k-mer data structures

Rayan Chikhi
CNRS, Univ. Lille, France

CGSI - July 24, 2018

Baseline problem

In-memory representation of a large set of short k-mers:

e.g.

ACTGAT

GTATGC

ATTAAA

GAATTG

...

(Indirect) applications

- Assembly
- Error-correction of reads
- Detection of similarity between sequences
- Detection of distances between datasets
- Alignment
- Pseudoalignment / quasi-mapping
- Detection of taxonomy
- Indexing large collections of sequencing datasets
- Quality control
- Detection of events (e.g. SNPs, indels, CNVs, alt. transcription)
- ...

Goals of this lecture

- Broad sweep of state of the art, with applications
- Refresher of basic CS elements

Au programme:

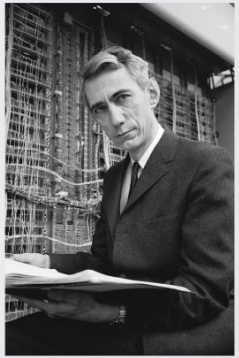
- Basic structures (Bloom Filters, CQF, Hashing, Perfect Hashing)
- k-mer data structures (SBT, BFT, dBG ds)
- Some reference-free applications

k-mers

Sequences of k consecutive letters, e.g. ACAG or TAGG for k=4

N.G. de Bruijn (1946),
de Bruijn sequences [1]

C. Shannon (1948),
information theory [2]



Framing the problem

Large set of k-mers : 10^{6} - 10^{11} elements

k in [11; 10^{3}]

Problem statement:

Representation of a set of k-mers:

```
ACTGAT

GTATGC

..
```

Operations to support

- Construction (from a disk stream)
- Membership (“is X in the set?”)
- Iteration (enumerate all elements in the set)
- ...

Extensions:

- Associate value(s) to k-mers (e.g. abundance)

- Navigate the de Bruijn graph

Problem statement:

Representation of a set of k-mers:

ACTGAT

GTATGC

..

10^6 - 10^{11} elements

k: 11 - 500

-

Data structures

*“In computer science, a **data structure** is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently.”*

https://en.wikipedia.org/wiki/Data_structure

“Data structures can implement one or more particular abstract data types (ADT) [..]”

Problem statement:

Representation of a set of k-mers:

`ACTGAT`

`GTATGC`

. .

10^6 - 10^{11} elements

k: 11 - 500

Operations:

- Construction
- Membership
- Iteration

Abstract data type

*"In computer science, an **abstract data type** (**ADT**) is a mathematical model for data types [..], a class of objects whose logical behavior is defined by a set of values and a set of operations"*

"[..] analogous to an algebraic structure in mathematics. " (e.g. set, group, ring, field, etc)

*"a **data structure** implements one (or more) **ADT**(s)"*

ADT examples

- List
- Set
- Multiset
- Map
- Multimap
- Graph
- Stack
- Queue
- Priority queue

Data structures examples

- Array
- Linked list
- B-tree
- Hash table
- FM-index

Corresponding data structures

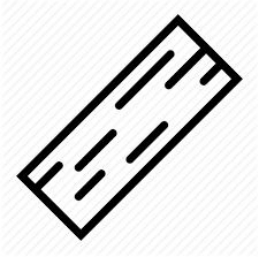
- List -> array, linked list
- Set -> B-tree, hash table
- Multiset -> array, linked list
- Map -> hash table
- Multimap -> hash table of lists
- Graph -> list of tuples, hash table
- Stack -> array, linked list

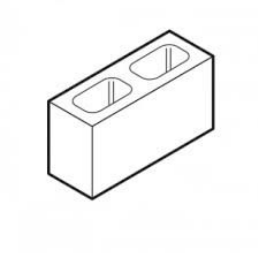
Analogy

ADTs

Data structures

Building blocks (which are also ADTs/data structures)





