Bloom Filter (1970)

The applications of Bloom Filter are:

- Weak password detection
- Internet Cache Protocol
- Safe browsing in Google Chrome
- Wallet synchronization in Bitcoin
- Hash based IP Traceback
- Cyber security like virus scanning

+ Bioinformatics

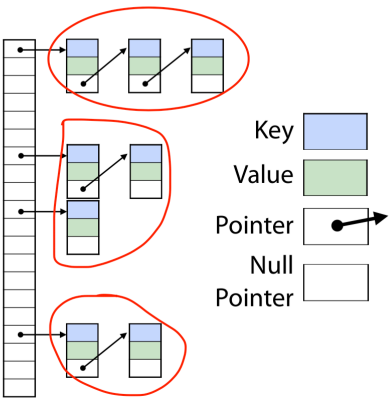
1

Bloom filter

Imagine we start with a hash table ...

... and start taking things away

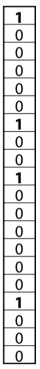
Query: c'è un elemento o no? Non mi interessa il valore, ma solo che c'è.



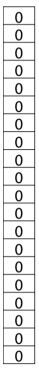
2

Bloom filter

sono array binari (facili da conservare)



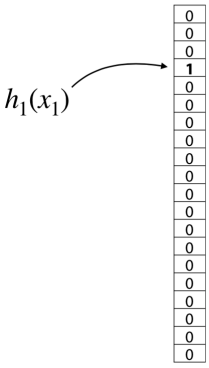
Bloom filter: inserting



Start with all 0s

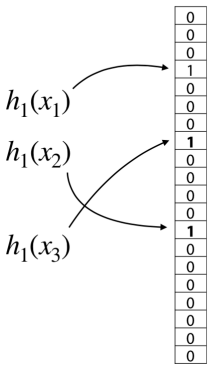
3

Bloom filter: inserting



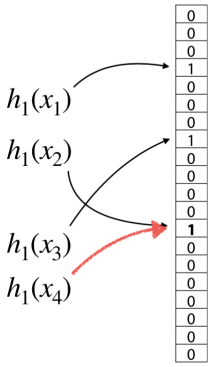
To insert item, hash it to a bucket and set bit to 1 (if not already)

Bloom filter: inserting



4

Bloom filter: inserting



If already 1, this is a *collision*

Life goes on

5

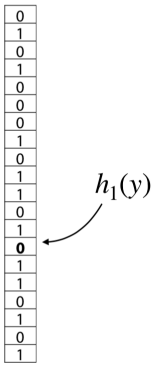
Bloom filter: querying

Use same hash function h_1 to hash query item; check if bucket is 0 or 1

6

Bloom filter: querying

If filter says "0" -- item is definitely not present



7

Bloom filter: querying

If filter says "0" -- item is definitely not present

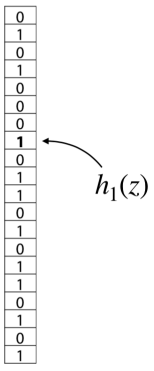
If filter says "1" -- item *may be* present

...or may be a *collision*

falso positivo

One-sided error

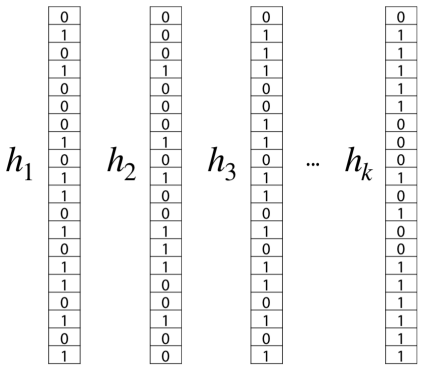
Randomized algorithms can exploit one-sided error via repeated trials



8

Bloom filter

What if we used many hashes/filters; adding each item to each?

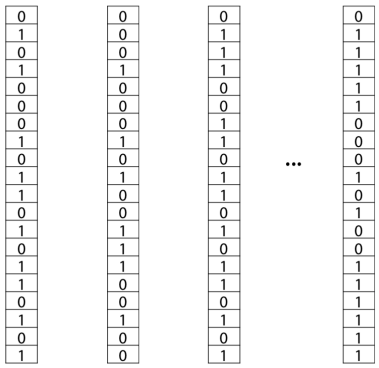


Will all filters have the same number of set (=1) bits?

