CS 5968/6968: Data Str & Alg Scalable Comp Spring 2023

Lecture 17 — Mar 22, 2023

Prof. Prashant Pandey *Scribe: Zixiao (Martin) Chen*

1 Solutions to build cardinality counting

Assume the universe size $|U| = 2^{64}$. For this problem, several solutions have been proposed:

1. Array

10	10	10	0	0	20	20	0	0	0

   The above is the model on how array implementation of the cardinality counting would work. The array will have the length of u where each index of the array representing the value of the item.

   Given an item value x, we can increment the count of the item by having arr$[x]+ = 1$.

2. Bit Array

1	0	0	1	0

   The bit array is established with length of $|U|$. The graph above illustrates how bit array would be like for implementing carnality representation.

   The bit array has the length of $|U|$. In the bit array, 1 is seen as item inserted while 0 is seen as item doesn't exist. If the item is first encountered, then the item will be checked at index i, if $arr[i] == 0$, then set $arr[i] = 1$. Otherwise, the item is already inserted and no changes is required.

3. Hash Table

10	20	30	null	null

   The above is a illustration of how hash table would be for the implementation of cardinality counting.

   The hash table has the length of $|U|$. The item will be checked using the hash value of it. If the item doesn't exist then the item will be inserted. Otherwise, no changes happen in the hash table.

Although these implementation provide the exact cardinality with $O(n)$ cost,these solutions are costly in terms of the space complexity, particuarly the hash table solution will incur $\Omega(n)$. Therefore, it is essential to have an implementation achieving time complexity of $O(n)$ while keeping the space complexity to be lower than linear complexity.

2 Techniques to estimate cardinality with examples

Before diving into the solution to this problem, let's introduce with the following hat problem:

- Given the fact that the cards are labelled 1-1000, and a random subset of size n to hide in hat is chosen. One representative value of the subset picked from this hat is known. However, one of median, minimum or maximum value get picked. The task to estimate n.

One possible solution for tackling this task is pick the minimum value and treat the cards such that the values are distributed uniformly (equal difference between consecutive two values of cards).

For example, if the minimum value is 40 for a subset with unknown items picked randomly from the hat. Using the property uniform distribution, I can build the equation of

$$1000/(n+1) = 40$$

in which $n = 24$.

Even through estimate should grow as minimum value shrinks under the assumption that the values are shown under the uniform distribution, it is still likely that the error margin is huge as cards with small value yet come in small numbers are also possible.

3 Two-hat problems

Another solution of estimating the cardinality and the Jaccard similarity between two sets is using two-hat problems model as follows:

- Given two sets A and B where the items are unknown, get the k smallest values from set A and B respectively.
- After getting k-min items from both sets, apply the intersection or union from the k-mins from both sets as a sketch to estimate the number of unique items using this data.
- Space of coincidences is large, Need to look at more than one representative. (Same representatives with minor difference in details)

The error margin is $O(1/\sqrt{k})$ for the above procedure. Hence, it seems that somehow bottom-k is better than getting the minimum value directly.

Nonetheless, the space of coincidences is so large that it would be problematic to calculate the

cardinality of two large sets.

Therefore, it is pretty necessary to look at more than one representative when estimating the number of unique items.

4 Hyperloglog

Before introducing a more sophisticated solution, let's look at the following model:

- Assume that a hash function producing hash values usingtotally randomness will be applied, we can use this hash function to hash input to create output. After getting the output, the number can be translated into p-bit representation.
- The bitwise representation may seem to look as follows with exactly three leading zeros on higher order for first few bits:

0	0	0	1	1	0	0	1	0	1

- The probability that higher order exactly 3 bits are all 0s is:

$$\mathbb{P}(\text{higher order exactly 3 bits are all zeros}) = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{16}$$

  Based on the above result, it can be inferred that for every 16 hash values there is one whose bitwise representation starts with 0001. In other words, if we see a sequence of exactly 3 zeros, these are probably 16 items.

- Under such metric, the probability of having higher order x bits are exactly 0s is $1/2^{x+1}$. Similarly, if in all hash values and the longest streak of zeros is L, then on average there are 2^{L+1} items.

Under this mapping between number X and Max LZC(Max Leading Zero Count) , the probability of having higher order x bits are exactly zeros is $1/2^{x+1}$, which further shows as the number of bits get larger, it becomes increasingly different to find many coincidences under one certain representative values with bitwise operation.

Hence, we can build the following analysis:

- Let M be maximum number of unique elements and L be the number of bits used for the values representing unique elements. Then

$$2^{L+1} \leq M$$

  as it is true that the probability of having higher order x bits exactly zeros is $1/2^{L+1}$.

- As the below inequality also holds:

$$L \leq \log M$$

We can establish a data structure where we only need to use L many bits to represent a single counter. However, the danger of doing so is such procedure will only estimate power of 2 and too much uncertainty is involved. What's worse, it still fails to solve the case in which items with large values exist yet come in few counts.

Therefore, it is necessary to have multiple counters under bitwise representation to get accurate estimate of number of unique items.

Actually, we can represent every counter value as bitwise representation. In other words, the space complexity will be $O(\log \log M)$ many bits. We can do so by using many counters with the representation of $O(\log \log M)$ many bits where M stands for the maximum number of unique items.

If we want to use multiple to get the average, then the final result will be

$$2^{\frac{L_1+L_2+...+L_k}{k}}$$

However such result is still having some error bias. Another solution producing less error margin is by calculating the harmonic mean of k counter, which is less sensitive to large outliers:

$$\frac{k}{\frac{1}{L_1}+...+\frac{1}{L_k}}$$

We call this calculation as hyperloglog approach. The name "hyper" is because such calculation is applying harmonic mean. "loglog" is because two bitwise operations are adopted with one dealing with the counter value and another dealing with the hash value of item.

References

[1] Daniel N. Baker, Ben Langmead. Dashing: fast and accurate genomic distances with HyperLogLog. Baker and Langmead Genome Biology (2019) 20:265.