(note the overlaps of functionalities)

k-mer ADTs and data structures

Building block ADT

1. Set
2. Tree
3. Full-text index

Corresponding **data structures**

1. Hash table, Bloom Filter, ..
2. Pointer-based or parenthesis tree
3. BWT, FM-Index, ..

k-mer ADT

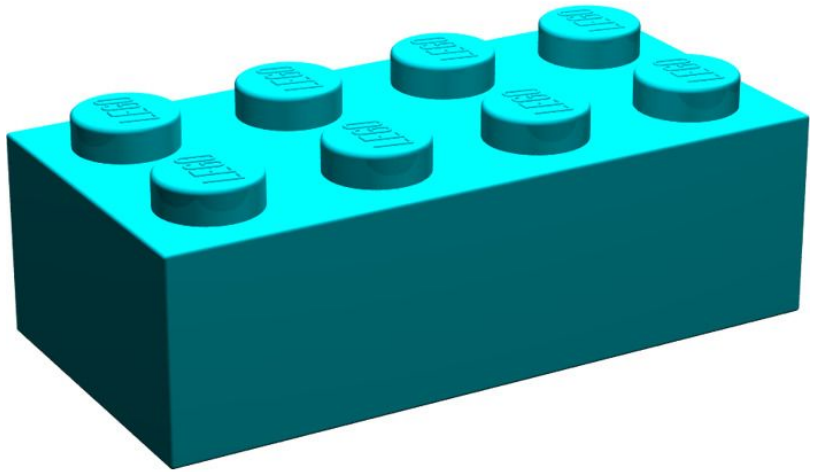
1. Set of k-mers
2. Dictionary of k-mer/values
3. De Bruijn Graph
4. Read index
5. Set of sets of k-mers
6. K-mer/abundance matrix

1. Bloom Filter, hash table, CQF, ..
2. Hash table, MPHF
3. BOSS, Minia, FDBG, ..
4. FM-Index
5. SBT, Mantis, BFT, ..
6. MPHF+compressed matrix

Recent methods are combinations of building blocks

- BOSS data structure: FM-index + bit arrays
- Minia: Bloom Filter + hash table
- Bloom Filter Tries: sort of Burst Tries + Bloom Filters
- Sequence Bloom Trees: Bloom Filter + tree
- Fully Dynamic De Bruijn Graphs: MPHF + tree
- Static/Dynamic Bit Arrays: compressed bit arrays
- SeqOthello: sort of MPHF
- Mantis: Counting Quotient Filters
- BIGSI: matrix of Bloom filters
- ..

Building blocks for k-mer set representations



Building block: Unsorted List

[GAGG, ACAT, CATC, …]

- O(k) insertion
- O(nk) deletion
- O(nk) search

Building block: Sorted List

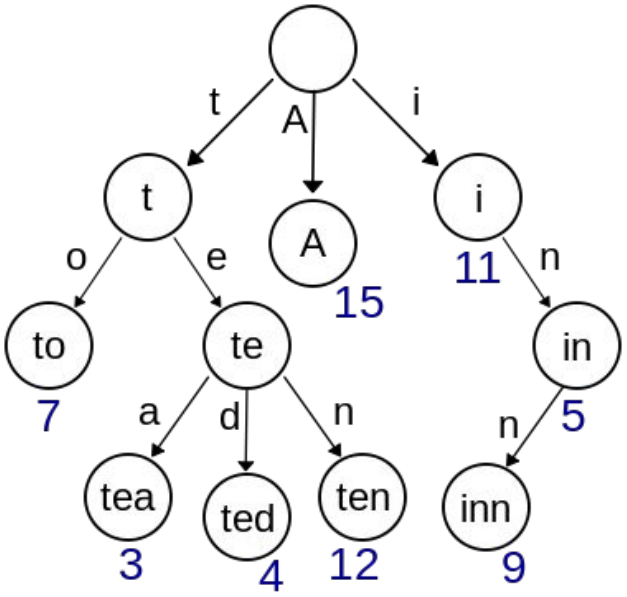
[ACAT, CATC, GAGG, …]

