streaming algorithms

data arrive one-by-one and it is not feasible to store the entire sequence

why streaming algorithms

- networking
  - billions of packets per hour per equipment
  - each provider has hundreds of equipments
  - detect anomalies, faults, and security problems
- telecommunications
  - there are billions telephone calls in North America each day
  - analyze

find the missing number

a sequence of distinct numbers arrives and one is missing

find the missing number

a sequence of n numbers a_1, …. , a_n arrives

- all numbers are different and have values between 1 and $n + 1$, which value is missing?
- we have only $\log n$ space
- example:
  - there are 11 soccer players with numbers 1, …, 11.
  - we see 10 of them one by one, which one is missing?
- because of the memory constraint we can remember just one number

1

8

5

11

3

9

2

6

7

4

which number is missing?

10

find the missing number – algorithm

- start a sum S with value 0
- when a number a_i arrives, add a_i to S
- when the stream is over, compute the sum $1 + 2 + 3 + 4, \ldots, + n + (n + 1)$
- output the difference between the sum and S

a counting problem

counting many elements using a few symbols

a counting problem

- a huge sequence of object arrives and a small memory is available
- objects arrive one-by-one and you have to count the objects
- of course if the objects are n a memory that can store $O(\log_{10} n)$ symbols is enough
- what happens if the available memory has less than such space?

approximate counting

- suppose to have a memory with size $\log_{10}\log_{10} n$
- can you still count the objects, at least approximately?
- since we can count n objects with $\log_{10} n$ symbols we can hope, with $\log_{10}\log_{10} n$ symbols, to count the number of symbols of the count
- we can hope to estimate the "order of magnitude" of n

approximate counting

- number: IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
- representation with $\log_{10} n$ symbols: 40
- representation with $\log_{10}\log_{10} n$ symbols: 2
  - representation with 2 symbols; hence it less than 100 and greater than 9

a simple algorithm for standard counting

- start a counter C with value 0
- when an object arrives, increase C by 1
- when the stream is over, output C

Morris' algorithm for approximate counting, 1977

- start a counter X with value 0
- when an object arrives, increase X by 1 with probability $\frac{1}{10^X}$
- when the stream is over, output 10^X-1

the algorithm is probabilistic: it exploits randomization

claim: the expected value of 10^X is $n + 1$; with math notation $E[10^X] = n + 1$

expected value

- the expected value is what one expects to happen *on average*
- suppose random variable X can take value x_1 with probability p_1, value x_2 with probability p_2, …., up to value x_k with probability p_k; the expectation of this random variable X is defined as

$$E[X] = x_1p_1 + x_2p_2 + \cdots + x_kp_k$$

expected value – example

- suppose X represents the outcome of a roll of a six-sided dice
- the possible values for X are $1, 2, 3, 4, 5, 6$, all equally likely (each having the probability of $\frac{1}{6}$)
- the expected value of X is $E[X] = 1\frac{1}{6} + 2\frac{1}{6} + 3\frac{1}{6} + 4\frac{1}{6} + 5\frac{1}{6} + 6\frac{1}{6} = 3.5$

approximate counting – intuition

- first object arrives: currently $X=0$; hence, X is incremented with probability $\frac{1}{10^0}=\frac{1}{1}=1$
- more objects arrive: for each of them X is incremented with probability $\frac{1}{10^1}=\frac{1}{10}$; hence, it is very likely that after 10 objects X will be equal to 2

Morris' algorithm for approximate counting – intuition

- more objects arrive: for each of them X is incremented with probability $\frac{1}{10^2}=\frac{1}{100}$; hence, it is very likely that after 100 objects X will be equal to 3
- ….
- hence, it is very likely that X is increased by 1 for the first 10 objects, by 1 for the next 100 objects, , by 1 for the next 1000 objects, ….

a majority problem

a majority problem

- stream of the purchase orders of NYSE
- each order is labeled with a number
  - identifies the stock
- is it true that the majority of the orders refer to one stock?
  - there is a stock leading the market

majority in a stream of numbers

- suppose a stream of n numbers arrives, each less or equal than n
- is there any number that appears in the stream more than $^{n}/_{2}$ times?

