Modern hashing for alignment-free sequence analysis

Part 1: Introduction (*k*-mers, alignment-free methods)

Jens Zentgraf & Sven Rahmann

GCB 2021

Tentative time table

1. Introduction: motivation, *k*-mers, alignment-free methods
   (09:00 - 09:45, Sven)
2. Hashing, hash functions, collision resolution
   (09:45 - 10:30, Jens)

   Short break (20 min)

3. Multi-way bucketed cuckoo hashing for DNA *k*-mers
   (10:50 - 11:30, Jens)
4. Performance engineering
   (11:30 - 12:00, Sven)

Foundation of most DNA sequence analysis tasks in bioinformatics

1. Read mapping: Find genomic origin(s) of a given DNA sequence (the "read")

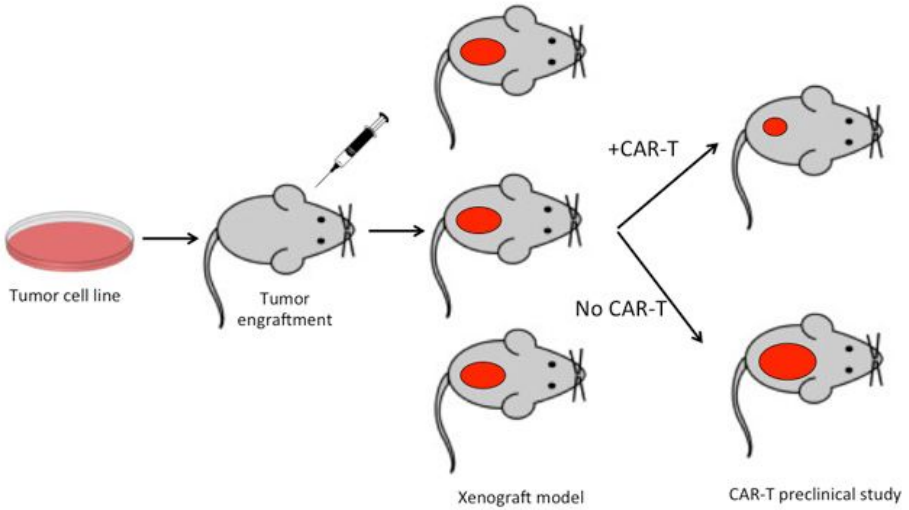
2. Read alignment: Base-by-base comparison of read and genome (often mingled together, but really 2 distinct steps!)

This tutorial: How to short-cut mapping and avoid alignment

- Find all exact occurrences of short *k*-mers (DNA substrings of length *k*)
- Do this fast, for billions for *k*-mers

