Advanced Algorithms
Lecture 12

Lecture by Shay Mozes
Typeset by Steven Karas

2017-01-29
Last edited 20:52:38 2017-01-29

1 Sublinear Algorithms

Streaming, sketching, property testing.

In many modern applications, the input is huge, and we can't possibly store or process all of the data. Even if we can store it, we may not be able to process it in a reasonable time. We still want to perform meaningful computations, so we need to change our computational model.

1.1 Sublinear space - streaming model

Input processed one element at a time. Space consumed should be far less than the size of the input (e.g. polylog). Sometimes, we allow multiple passes over the data.

Example: Routing Internet backbone routers process millions of packets per second. Want to compute statistics on network activity. Can't store all the data going through.

Example: Web Search Determine statistics for ad placement. Huge database, constantly changing. Want to compute in single (or few) passes.

Similarities to Online algorithms We get data one element at a time, and the future is unknown. Usually, we will analyze the worst-case stream.

Differences from Online algorithms Can delay decisions until output. Memory is limited, and is sublinear by definition.

Example: Estimating Frequency Given an input $x_1, ..., x_n$, provide $F(t)$ which is the number of times t appears in the input. We can think of $F(t)$ as a vector of length n where $F(t) \in [m]$. But this solution is too inefficient for large $m = poly(n)$.

We want to keep a "sketch" of data.

1.1.1 Misra-Gries

Invented in 1982 by Misra and Gries. Maintain a set S of k counters. If $x_i \in S$: increment counter for x_i. If there is space left in S, add x_i to S. Otherwise, decrement all counters in S and remove any counters that reach 0.

$F(t)$ is a counter for t if $t \in S$, otherwise 0.

Space

$$k \cdot (\log n + \log m) = O(k \log n)$$

Accuracy Let $F'(t)$ be the frequency as reported by this algorithm. Let $F(t)$ be the actual frequency.

$$F'(t) \leq F(t)$$

$F'(t) < F(t)$ can happen in two cases:

1. t wasn't in S when it appeared and S was full.
2. t was in S when some $s \notin S$ caused $F'(t)$ to be decremented.

Both cases cause k decrements, and the total number of occurrences is at most m/k. Therefore, because the sum of decrements cannot be more than the increments, it follows that the additive error is at most m/k. [1]

1.1.2 Turnstile Model

Input consists of pairs: x_i, c_i, where c_i indicates the change in x_i. c_i can be negative. Total at every point must be non-negative.

We will show a randomized algorithm with an additive error of $\varepsilon \cdot |F|_1$, where $|F|_1 = \sum |c_i|$ guaranteed with probability $1 - \delta$.

1.2 Count Min Sketch

Invented by Cormode and Muthukrishnan '05.

Maintain t arrays with k counters. For array i choose a random hash function $h_i : [n] \to [k]$. On input (x_j, c_j), for $i = 1...t$: add c_j to entry $h_i(x_j)$ of array i.

$$F'(x) = \min_i A[i, h_i(x)]$$

Accuracy Let $k = 2/\varepsilon$ and $t = \log(1/\delta)$.

$$\Pr[|F'(x) - F(x)| > \varepsilon \cdot |F|_1] < \delta$$

[1] A visual example was made on the whiteboard

Proof Construct an element a such that $X_i = \underbrace{A[i, h_i(a)]}_{\text{The counter for a in row i}} - F(a)$ is the error for a. Let $Y_{ib} = \{h_i(a) = h_i(b)\}$ be a random variable that indicates a hash collision between a and b in row i. We used random hash functions h_j, so it follows that $E[Y_{ib}] = \frac{1}{k}$.

$$E[X_i] = \sum_{b \neq a} \Pr[Y_{ib} = 1] F(b) = \sum_{b \neq a} E[Y_{ib}] F(b) \leq \frac{1}{k} \sum_b F(b) \leq \frac{1}{k} |F|_1$$

$$\Pr[X_i > \overbrace{\frac{2}{k}}^{=\varepsilon} |F|_1] \leq \frac{E[X_i]}{\frac{2}{k}|F|_1} \leq \frac{\frac{1}{k}|F|_1}{\frac{2}{k}|F|_1} = \frac{1}{2}$$

Space

$$O(t \cdot k \cdot \log |F|_1 + t \cdot \log k \cdot n) = \Omega(n)$$

1.2.1 Pairwise-Independent Hashing

Let H be a family of functions from $[0...n-1]$ to a $[0...k-1]$. We say that H is pairwise independent if for every $x \neq y \in [0...n-1]$ and $c, d \in [0...k-1]$: When h is chosen uniformly from H, $\Pr_h[h(x) = c \text{ AND } h(y) = d] = \frac{1}{k^2}$.