1 2 2 3 1 1 2 1 1 1 3 1 1 1 2 1 yes

1 2 2 3 1 1 2 1 3 2 3 1 1 2 2 1 no

if the memory is not a problem

- we can use a counter for each number
- plus a counter for the stream

small memory

- what happens if the memory is small?
- suppose to have only one counter plus one cell of memory to store exactly one number
  - an amount of memory that is $O(\log n)$
- we name C the counter and M the cell of memory for just one number

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C
- output M

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0$ $M =$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0$ $M =$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0$ $M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0$ $M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 2$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 2$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0 \quad M = 2$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 0 \quad M = 2$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 1 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 2 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 2 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 3 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 3 \quad M = 1$

Boyer-Moore algorithm

- set counter C to 0
- when a number i arrives
  - if $C = 0$ then set M to i and add 1 to C
  - else if $M = i$
    - then add 1 to C
    - else subtract 1 to C

1 2 2 3 1 1 1 1 1 1 3 1 1 1 2 1

$C = 4 \quad M = 1$

analysis – why it works?

- suppose that the stream contains a number, say 7, that appears more than $n/2$ times
- the numbers different from 7 decrease the counter of 7 each at most once

does it really work?

- notice that if there is no number with the majority then the algotithm outputs the wrong number

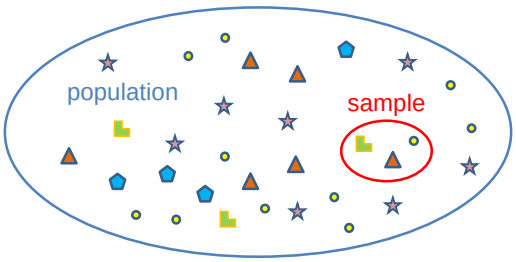
how to make it work?

- at the end of the stream we do a second pass on the same stream cheching if the selected number has the majority
- the algorithm performs two scans on the stream
- this is semi-streaming

sampling a stream of objects

sampling

- *sampling* is the selection of a subset of objects (*sample*) within a large population of objects
- the target is to study the characteristic of the entire population looking only at the sample



sampling

- so we can ask queries about the selected subset and have the answers be statistically representative of the stream as a whole

sampling a sequence of n objects

- suppose to have a sequence of n objects, where n is known in advance
- we want to determine a sample of the sequence consisting of just one object, so that all the objects have the same probability to be in the sample

sampling a sequence of n objects

- a simple algorithm for computing the sample
  - randomize an index i between 1 and n with equal probability
  - put in the sample the i-th element of the sequence

if n is not known in advance?

- suppose that the sequence is a stream whose length is not known
- how to compute the one-element-sample so that all the objects have the same probability to be seleced?

if n is not known in advance?

- a simple algorithm
  - let R be the sample
  - when the first object arrives put it into R
  - when the i-th object arrives ($i > 1$)
    - with probability $1/i$ subsitute the object contained into R with the new object
    - with probability $1 - 1/i$ do nothing

if n is not known in advance?

- analysis
  - if the stream has one item it is kept with probability 1
  - if the stream has two items each is kept with probability $1/2$
  - if the stream has three items:
    - the third is kept with probability $1/3$
    - the second substituted the first with probability $1/2$ and then was substituted by the third with probability $(1 - 1/3)$; hence it is in the sample with probability $1/2\ (1 -$

sampling k elements of a stream

- Vitter algorithm – reservoir sampling
  - let R be the sample, with k slots, each denoted $R[k]$
  - for $i \leq k$ (first portion of the stream) simply assign the i-th element of the stream to $R[i]$
  - for $i > k$ (second portion of the stream), when the i-th element arrives select a random number j between 1 and i; if $j \leq k$ then substitute $R[j]$ with the i-th element