Motivation: Xenograft sorting

(Patient-derived) xenografts

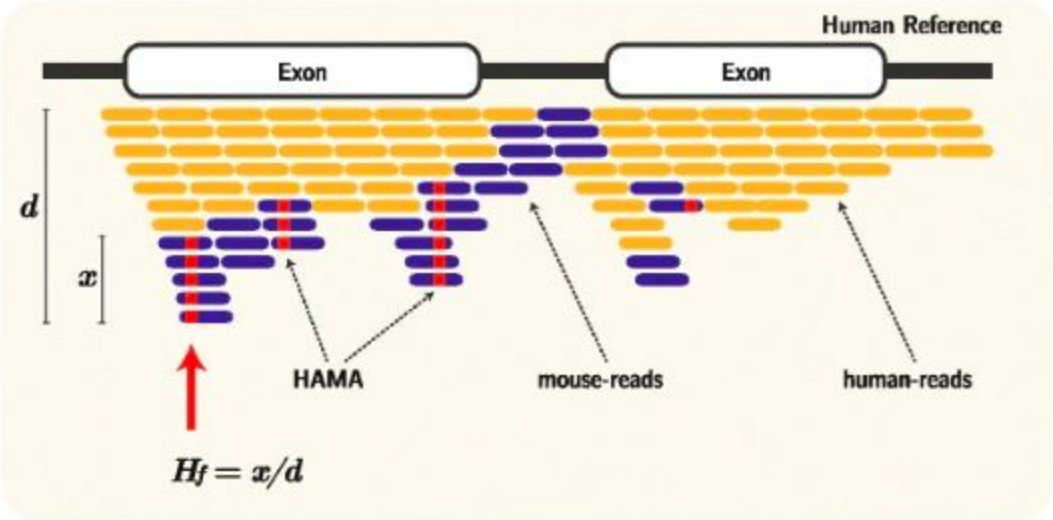


- tumor cell lines
  or patient tumor samples
  implanted in mice
- study tumor heterogeneity,
  evolution
- sequencing of samples
- mixture of human+mouse DNA
- First task: separate/sort reads
  ("xenograft sorting"), or:
  extract graft (human) reads

Source: Creative AniModel,
https://www.creative-animodel.com/Featured-Service/Human-Tumor-Xenograft-Model.html

Problem: Human-Aligned Mouse Alleles (HAMAs)

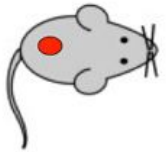
- mouse reads may align to human genome
- may lead to false human (tumor) variant calls
- oncogenes particularly prone to this effect



S. Y. Jo, E. Kim, and S. Kim.
Impact of mouse contamination in genomic profiling of patient-derived models and best practice for robust analysis.
Genome Biology, 20(1):Article 231, Nov 2019.

The xenograft sorting problem

Given: sequenced xenograft sample (reads from two species),
paired-end or single-end,
genomic or transcriptomic reads,



sort the reads into five categories according to species of origin:
host (mouse), graft (human), both, neither, ambiguous

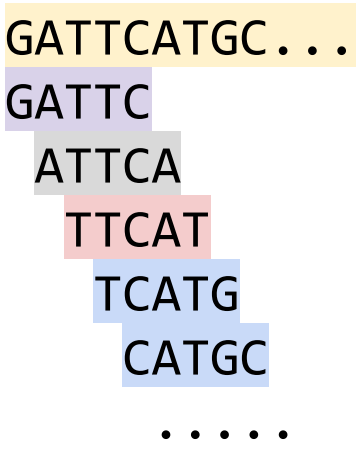
or: partially sort using fewer categories (host, graft, other),
or: count how many reads are in each category,
or: filter (select) only graft (human) reads.

k-mer methods for xenograft sorting

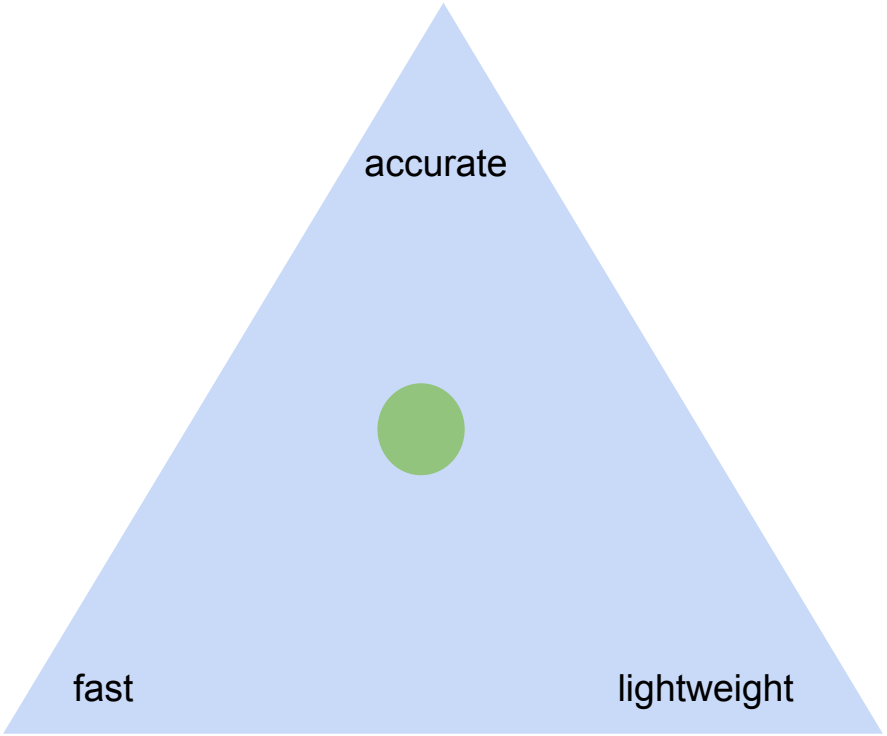
- Partition each read into its *k*-mers

- Look up information on each *k*-mer in hash table
  [*k*-mer ↦ human | mouse | both]
- Absent k-mers occur in neither species.

