0.03)
A= 4 B= 160  correct rate of A1 and A2:  (1.0, 0.04)
A= 4 B= 500  correct rate of A1 and A2:  (0.72, 0.17)
A= 7 B= 10  correct rate of A1 and A2:  (0.0, 0.0)
A= 7 B= 20  correct rate of A1 and A2:  (0.0, 0.0)
A= 7 B= 40  correct rate of A1 and A2:  (0.0, 0.03)
A= 7 B= 80  correct rate of A1 and A2:  (0.03, 0.05)
A= 7 B= 160  correct rate of A1 and A2:  (0.89, 0.15)
A= 7 B= 500  correct rate of A1 and A2:  (1.0, 0.5)
A= 10 B= 10  correct rate of A1 and A2:  (0.0, 0.02)
A= 10 B= 20  correct rate of A1 and A2:  (0.0, 0.02)
A= 10 B= 40  correct rate of A1 and A2:  (0.0, 0.01)
A= 10 B= 80  correct rate of A1 and A2:  (0.1, 0.02)
A= 10 B= 160  correct rate of A1 and A2:  (0.57, 0.08)
A= 10 B= 500  correct rate of A1 and A2:  (1.0, 0.26)
A= 100 B= 10  correct rate of A1 and A2:  (0.0, 0.01)
A= 100 B= 20  correct rate of A1 and A2:  (0.0, 0.0)
A= 100 B= 40  correct rate of A1 and A2:  (0.0, 0.0)
A= 100 B= 80  correct rate of A1 and A2:  (0.0, 0.0)
```

Figure 2: Uniformly Data Test M and N Fixed

From $Figure2$, we can find that when B is small, such as $B = \{10, 20\}$, the correct rate of algorithm2 is higher, but when B is large, such as $B = \{80, 160, 500\}$, algorithm1 correct rate is higher.

4.3 Exponential Distribution, Most Frequently Items

In this group of test, we tested the correct rate of the first 20% most frequently appeared items in the stream. We first fixed $A = 10, B = 500$, then fixed $M = 100, N = 1000$.

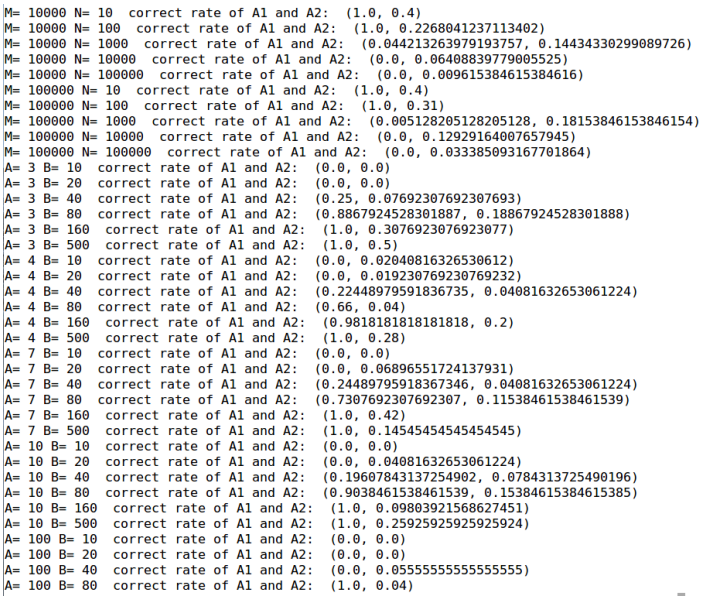

Figure 3: Exponential Distribution Test on Most Frequently Items

Then in *Figure*3 we find the similar result of comparison of algorithm1 and algorithm2, that when N is large or B is small, algorithm2's correct rate is higher. When N is small and B is large, algorithm1's correct rate is higher.

4.4 Exponential Distribution, Least Frequently Items

In this group of test, we tested the correct rate of the last 20% least frequently appeared items in the stream. The parameters A, B, M, N value setting is the same with 4.3. Then in *Figure*4 we find

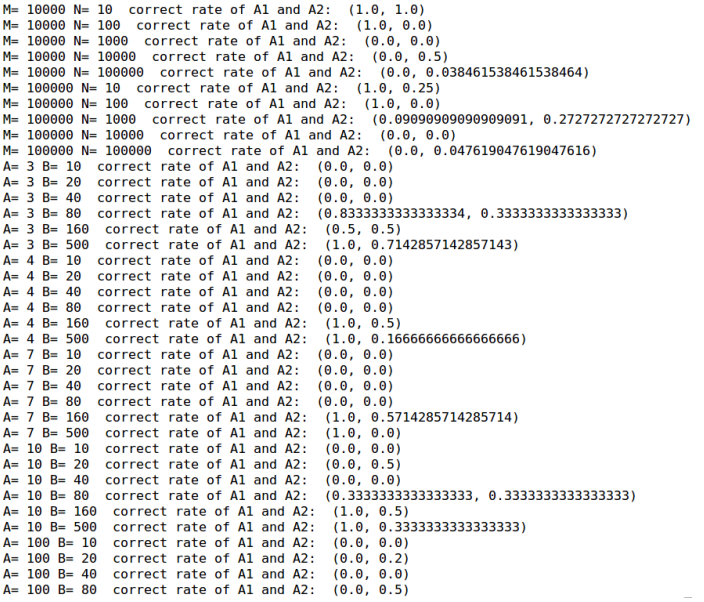

Figure 4: Exponential Distribution Test on Least Frequently Items

the same result as we said above other conditions, that when N is large or B is small, algorithm2's correct rate is higher. When N is small and B is large, algorithm1's correct rate is higher.

5 Conclusions and Discussion

From the observation of the experimenting result, I draw conclusions below:

a. When N is large or when B is small, algorithm2 is more accurate then algorithm1, while algorithm1 perform better when N is small or B is big.

b. The distribution of input stream does not influence which algorithm is better on accuracy.

c. The different elements in the stream does not influence which algorithm is better on accuracy.

I think the reason of conclusion (a) is might because algorithm2 is designed to subtracting some of the noise from nearby counters, so when N is large and B is small, we can easily think of that the noise become bigger, since more items are mapped into smaller buckets.

And the reason of conclusion (b) and (c) is might because the hash function we use is random enough(since we choose big enough $Prime$), so no matter how the input data is, we can map them uniformly into our B buckets.