

CS5228: Knowledge Discovery and Data Mining

Lecture 11 — Data Stream Mining

Course Logistics

- **Reminder for submission deadlines**
 - A4: Nov 14, 11.59 pm
 - Project Report: Nov 4, 11.59 pm
- **Project submission**
 - Submission = project report (PDF, max 8 pages) + source code
 - Only 1 submission per team needed
 - Submission files should include team name
- **Last Lecture Quiz**
 - Friday, Nov 15, ~19:15 (30 min)
 - MCQs/MRQs, Lectures 7-11

Quick Recap — Graph Mining

- **Community Detection**

- Identification of "interesting" subgraphs
(\approx nodes in subgraph more tightly compared to other nodes)
- Similar to the task of clustering
(clustering algorithms can be adopted to find communities)

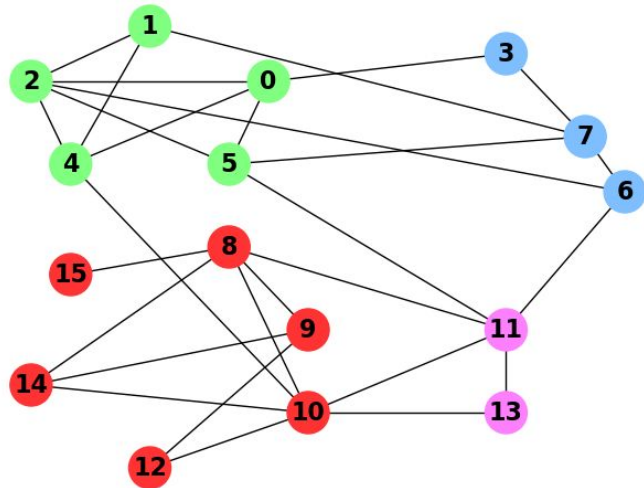
- No single definition of "community"

- Many algorithms for community detection

- Similarity between nodes (e.g., AGNES)
- Density-based (modularity + Louvain algorithm)
- Split-based (Edge Betweenness, Min-Cut)

Girvan Newman algorithm

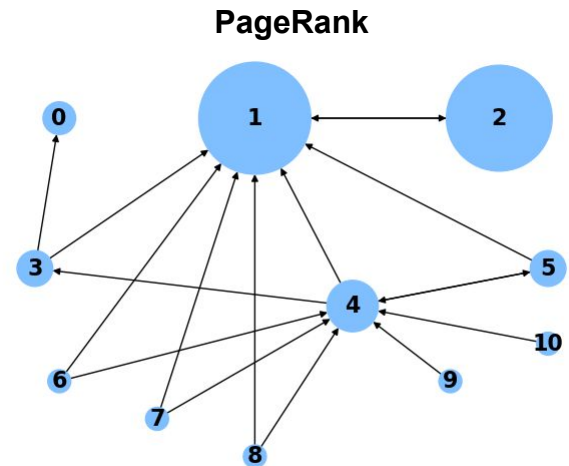
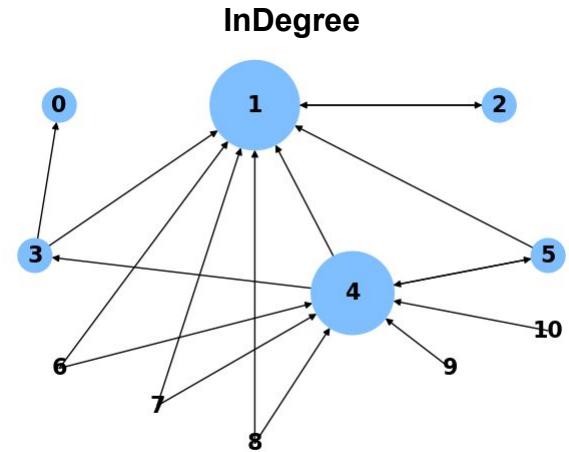
Karger's algorithm



Shared focus: **connectedness**

Quick Recap — Graph Mining

- **Centrality = importance of a node**
 - Based on a node's topological position in a graph
 - Different centrality measures focusing on different topological features
 - Not all measures applicable to all types of graphs
- **Popular centrality measures covered**
 - Local measures (Degree, InDegree, OutDegree)
 - Eigenvector-based measures (Eigenvector Centrality, PageRank)
 - Distance-based or path-based measures (Closeness, Betweenness)



Outline

- **Motivation**
 - Basic setup
 - Example Applications
- **Core Techniques**
 - Sampling
 - Filtering
 - Counting (distinct items)
- **Summary**

Data Stream Mining

- Data Mining so far

- Access to complete dataset (at the same time)
- Virtually unlimited storage and computing resources
(also: runtime of algorithms typically not that important, compared to the results)
- Support of arbitrary complex patterns

- Now: data items arrive one-by-one
in real time...like a stream

- All data never fully available
- Often very high arrival speeds
- Often limited amount of resources
- Often time-critical decisions
(common: execution in main memory only)

→ Focus on simple patterns (but not too simple)

Quick Quiz

Given is a stream of temperature sensor values in °C.

What is the only **non-trivial** pattern to monitor?

