# Misra-Gries Summary: Finding Heavy Hitters in Data Stream

**COMPCSI 753: Algorithms for Massive Data**

Instructor: Ninh Pham

University of Auckland

Auckland, Aug 18, 2020

# Basic definitions

- Let $\mathbf{U}$ be a universe of size $\mathbf{n}$, i.e. $\mathbf{U} = \{\mathbf{1, 2, 3, \dots, n}\}$

- Cash register model stream:
  - Sequence of $\mathbf{m}$ elements $\mathbf{a_1}, \dots, \mathbf{a_m}$ where $\mathbf{a_i} \in \mathbf{U}$
  - Elements of $\mathbf{U}$ may or may not occur once or several times in the stream

- Finding heavy hitters in data stream (today's lecture):
  - Given a stream, finding frequent items.

# Frequent items

- Each element of data stream is a tuple.
- Given a stream of $\mathbf{m}$ elements $\mathbf{a_1}, \ldots, \mathbf{a_m}$ where $\mathbf{a_i} \in \mathbf{U}$, finding the most/top-$\mathbf{k}$ frequent items.

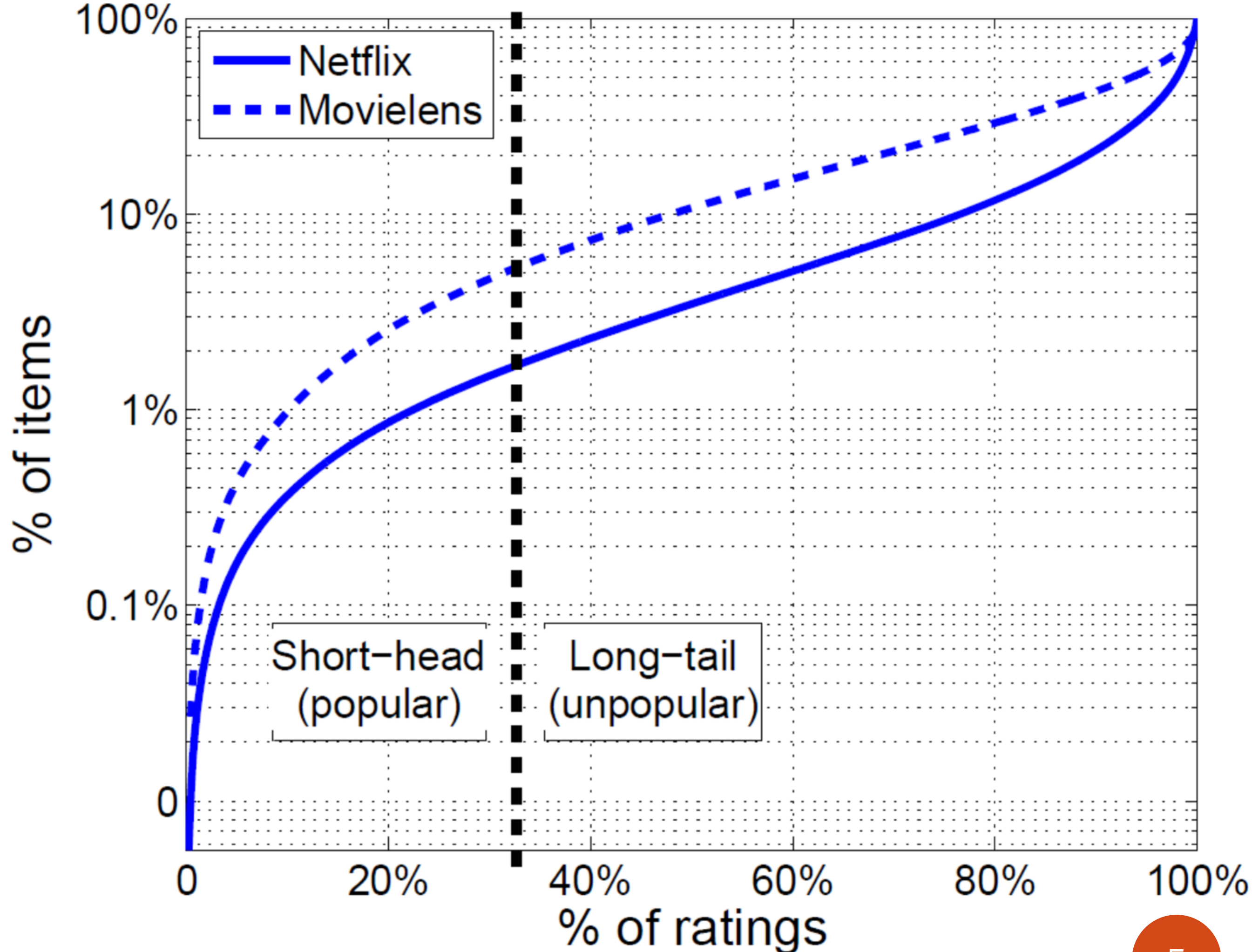
- Example:
  - $\{\underline{1}, 2, \underline{1}, 3, 4, 5\} \rightarrow \mathbf{f} = \{\underline{2}, 1, 1, 1, 1\}$
  - $\{\underline{1}, \underline{2}, \underline{1}, 3, \underline{1}, \underline{2}, 4, 5, \underline{2}, 3\} \rightarrow \mathbf{f} = \{\underline{3}, \underline{3}, 2, 1, 1\}$