Not necessarily; one hash/filter might have more/fewer collisions than another

9

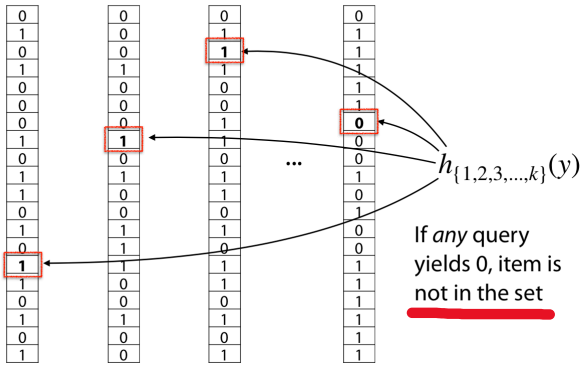
Bloom filter



$h_{\{1,2,3,\ldots,k\}}(y)$

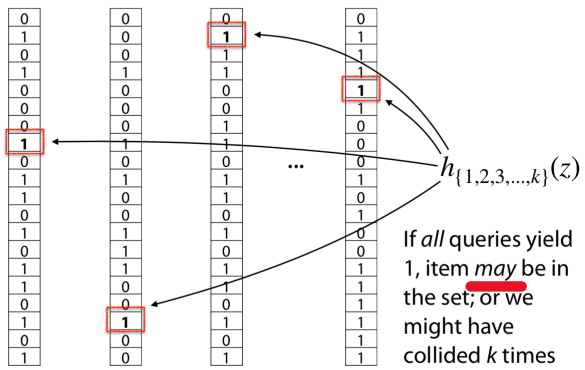
10

Bloom filter



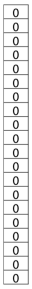
11

Bloom filter



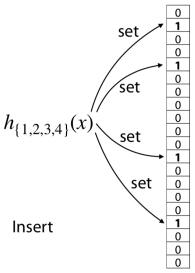
12

Bloom filter



Rather than use a new filter for each hash, one filter can use k hashes

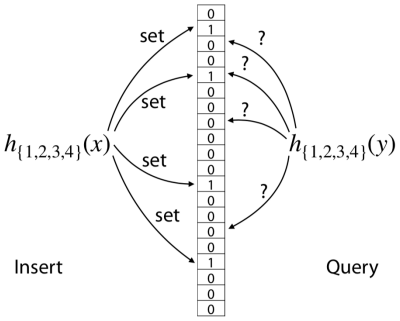
Bloom filter



Rather than use a new filter for each hash, one filter can use k hashes ($k = 4$ here)

13

Bloom filter



Rather than use a new filter for each hash, one filter can use k hashes ($k = 4$ here)

Come li uso?
Memorizzo i kmer di una sequenza genomica:

- Se voglio sapere se un kmer c'è o no, uso un BF
- Occupa poco spazio.

14

Problem definition

- Input:
  - set ***T*** of texts
  - Integer value ***k***
- Output:
  - Set of pairs ***(s, occ(s))***
  - ***s*** is a k-mer
  - ***occ(s)*** is the number of occurrences of s in T

15

Una breve lista di k-mer counter

- Tallymer
- Jellyfish
- BFCounter
- DSK
- KMC
- BLESS
- khmer
- Turtle
- KMC2

K-mer counting is arguably one of the simplest (both conceptually and programmatically) tasks in computational biology, *if we do not care about efficiency*. The number of existing papers on this problem suggests, however, that efficient execution of this task, with reasonable memory use, is far from trivial. The most successful

"KMC 2: fast and resource-frugal k-mer counting"
(Bioinformatics, 2015)
Deorowicz et al.

16

BFCounter (2011)

Melsted and Pritchard *BMC Bioinformatics* 2011, **12**:333
http://www.biomedcentral.com/1471-2105/12/333

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

Efficient counting of *k*-mers in DNA sequences using a bloom filter

Páll Melsted[1*] and Jonathan K Pritchard[1,2*]

Abstract

Background: Counting *k*-mers (substrings of length *k* in DNA sequence data) is an essential component of many methods in bioinformatics, including for genome and transcriptome assembly, for metagenomic sequencing, and for error correction of sequence reads. Although simple in principle, counting *k*-mers in large modern sequence data sets can easily overwhelm the memory capacity of standard computers. In current data sets, a large fraction-often more than 50%-of the storage capacity may be spent on storing *k*-mers that contain sequencing errors and which are typically observed only a single time in the data. These singleton *k*-mers are uninformative for many algorithms without some kind of error correction.