A

temp > 100 °C

B

Average of all temperature values

C

Median of all temperature values

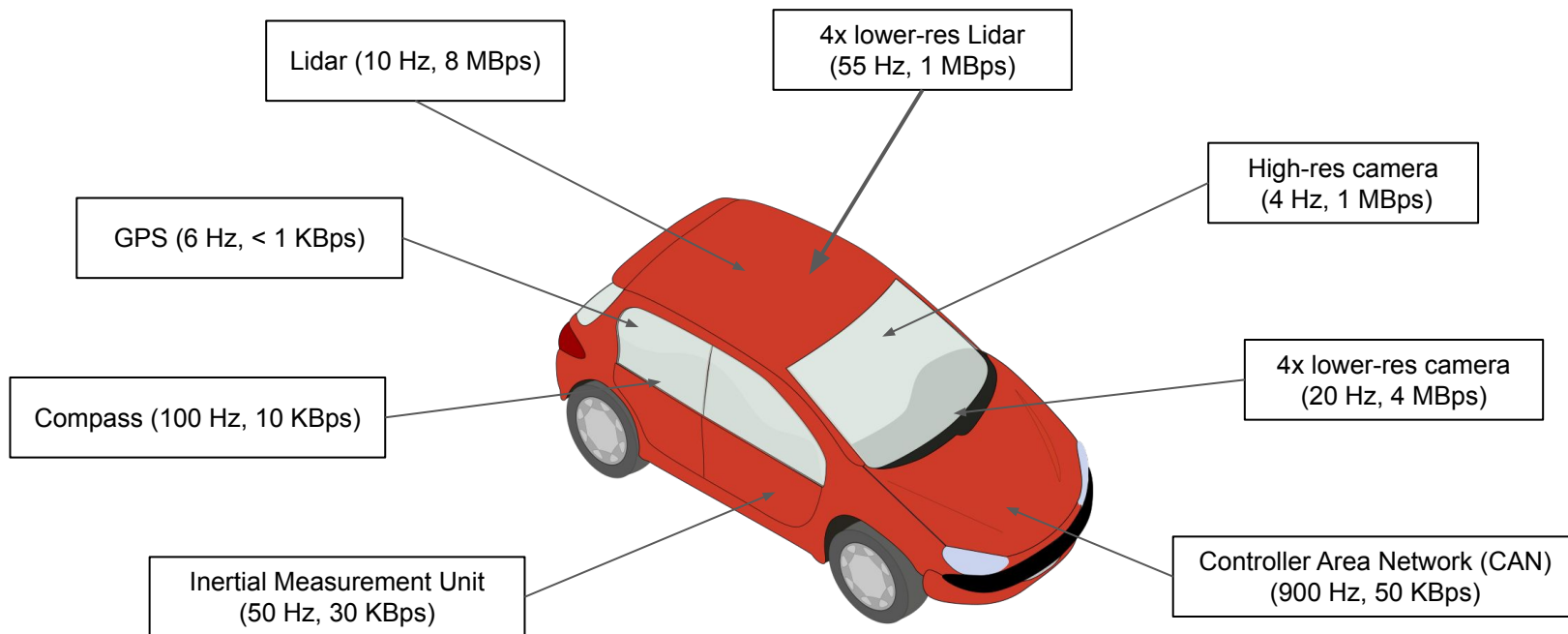
D

Maximum range of all temperature values

(Self-Driving) Cars

- Sensor data generated during a 40-minutes trip: 30 GB (13 MBps)

Source: [Exploring big volume sensor data with Vroom](#) (Moll et al., 2017)



Smart Cities

- Example: Lamppost-as-a-Platform (LaaP)

Source: <https://www.developer.tech.gov.sg/technologies/sensor-platforms-and-internet-of-things/lamppost-as-a-platform>



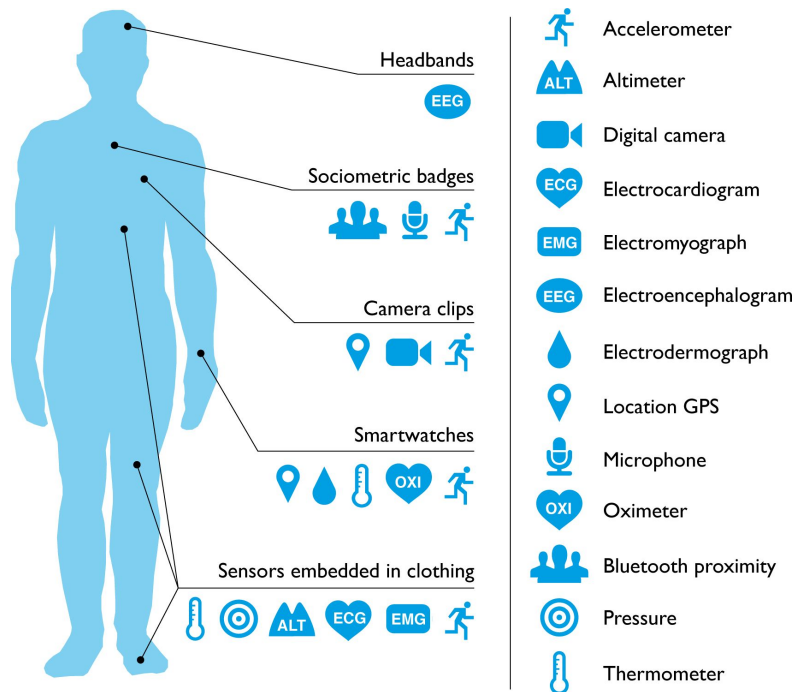
- Camera
- Temperature
- Humidity
- Gas (e.g., carbon monoxide)
- Air quality
- Rain