# Applications

- Networking:
  - Tracking the most popular source, destinations, or source-destination pairs (those with the highest amount of traffic).
- Web analytics:
  - Tracking the most popular queries to a search engine, or the most popular pieces of content in a large content host.
- Facts:
  - Typical frequency distribution are highly skewed.
  - Top **10%** elements have **90%** of total occurrences (active rating users, most rated movies in Netflix).

# Skewed distribution

- Rating distribution for Netflix (solid line) and Movielens (dashed line) datasets.
- Items are ordered according to popularity (most popular at the bottom).

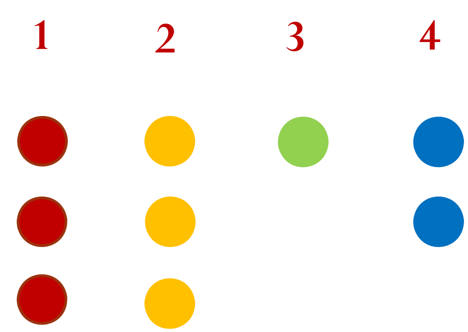



Cremonesi et al. RecSys'10

# Exact solution

- Create a counter for each distinct element on its first occurrence.
- When processing an element, increase its counter.

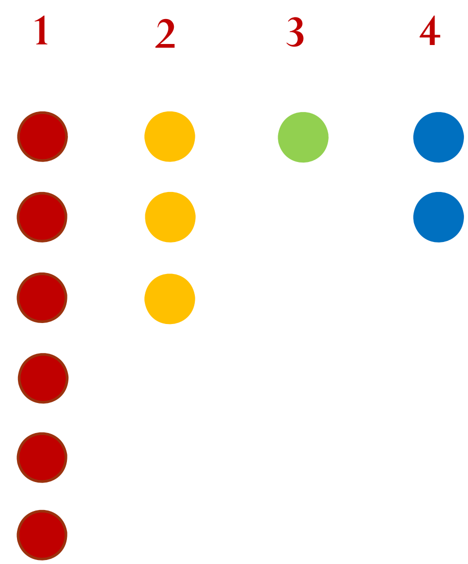
- Example:
  - Stream: {1, 2, 3, 1, 4, 2, 1, 4, 2}




- Problem:
  - Maintain **n** counters.
  - We can only maintain **k** << **n** counters.

# Sampling solution

- Reservoir sampling:
  - Reservoir sampling of size **k** to maintain **k** elements so far and the size of stream **m**.
  - Estimate frequency based on the reservoir summary.
  - Note that frequency distribution are highly skewed. What occurs if some elements have frequency **>> m/k**?
- Example:
  - Stream: {2, 2, 2, 4, 3, 4, 1, 1, 1, 1, 1, 1}
  - Reservoir sampling with **k = 3**



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

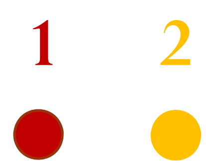
  {1}

**1**

# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

  {1, 2}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

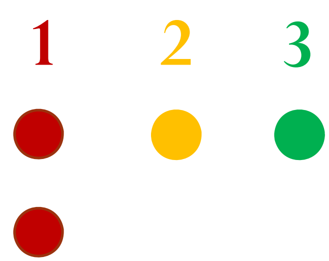
  {1, 2, 3}

**1** **2** **3**

# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

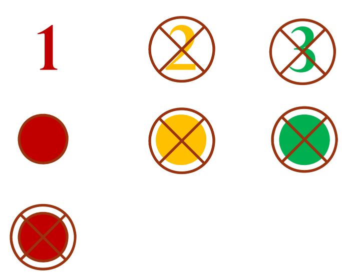
  {1, 2, 3, 1}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

