Theory behind the Technology

Thoughts on computer science theory, math and technology.

Ravi Bhide

Ravi is an armchair futurist and an aspiring mad scientist. His mission is to create simplicity out of complexity and order out of chaos.

View my complete profile

SUNDAY, APRIL 3, 2011

Flajolet-Martin algorithm

Flajolet-Martin algorithm approximates the number of unique objects in a stream or a database in one pass. If the stream contains n elements with m of them unique, this algorithm runs in $O(n)$ time and needs $O(log(m))$ memory. So the real innovation here is the memory usage, in that an exact, brute-force algorithm would need $O(m)$ memory (e.g. think "hash map").

As noted, this is an approximate algorithm. It gives an approximation for the number of unique objects, along with a standard deviation σ, which can then be used to determine bounds on the approximation with a desired maximum error ϵ, if needed.

Given below are the following:

- intuition behind the algorithm
- the algorithm itself
- a java-based implementation
- some results using that implementation and
- some closing thoughts.

Intuition

If we had a good, random hash function that acted on strings and generated integers, what can we say about the generated integers? Since they are random themselves, we would expect:

- $1/2$ of them to have their binary representation end in 0 (i.e. divisible by 2),
- $1/4$ of them to have their binary representation end in 00 (i.e. divisible by 4)
- $1/8$ of them to have their binary representation end in 000 (i.e. divisible by 8)
- and in general, $1/2^n$ of them to have their binary representation end in 0^n.

Turning the problem around, if the hash function generated an integer ending in 0^m bits (*and* it also generated integers ending in 0^{m-1} bits, 0^{m-2} bits, ..., 0^1 bits), intuitively, the number of unique strings is around 2^m.

To facilitate the above, this algorithm maintains 1 bit for each 0^i seen - i.e. 1 bit for 0, another for 00, another for 000, and so on. The output of the algorithm is based on the maximum of consecutive 0^i seen.

The Flajolet-Martin algorithm

This is an informal description. Formal treatment can be found in the original paper listed in the reference section.

1. Create a bit vector (bit array) of sufficient length L, such that $2^L > n$, the number of elements in the stream. Usually a 64-bit vector is sufficient since 2^{64} is quite large for most purposes.
2. The i-th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in 0^i. So initialize each bit to 0.
3. Generate a good, random hash function that maps input (usually strings) to natural numbers.
4. Read input. For each word, hash it and determine the number of trailing zeros. If the number of trailing zeros is k, set the k-th bit in the bit vector to 1.
5. Once input is exhausted, get the index of the first 0 in the bit array (call this R). By the way, this is just the number of consecutive 1s (i.e. we have seen 0, 00, ..., 0^{R-1} as the output of the hash function) plus one.
6. Calculate the number of unique words as $2^R/\phi$, where ϕ is 0.77351. A proof for this can be found in the original paper listed in the reference section.
7. The standard deviation of R is a constant: $\sigma(R) = 1.12$. In other words, R can be off by about 1 (for 68% of the observations, off by 2 for about 95% of the observations, off by 3 for 99.7% of the observations using the Empirical rule of statistics). This implies that our count can be off by a factor of 2 (for 68% of the observations, off by 4 for 95% of the observations and so on).

To improve accuracy of this approximation algorithm, we do the following:

8. (Averaging) Use multiple hash functions and use the average R instead.
9. (Bucketing) Averages are susceptible to large fluctuations. So use multiple buckets of hash functions from the above step and use the median of the average R. This gives fairly good accuracy.
10. Overall accuracy of this algorithm can be tuned by using appropriate number of hash functions in the averaging and bucketing steps. Of course, if more accuracy is desired, more hash functions need to be used, which implies higher computation cost.

Java-based implementation

The code can be found here: FlajoletMartin.java. It uses lucene-core-3.0.3.jar or better.

Results using the above implementation

- Wikipedia article on "United States Constitution" had 3978 unique words. When run ten times, Flajolet-Martin algorithm reported values of 4902, 4202, 4202, 4044, 4367, 3602, 4367, 4202, 4202 and 3891 for an average of 4198. As can be seen, the average is about right, but the deviation is between -400 to 1000.
- Wikipedia article on "George Washington" had 3252 unique words. When run ten times, the reported values were 4044, 3466, 3466, 3466, 3744, 3209, 3335, 3209, 3891 and 3088, for an average of 3492.

Closing thoughts

- The Flajolet-Martin algorithm approximates the number of unique elements quite well, using just O(log m) memory, where m is the number of unique words.
- During implementation, it was observed that this algorithm is quite sensitive to the hash function parameters. The hash functions suggested in the original paper ($h(x) = (M + N \sum ord(x_j) * 128^j) \mod (2^L)$) work properly only when n is odd. Otherwise, the algorithm always reports the number of unique elements as 1 or 2! I chose both M and N as odd, as can be seen in the implementation.

References

1. Flajolet, P. and Martin, N., *Journal of Computer and System Sciences*, 1985. PDF - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.3869&rep=rep1&type=pdf
2. J. Ullman and A. Rajaraman, *Mining of Massive Datasets*, *Chapter 3* - available at http://infolab.stanford.edu/~ullman/mmds/ch4.pdf

Posted by Ravi Bhide at 6:02 PM

4 comments:

ppsly November 21, 2011 at 9:12 PM

great article :)

Reply

megha February 5, 2012 at 6:00 PM

thanks for the simple explanation!

Reply

Rahul November 22, 2012 at 10:07 PM

A wonderful explanation. Only one thought : If you can't explain it simply, you don't understand it well enough. – Albert Einstein

You have explained it very lucidly.
Thanks!!

Reply

Anton Zuenko March 30, 2014 at 11:19 AM

Thanks for the article!

Reply

Subscribe to: Post Comments (Atom)