- O(nk) insertion
- O(nk) deletion
- O(k log2(n)) search

Building block: Tries

Worst case

- O(k) insertion
- O(k) deletion
- O(k) search



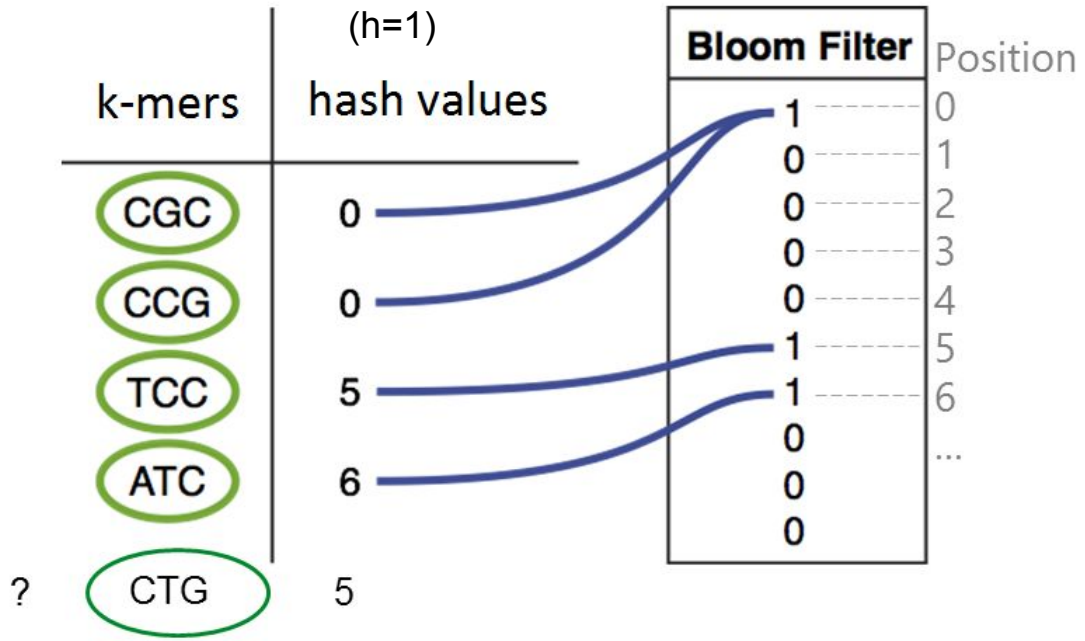
Building Block: Bloom filter

Init a bit array
Take **h** hash functions

Insertion: put 1's at positions given by hash functions

Query: are there 1's at all positions given by hash functions?

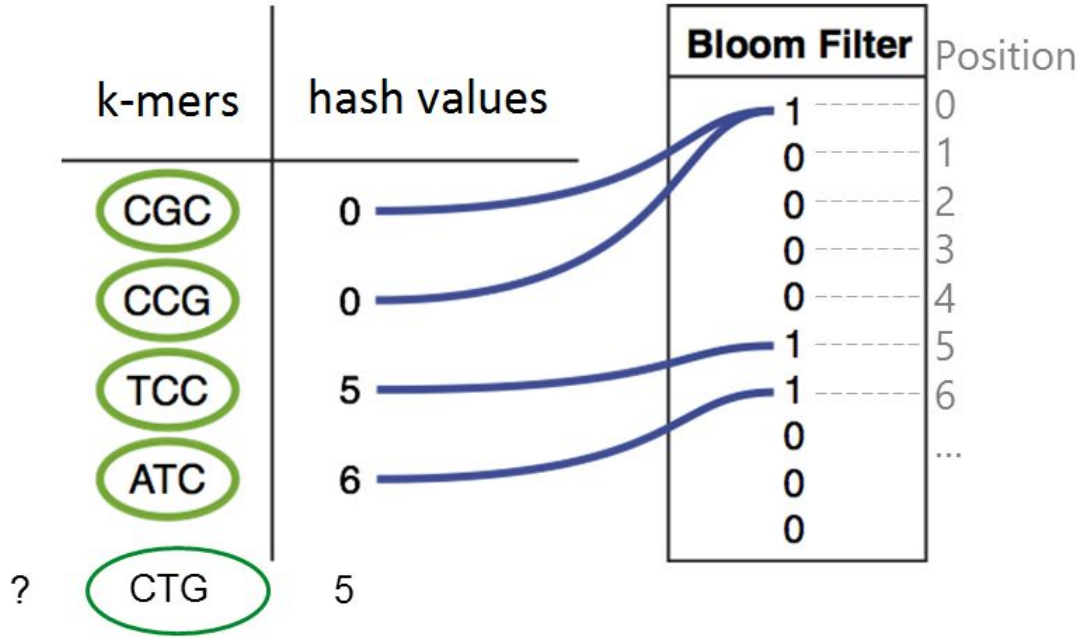
- O(hk) insertion
- (O(hk) deletion)
- O(hk) query



Building Block: Bloom filter

- Are the queries exact?
- Can it support iteration?

