ECS-189A Special Topics in TCS May 9, 2023

Lecture 7

Lecturer: Slobodan Mitrović

1 Introduction

We are now switching gears to the streaming model of computation. The history of the first streaming algorithms can be traced back to the late seventies and the early eighties, attributed to works [Mor78, MP80, FM85]. This research direction was popularized thanks to the seminal work by Noga Alon, Yossi Matias, and Mario Szegedy [AMS96]. "For their foundational contribution to streaming algorithms", to quote, these authors even received a Gödel Prize in 2005. Since then, streaming algorithms have become one of the typical playgrounds for theoretical computer scientists. Being tailored for settings where the available memory is scarce, the streaming setting got significant attention in practice as well.

2 Preliminaries

2.1 Notation

We will use the notation $\tilde{O}(f)$ to hide poly-logarithmic factors in f, i.e., $\tilde{O}(f) = O(f \cdot \operatorname{poly}\log f)$.

By saying that an event $\mathcal{E}$ happens *with high probability* (whp), we refer that $\Pr[\mathcal{E}] \geq 1 - n^{-c}$ for some constant $c \geq 1$.

2.2 Streaming setting

An algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ is said to be a *streaming algorithm* if:

- $\mathcal{A}$ scans the input of size N element by element, e.g., edge by edge of a graph or entry by the entry of an array. We also say that element by element *arrive*.
- $\mathcal{A}$ outputs a solution to $\mathcal{P}$ while at any moment its memory consumption is $\operatorname{poly}\log N$ bits.

There are studies of streaming algorithms concerning how many scans/passes over the input $\mathcal{A}$ can make. In this class, we assume that $\mathcal{A}$ makes only one pass/scan over the inputs.

2.3 Semi-streaming setting

Many problems are incredibly challenging in the streaming setting. Moreover, output for many graph problems cannot be stored in $\operatorname{poly}\log n$ memory. That inspired a number of researchers to formalize a *semi-streaming* setting [FKM+05]. In this setting, if we are solving a problem on an n-vertex graph, the algorithm can use $O(n \operatorname{poly}\log n)$ bits of memory.

2.4 Probability tools

Theorem 1 (Markov's inequality)**.** *If X is a nonnegative random variable and $a > 0$, then*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a}.$$

3 Toy problems

As a warm-up, we will see two examples of simple streaming algorithms.

3.1 Sum of the input values

Problem: Given a stream of integers, we aim to compute their sum.

Solution: Initialize $S \leftarrow 0$. Scan integer by integer. When an integer x is scanned, update $S \leftarrow S + x$. After processing the stream, output S.

This is a trivial streaming algorithm that also illustrates that we have never stored any of the actual elements from the stream of integers.

3.2 Computing a maximal matching in semi-streaming

We now provide a semi-streaming algorithm (Algorithm 1) for finding a maximal matching. Algorithm 1 simulates the classical greedy one.

```
   Input : A graph $G = (V, E)$ presented as a stream of edges
1  $M \leftarrow \emptyset$
2  Label each vertex as “free”.
3  for edge $e = \{u, v\}$ on the stream do
4      if both $u$ and $v$ are “free” then
5          Label $u$ and $v$ as “taken”.
6          $M \leftarrow M \cup \{e\}$
7  return $M$
```

Algorithm 1: A streaming algorithm for finding a maximal matching.

It is easy to maintain M and the labels of the vertices in $O(n \log n)$ memory.

4 Sampling an element from a stream

Consider a stream of elements, each represented by $O(B)$ bits. Our goal is to uniformly at random sample precisely one element from this stream by using only $O(B + \log n)$ bits of memory. The idea we use is quite simple: we ensure that for a given prefix of length t of the stream containing elements $x_1, \ldots, x_t$, our algorithm in its memory stores only a single element chosen uniformly at random from that prefix. An algorithm following this idea is presented as Algorithm 2.

```
   Input : A sequence of elements $(x_1, \ldots, x_n)$ presented as a stream
1  $e \leftarrow x_1$
2  for $i = 2 \ldots n$ do
3      With probability $1/i$ replace $e$ by $x_i$.
4  return $e$
```

Algorithm 2: A streaming algorithm for sampling an element from a stream.

