The Count Distinct Problem

Steven Rosendahl

Today's modern world is completely connected; nearly everyone has access to the internet at some point during the day. This constant connection to the internet means that data is constantly being transferred and collected. Companies such as Google, Facebook, and Twitter often collect the data that they

The problem of processing data has not always been a problem. Before the internet was widely used by the population, storing data was expensive, and as a result most companies did not want to spend money on the the hardware and space that would be required to store large amounts of data. In addition, data was not transferred and gathered at a fast enough rate to warrant finding a way to count the data; even if companies wanted to count the data, the computers at the time were not powerful enough to perform the necessary calculations. The problem of counting big data did not arise until the 1990's, when the internet began to facilitate the easy transfer and collection of data. Today, more than 25 exabytes of data are created a day, and out of that data, nearly 50% of it is stored for later use. The main factor that contributes most to the problem is that gathering and storing data is cheap, but analyzing it is not. The volume of the data is what leads to the count distinct problem that we will analyze.

The first solution to the count distinct problem is called the *Hash Table*. The algorithm is as follows:

Let $\mathbb{S}$ be a set of random elements. In order to count the number of distinct elements in $\mathbb{S}$,

1. Apply a bijection, $h(n)$ to all $n \in \mathbb{S}$, and store the result in a set $\mathbb{V}$.
2. Count the number of elements in $\mathbb{V}$.

Since $\mathbb{V}$ is a bijection, it is surjective. As a result, if we take any $x \in \mathbb{S}$ and $y \in \mathbb{S}$, then if $x = y$, they will hash to the same value. For example, Say we want to count the number of distinct names in the set $\mathbb{S} = \{\text{“Alice”}, \text{“Emily”}, \text{“Alice”}\}$. We can create a simple hash function where we map each letter to its corresponding value in the alphabet, sum up each value in a name, and mod it with the number of letters in the name. Applying the function on each element of $\mathbb{S}$ yields a set $\mathbb{V} = \{0, 4\}$. $||\mathbb{V}|| = 2$, so there were

two distinct elements in $\mathbb{S}$. In this example, the hash function we chose was bijective. However, this may not always be the case.

Imagine that we are solving the twitter problem where we want to count the number of distinct hashtags on twitter on a given day. Our set $\mathbb{S}$ will now contain every hashtag that was made on a given day. We know that we want a hash function that is surjective, but this is an unrealistic, since we could have some $x \in \mathbb{S}$ and $y \in \mathbb{S}$ where $x \neq y$ but $h(x) = h(y)$. To model this issue, we can consider the following scenario:

> On October 30th, 2015, Mojang released a special Halloween cape to all players of the popular video game Minecraft. As a result, "#cape" began trending on Twitter. On the same day, Target announced that all their *Bob's Burgers* costumes would be on sale for the low price of $12.95, and "#bob" began trending on twitter.

Our hash function from earlier will not work in this case, since both "#bob" and "#cape" hash to the same value. On a larger scale, it is easy to see that we would need to either handle collisions or create a better hash function. Creating a way to handle collisions would not benefit us in this case, since the data is being processed in real time. We would not be able to tell if we were hashing a value we already have seen or hashing a new value that happens to hash to the same value as another element we have already seen. Since the collision policy would not help us, we need to come up with a better hashing function.

More complex hash function, such as SHA-2 (Secure Hash Algorithm 2) or MD5 (Message Digest 5) offer a solution to the problems at which we are looking. SHA-2 is used by the NSA to encrypt data, and has yet to produce any collisions when hashing values. However, this is not a viable solution to the count distinct problem either. The problem arises from the volume of the data being processed. In both the Twitter and Facebook problems, we are analyzing exabytes of data. The SHA-2 function outputs a 512 byte long integer for every input, which means that the resulting size of the set $\mathbb{V}$ would be in the worst case $512 \times ||\mathbb{S}||$. All of $\mathbb{V}$ would have to be held in memory, which means that analyzing 4 billion hashtags (which is a generously low estimate) could possibly result in $(4 \times 10^{13}) \times 512$ bytes of data being stored at one time, assuming every hashtag was 10 bytes long. MD5 outputs smaller sized integers (128 in this case), but the collision probability is much higher. The hash method is not a viable solution to the count distinct problem when we are dealing with a large volume of data. Hashing, however is used in the other solutions that we will discuss.

The most beneficial aspect of the hash is that it has a 0% error rate. In other words, the cardinality of $\mathbb{V}$ will be exactly how many distinct elements are in $\mathbb{S}$, assuming the hash function is surjective.

However, if we are looking for a *good enough* estimate of the number of distinct elements in $\mathbb{S}$, we can use an algorithm called a Linear Probabilistic Counter. With the LPC, the user can specify how large they want the error to be. This method uses a data structure called a bit-map. Bit-maps are a matrix where either a 1 or 0 is stored in a cell, which allows for the map to be incredibly space efficient. The LPC also uses a hash, but it only needs to temporarily produce a position and then it can "forget" the value it just produced. The number of distinct elements is determined by the number of 1's in the bit-map.