Space: m = 1.44n log2(ε) where ε is the false positive rate



Building Block: Bloom Filter

- Good for error-tolerant membership testing (e.g. initial filter)
- Easy to implement

- Can use off-the-shelf hash functions (but *not* std::hash)

```
unsigned int hash(unsigned int x) {
      x = ((x >> 16) ^ x) * 0x45d9f3b;
      x = ((x >> 16) ^ x) * 0x45d9f3b;
      x =  (x >> 16) ^ x;
return x; }
```

- `from pybloom import BloomFilter`

Data structure: Sequence Bloom Trees

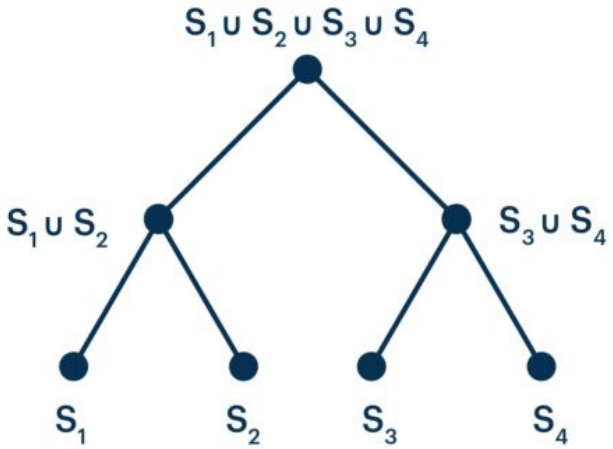


Fig: https://www.sevenbridges.com/sequence-bloom-trees-principles/

Leaf: Bloom Filter of a sequencing dataset
Internal nodes: Bit-wise union of children BF's

- Representation of sets of k-mer sets
- Approximate membership across all datasets in O(hits) instead of O(datasets)
- No k-mer iteration
- Insertion/deletion of complete datasets
- Whole structure resides on disk

Application: fast sequence search in 1000's of RNA-seq experiments

Solomon, Kingsford, *Nat Biotech 2017*
Sun, Harris, Chikhi, Medvedev, *RECOMB 2017*
Solomon, Kingsford, *RECOMB 2017*