- Aggregate *k*-mer information
  into a statement about the read
  (majority vote, complex decision rule, …).



Goal: "Fast lightweight accurate xenograft sorting"



fast:

- slow random memory accesses
- 3-way bucketed Cuckoo hashing
- buckets fit within a cache line

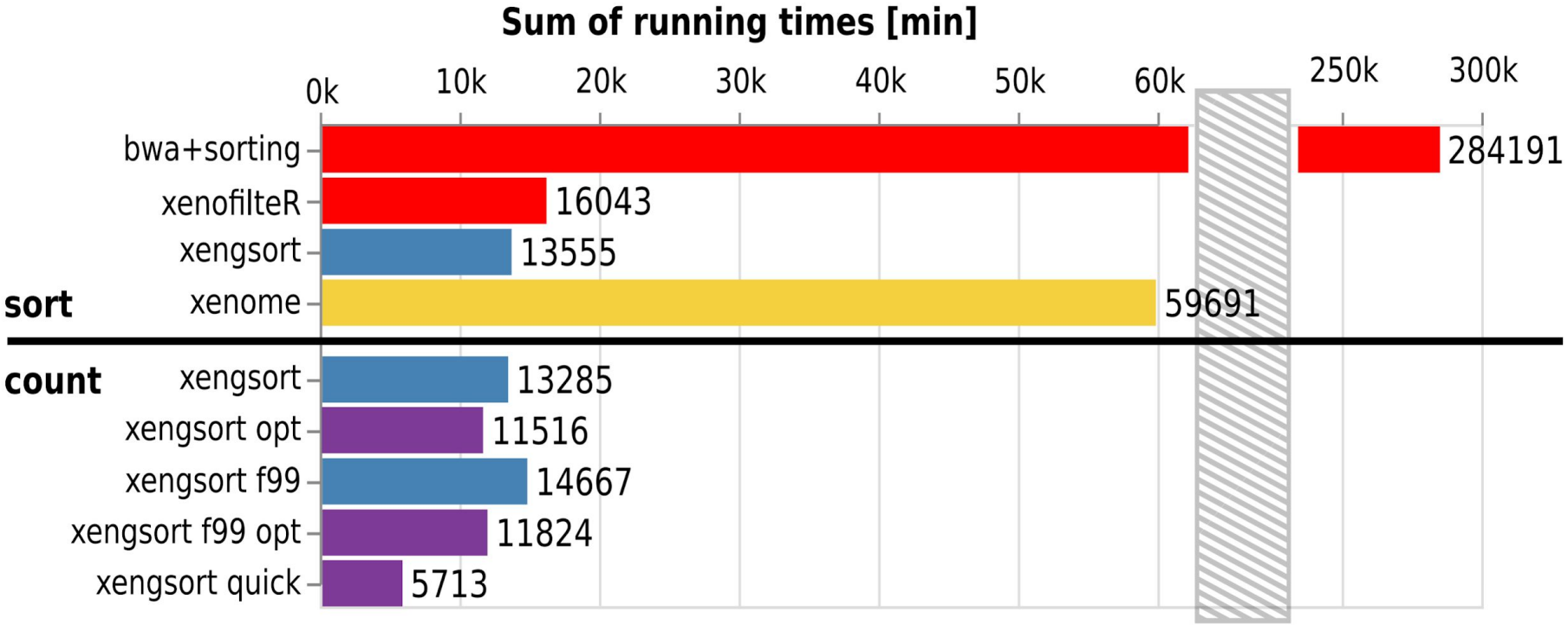
lightweight (small memory footprint):

- 4.5 billion 25-mers + values
- high load (little wasted space)
- quotienting

accurate:

- identical + highly similar sequences
- "weak" *k*-mers
- multi-level decision rule