Health Monitoring

- Consumer Health Wearables

Source: <https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1001953>

- Smartphones
- Smartwatches
- Fitness bands
- Body cameras



Social Media

- Example: Daily data volume on Facebook

Source: <https://blog.wishpond.com/post/115675435109/40-up-to-date-facebook-facts-and-stats>

- 4.5 billion Likes
- 17 billion location-tagged posts
- 350 million photos uploaded
- 4.75 billion items shared
- 10 billion messages sent

facebook®

FOURSQUARE

twitter 

 TikTok

 YouTube

 Instagram

Linked 

Quick Sidenotes

- Covered techniques and algorithms not specific to streams
 - Applicable to many use cases involving large volumes of data
- Not of interest here: sliding window approaches
 - Keep the k most recent data items
 - Ignore all older items (delete from window)

} → window = complete dataset (snapshot)
- Commonly used throughout the lecture: hash functions
 - We will assume "well-behaved" hash functions
(same input → same output, minimize duplication, avalanche effect)
 - No deeper discussion what this means here

Outline

- Motivation
 - Basic setup
 - Example Applications
- **Core Techniques**
 - **Sampling**
 - Filtering
 - Counting (distinct items)
- Summary

Sampling

- Sampling — basic definition

- Process of selecting members of a population of interest (e.g., HDB residents)
- Consideration of whole population typically impractical (e.g., too costly to survey all HDB residents)
- Goal: statistical analysis of population (e.g., average happiness, most common complaints)

- Sampling data streams

- Population = stream of incoming data items
- Time and/or resource constraint generally make it impossible to consider all data items
- Relevant patterns based on statistical analysis

→ **Core challenge: How to get a representative sample?**

Problems with Naive Approach

- Toy example setup

- Stream of search queries — items are tuples (user, query, time)
- Goal: Fraction of queries issued more than once (here: twice) in the last 24h (by the average user)
- Restriction only 10% of all tuples should be stored

- Naive approach: Store latest tuple with a probability of 1/10

- Let s = number of time a user issued a query **once**
- Let d = number of time a user issued a query **twice**

Correct estimate:
(assuming all tuples)

$$\frac{d}{s + d}$$

**Estimate based on
naive sample:**

$$\frac{d}{10s + 19d}$$

Problems with Naive Approach — Explanation

- The issue: of d queries issued twice

- $d/100$ will be both in the sample

$$\left(\frac{1}{10}\right) \left(\frac{1}{10}\right) \cdot d = \frac{1}{100}d$$

- $18d/100$ will be only once in the sample

$$2 \cdot \left(\frac{1}{10}\right) \left(\frac{9}{10}\right) \cdot d = \frac{18}{100}d$$

- Fractions of queries issued twice in the sample:

$$\frac{\frac{1}{100}d}{\frac{1}{100}d + \frac{18}{100}d + \frac{1}{10}s} = \frac{d}{10s + 19d}$$

Sample with a Given Probability

- General Approach

- Given: items/tuples with n components, e.g., (user, query, time)
- Select subset of components as key on which the selection of the sample is based
(for toy example: key = "user" → consider only 10% of users instead of arbitrary tuples)
- Choice of key depends on goal of analysis

- Question: How to obtain a sample consisting of any fraction a/b of keys?

(in other words: How to decide whether to keep or discard a new tuple?)

- Common implementation via hash function

- Define hash function h that maps $h(key)$ in to b buckets $1..b$
- For each new tuple, if $h(key) \leq a$ (with $a \leq b$), add tuples to sample

Sample with a Given Size Limit — Reservoir Sampling

- Goal: Maintain a uniform random sample of fixed size B
 - Uniform random = each item has the same probability to be sampled
 - Allows to approximate basic statistics such as mean, variance, median, etc.
- Basic algorithm
 - Input stream of items $\{a_1, a_2, \dots\}$
 - Maximum reservoir size B
 - It can be shown that each item in the reservoir was sampled with the same probability B/t
(at any time t)

- 1) Add a_t with $t \leq B$ to reservoir
- 2) When receiving a_t with $t > B$:
with probability B/t , replace random item in reservoir with new item a_t

Reservoir Sampling — Example

- Initialization: $t = 0$, $B = 3$

→ **Stream:**

t = 1	A
t = 2	B
t = 3	C
t = 4	A
t = 5	C
t = 6	B
t = 7	B
t = 8	A
t = 9	C


Reservoir:

- 1) Add a_t with $t \leq B$ to reservoir
- 2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Reservoir Sampling — Example

- $t = 3$

Stream:



t = 1	A
t = 2	B
t = 3	C
t = 4	A
t = 5	C
t = 6	B
t = 7	B
t = 8	A
t = 9	C

Reservoir:

A
B
C

1) Add a_t with $t \leq B$ to reservoir

2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Just fill up the reservoir...