Data structure: Bloom Filter Tries

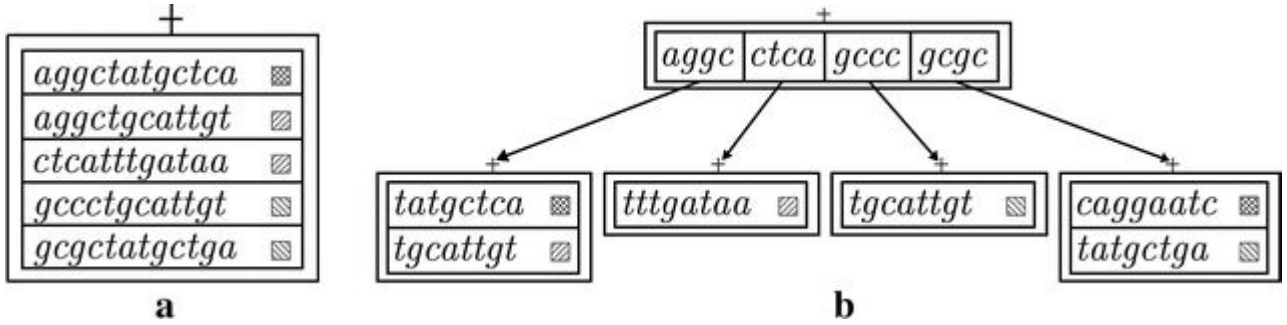


Fig:BFT article

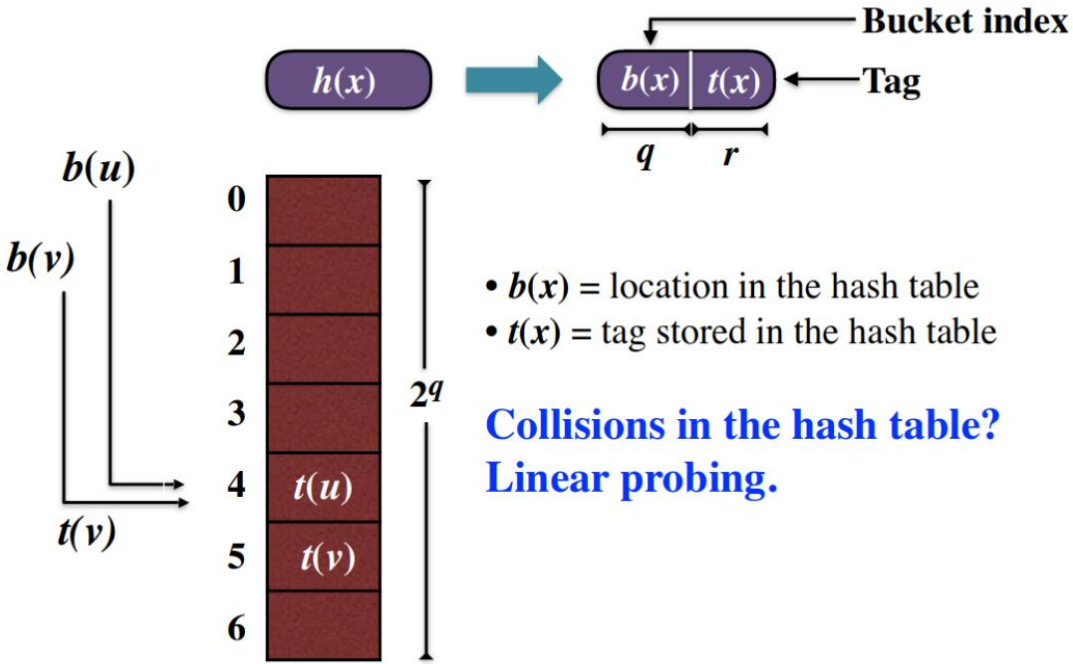
Principle: cut k-mers into chunks, insert in a burst trie, Bloom Filters added for speed

- Representation of sets of k-mer sets
- Tailored to pan-genomes: a single k-mer belongs to many sets
- Explicit dBG operations support

Application: indexing and compression of pan-genomes

Holley, Wittler, Stoye, *WABI 2016*

Data structure: Counting Quotient Filter



Hybrid between a *compact hash table* and a Bloom Filter.

Approximate membership

- O(k) insertion
- (O(k) deletion)
- O(k) query

Fig: P. Pandley, SIGMOD 2017
https://pdfs.semanticscholar.org/6bde/f4a86108309086de4071c9d28d97565a84a4.pdf

Pandley, Bender, Johnson, Patro, *SIGMOD 2017*

Building block: Hash table

On average:

- O(k) insertion
- O(k) deletion
- O(k) search

Worst case:

- O(nk) everything

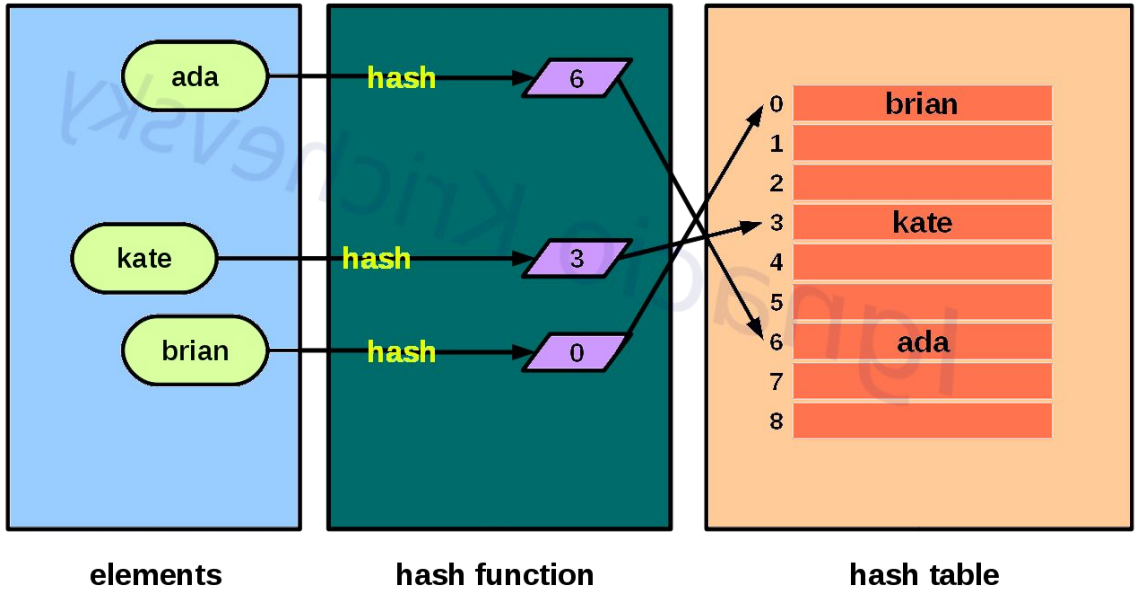
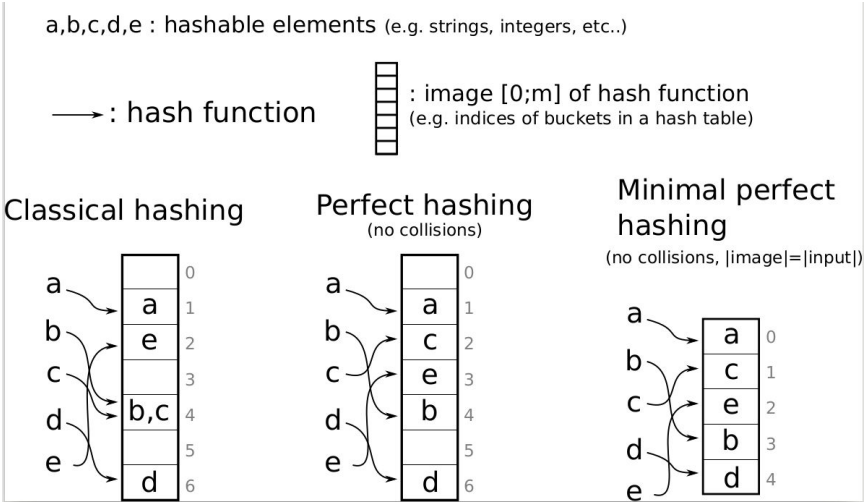