Results: We present a new method that identifies all the *k*-mers that occur more than once in a DNA sequence data set. Our method does this using a Bloom filter, a probabilistic data structure that stores all the observed *k*-mers implicitly in memory with greatly reduced memory requirements. We then make a second sweep through the data to provide exact counts of all nonunique *k*-mers. For example data sets, we report up to 50% savings in memory usage compared to current software, with modest costs in computational speed. This approach may reduce memory requirements for any algorithm that starts by counting *k*-mers in sequence data with errors.

Conclusions: A reference implementation for this methodology, BFCounter, is written in C++ and is GPL licensed. It is available for free download at http://pritch.bsd.uchicago.edu/bfcounter.html

17

FACS (2010)

BIOINFORMATICS ORIGINAL PAPER Vol. 26 no. 13 2010, pages 1595–1
doi:10.1093/bioinformatics/btq

Sequence analysis Advance Access publication May 13, 2

Classification of DNA sequences using Bloom filters

Henrik Stranneheim[1,*], Max Käller[2], Tobias Allander[3], Björn Andersson[4], Lars Arvestad[5] and Joakim Lundeberg[1,*]

[1]Science for Life Laboratory, KTH Royal Institute of Technology, SE-100 44 Stockholm, [2]LingVitae AB, Roslagstullsbacken 33, 114 21 Stockholm, [3]Department of Microbiology, Laboratory for Clinical Microbiology, Tur and Cell Biology, Karolinska University Hospital, Karolinska Institutet, SE-17176 Stockholm, [4]Department of Cell a Molecular Biology, Karolinska Institutet, SE-17177 Stockholm and [5]School of Computer Science and Communication, Stockholm Bioinformatics Center, AlbaNova University Center, Royal Institute of Technology, 106 91 Stockholm, Sweden

Associate Editor: Alex Bateman

3.2 Classification of reads using FACS

For a query to be classified as belonging to a reference genome, it needs to accumulate a match score surpassing a chosen cut-off value, which is based on sequence similarity.

In this study, each query sequence was divided by a sliding window into K-mers and each K-mer was interrogated against the Bloom filter. If a match was found for a K-mer then the sequential K-mers beginning within the matching K-mer were not queried further. Each query accumulated a match score that was related to the K-mer size for every positive hit in the filter, i.e. 1 point for every base. A filtering step called Quick Pass, was included in the algorithm, to faster discard non-significant alignments. Quick Pass was set to use a K-mer offset sliding window approach on the query and to require only one K-mer match in the reference to initiate the more thorough analysis with a 1-base offset sliding window approach. Initial tests showed a high degree of false positives in classification when analysing very short sequences; this was expected due to the small ratio of the query sequence and K-mer size. Therefore, query sequences shorter than 61 bases were removed from the analysis. Sequences classified in each analysis as belonging to a reference were withdrawn from further querying since they do not represent 'novel' reads.

18

FACS (2010)

BIOINFORMATICS ORIGINAL PAPER Vol. 26 no. 13 2010, pages 1595–1
doi:10.1093/bioinformatics/btq

Sequence analysis Advance Access publication May 13, 2

Classification of DNA sequences using Bloom filters

Henrik Stranneheim[1,*], Max Käller[2], Tobias Allander[3], Björn Andersson[4], Lars Arvestad[5] and Joakim Lundeberg[1,*]

[1]Science for Life Laboratory, KTH Royal Institute of Technology, SE-100 44 Stockholm, [2]LingVitae AB, Roslagstullsbacken 33, 114 21 Stockholm, [3]Department of Microbiology, Laboratory for Clinical Microbiology, Tum and Cell Biology, Karolinska University Hospital, Karolinska Institutet, SE-17176 Stockholm, [4]Department of Cell a Molecular Biology, Karolinska Institutet, SE-17177 Stockholm and [5]School of Computer Science and Communication, Stockholm Bioinformatics Center, AlbaNova University Center, Royal Institute of Technology, 106 91 Stockholm, Sweden

Associate Editor: Alex Bateman

Se ho coverage 30, e kmer che compaiono 2 volte, li tolgo.

Li posso eventualmente correggere computando la distanza da quelli più frequenti.

3.2 Classification of reads using FACS

For a query to be classified as belonging to a reference genome, it needs to accumulate a match score surpassing a chosen cut-off value, which is based on sequence similarity.

