Analysis

yourGrand

November 30, 2024

Consider a stream $a_1, \ldots, a_m$ where each $a_i \in \{1, \ldots, n\}$

Problem 1. Give a randomised streaming algorithm which approximates the sum $a_1 + \cdots + a_m$ using $O(\log \log m + \log n)$ space

Algorithm 1 Approximate Stream Sum

```
procedure Increment(counters, a_i, c)                    ▷ Increment operation
    rands ← array of random values between 0 and 1 of size c
    probs ← (2^(−counters)) × a_i                         ▷ Element-wise computation

    for each element c_j in counters do
        if rands[j] < probs[j] then
            c_j ← c_j + 1
        end if
    end for
end procedure

function Estimate(counters)                               ▷ Estimate total
    estimates ← 2^counters − 1                            ▷ Element-wise computation
    return mean(estimates)
end function

function Approximate_Stream_Sum(stream, c)
    Initialise counters ← array of zeros of size c

    for each element a_i in stream do
        Increment(counters, a_i, c)
    end for

    return Estimate(counters)
end function
```

Theorem 1. The streaming algorithm provides an unbiased estimator of the stream sum

$$S = \sum_{i=1}^{m} a_i.$$

Proof. The algorithm aims to approximate the stream sum by incrementing counters with a certain probability and using the counter value to estimate the sum. Let $X_i = 2^{C_m^{(i)}} - 1$ represent the estimate from the i-th counter at the end of the stream. Let the value of the counter at time t be C_t.

Step 1: Single Counter Estimate For a single counter, the expected change in $2^{C_t} - 1$ at time t is given by:

$$\Delta_t = \begin{cases} 2^{h+1} - 2^h, & \text{if increment occurs} \\ 0, & \text{otherwise.} \end{cases}$$

The probability of an increment, conditioned on the counter value $C_{t-1} = h$, is:

$$P(\text{increment} \mid C_{t-1} = h) = a_t \cdot 2^{-h}.$$

The expected value of Δ_t is then:

$$\mathbb{E}[\Delta_t \mid C_{t-1} = h] = (2^{h+1} - 2^h) \cdot P(\text{increment} \mid C_{t-1} = h).$$

Substituting $P(\text{increment})$:

$$\mathbb{E}[\Delta_t \mid C_{t-1} = h] = (2^{h+1} - 2^h) \cdot (a_t \cdot 2^{-h}) = a_t \cdot (2 - 1) = a_t.$$

Therefore, the expected change in $2^{C_t} - 1$ at time t is exactly a_t.

Step 2: Final Estimate By linearity of expectation:

$$\mathbb{E}[2^{C_m} - 1] = \mathbb{E}\left[\sum_{t=1}^{m} \Delta_t\right] = \sum_{t=1}^{m} \mathbb{E}[\Delta_t] = \sum_{t=1}^{m} a_t = S.$$

Step 3: Multiple Counters The final estimator Y is the mean of c independent random variables $X_1, X_2, \ldots, X_c$ each representing an estimate from one counter. When using c independent counters, the final estimate is:

$$Y = \frac{1}{c} \sum_{i=1}^{c} X_i, \quad \text{where } X_i = 2^{C_m^{(i)}} - 1.$$

Since each counter is independent and unbiased, the estimator Y is also unbiased:

$$\mathbb{E}[Y] = \frac{1}{c} \sum_{i=1}^{c} \mathbb{E}[X_i] = \frac{1}{c} \sum_{i=1}^{c} S = S.$$

Step 4: Variance of the Estimator The variance of a single counter estimate $X_1 = 2^{C_m^{(1)}} - 1$ is:

$$\mathrm{Var}[X_1] = \mathbb{E}[X_1^2] - \mathbb{E}[X_1]^2.$$

If $X_1, X_2, \ldots, X_c$ are independent random variables with the same variance $\mathrm{Var}[X_1]$, the variance of their mean is:

$$\mathrm{Var}[Y] = \mathrm{Var}\left(\frac{1}{c}\sum_{i=1}^{c} X_i\right).$$

By the properties of variance:

$$\mathrm{Var}[Y] = \frac{1}{c^2}\sum_{i=1}^{c} \mathrm{Var}[X_i].$$

Since all the X_i have the same variance $\mathrm{Var}[X_1]$ because they are derived from identical update rules and are independent, we get:

$$\mathrm{Var}[Y] = \frac{1}{c^2} \cdot c \cdot \mathrm{Var}[X_1] = \frac{\mathrm{Var}[X_1]}{c}.$$

Conclusion: The algorithm provides an unbiased estimator for S, with variance that decreases as $O(1/c)$. By adjusting c, the trade-off between accuracy and space complexity can be controlled. □

Space Complexity Analysis:

- In this algorithm, we increment each counter with a probability of $(2^{-\text{counters}}) \times a_i$, where a_i is the value from the stream.
- The effect of multiplying by a_i is equivalent to splitting each value in the stream into 1s. For example, a value of 6 would be represented as six 1s, and each 1 would increment the counter with probability $2^{-\text{counter}}$, as in the original Morris's algorithm.
- Morris's algorithm achieves space efficiency by approximating x with a space complexity of $O(\log(\log(x)))$, where x is the value being approximated. In our case, the value x is $m \times n$, where m is the number of elements in the stream, and n is the maximum value of any element in the stream.
- Therefore, the space complexity of our algorithm in bits is $O(c \times \log(\log(m \times n)))$, where c is the number of counters.
- Since the number of counters c is a constant, the space complexity in bits simplifies to $O(\log(\log(m \times n))) = O(\log(\log(m) + \log(n)))$, which is smaller than the upper bound $O(\log(\log(m)) + \log(n))$.

Thus, the space complexity of the algorithm is $O(\log(\log(m) + \log(n)))$.

Note: The implementation of this algorithm in Python can be found on my GitHub:
`https://github.com/yourGrand/approximate_stream_sum`