Fig: http://www.kriche.com.ar/root/programming/data%20structures%20&%20algorithms/hashTable.html

Building block: Perfect hashing

- Smaller than a classical hash table
- Only needs to store the hash function and values, not the keys, because there is no need to check for collisions
- Cannot test for membership, but can do key-value dictionary



Building Block: Perfect hashing

Naive method to create a perfect hash function.

Let’s pick a random hash function and see if it’s perfect.

How random? Fully random?

If so, would need to store this hash function somehow.

m = universe size, so **m*log2(m)** bits (with log2(m) = 32 typically) to store function

Need another type of function than fully random.

Source https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf

Building Block: Universal hashing

A family of hash functions having the same desirable properties as random hash functions:

- A randomized algorithm for constructing hash function over universe {1...m} is **universal** if, for x and y fixed (x≠y), $\mathbf{Pr_h(h(x) = h(y)) \leq 1/m}$

This is exactly the probability of collision we would expect if the hash function was truly random.

Universal hashing example

Universal hash function:
$Pr_h(h(x) = h(y)) <= 1/m$

Universe size: m

$h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$ (a,b **randomly chosen mod p**, a!=0, p prime)

is universal

Proof: collision if $ax+b = ay+b+t*m \pmod p \Leftrightarrow a = t*m*(x-y)^{-1} \pmod p$
With t taking values in [0; integer value of p/m]

p-1 choices for a ; (p-1)/m possible non-zero values for $(t*m*(x-y)^{-1})$

Collision probability: $((p-1)/m)/(p-1) <= 1/m$

Source: https://en.wikipedia.org/wiki/Universal_hashing

Building Block: Perfect hashing

Naive method to create a perfect hash function

- To hash **n** elements, just set **$m = \alpha n^2$** with **α**>=2**,** run a **randomized algorithm:**
  - choose a random hash function from an **universal** collection
  - check for collision
  - If none, return the function.
  - Otherwise, retry

- Will it run forever? No, only 2 tries are expected

Building Block: Perfect hashing of k-mers, wrap-up

- Recommended method for large dictionaries of kmers
- Only if one doesn't need to test membership (*keys are not stored*)

- Can store all distinct k-mers of human genome (around 2.5 billion) in **~3** bits/kmer, i.e. 1 GB.

Fast and scalable minimal perfect hashing for massive key sets

Antoine Limasset, Guillaume Rizk, Rayan Chikhi, Pierre Peterlongo (2017)

ADT: K-mer matrices

	Sample 1	Sample 2	Sample 3
AAAGT	0	0	20
AACTG	20	10	0
...	..	..	..

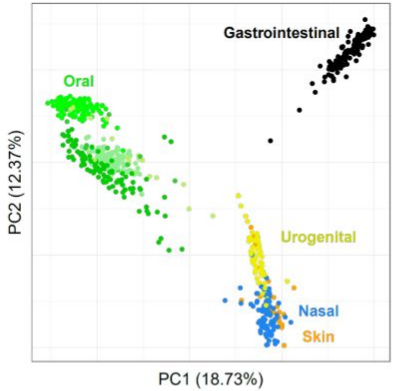
Applications:

- Clustering of metagenomes
- Reference-free detection of events in collections of transcriptomes and genomes
- more?