Reservoir Sampling — Example

- $t = 4$

Stream:

t = 1	A
t = 2	B
t = 3	C
t = 4	A
t = 5	C
t = 6	B
t = 7	B
t = 8	A
t = 9	C



Reservoir:

A
B
C A

1) Add a_t with $t \leq B$ to reservoir

2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Probability $B/t = 3/4$

→ Randomized decision: add $a_4 = A$ to reservoir

→ Replace random element — here $B_3 = C$ with A

Reservoir Sampling — Example

- $t = 5$

Stream:

t = 1	A
t = 2	B
t = 3	C
t = 4	A
→ t = 5	C
t = 6	B
t = 7	B
t = 8	A
t = 9	C

Reservoir:

A
B C
A

1) Add a_t with $t \leq B$ to reservoir

2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Probability $B/t = 3/5$


→ Randomized decision: add $a_5 = C$ to reservoir

→ Replace random element — here $B_2 = B$ with C

Reservoir Sampling — Example

- $t = 6$

Stream:

t = 1	A
t = 2	B
t = 3	C
t = 4	A
t = 5	C
 t = 6	B
t = 7	B
t = 8	A
t = 9	C

Reservoir:

A
C
A

1) Add a_t with $t \leq B$ to reservoir

2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Probability $B/t = 3/6$

→ Randomized decision: discard $a_6 = B$

Reservoir Sampling — Example

- $t = 7$

Stream:

t = 1	A
t = 2	B
t = 3	C
t = 4	A
t = 5	C
t = 6	B
→ t = 7	B
t = 8	A
t = 9	C

Reservoir:

A
C
A B

1) Add a_t with $t \leq B$ to reservoir

2) When receiving a_t with $t > B$:
with probability B/t , replace random
item in reservoir with new item a_t

Probability $B/t = 3/7$

→ Randomized decision: add $a_7 = B$ to reservoir

→ Replace random element — here $B_3 = A$ with B

Proof Sketch — All Elements in B sampled with B/t

- Obvious case: $i = t$
 - a_i was inserted into B with probability B/t (direct result from algorithm)
- Otherwise: $i < t$
 - Observation: in step i , an item gets replaced with probability $B/i * 1/B = 1/i$
 - Probability of an item in reservoir

$$B/i \cdot \left(1 - \frac{1}{i+1}\right) \cdot \left(1 - \frac{1}{i+2}\right) \cdot \dots \cdot \left(1 - \frac{1}{t}\right) = B/t$$

Probability that a_i
was inserted in B

Probability that a_i
was NOT replaced
in B at step $(i+1)$

Probability that a_i
was NOT replaced
in B at step $(i+2)$

Probability that a_i
was NOT replaced
in B at step t

Outline

- Motivation
 - Basic setup
 - Example Applications
- **Core Techniques**
 - Sampling
 - **Filtering**
 - Counting (distinct items)
- Summary