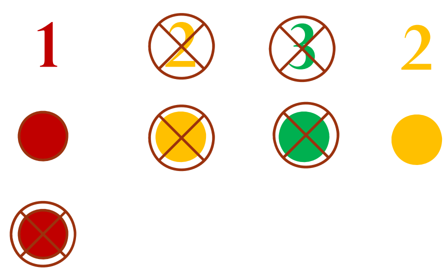
  {1, 2, 3, 1, 4}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

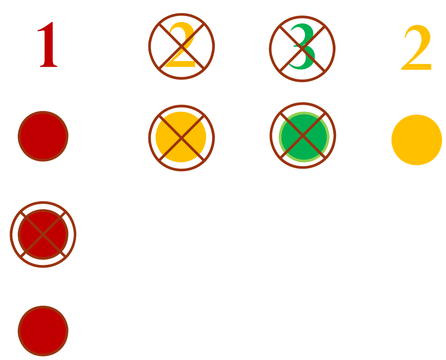
  {1, 2, 3, 1, 4, 2}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

  {1, 2, 3, 1, 4, 2, 1}



# Misra Gries'82

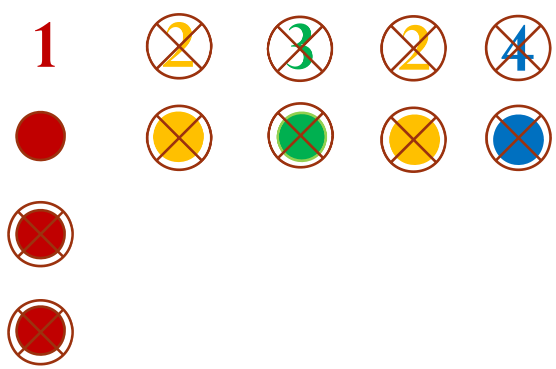
- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

  {1, 2, 3, 1, 4, 2, 1, 4}

# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2, 3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

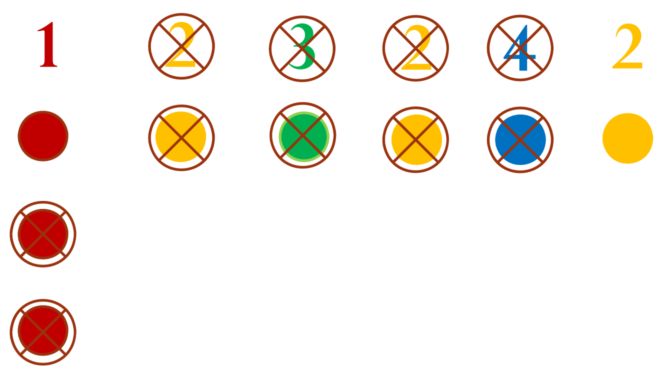
  {1, 2, 3, 1, 4, 2, 1, 4, 5}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2 ,3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

  {1, 2, 3, 1, 4, 2, 1, 4, 5, 2}



# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).
- Example: {1, 2 ,3, 1, 4, 2, 1, 4, 5, 2, 6}, **n**=6, **k**=3, **m**=11.

  {1, 2, 3, 1, 4, 2, 1, 4, 5, 2 , 6}

# Misra Gries'82

- Process an element **a**:
  - If we already have a counter for **a**, increment it.
  - Else, if there is no counter for **a**, but fewer **k** counters, create a counter for **a** initialized to **1**.
  - Else, decrease all counters by **1**. Remove **0** counters (key step).

- Query: How many times the element **a** occurred?
  - If we have a counter for **a**, return its value.
  - Else, return **0**.

- Observation: We always under-estimate the frequency!

# Why it works?

- Question:
  - How many decrements to a particular **a** can we have?
  - How many decrement step can we have?

- Answer:
  - The number of elements in stream: **m**.
  - The number of elements in the summary: **m'**.
  - Given **a** does not occur in the summary, one decrement step takes out **k** items and not count **a**. That is **k + 1** "uncounted" occurrences.
  - There is at most **(m – m') / (k + 1)** decrement steps.