Usually represented as a flat disk file
or
Stored in memory using MPHF

Benoit *et al, PeerJ 2016*
Audoux *et al, Genome Biology 2018*
Rahman *et al, bioRxiv 2017*

Application: direct comparison of metagenomes

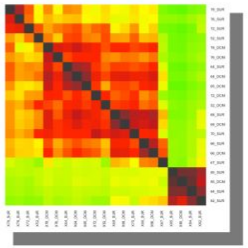


Human Microbiome Project
n=690 samples
(**32 billion reads**)
0.5 day computation

k-mer matrix

	A	B	…	N
ACGAG	2	4		0
CGAGC	2	1		9
GAGCT	0	0		5

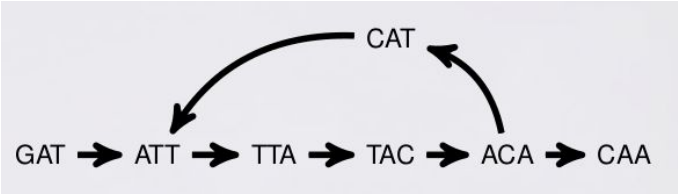
all-vs-all dataset similarity estimation
(Jaccard, Bray-Curtis)



Benoit *et al, PeerJ 2016*

ADT: de Bruijn graphs

- k-mer set, and also..
- Iteration of neighbors in the dBG
- Implicit support for reverse-complements (bi-directed)



Construction algorithms:

- BCALM 2
- ABySS

Blog post: “**Bi-Directed Graphs in BCALM 2”**

Simpson *et al, Genome Res 2009*
Chikhi *et al, RECOMB 2014*

Data Structures for de Bruijn graphs

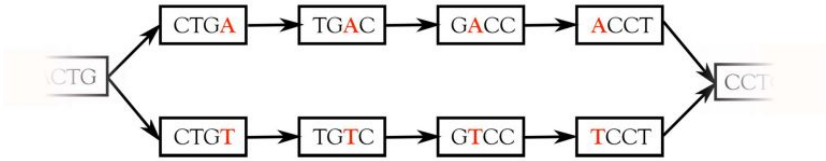
- BOSS: FM-index over k-mers
- Dbgfm: FM-index over unitigs
- Minia: Bloom filter with hash table of false positives
- Fully Dynamic DBG: MPHF with a tree for false positives
- ...

Applications:

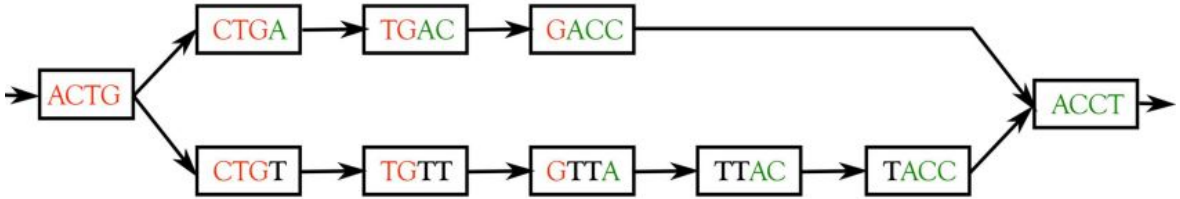
- de novo assembly of genomes and metagenomes
- Error-correction of 2nd and 3rd generation sequencing data
- Reference-free variant detection
- Transcriptome quantification
- Pan-genome representation (*colored de Bruijn graph*)

Bowe, Onodera, Sadakane, Shibuya, *WABI 2012*
Chikhi, Rizk, *WABI 2012*
Chikhi *et al, RECOMB 2014*
Boucher *et al, DCC 2015*
Crawford *et al, Bioinformatics 2018*

Application: Reference-free SNP detection



- Two paths of length *k* nodes:
- Provides two sequences of length 2k-1:
  - CTGACCT
  - CTGTCCT