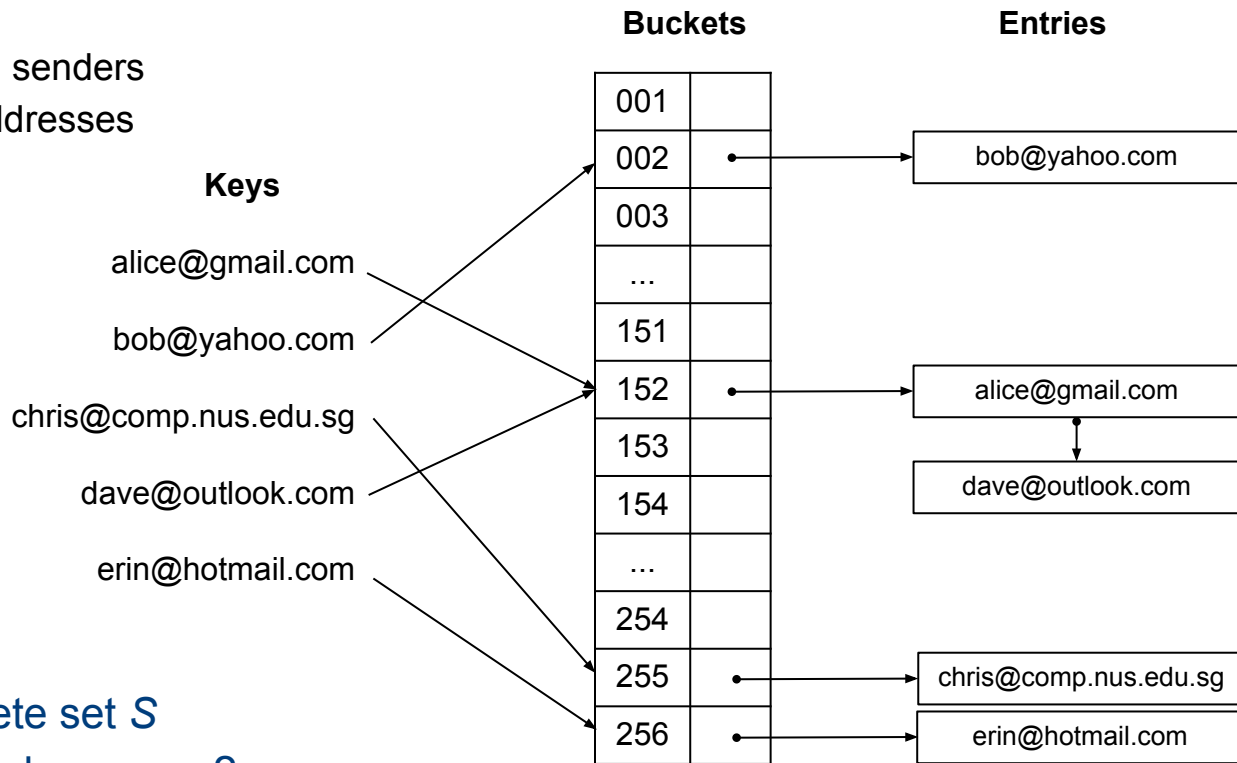
Filtering Data Streams

- Goal: Only accept items that meet certain criterion
- Simple: criterion is a property of item that can be (easily) calculated, e.g.:
 - Search queries with more than 2 keywords
 - Sensor values with valid status code
- Challenge: criterion involves lookup for membership in a (very large) set S , e.g.:
 - Emails with sender addresses that are whitelisted (spam filter)
 - Page visits to a predefined set of websites
 - Tweets from a selected group of users

Basic Solution — Create Hash Table for Set S

- Example: spam filter

- Only accept emails from senders with whitelisted email addresses



- Requires to store complete set S
- What if there is not enough memory?

Hashing without Storing S

- Create lookup table

- Create bit array B with $1..n$ bits and $B[i] = 0$
- Choose hash function $h(key) \in [1, n]$
- For each key $s \in S$ set $B[h(s)] = 1$

- Filter step for new data item with key k

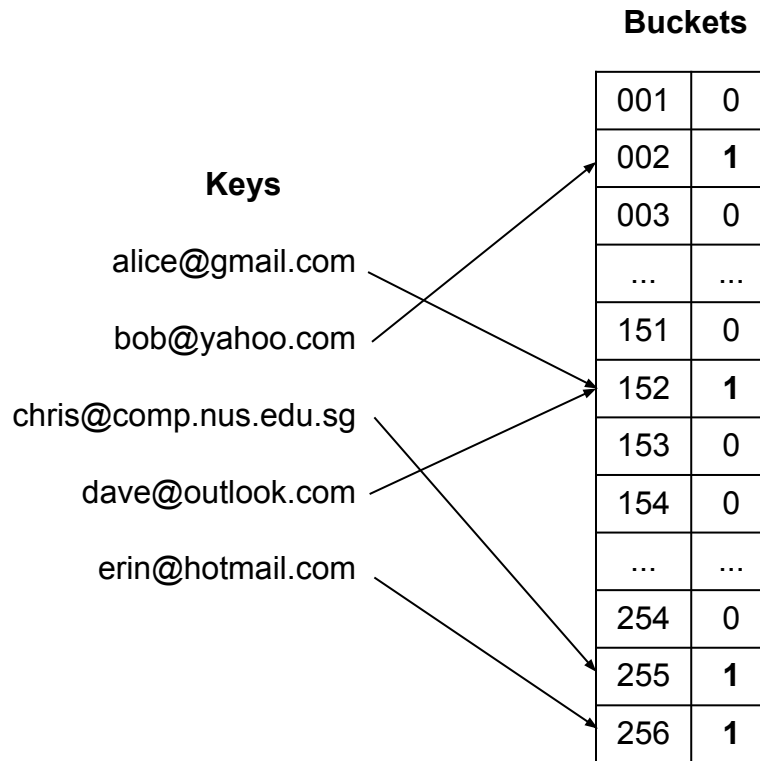
- Accept if $B[h(k)] = 1$, discard otherwise

→ Problem: **False Positives**

- $a \in S, b \notin S$ and $h(a) = h(b)$



How bad is it?



Quick Quiz

Can **false negatives** occur?

A

No

B

Yes, but the probability is negligible

C

Yes, but they do not matter for the application context.

D

Yes, and that's why this is a bad approach.

False Positive Rate — Analysis

Probability of $B[l] = 1$ after inserting **one key** from S :

$$\frac{1}{|B|}$$

Probability of $B[l] = 0$ after inserting **one key** from S :

$$1 - \frac{1}{|B|}$$

Probability of $B[l] = 0$ after inserting **all keys** from S :

$$\left(1 - \frac{1}{|B|}\right)^{|S|} = \left(1 - \frac{1}{|B|}\right)^{|B| \frac{|S|}{|B|}} \approx e^{-\frac{|S|}{|B|}}$$