There is a family of pairwise-independent functions that require just $\log(n) + \log(k)$ space.

Note that $\Pr_h[h(x) = h(y)] = 1/k$, and therefore Count-min sketches are efficient.

Construction Assume that k is prime and $n = k^r$. Think of $u \in [0...n-1]$ as an integer in base k such that $u = \sum_i u_i k^i$. The following family H is pairwise independent:

$$h_{\bar{a},b}(u) = b + \sum_{i=0}^{r-1} a_i u_i \mod k \quad ; \quad 0 \leq a_i, b \leq k-1$$

To represent one function, we need to store $r + 1 = O(\log n / \log k)$ numbers, each in $\log k$ bits.

Proof We need to compute the probability that $h(u) = c$ and $h(v) = d$. If $u \neq v$, then there is some i such that $u_j \neq v_i$. Given any choice of the other a_j's, we have:

$$a_i u_i + b = \left(c - \sum_{j \neq i} a_j u_j \right) \mod k$$

$$a_i v_i + b = \left(d - \sum_{j \neq i} a_j v_j \right) \mod k$$

This set of equations has a unique solution (k is prime). This means there are k^{n-1} choices for the other a_j's and k^{r+1} choices overall.

$$\Pr[h(u) = c \text{ AND } h(v) = d] = \frac{1}{k^2}$$

1.2.2 Estimating Distinct Elements

We skipped this section, because we wanted to see a non-streaming algorithm.

1.2.3 More streaming algos

- Graph algorithms - input is sequence of edges; questions such as "how many triangles", "max-weight matching"
- Geometric algorithms - input is sequence of points; questions such as "k-center clustering"
- Multiple passes
- Random Order
- and many more

1.3 Sublinear time

We model this with us having random access to the input. Time consumed should be small compared to the input (polylog). Sample of the input (sketch) is used to deduce approximate answer. Using this for decision problems is called property testing.

1.3.1 Example: Counting the fraction of 1s

Input is a binary string s of n bits. We need to output the fraction $\mu$of 1s in s.

Algorithm Return the fraction of μ' of 1s in a sample of k locations chosen uniformly and independently at random.

$$\Pr[|\mu' - \mu| \geq \varepsilon] \leq 2e^{-2k\varepsilon^2} \quad \text{[Chernoff]}$$

1.3.2 Example: Property Testing of Monotonicity

A decision problem, where the input an array $s[]$ of distinct members. We need to output an approximate decision of whether or not the array is sorted.

A natural algorithm for this is to use the birthday paradox, but it fails horribly.

Binary search Discovered by Ergun Kannan Kumar Ravi Rubinfeld Vishwanathan. Repeat $\Theta(1/\varepsilon)$ times: Pick an element x uniformly at random. Perform a binary search for x. If x was not found, reject. If all x's found, then accept.

Claim The entries for which binary search would succeed form a sorted sequence. [2]

[2] There was a proof on the whiteboard, but I missed it

1.3.3 Additional Property Testing Examples

- Checking if an array is monotonously sorted
- Checking if a string is in a regular language
- Checking if a function is linear
- Checking if a graph is 3-colorable

It's interesting to note that some NP-hard problems can be tested in sublinear time, which can save us a lot of time.

2 Review

- Stable marriage and induction
- Divide and conquer
- Approximation algorithms
- Structured inputs - dynamic programming and tree decompositions
- Parametrized complexity - FPT, Kernalization, branching
- Linear Programming - Approximation via relaxation, LP duality
- Randomized algorithms - Markov and Chernoff bounds, sampling
- Online algorithms
- Sublinear algorithms