174 PDX datasets: Running times [CPU minutes]



Zentgraf and R., WABI 2020

Examples of alignment-free methods

1. **Xengsort:** Xenograft sorting (already discussed)
   (https://gitlab.com/genomeinformatics/xengsort - 2020)
2. **BCOOL:** Sequencing error correction
   (https://github.com/Malfoy/BCOOL - 2019)
3. **Kraken 2:** metagenomic species identification and quantification
   (https://ccb.jhu.edu/software/kraken2/ - Sep 2019)
4. **Kallisto:** RNA-seq transcript quantification
   (https://pachterlab.github.io/kallisto/ - 2016);
   not for differential expression; use additional tools like sleuth
5. **DE-kupl:** discovery of novel (differentially expressed) transcripts
   (https://transipedia.github.io/dekupl/ - 2017)

K-mers and their encodings, Minimizers and sketches

k-mers and their integer encodings

***k*-mer:** any DNA/RNA sequence of length *k*.
There are 4^k different DNA *k*-mers.

Other names: *k*-mer, *q*-gram, *n*-gram, *ℓ*-mer, shingle, ...

k-mer code / encoded *k*-mer: Translating A=0, C=1, G=2, T=3
(or any other bijective map {A,C,G,T} → {0,1,2,3}) for fixed *k*,
a *k*-mer becomes an integer (base-4 number) in {0, 1, …, 4^k-1}.

Example: TATCG ↦ $(30312)_4 = 3 \cdot 256 + 0 \cdot 64 + 3 \cdot 16 + 1 \cdot 4 + 2 \cdot 1 = 822$

Canonical *k*-mers

canonical *k*-mer: DNA is double-stranded;
a *k*-mer ist the same molecule as its reverse complement,
the canonical representation is the **lexicographically smaller** one.
Example: TATCG = CGATA, canonical: CGATA.

canonical code: integer code of canonical *k*-mer
minimum of encodings of *k*-mer and its reverse complement;
always need to evaluate both *k*-mer *x* and rc(*x*).
Example: code(TATCG) = code(CGATA) = min(822, 716) = 716.

Note: works equally well with max() instead of min()

Contiguous vs. gapped *k*-mers

contiguous *k*-mer (standard):

k-mer that occurs as one contiguous substring

gapped / spaced *k*-mer and mask:

- gap pattern given by (symmetric!) mask: e.g.: #__#__#__#
- #: significant positions (*k*) vs. _: gap positions / spacers (*s*)
- *k*-mer by concatenating significant positions (**weight** *k*, **span/width** *w* = *k*+*s*)
- advantage: cover sequence width in fewer steps

```
Example: AGGTCGGTAGGC          AGGTCGGTAGGC      3 k-mers cover
         #__#__#__#  ATGG      ####             12 positions (gapped)
          #__#__#__#  GCTG      ####             6 positions (cont.)
           #__#__#__#  GGAC      ####
```

Key-value stores

General definition:

A key-value store ("key-value database") is a data structure that stores objects or records ("values"), each of which is associated to an immutable "key" object.

Examples:

- Java HashMap
- Python dict
- Databases: redis, Oracle NoSQL, memcached, ...

Restricting values:

Values in key-value databases may be any object, even with different types!
Keys can be any immutable hashable object (often strings or tuples of numbers).

We assume that the value type is known and fixed (value set $V = \{0, .., |V|-1\}$, so values have fixed bit width (e.g. 32 bits).

(Circumvented by storing pointers to arbitrary objects -- what the databases do anyway)

Minimizers

Given: Two integers $k \leq w$ (w: "window width" in a DNA sequence)

Definition: A (canonical) k-mer m is a **minimizer** in a window of length w iff

- m is a (canonical) k-mer in the window,
- its (canonical) code is the **smallest** of all (canonical) codes in the window.
- The "smallest" may be with respect to a permutation of k-mers.

Advantages of minimizers

- Minimizers tend to stay (locally) constant for overlapping windows
- There are fewer different minimizers than windows
- Similar sequences have high probability of having the same minimizer(s).
- Sequence of minimizers is also called a **sketch** of the original sequence.