sampling k elements of a stream

- observe that Vitter algorithm is a generalization of the algorithm for sampling just one element
- the probability of any element to be in the sample is k/n

a filtering problem

only the elements belonging to a certain set should be retained

a filtering problem

- filter stream so that elements that belong to a particular set are allowed through, while “most” nonmembers are deleted
- the set is too large to store in main memory

spam or not spam

- we have a set S of one billion allowed email addresses – those that we will allow through because we believe them not to be spam
  - typical email address is 20 bytes or more, it is not reasonable to store S in main memory
- stream emails
  - for each email we have either to forward or to discard
  - we tolerate that a small percentage of spam gets through

Bloom Filter

- a Bloom filter is a succinct description of a set of items

66

hash function

- a hash function is used to map data of arbitrary size to data of fixed size

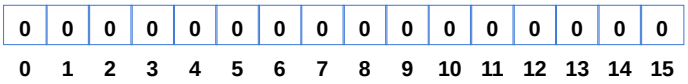
Bloom Filter

- A Bloom filter implements a dictionary with
  - A bit vector F of size m
  - A collection of hash functions $h_1, \cdots, h_k$ mapping an item to a integer in the range $[0, m-1]$
- Add an item x
  - **for** $i = 1$ to k
    - $F[h_i(x)] = 1$
- Check for the presence of item x
  - **for** $i = 1$ to k
    - **if** $F[h_i(x)] = 0$ **return** false
  - **return** true

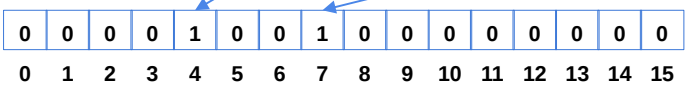
68

Bloom Filter example

- Let F be a vector with m = 16 bits



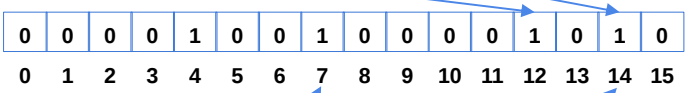
- We use hash functions h_1 and h_2 and add "roberto"
- Assuming h_1(roberto) = 4 and h_2(roberto) = 7 we obtain



69

Bloom Filter example

- Next, we add "pino"
- Assuming h_1(pino) = 12 and h_2(pino) = 14 we obtain



- We check for the presence of "pino"
- Since h_1(pino) = 12 and h_2(pino) = 14, and since F(12) = 1 and F(14) = 1 we return true (ok)
- We check for the presence of "paolo"
- Assuming h_1(paolo) = 7 and h_2(paolo) = 14, since F(7) = 1 and F(14) = 1 we return true (false positive)

70

Properties of a Bloom Filter

- a Bloom filter does not have false negatives
- a Bloom filter may have false positives
  - due to collisions in the hash functions
  - for n elements stored in a Bloom filter of size m with k hash functions, the probability of a false positive is about

$$P \approx (1 - e^{-kn/m})^k$$

  - P is minimized for k = (ln 2) m / n, which gives $P \approx 0.7^{m/n}$

71

back to the email

- false positives
  - for n elements stored in a Bloom filter of size m with k hash functions, the probability of a false positive is about

$$P \approx (1 - e^{-kn/m})^k$$

  - P is minimized for k = (ln 2) m / n, which gives $P \approx 0.62^{m/n}$

if n=1,000,000,000 - suppose m=5,000,000,000 bits - we use k=4 hash functions and obtain a probability of false positives P=0.09

72

streaming algorithm – conclusions

- a streaming algorithm is an algorithm for processing a data stream
  - the input is presented as a sequence of items and can be examined in one pass (or a few passes)
  - the available memory is limited
    - much less than the input size
  - the processing time per item is also limited
    - the time for processing an item is less than the interarrival time between two consecutive items