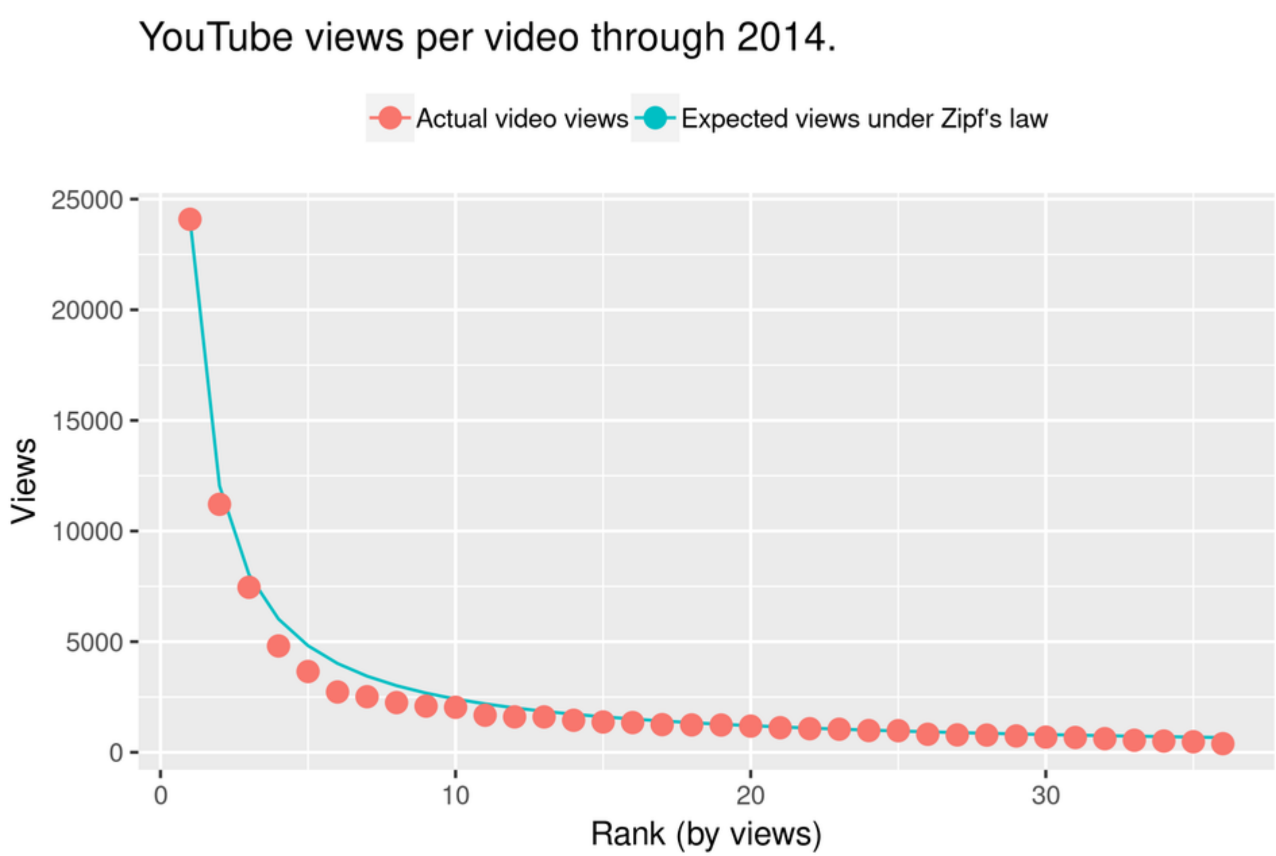
- Estimate is smaller than exact count by at most $\frac{m - m'}{k + 1}$.

# Why it works?

- Estimate is smaller than exact count by at most $\frac{m - m'}{k + 1}$.
  - We can find the most frequent items if their frequencies $> \frac{m - m'}{k + 1}$.
  - We get good estimate for a if its frequency $>> \frac{m - m'}{k + 1}$.
- Error bound:
  - Inversely proportional to **k**. Hence larger **k** gives better accuracy.
  - Can be computed by knowing **k** and **m'**, and tracking **m.**

# Why it works?

- Misra Gries works because typical frequency distributions have few very popular elements "Zipf law".




Carl Colglazier
GitHub

# Homework

- Implement the Misra-Gries algorithm on the dataset from Assignment 1:
  - Description: Each line (doc ID, word ID, freq.) as a stream tuple.
  - Query: What are the most and top-**10** frequent word ID have been used?