Flajolet-Martin Sketches

Omar Zeidan

Count-distinct problem

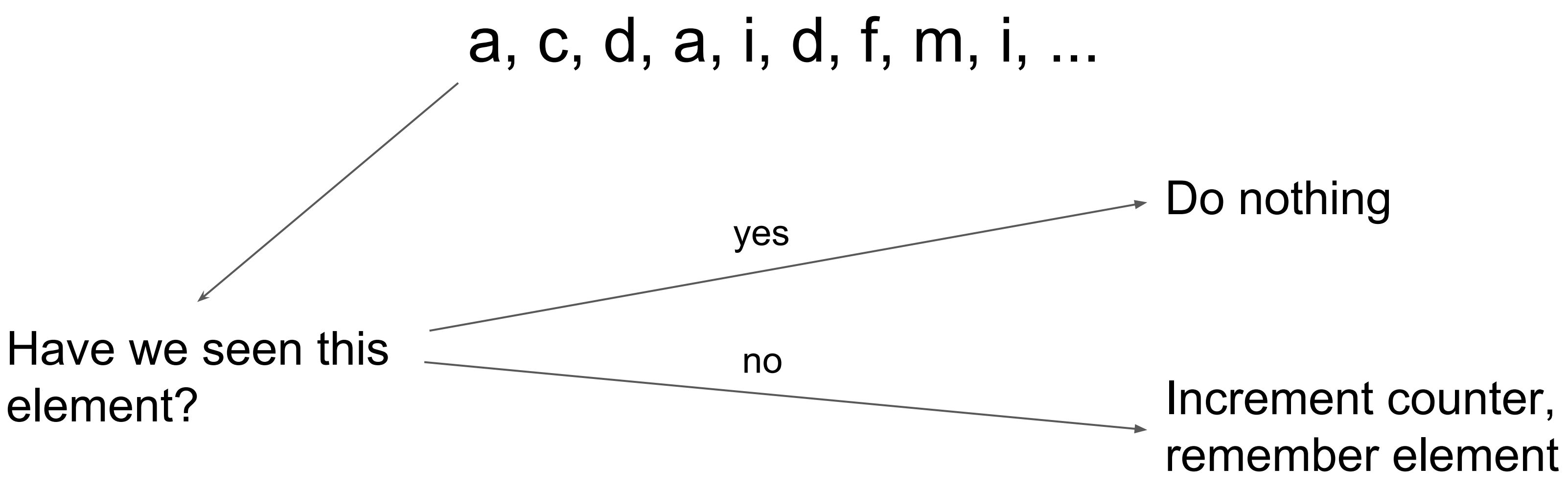
Given is a stream of elements from a finite dataset:

a, c, d, a, i, d, f, m, i, ...

How many distinct elements are in the set?

```sql
SELECT COUNT(DISTINCT x)
FROM table;
```

Naive Approach



- O(n) in memory ⟶ large datasets don't fit into RAM
- Performance depends on data structure

Flajolet-Martin algorithm [1]

Better approach: smart estimation (sketch)

Idea: count leading zeros

Bit Pattern	Probability (if uniformly distributed)
0XXXXXXX	1/2
00XXXXXX	1/4
000XXXXX	1/8

example: 4 leading zeros → $2^4 = 16$

Flajolet-Martin algorithm

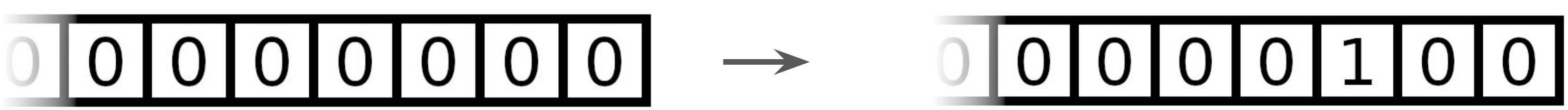
1. Hash element (uniformly)

a → 00101101

2. Determine count of leading zeros (clz)

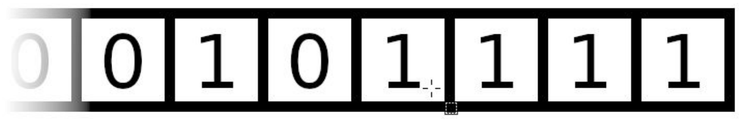
00101101 → 2

3. Set bitmap at index clz



Flajolet-Martin algorithm

After every element is processed:



Get smallest index that is not set: 4

Final estimate: $2^R/\phi$ (***R*** is the smallest index, **φ** is the correction factor)

In this case: **20,684929736**

Flajolet-Martin algorithm

Results:

- Sensible estimate of the ‘cardinality’
- Constant memory footprint (typically 4 or 8 bytes)

But:

- Still some room of improvement for the accuracy
- Results have a high variation (standard deviation =~ 1.12)

(Hyper)LogLog [2]

Builds on Flajolet-Martin algorithm

Improvements:

1. Split elements into 2^p substreams, where ***p*** is the 'precision'

first p bits

10111010 → Substream 5

00101101 → Substream 1

(Hyper)LogLog

Process every substream similarly as with Flajolet-Martin

2. Start counting the leading zeros after the first p bits

10100110 → 2

3. Remember highest clz for every substream

HyperLogLog

4. Calculate the estimate, the harmonic mean of all maximum clzs:

Harmonic Mean → $$\alpha_m \cdot m^2 \cdot \left(\sum_{j=1}^{m} 2^{-M[j]} \right)$$

$$\alpha_m := \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}$$

(Precalculated in my implementation)

HyperLogLog

Results

"The HyperLogLog algorithm is able to estimate cardinalities of > 10^9 with a typical accuracy of 2%, using 1.5 kB of memory." [2]

- Accuracy is improved, outliers 'hurt' the result less
- Memory footprint is still very small:

typically 4 <= p <= 16 ⟶ max. 2^{16} = 65536 buckets

typically 32 bit hashes ⟶ 5 bits per bucket (at p = 16)

⟶ memory footprint is 40kb at highest precision

HyperLogLog

Some more improvements:

1. Small range corrections for small *n* (linear counting)
2. Large range corrections when *n* starts to approach 2^{32}

HyperLogLog++ [3]

Does everything HyperLogLog does, and:

- Uses a 64 bit hash function:

  Stays accurate for higher cardinalities, no need for large range corrections

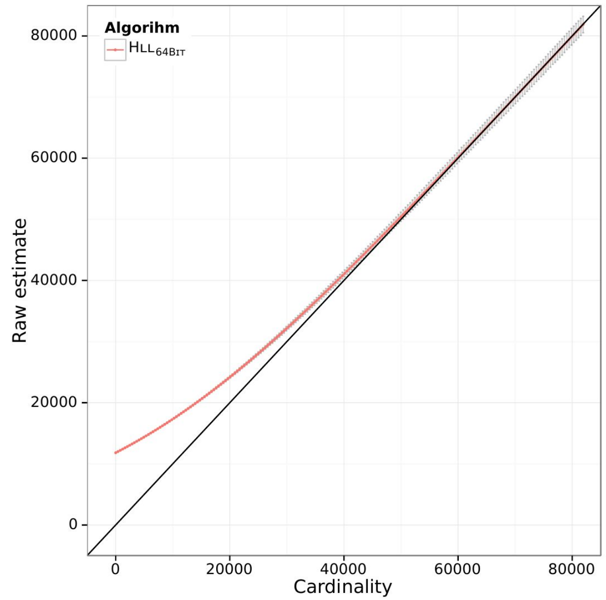
- Eliminates bias:

  HLL is biased for small cardinalities

  →

  Empirically measure the bias and

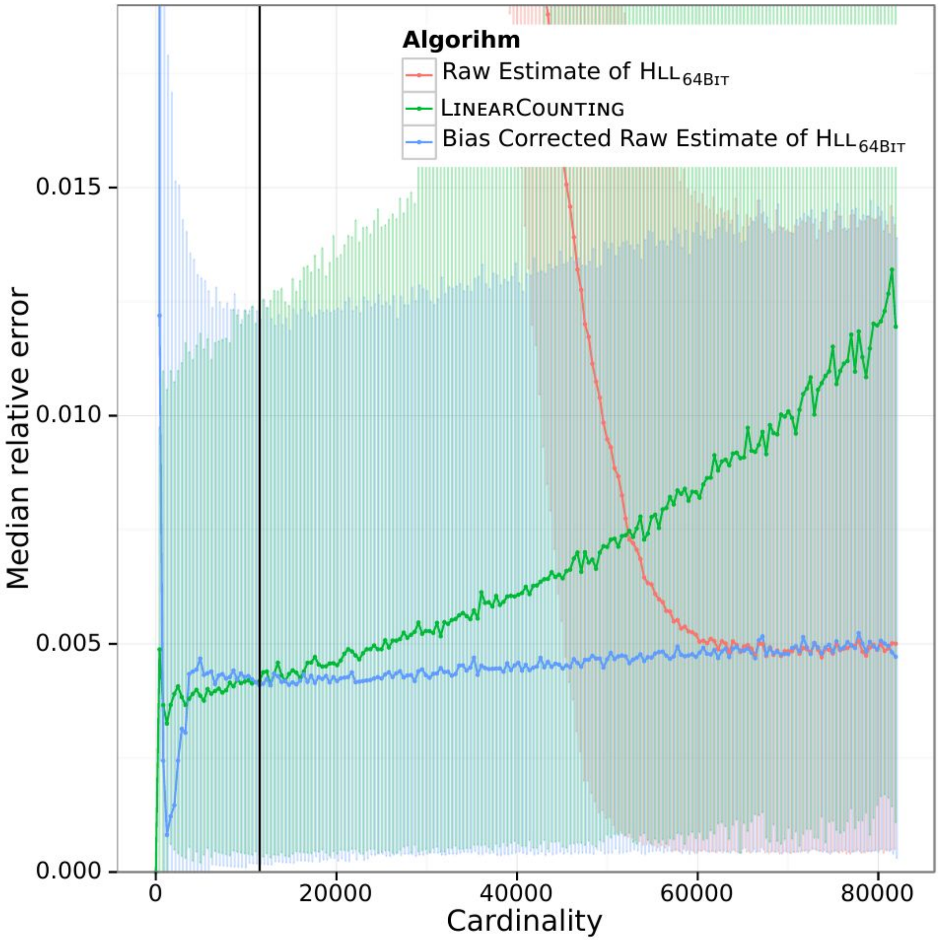
  subtract it from our estimate



Estimates of HLL-64bit [3]

HyperLogLog++

- Uses the bias corrected estimate OR linear counting, depending on the cardinality



Effects of the bias correction [3]

HyperLogLog++

- Compresses the buckets in a complex manner to further save more memory

Memory footprint

- Flajolet-Martin:

  8 bytes

- HyperLogLog:

  2^p * 5 bits

- (my) HyperLogLog++:

  2^p * 6 bits (+ bias elimination data)

Benchmarks: Performance (one element)

At sample size 67.108.864

- Flajolet-Martin:
  32.6233 ns 25.9636 ns (without hash)

- HyperLogLog/HyperLogLog++:
  32.4092 ns 25.8448 ns (without hash)

Benchmarks: Accuracy

At cardinality 42000 with 10000 data points and p = 14

	Mean Relative Error (%)	Standard Deviation of Error (%)
Flajolet-Martin	59.5372	41.2034
HyperLogLog	2.17655	0.628657
HyperLogLog++	0.533857	0.407194

Benchmarks: HyperLogLog++ Accuracy

At cardinality 65536 with 2000 data points

p	Mean Relative Error (%)	Standard Deviation of Error (%)
4	21.8203	17.9068
6	10.5101	8.26347
8	5.28637	5.28815
10	2.68807	2.13092
12	1.25001	0.935579
14	0.567714	0.424141
16	0.257173	0.195455

Sources

[1] Flajolet, Philippe; Martin, G. Nigel (1985). "Probabilistic counting algorithms for data base applications"

[2] Flajolet, Philippe; Fusy, Éric; Gandouet, Olivier; Meunier, Frédéric (2007). "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm"

[3] S Heule, M Nunkesser, A Hall (2013). "HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm"