Streaming Algorithms

Meng-Tsung Tsai

03/27/2018

Reminder

Programming Assignment #0 is due by 23:59, March 30. You need to submit your program on OJ (https://oj.nctu.me).

Written Assignment #1 is due by 23:59, Apr 10. You need to LaTex your solution and submit it on New E3 (https://e3new.nctu.edu.tw).

References

- "Pairwise Independence and Derandomization," Luby and Wigderson (2005)
- "The space complexity of approximating the frequency moments," Alon, Matias, Szegedy (Gödel Prize 2005)

Frequency Moment

Problem Definition

Input: a sequence of n (possibly repeat) elements $a_1, a_2, \ldots, a_n$ in $[U] = \{1, \ldots, U\}$ and an integer k. Define

$$m_j = \sum_{1 \le i \le n} \mathbf{1}[a_i = j].$$

Output:

$$F_k = \sum_{i \in U} (m_j)^k.$$

Goal: use $o(n \log |U|)$ bits.

> F_0 is the count of distinct elements in the input.
> Have we seen an application of F_0?

Estimating F_0

Fajolet-Martin Sketch

Suppose that we have a succinct independent hash function

$$h : U \to [n]$$

so that

$$\Pr[\cap_{i \in S} h(s_i) = x_i] = 1/n^{|S|} \text{ for every } x_1, x_2, \ldots, x_{|S|} \in [n], \text{ all } S \subseteq U,$$

i.e. fully independent.

> If we don't require h to be succinct, is there an simple construction of h?

Algorithm

```
bool C[log n] = {0}; // initialize a bit vector of length log n to zero's

foreach a_i {
   C[ctz(h(a_i))] = 1; // ctz(x) returns the number of trailing zero's in
                          the binary representation of x
}

r ← the smallest i so that C[i] = 0;

return 2^r;
```

> Note that $E[C[i]] = F_0/2^i$.
> If $2^i \gg F_0$, then C[i] is likely to be 0.
> If $2^i \ll F_0$, then C[i] is likely to be 1.
> Hence, 2^r is a good estimate.

Issues

It seems that storing a fully independent hash function needs

$$\Omega(|U| \log n) \text{ bits,}$$

but we require our algorithm to use $o(|U| \log n)$ bits.

Use pairwise independent hash function as an alternative. It is easy to bound the variance.

Flajolet-Martin Sketch (AMS variation)

```
Let p be the smallest prime > |U|;
Sample α, β independently, uniformly at random from [p];
Let h(a_i) = α a_i + β; // pairwise independent hash function

bool C[log n] = {0}; // initialize a bit vector of length log n to zero's

foreach a_i {
   C[ctz(h(a_i))] = 1; // ctz(x) returns the number of trailing zero's in
                          the binary representation of x
}

r ← the largest i so that C[i] = 1;

return 2^r;
```

We claim that, for every $c > 2$
$\Pr[\text{the ratio between } 2^r \text{ and } F_0 \notin [1/c, c]] \le 2/c$.

Analysis

Let $z_i = ctz(h(a_i))$ for each $i \in [n]$;

Because h is a pairwise independent function, for every $a_i \neq a_j$ we get

$$\text{(1) } \Pr[z_i \ge r] = 1/2^r$$
$$\text{(2) } \Pr[z_j \ge r] = 1/2^r$$
$$\text{(3) } \Pr[z_i \ge r \text{ and } z_j \ge r] = 1/2^{2r}$$

For each distinct element x in $\{a_1, a_2, ..., a_n\}$, let W_x be the indicator variable denoting whether $ctz(h(x)) \ge r$. Let

$$Z = \sum_x W_x.$$

By definition, $E[Z] = F_0/2^r$ and $Var[Z] = \sum_x Var[W_x] = F_0(1/2^r)(1-1/2^r)$

Case 1: if $2^r > cF_0$

We hope $C[r] = 0$;

By Markov Inequality,

$$\Pr[C[r] = 1] \le \Pr[Z > 0] \le E[Z] = F_0/2^r < 1/c.$$

Case 2: if $2^r < F_0/c$

We hope C[r] = 1;

By Chebyshev Inequality,

$$\Pr[C[r] = 0] = \Pr[Z = 0] \leq \mathrm{Var}[Z]/(E[Z])^2 = \frac{F_0(1-1/2^r)(1/2^r)}{(F_0(1/2^r))^2}$$

$$< 2^r/F_0 < 1/c$$

By the Union Bound, the total failure probability is at most 2/c.

Results

One an estimate F_0 to within the error range [1/c, c] for any constant c > 2 using O(n) time and O(log |U| + log log n) bits.

Allow Deletions

Let p be the smallest prime > |U|;
Sample α, β independently, uniformly at random from [p];
Let $h(a_i) = \alpha\, a_i + \beta$; // pairwise independent hash function

```
int C[log n] = {0}; // initialize an integral array of length log n to 0's

foreach (+/-) a_i {
  C[ctz(h(a_i))] (+/-)= 1; // ctz(x) returns the number of trailing 0's in
                           the binary representation of x
}

r ← the largest i so that C[i] ≥ 1;

return 2^r;
```

Space requirement increases by a factor of log n.

Decreasing the failure probability

This can be achieved as before.

Run the algorithm in parallel and take the median of the individual outputs, i.e. median of $\{2^{r1}, 2^{r2}, \ldots, 2^{r^*}\}$.

Shrinking the error range

We currently partition the range of the hash function into subsets of size

1, 2, 4, ...

To shrink the error range, we can parition the range into subsets of size

$1, 1+\varepsilon, (1+\varepsilon)^2, \ldots$

Exercise 1

How to estimate F_1?