The Count Distinct Problem

Steven Rosendahl

November 16, 2015

Summary

The Problem

The Hash Table

The HyperLogLog

The Question

The Question

- What is the problem?

The Question

- What is the problem?
- Image we have a set called $\mathbb{V}$ that contains a billion elements of the same type.

The Question

- What is the problem?
- Image we have a set called $\mathbb{V}$ that contains a billion elements of the same type.
- How many *unique* elements are in $\mathbb{V}$?

Questions

Questions

1. Pokémon Problem: How many unique Pokémon will a player encounter in a given playthrough of all the games?

Questions

1. Pokémon Problem: How many unique Pokémon will a player encounter in a given playthrough of all the games?
2. Facebook Problem: How many unique application installs are made a day on Facebook?

Questions

1. Pokémon Problem: How many unique Pokémon will a player encounter in a given playthrough of all the games?
2. Facebook Problem: How many unique application installs are made a day on Facebook?
3. Twitter Problem: How many unique hashtags are made a day on Twitter?

Questions

1. Pokémon Problem: How many unique Pokémon will a player encounter in a given playthrough of all the games?
2. Facebook Problem: How many unique application installs are made a day on Facebook?
3. Twitter Problem: How many unique hashtags are made a day on Twitter?

We will use $\mathbb{S}$ to represent the set of all the data, and $\mathbb{V}$ to represent the set of unique elements.

Summary

The Problem

The Hash Table

The HyperLogLog

Hashing

Hashing

- What is *hashing*?

Hashing

- What is *hashing*?
- Applying a function $h(x)$ to every element in $\mathbb{S}$, and storing the result in $\mathbb{V}$.

Hashing

- What is *hashing*?
- Applying a function $h(x)$ to every element in $\mathbb{S}$, and storing the result in $\mathbb{V}$.
- Ideally, $h(x)$ is

Hashing

- What is *hashing*?
- Applying a function $h(x)$ to every element in $\mathbb{S}$, and storing the result in $\mathbb{V}$.
- Ideally, $h(x)$ is
  1. Onto (surjective)

Hashing

- What is *hashing*?
- Applying a function $h(x)$ to every element in $\mathbb{S}$, and storing the result in $\mathbb{V}$.
- Ideally, $h(x)$ is
  1. Onto (surjective)
  2. One-to-one (injective)

Hashing

- What is *hashing*?
- Applying a function $h(x)$ to every element in $\mathbb{S}$, and storing the result in $\mathbb{V}$.
- Ideally, $h(x)$ is
  1. Onto (surjective)
  2. One-to-one (injective)
- We can ignore the duplicate values in $\mathbb{V}$.

Solving the Pokémon Problem

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.
- The hash function:
  1. Sum up the ASCII value of each character in a Pokémon's name. Call this n.

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.
- The hash function:
  1. Sum up the ASCII value of each character in a Pokémon's name. Call this n.
  2. Add n to the Pokémon's corresponding National Pokédex number. Call this m.

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.
- The hash function:
    1. Sum up the ASCII value of each character in a Pokémon's name. Call this n.
    2. Add n to the Pokémon's corresponding National Pokédex number. Call this m.
    3. Find $m \mathit{mod}$ 721.

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.
- The hash function:
  1. Sum up the ASCII value of each character in a Pokémon's name. Call this n.
  2. Add n to the Pokémon's corresponding National Pokédex number. Call this m.
  3. Find $m \ mod \ 721$.
  4. Results:

Solving the Pokémon Problem

- How can we solve the Pokémon Problem using a hash?
- Create a hash function that turns a given Pokémon into a numerical value
- Store the result in $\mathbb{V}$ if it is not already there.
- The hash function:
  1. Sum up the ASCII value of each character in a Pokémon's name. Call this n.
  2. Add n to the Pokémon's corresponding National Pokédex number. Call this m.
  3. Find $m \; mod \; 721$.
  4. Results:
     - On average: 461 unique encounters

Problems With The Hash Table

Problems With The Hash Table

- Memory Intensive
  - Pokémon problem only dealt with a set $\mathbb{S}$ of size 6000

Problems With The Hash Table

- Memory Intensive
  - Pokémon problem only dealt with a set $\mathbb{S}$ of size 6000
  - Twitter Problem deals with $\mathbb{S}$ of size 200,000,000.

Problems With The Hash Table

- Memory Intensive
  - Pokémon problem only dealt with a set $\mathbb{S}$ of size 6000
  - Twitter Problem deals with $\mathbb{S}$ of size 200,000,000.
  - Collisions and collision policies also add to the amount of memory required.

Summary

The Problem

The Hash Table

The HyperLogLog

The Algorithm

The Algorithm

1. Create a bitmap in memory. We will call this $\mathbb{V}$.

The Algorithm

1. Create a bitmap in memory. We will call this $\mathbb{V}$.
2. For each value s is $\mathbb{S}$, hash s to a binary number.

The Algorithm

1. Create a bitmap in memory. We will call this $\mathbb{V}$.
2. For each value s is $\mathbb{S}$, hash s to a binary number.
   - We will use a *Murmur Hash* to do this.

The Algorithm

1. Create a bitmap in memory. We will call this $\mathbb{V}$.
2. For each value s is $\mathbb{S}$, hash s to a binary number.
   - We will use a *Murmur Hash* to do this.
3. Analyze the first sequence of 0's in the binary value, and store the number of leading 0's into the bitmap.
   - If the sequence of 0's has been seen before, there is a high probability that the term is a duplicate

The Algorithm

1. Create a bitmap in memory. We will call this $\mathbb{V}$.
2. For each value s is $\mathbb{S}$, hash s to a binary number.
   - We will use a *Murmur Hash* to do this.
3. Analyze the first sequence of 0's in the binary value, and store the number of leading 0's into the bitmap.
   - If the sequence of 0's has been seen before, there is a high probability that the term is a duplicate
4. Take the harmonic average of all the totals in the bitmap.

The Math

The Math

- How much memory is required?

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2\left(\log_2\left(M\right)\right)$$

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2\left(\log_2\left(M\right)\right)$$

- M is the size of the original set of data ($\mathbb{S}$).

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2(\log_2(M))$$

- M is the size of the original set of data ($\mathbb{S}$).
- For the Twitter Problem:

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2(\log_2(M))$$

- M is the size of the original set of data ($\mathbb{S}$).
- For the Twitter Problem:

$$\begin{aligned}\text{Memory Required} &\approx \log_2(\log_2(200{,}000{,}000 \times 10)) \\ &\approx 4.94kb\end{aligned}$$

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2\left(\log_2\left(M\right)\right)$$

- M is the size of the original set of data ($\mathbb{S}$).
- For the Twitter Problem:

$$\begin{aligned}\text{Memory Required} &\approx \log_2\left(\log_2\left(200,000,000 \times 10\right)\right)\\ &\approx 4.94kb\end{aligned}$$

- What is the predicted error?

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2\left(\log_2\left(M\right)\right)$$

- M is the size of the original set of data ($\mathbb{S}$).
- For the Twitter Problem:

$$\begin{aligned}\text{Memory Required} &\approx \log_2\left(\log_2\left(200,000,000 \times 10\right)\right)\\ &\approx 4.94kb\end{aligned}$$

- What is the predicted error?

$$\text{Error} = \frac{\sqrt{3\log\left(2\right) - 1}}{m}$$

The Math

- How much memory is required?

$$\text{Memory Required} = \log_2\left(\log_2\left(M\right)\right)$$

- M is the size of the original set of data ($\mathbb{S}$).
- For the Twitter Problem:

$$\begin{aligned}\text{Memory Required} &\approx \log_2\left(\log_2\left(200,000,000 \times 10\right)\right)\\ &\approx 4.94kb\end{aligned}$$

- What is the predicted error?

$$\text{Error} = \frac{\sqrt{3\log\left(2\right) - 1}}{m}$$

- m is the number of spaces in the bitmap ($\mathbb{V}$).

Solving the Twitter Problem

Solving the Twitter Problem

- How can we process count 200,000,000 hashtags on an average computer?

Solving the Twitter Problem

- How can we process count 200,000,000 hashtags on an average computer?
- We can lower the sample size and apply a best fit line to the data.

Solving the Twitter Problem

- How can we process count 200,000,000 hashtags on an average computer?
- We can lower the sample size and apply a best fit line to the data.
    1. For 24 hours, gather 2000 tweets containing “#” every 2 minutes.

Solving the Twitter Problem

- How can we process count 200,000,000 hashtags on an average computer?
- We can lower the sample size and apply a best fit line to the data.
  1. For 24 hours, gather 2000 tweets containing "#" every 2 minutes.
  2. Using the HyperLogLog, determine the unique number of total hashtags every time a new sample is gathered.

Results

Results

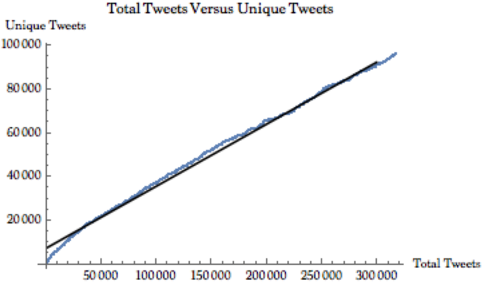

Figure: $0.284356x + 7361.39$

Results

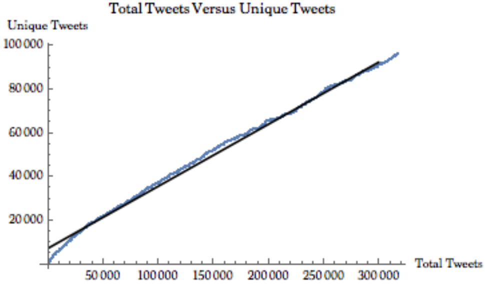

Figure: $0.284356x + 7361.39$

- Plugging in 200,000,000 gives us

Results

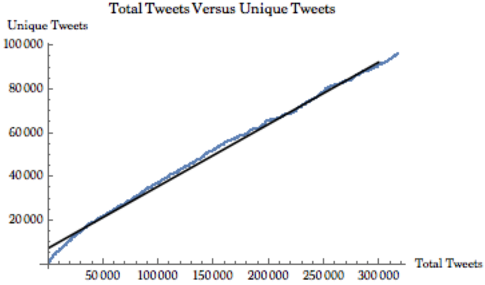

Figure: $0.284356x + 7361.39$

- Plugging in 200,000,000 gives us
- 5.68785×10^7 unique hashtags.