UNIVERSITÄT
DES
SAARLANDES

ZBI ZENTRUM FÜR BIOINFORMATIK

Locality Sensitive Hashing

Algorithms for Sequence Analysis

Sven Rahmann
based on slides by Ali Ghaffaari

Summer 2021

Overview

Previous lectures

- Various **index structures for strings**:
  Suffix trees, suffix arrays, BWT, FM index, q-gram (k-mer) index
- Employ these indexes for exact and approximate search, **read mapping**

Today's lecture

- Finding exact and approximate matches by **hashing techniques**
- k-mers: encoding vs. hashing
- locality sensitive hashing (in general)
- min-hashing (on k-mers)

Part I: Remarks on the k-mer index

Implementations for a k-mer Index

Observation

The k-mer index is a multimap
which associated each k-mer to its occurrences in the text T.

Implementations for a k-mer Index

Observation

The k-mer index is a multimap
which associated each k-mer to its occurrences in the text T.

Implementations

- lexicographically sorted starting positions `pos` of all possible k-mers
  $+$ table `start` of k-mer bucket starting ranks in `pos`

Implementations for a k-mer Index

Observation

The k-mer index is a multimap
which associated each k-mer to its occurrences in the text T.

Implementations

- lexicographically sorted starting positions `pos` of all possible k-mers
  + table `start` of k-mer bucket starting ranks in `pos`
- positionally sorted starting positions of all possible k-mers
  + table `start`, as above
- **Notes:** Sorting order of `pos` within a k-mer bucket is irrelevant.
  Suffix array `pos` is useful for **every value of** k.

Implementations for a k-mer Index

Observation

The k-mer index is a multimap
which associated each k-mer to its occurrences in the text T.

Implementations

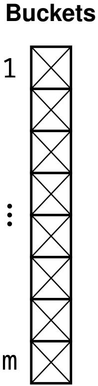
- lexicographically sorted starting positions `pos` of all possible k-mers
  $+$ table `start` of k-mer bucket starting ranks in `pos`
- positionally sorted starting positions of all possible k-mers
  $+$ table `start`, as above
- **Notes:** Sorting order of `pos` within a k-mer bucket is irrelevant.
  Suffix array `pos` is useful for **every value of** k.
- Today: Implementation as a hash table

Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

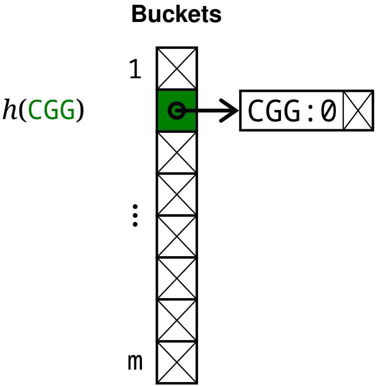


Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

```
0123456789
```

$T =$ `CGGCATCATG`

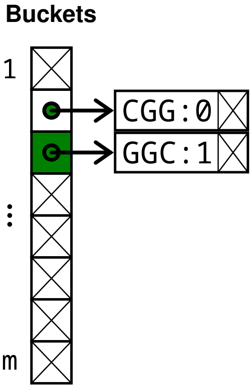


Indexing k-mers: Hash Map

Example: Building a 3-mer index

```
    0123456789
T = CGGCATCATG
```

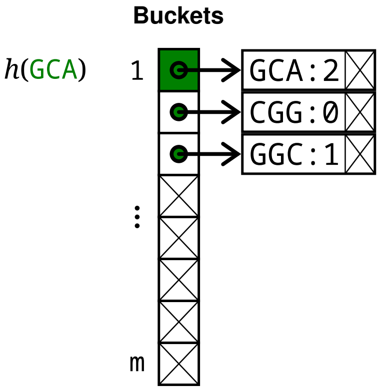
$h(\texttt{GGC})$



Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

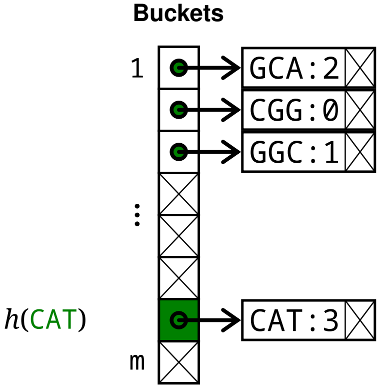
$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \end{array}$$



