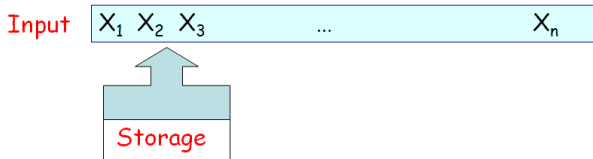


# Massive Data Algorithmics

## Lecture 13: Streaming Model

# Definition

- The input is a sequence  $\langle a_1, a_2, \dots, a_m \rangle$  receiving one by one (imagine the input is on a tape).
- $a_i$  is an element of the universe  $[n]$  where  $[n] = \{1, 2, \dots, n\}$
- Goal is to process the stream using a small amount of space  $s$
- Since  $m$  and  $n$  are to be thought of as huge, we want to make  $s$  much smaller than these
- Specifically, we want  $s$  to be sublinear in both  $n$  and  $m$
- The holy grail is to achieve  $s = O(\log n + \log m)$
- $k$ -passes streaming: allowed to make  $k$  passes over the stream



# Approximation Algorithms

- It is usually impossible to design exact algorithms in the streaming model
- Let  $A(\sigma)$  denote the output of a randomized streaming algorithm  $A$  on input  $\sigma$ . Let  $\Phi$  be the function that  $A$  is supposed to compute.
  - We say that the algorithm  $A$   $(\epsilon, \delta)$ -approximates  $\Phi$  if we have:

$$Pr\left(\left|\frac{A(\sigma)}{\Phi(\sigma)} - 1\right| > \epsilon\right) \leq \delta$$

- We say that the algorithm  $A$   $(\epsilon, \delta)$ -additively approximates  $\Phi$  if we have:

$$Pr(|A(\sigma) - \Phi(\sigma)| > \epsilon) \leq \delta$$

# Frequency Vector

- Let  $f_j = |\{i : a_i = j\}|$ , # occurrences of  $j$  in the stream
- $F = (f_1, f_2, \dots, f_n)$  is called a frequency vector
- We sometimes would like to compute  $\Phi(F)$
- You can imagine that the term  $a_i$  of the stream is of form  $(j, c)$  which means  $f_j$  must be set to  $f_j + c$  (in the standard streaming model  $c = 1$ )
- $\|F\|_k = (f_1^k + \dots + f_n^k)^{1/k}$ 
  - $\|F\|_0$  is the number of distinct elements in the stream
  - $\|F\|_1$  is the number of elements in the stream which is  $m$

# The Problem

- Majority Problem

- Input: the stream  $\sigma = \langle a_1, \dots, a_m \rangle$  where  $a_i \in [n]$
- Output: if  $\exists j: f_j > m/2$ , output  $j$ . Otherwise output null.

- Frequent Problem

- Input: the stream  $\sigma = \langle a_1, \dots, a_m \rangle$  where  $a_i \in [n]$ , and  $k$
- Output: output set  $\{j: f_j > m/k\}$ .

Note: we have to report an output upon arrival of  $a_i$  for any  $i$

# Misra-Gries Algorithm

- A deterministic algorithm to approximate  $f_j$

**Initialize** :  $A \leftarrow$  (empty associative array) ;


**Process  $j$** :

```

1 if  $j \in \text{keys}(A)$  then
2   |  $A[j] \leftarrow A[j] + 1$  ;
3 else if  $|\text{keys}(A)| < k - 1$  then
4   |  $A[j] \leftarrow 1$  ;
5 else
6   | foreach  $\ell \in \text{keys}(A)$  do
7     |  $A[\ell] \leftarrow A[\ell] - 1$  ;
8     | if  $A[\ell] = 0$  then remove  $\ell$  from  $A$  ;

```

in the end of stream not this  
program



**Output** : On query  $a$ , if  $a \in \text{keys}(A)$ , then report  $\hat{f}_a = A[a]$ , else report  $\hat{f}_a = 0$

# Analysis

- space:  $O(k(\log n + \log m))$
- approximation:  $f_a - \frac{m}{k} \leq \hat{f}_a \leq f_a$ 
  - Counter  $A[a]$  is incremented only when we process an occurrence of  $a$ . So  $\hat{f}_a \leq f_a$ .
  - Whenever  $A[a]$  is decremented (in lines 7 and 8, we pretend that  $A[j]$  is incremented from 0 to 1, and then immediately decremented back to 0), we also decrement  $k-1$  other counter, corresponding to distinct items in the stream. Then each decrement of  $A[a]$  is witnessed by a collection of  $k$  distinct items (one of which is  $a$  itself) from the stream.
  - Since the stream consists of  $m$  items, there can be at most  $m/k$  such decrements. So,  $f_a - \frac{m}{k} \leq \hat{f}_a$

# Example

Assume  $k = 3$  and  $a = 3$

$\sigma$	keys	$A[3]$
$\langle 2 \rangle$	$\{(2, 1)\}$	0
$\langle 2, 1 \rangle$	$\{(2, 1), (1, 1)\}$	0
$\langle 2, 1, 2, 2, 1 \rangle$	$\{(2, 3), (1, 2)\}$	0
$\langle 2, 1, 2, \underline{2}, \underline{1}, \underline{3} \rangle$	$\{(2, 2), (1, 1)\}$	0
$\langle 2, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3} \rangle$	$\{(2, 1)\}$	0
$\langle 2, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3}, 3 \rangle$	$\{(2, 1), (3, 1)\}$	1
$\langle 2, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3}, 3, 3 \rangle$	$\{(2, 1), (3, 2)\}$	2
$\langle \underline{2}, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3}, 3, \underline{3}, \underline{1} \rangle$	$\{(3, 1)\}$	1
$\langle \underline{2}, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3}, 3, \underline{3}, \underline{1}, 1, 1, 1, 1 \rangle$	$\{(3, 1), (1, 4)\}$	1
$\langle \underline{2}, \underline{1}, \underline{2}, \underline{2}, \underline{1}, \underline{3}, \underline{3}, \underline{3}, \underline{3}, \underline{1}, 1, 1, 1, \underline{1}, \underline{2} \rangle$	$\{(1, 3)\}$	0



## A 2-Passes Algorithm for The Frequent Problem

- Each  $j$  s.t.  $f_j > m/k$  exists in  $\text{keys}(A)$
- We can make a second pass over the stream, counting exactly the frequencies  $f_j$  for all  $j \in \text{keys}(A)$ , and then output the desired set of items

## References

- **Data Stream Algorithms** (Chapters 0 and 1)  
Lecture notes by A. Chakrabbarti and D. College