Principle: detect SNPs using only the dBG

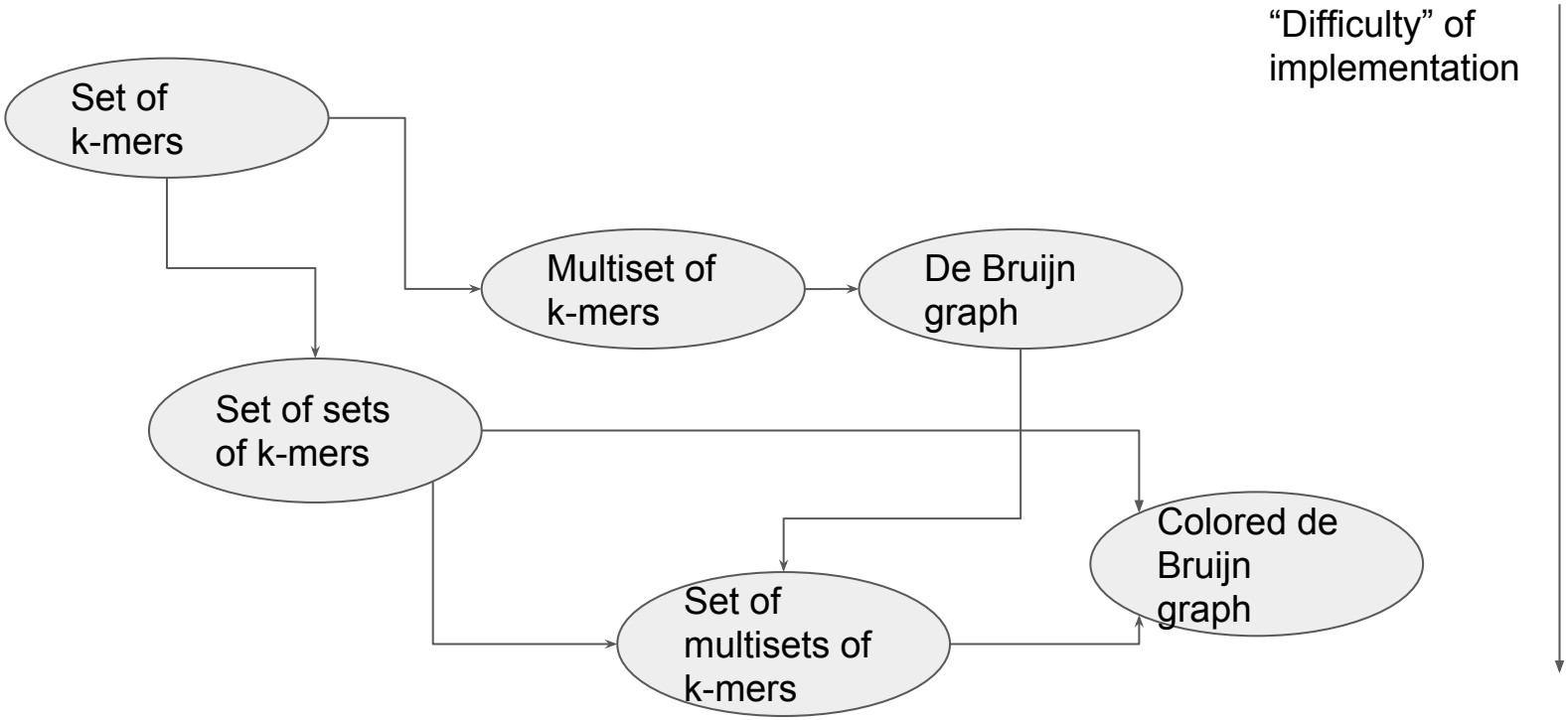
Steps:

1. Construct dBG
2. Detect bubble motifs
3. Scan reads to compute coverage per event
4. Generate .fa or .vcf results

Pros: no reference necessary, no read mapping ambiguity
Cons: more resources-intensive, more false positives

Iqbal *et al, Nat Gen 2012*
Uricaru *et al, NAR 2015*
Alipanahi *et al, RECOMB-Seq 2018*

Reductions


“Difficulty” of implementation
Set of k-mers
Multiset of k-mers
De Bruijn graph
Set of sets of k-mers
Colored de Bruijn graph
Set of multisets of k-mers

Conclusion

- k-mer data structures are pervasive in sequence bioinformatics
- Building blocks:
  - Hash tables
  - Perfect hash functions
  - Bloom filters
  - Trees
- More advanced k-mer ADTs and data structures:
  - (Colored) de Bruijn graphs
  - Set of sets of k-mers
  - K-mer matrices
- Applications:
  - Virtually all aspects of sequencing data analysis
- GATB library, www.gatb.fr