Minimizers

Example: k = 3, w = 6 (4 k-mers), **AGGTCGGTAGGC**

k-mer	cc_{max}	minim(6)
AGG	23 (CCT)	23
GGT	43	26
GTC	45	26
TCG	54	26
CGG	26	26
GGT	43	23
GTA	49	23
TAG	50	-/-
AGG	23	-/-
GGC	41	-/-

cc_{max}	cc xor (101010)	minim(6)
23 (CCT)	61	1
43	1	1
45	7	1
54	28	1
26	48	1
43	1	1
49	27	3
50	24	-/-
23	61	-/-
41	3	-/-

Note:
xor-ing canonical codes with random numbers and taking the minimum "simulates" different random permutations of numbers w.r.t. taking the minimum.

$43 = (101011)_2$
xor $(101010)_2$
$=1 = (000001)_2$

Data structures for key-value-stores (in memory)

Two basic possibilities to look up keys fast:

- **sorting** (binary search)
  - variants of lists (e.g. skip lists)
  - (balanced) search trees
- **hashing** (compute an address / index in an array)
  - typically arrays, but may need to be re-sized
  - collisions must be resolved
- **hybrids** (binning/hashing by prefix, sorted within bin)

Note: on small datasets, do nothing, linear scan is fast enough!

Applications of the alignment-free paradigm

Examples of alignment-free methods

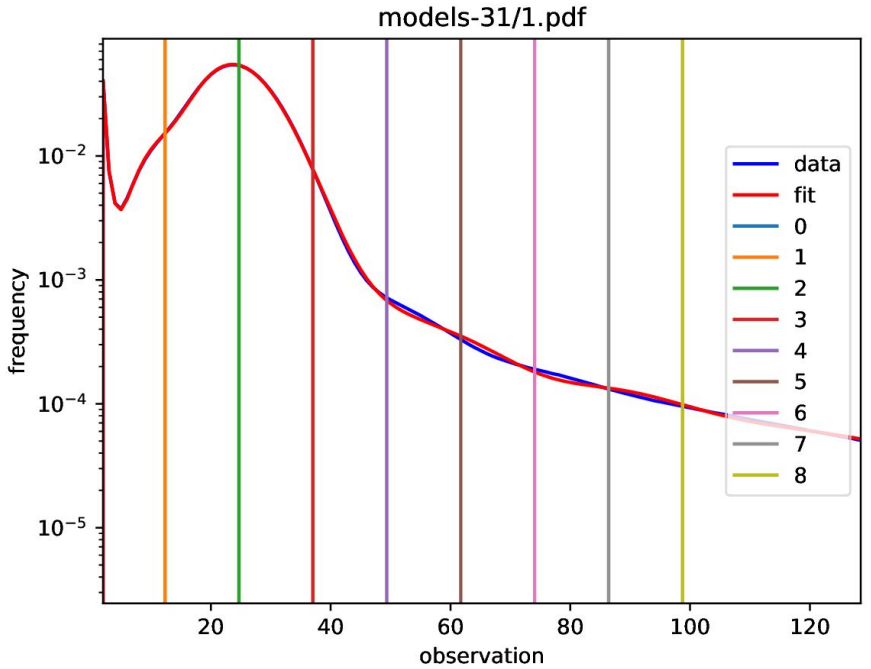
1. **Xengsort:** Xenograft sorting (already discussed)
   (https://gitlab.com/genomeinformatics/xengsort - 2020)
2. **BCOOL:** Sequencing error correction
   (https://github.com/Malfoy/BCOOL - 2019)
3. **Kraken 2:** metagenomic species identification and quantification
   (https://ccb.jhu.edu/software/kraken2/ - Sep 2019)
4. **Kallisto:** RNA-seq transcript quantification
   (https://pachterlab.github.io/kallisto/ - 2016);
   not for differential expression; use additional tools like sleuth
5. **DE-kupl:** discovery of novel (differentially expressed) transcripts
   (https://transipedia.github.io/dekupl/ - 2017)

BCOOL - sequencing error correction

Software: https://github.com/Malfoy/BCOOL
Papers: arXiv: https://arxiv.org/pdf/1711.03336.pdf;

Ideas:

- count number of occurrences of each k-mer in all reads
- build k-mer histogram
- k-mers occurring rarely are probably errors and must be corrected
- Reads are mapped to De Bruijn graph

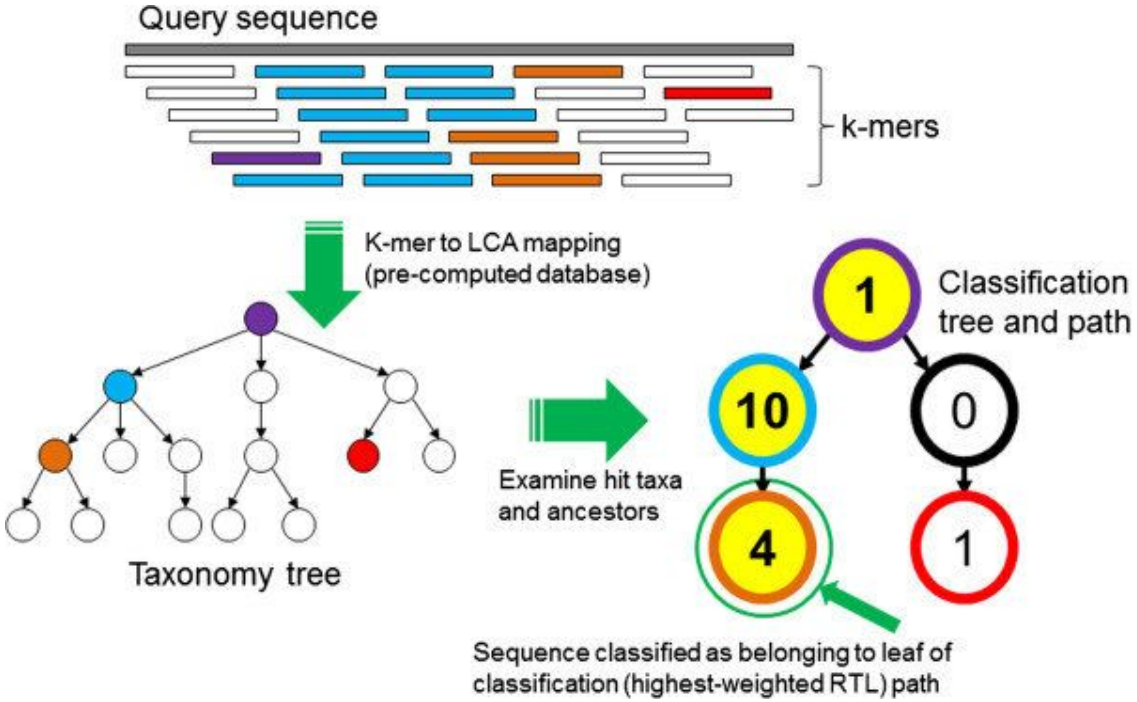


Kraken 2 - Metagenomic species identification

Software: https://ccb.jhu.edu/software/kraken2/
Preprint: https://www.biorxiv.org/content/10.1101/762302v1

Examines *k*-mers of a query read to find the most probable species of origin in a taxonomy tree.

Each *k*-mer is mapped to a tree node (lowest common ancestor, LCA) of all species containing the *k*-mer.



Kallisto: RNA-seq transcript quantification

Software: https://pachterlab.github.io/kallisto/
Paper: https://www.nature.com/articles/nbt.3519

Ideas:

- map each k-mer of a read to a set ("compatibility class") of transcripts (typically from .cdna.fasta files)
- take (soft) intersection of compatibility classes (perhaps do read error correction before mapping)
- run a decoding algorithm on reads that cannot be uniquely placed

DE-kupl: Discovery of new differential transcripts

Software: https://transipedia.github.io/dekupl/

Paper: https://genomebiology.biomedcentral.com/articles/10.1186/s13059-017-1372-2

Ideas:

- Count occurrence of each k-mer in RNA-seq datasets (from two classes)
- Remove k-mers from known transcripts
- Do test of differential expression on remaining ("novel") k-mers
- Locally assemble differential novel k-mers;
  yields novel differentially expressed transcripts (parts of them)

Next part:
Hashing, hash functions
collision resolution