Probability of $B[l] = 1$ after inserting **all keys** from S :

$$1 - e^{-\frac{|S|}{|B|}}$$



Probability of $B[h(s)] = 1$ with key $s \notin S$:
(probability of a false positive)

$$1 - e^{-\frac{|S|}{|B|}}$$

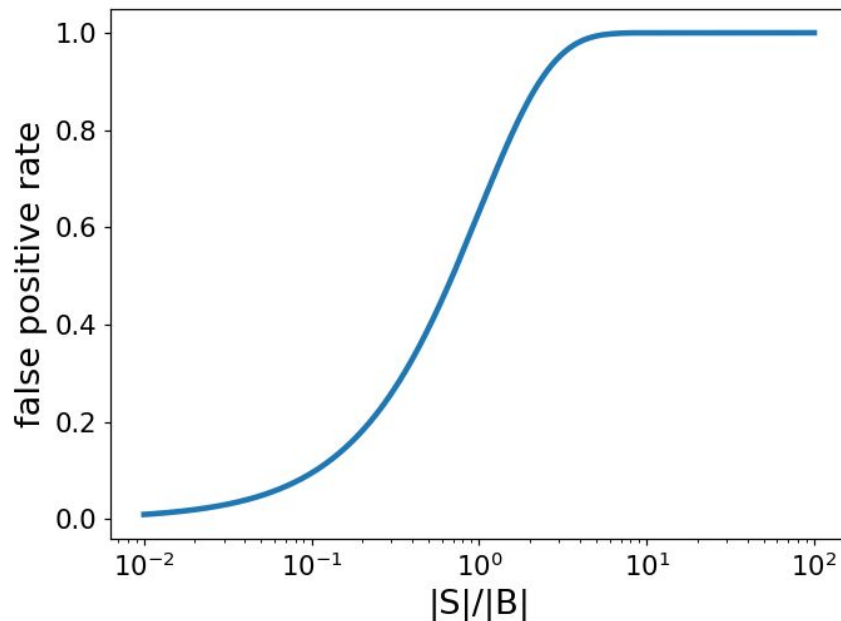
$$e^p = \left[\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n \right]^p$$



$$\approx e^{-\frac{|S|}{|B|}}$$

False Positive Rate — Visualization

- Effect of ratio $|S|/|B|$ on false positive rate
 - Example calculation
 - $|S| = 10^9$ whitelisted email addresses
 - $|B| = 8 \cdot 10^9$ (1 GB of main memory)
- $|S|/|B| = 1/8$
- False positive rate: $1 - e^{-1/8} = \mathbf{11.75\%}$



How to reduce false positive rate?

(without increasing the size of bit array B)

Bloom Filters

- Bloom Filters — basic idea

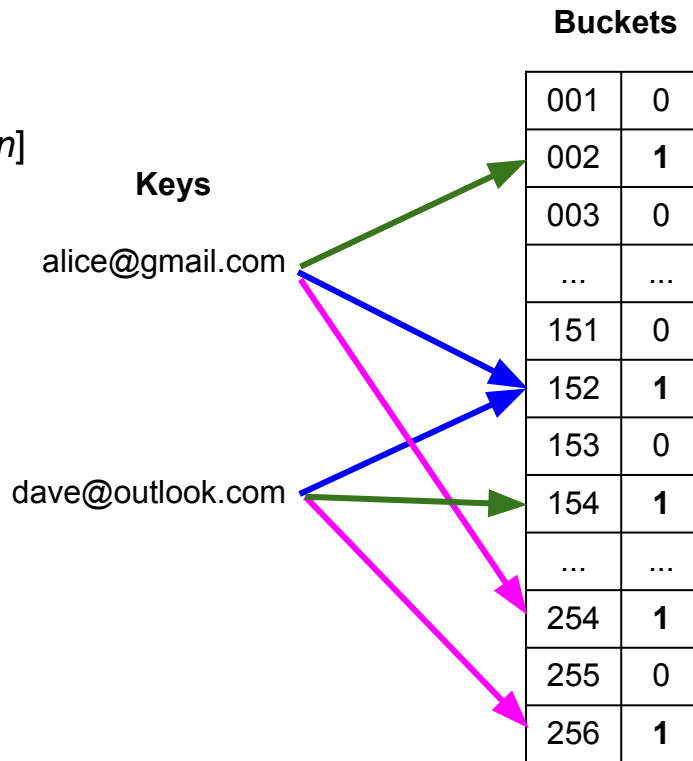
- Create bit array B with $1..n$ bits and $B[i] = 0$
- Choose **m independent hash functions** $h_i(key) \in [1, n]$
- For each key $s \in S$ set $B[h_i(s)] = 1$, with $1 \leq i \leq m$

- Example

- 3 hash functions: h_1 , h_2 , h_3

- Filter step for new data item with key k

- Accept if $B[h_i(k)] = 1$ for all $1 \leq i \leq m$,
discard otherwise



False Positive Rate — Analysis

Probability of $B[h(s)] = 1$ with key $s \notin S$:

(probability of a false positive)

$$1 - e^{-\frac{|S|}{|B|}}$$



from 1 to m hash functions

Probability of $B[h_i(s)] = 1$ ($1 \leq i \leq m$) with key $s \notin S$:

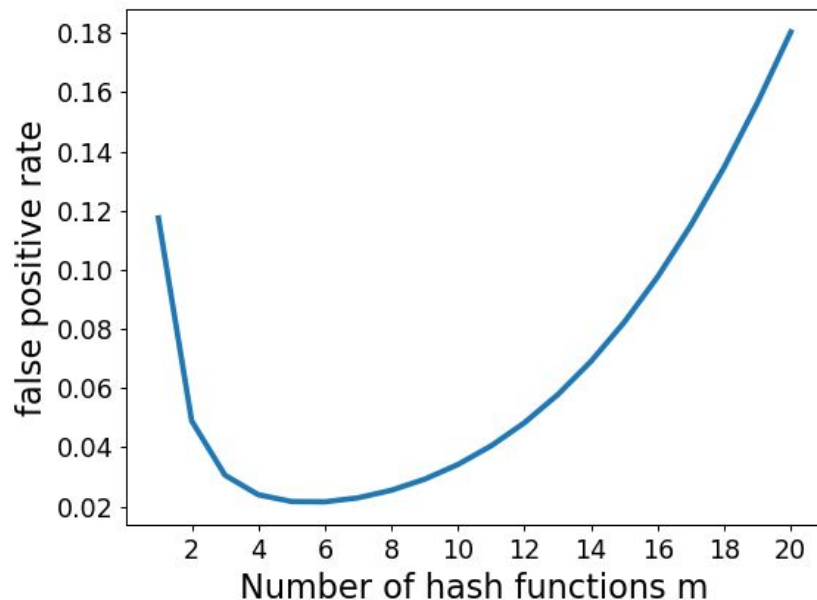
(probability of a false positive)

$$\left(1 - e^{-\textcolor{red}{m} \frac{|S|}{|B|}}\right)^{\textcolor{red}{m}}$$

False Positive Rate — Visualization

- Effect of number of hash functions m on false positive rate
 - Here, $|S|/|B| = 1/8$ (see previous example)
- Global optimum — intuition: large m
 - More hash results need all be 1
 - But: also more 1s in bit array
- Optimal value for m

$$m_{best} = \frac{|B|}{|S|} \ln 2$$



$$m_{best} = \frac{8}{1} \ln 2 = 5.5 \approx 6$$

→ False positive rate: **2.2%**

Bloom Filter — Discussion

- Memory and space consumption
 - Time complexity: $O(m)$
 - Space complexity: $O(|B|)$
- Limitation: no support of removing keys from S
 - E.g.: not able to remove a whitelisted email address
 - Only workaround: rebuild lookup table (bit array) from scratch

Why?

Extension — Counting Bloom Filters

- Counting Bloom Filters

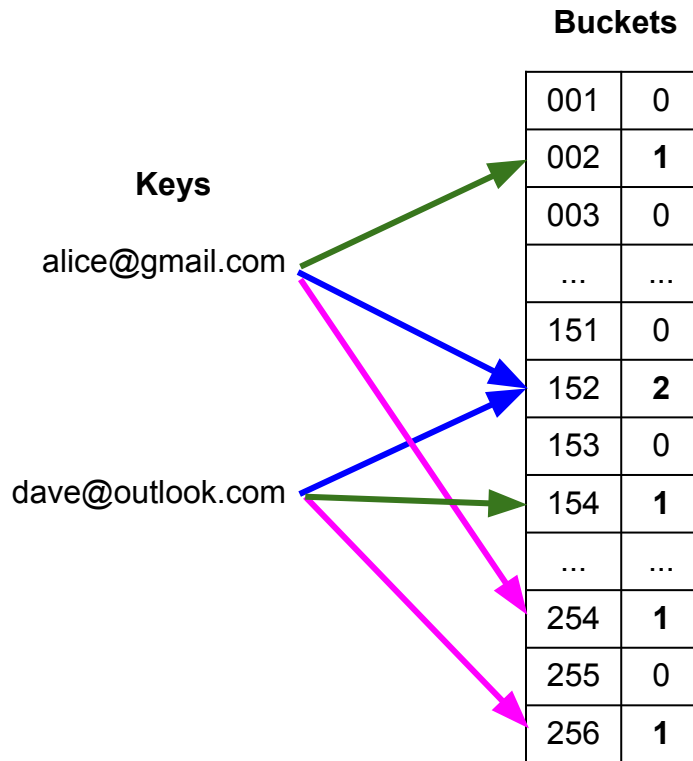
- Replace bits for counters (typically 3-4 bits per counter)
- Increased size of lookup table (3-4 times)

- Operations

- Insert s : $B[h_i(s)] += 1$, with $1 \leq i \leq m$
- Delete s : $B[h_i(s)] -= 1$, with $1 \leq i \leq m$
- Lookup k : Accept if $B[h_i(k)] > 0$ for all $1 \leq i \leq m$

- False positive rate

- Same as normal Bloom Filter: $\left(1 - e^{-m \frac{|S|}{|B|}}\right)^m$



Outline

- Motivation
 - Basic setup
 - Example Applications
- **Core Techniques**
 - Sampling
 - Filtering
 - **Counting** (distinct items)
- Summary

Counting Unique/Distinct Elements

- Applications

- Number of unique Facebook visitors (identified by user id)
- Number of unique website visitors (identified by IP address)
- Number of unique words in a (large) text document

- Straightforward solution

- Maintain set S of items seen so far
- Number distinct elements $\rightarrow |S|$

→ What if set S can grow very large?

```
1 S = set()
2
3 with open('data/ip-only-nasa-access.log') as file:
4     for line in file:
5         ip = line.strip() # Get IP address
6         S.add(ip)         # Add IP address to set
7
8 print('|S| = {}'.format(len(S)))
```

$|S| = 7637$

Flajolet-Martin Algorithm

- Approximation approach

- Estimate distinct count in an unbiased way
- Accept errors but minimize their probability