4.1 Memory complexity

Algorithm 2 maintains only a single element that by our assumption fits into B bits.

4.2 Correctness

Our goal is to show that it holds $\Pr[e = x_i] = 1/n$. This can be proved by induction. Let e_t be the value of element e after seeing $\{x_1, \dots, x_t\}$. For $i \in \{1, \dots, t\}$, we will prove that $\Pr[e_t = x_i] = 1/t$.

The base case holds trivially as $e_1 = x_1$ and hence $\Pr[e_1 = x_1] = 1$.

Now, consider e_{t+1} and assume that $\Pr[e_t = x_i] = 1/t$ for $i \in \{1, \dots, t\}$. Then, we have by the construction of our algorithm

$$\Pr[e_{t+1} = x_{t+1}] = \Pr[\text{Algorithm 2 replaces } e_t \text{ by } x_{t+1}] = \frac{1}{t+1}.$$

For $i \in \{1, \dots, t\}$ we have

$$\Pr[e_{t+1} = x_i] = \Pr[\text{Algorithm 2 does not replace } e_t \text{ by } x_{t+1} \text{ and } e_t = x_i] = \frac{t}{t+1} \cdot \frac{1}{t} = \frac{1}{t+1},$$

as desired.

5 The majority element

Consider a stream of elements $\{x_1, \dots, x_n\}$, where each element is represented by $O(B)$ bits. Assume that an element appears **more** than $n/2$ times in the stream; we call such an element *majority*. We aim to design an algorithm that outputs the majority element using $O(B + \log n)$ bits of memory.

```
Input : A sequence of elements $(x_1, \dots, x_n)$ is presented as a stream.
$e \leftarrow \emptyset$
$c \leftarrow 0$ /* A counter. */
for $i = 1 \dots n$ do
    if $e = x_i$ then
        $c \leftarrow c + 1$
    else if $c > 0$ then
        $c \leftarrow c - 1$
    else
        $c \leftarrow 1$
        $e = x_i$
return $e$
```

Algorithm 3: A streaming algorithm for finding the majority element, if there exists one.

5.1 Memory complexity

Algorithm 3 maintains two numbers: an element e that fits into B bits and a non-negative counter c that counts up to n. That counter uses $O(\log n)$ bits.

5.2 Correctness

Let m be the majority element. We want to show that under the assumption that m appears in the stream more than $n/2$ it holds that Algorithm 3 returns $e = m$.

This can be proved inductively on the length of the stream. When $n = 1$, the algorithm outputs m. Assume that Algorithm 3 outputs m for any stream of length less than n. Consider a stream of length n.

Assume that $x_1 \neq m$. Then, at some point after scanning x_1, the counter c has to reach 0 as x_1 is not the majority. Consider the first such moment. Assume that the algorithm saw x_1 for k times before c reached 0. It means the algorithm scanned k elements different than x_1, and hence the rest of the stream has length $n - 2k$. The rest of the stream has more than $n/2 - k$ elements m, and hence m is the majority in the rest of the stream so, by the inductive hypothesis, Algorithm 3 outputs m as desired.

Now assume that $x_1 = m$. If c is never again 0 after x_1 is scanned, then the algorithm trivially outputs m. Otherwise, c reaches 0 before the end of the stream. In that case, as in the prior case, Algorithm 3 scanned k times element m and k times elements different than m. Note that $n > 2k$ since m is the majority. In the rest of the stream of length $n - 2k \geq 1$, there are more than $n/2 - k$ appearances of m, which again by induction proves that Algorithm 3 outputs m in this case as well.

References

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996.

[FKM$^+$05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[MP80] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.