In this study, each query sequence was divided by a sliding window into K-mers and each K-mer was interrogated against the Bloom filter. If a match was found for a K-mer then the sequential K-mers beginning within the matching K-mer were not queried further. Each query accumulated a match score that was related to the K-mer size for every positive hit in the filter, i.e. 1 point for every base. A filtering step called Quick Pass, was included in the algorithm, to faster discard non-significant alignments. Quick Pass was set to use a K-mer offset sliding window approach on the query and to require only one K-mer match in the reference to initiate the more thorough analysis with a 1-base offset sliding window approach. Initial tests showed a high degree of false positives in classification when analysing very short sequences; this was expected due to the small ratio of the query sequence and K-mer size. Therefore, query sequences shorter than 61 bases were removed from the analysis. Sequences classified in each analysis as belonging to a reference were withdrawn from further querying since they do not represent 'novel' reads.

19

Jellyfish (2011)

BIOINFORMATICS ORIGINAL PAPER Vol. 27 no. 6 2011, pages 764–770
doi:10.1093/bioinformatics/btr011

Sequence analysis Advance Access publication January 7, 2011

A fast, lock-free approach for efficient parallel counting of occurrences of *k*-mers

Guillaume Marçais[1,*] and Carl Kingsford[2]

[1]Program in Applied Mathematics, Statistics and Scientific Computation and [2]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

Associate Editor: Alex Bateman

- Designed for shared memory parallel PCs
- Uses lock-free data structures
  - Lock-free queues for communication between threads
  - Lock-free hash tables
- Memory-aware
  - Key compression scheme (constant space for each hash table entry)

20

KMC3…

Bioinformatics, 31(10), 2015, 1569–1576
doi: 10.1093/bioinformatics/btv022
Advance Access Publication Date: 20 January 2015
Original Paper

Sequence analysis

KMC 2: fast and resource-frugal *k*-mer counting

Sebastian Deorowicz[1,*], Marek Kokot[1], Szymon Grabowski[2] and Agnieszka Debudaj-Grabysz[1]

[1]Institute of Informatics, Silesian University of Technology, Akademicka 16, 44-100 Gliwice and [2]Institute of Applied Computer Science, Lodz University of Technology, Al. Politechniki 11, 90-924 Łódź, Poland

- Reduces the disk usage of KMC1
- Improves the speed
- Uses signatures of k-mers **(minimizer)**

ci ritorneremo mercoledì

21

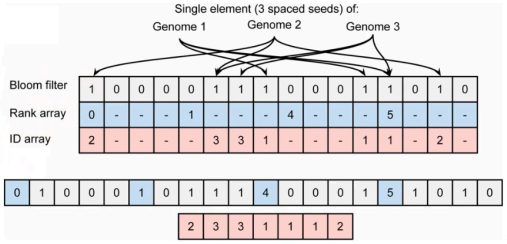
Data structure need to be scalable and have fast access time:
multi-index Bloom filter

Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multiindex Bloom filters

Justin Chu, Hamid Mohamadi, Emre Erhan, +3, and Inanc Birol, Authors Info & Affiliations

Edited by David L. Donoho, Stanford University, Stanford, CA, and approved June 12, 2020 (received for review March 1, 2019)
July 8, 2020 | 117 (29) 16961-16968 | https://doi.org/10.1073/pnas.1903436117

https://github.com/bcgsc/btllib



- Associates read signatures with an ID
- Reduction in memory footprint vs a 32-bit index vector

usato in GoldRush

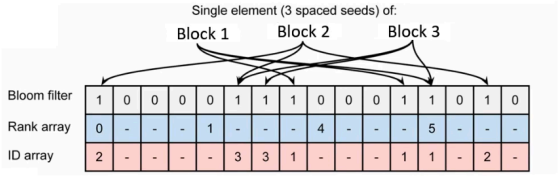
22

usati in goldrush

The read is divided into blocks of length t × b

- t is the length of a tile
- b is the number of tiles in a block

All read signatures inside a block are inserted with the same ID



23

cascade Bloom Filter (per gestire i falsi positivi)

Salikhov *et al. Algorithms for Molecular Biology* 2014, **9**:2
http://www.almob.org/content/9/1/2

AMB ALGORITHMS FOR MOLECULAR BIOLOGY

RESEARCH Open Access

Using cascading Bloom filters to improve the memory usage for de Brujin graphs

Kamil Salikhov[1], Gustavo Sacomoto[2,3] and Gregory Kucherov[4,5*]

24

Minia usa BF per il dBG

guardiamo il pdf

25

Abyss 2.0 usa BF per il dBG

guardiamo il pdf

26

https://blog.roblox.com/2023/11/roblox-reduces-spark-join-query-costs-machine-learning-optimized-bloom-filters/

27