- Basic algorithm

- (1) Choose hash function that maps each of the n elements to at least $\log_2 n$ bits
- (2) For each element s in the stream
 - (a) Calculate hash $h(s)$ → bit string of length $\log_2 n$
 - (b) Let $r(s)$ = number of trailing 0s in $h(s)$
 - (c) Keep track of largest $r(s)$ → R
- (3) Return estimate for distinct count as 2^R

} Example: 2^{32} IPv4 addresses
→ at least 32 bits

Flajolet-Martin Algorithm — Example

a: IPv4 address	h(a)
13.66.139.0	01010100110100100110111000001001
157.48.153.185	110010100100101001001011011111 <u>00</u>
157.48.153.185	110010100100101001001011011111 <u>00</u>
216.244.66.230	0001110001010101100111010001001 <u>0</u>
54.36.148.92	010101010001010110010111000011 <u>00</u>
92.101.35.224	10100011100001100000000100000001
73.166.162.225	001001001001111100100101101011 <u>000</u>
73.166.162.225	001001001001111100100101101011 <u>000</u>
54.36.148.108	00001100010100011000000001010001
54.36.148.1	011001011000101000101100100001 <u>00</u>
162.158.203.24	1011100001001001010011100010111 <u>0</u>
157.48.153.185	110010100100101001001011011011 <u>00</u>
157.48.153.185	110010100100101001001011011011 <u>00</u>
...	...

$R = 3$ (largest number of trailing 0s so far)

→ Estimate for distinct count: $2^3 = 8$

Quick "Quiz"

Someone is telling you that s/he flipped a fair coin **3 times** and got **3 Heads** after **k tries**?

Which **number of tries** is the most believable?

A

$k = 1$

B

$k = 10$

C

$k = 100$

D

$k = 1,000$

Flajolet-Martin Algorithm — Intuition & Proof Sketch

- Basic intuition

- More distinct elements \rightarrow more different hash values \rightarrow "unusual" hash values more likely
- "Unusual" hash value = hash value with rare bit pattern (e.g., number of trailing 0s)

Probability that $h(a)$ ends in **at least** k trailing 0s

$$\overbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{2}}^{k \text{ factors}} = \frac{1}{2^k} = s^{-k}$$

Probability that $h(a)$ ends in **less than** k trailing 0s

$$1 - \frac{1}{2^k}$$

Given m distinct elements, probability that **all** $h(a)$ end in **less** than k trailing 0s

$$\left(1 - \frac{1}{2^k}\right)^m$$

Given m distinct elements, probability that **$R \geq k$**
(i.e., at least one of the m elements has $h(a)$ with at least k trailing 0s)

$$1 - \left(1 - \frac{1}{2^k}\right)^m$$

Flajolet-Martin Algorithm — Proof Sketch

Given m distinct elements, probability that $\mathbf{R} \geq k$
(i.e., at least one of the m elements has $h(a)$ with at least k trailing 0s)

$$e^p = \left[\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n \right]^p$$

$$1 - \left(1 - \frac{1}{2^k} \right)^m = 1 - \left(1 - \frac{1}{2^k} \right)^{2^k \frac{m}{2^k}} \approx 1 - e^{-\frac{m}{2^k}}$$

Case 1: $2^k \ll m \rightarrow 1 - e^{-\frac{m}{2^k}} \approx 1 - 0 = 1$

Case 2: $2^k \gg m \rightarrow 1 - e^{-\frac{m}{2^k}} \approx 1 - \underbrace{\left(1 - \frac{m}{2^k} \right)} \approx \frac{m}{2^k} \approx 0$

$$e^x = 1 + \frac{x}{1!}, \text{ if } x \ll 1$$

First 2 terms of the
Taylor expansion of e^x

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Flajolet-Martin Algorithm — Proof Sketch

- Interpretation

Case 1: $2^k \ll m \rightarrow P(R \geq k) \approx 1$

The probability to get an $h(a)$ with
enough trailing 0s is rather high

Case 2: $2^k \gg m \rightarrow P(R \geq k) \approx 0$

The probability to get an $h(a)$ with
too many trailing 0s is rather low

→ R is typically in the right ballpark

Flajolet-Martin Algorithm — Problems & Extensions

- Obvious problem with basic Flajolet-Martin algorithm — given 2^R
 - An estimate is always a power of 2
 - If R is off by just 1, estimates doubles or halves
- Practical solution: use multiple hash functions $h_i(a)$ — e.g.:
 - $p \cdot q$ hash functions $\rightarrow p \cdot q$ R values $\rightarrow p \cdot q$ estimates for the distinct counts
 - Put all estimates into p groups, each of size q
 - Calculate median of each group $\rightarrow p$ medians
 - Calculate the mean over all p medians

Outline

- Motivation
 - Basic setup
 - Example Applications
- Core Techniques
 - Sampling
 - Filtering
 - Counting (distinct items)
- Summary

Summary

- Data streams — main challenges

- Large data volumes + high arrival speeds
- Limited resources + real-time requirements



→ **patterns = statistical analysis**

- Common tasks on streams

(or very large datasets in general)

- Sampling
- Filtering
- Counting
(distinct elements)



→ trade-off: **speed / resource-efficiency** vs. **accuracy / errors**

Solutions to Quick Quizzes

- Slide 7: C
- Slide 30: A
- Slide 42: B