Indexing k-mers: Hash Map

Example: Building a 3-mer index

$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \end{array}$$



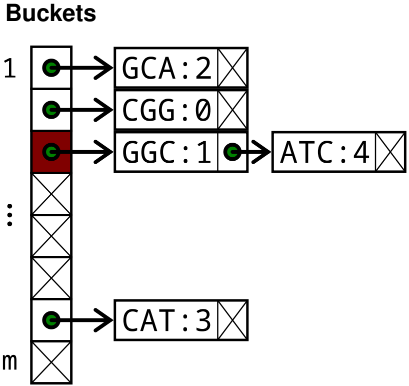
Indexing k-mers: Hash Map

Example: Building a 3-mer index

`0123456789`

$T =$ `CGGCATCATG`

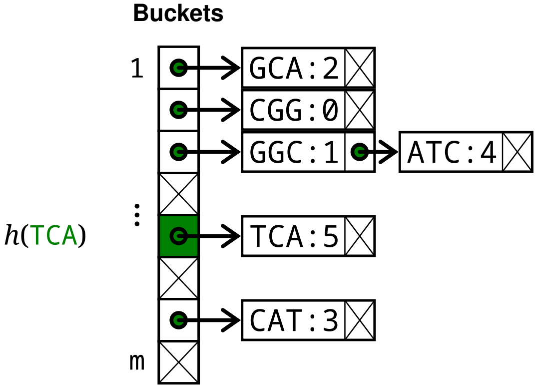
$h($`ATC`$)$



Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

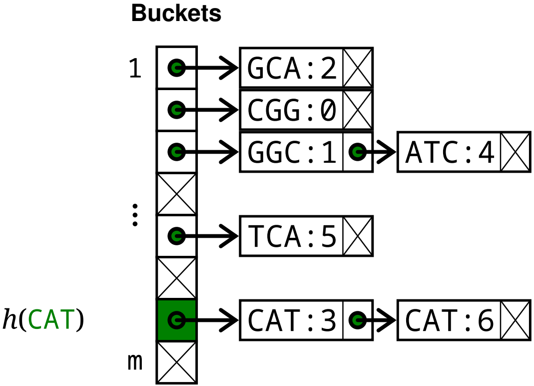
$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \end{array}$$



Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

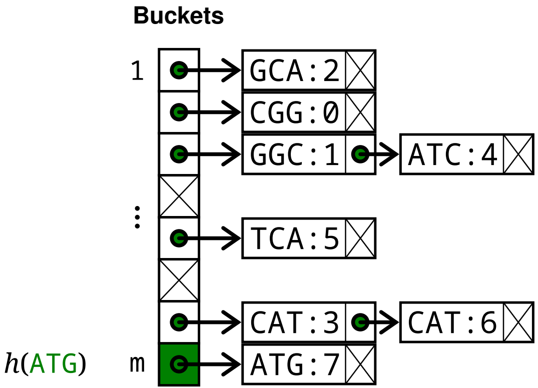
$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \end{array}$$



Indexing *k*-mers: Hash Map

Example: Building a 3-mer index

$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \end{array}$$



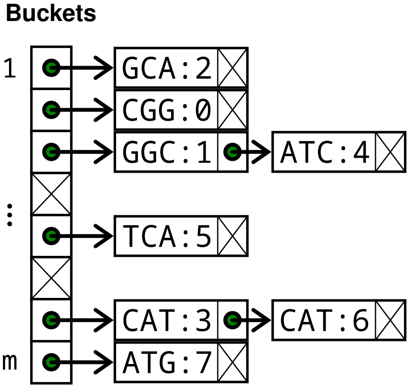
Querying a *k*-mer Index: Hash Map

Example: Querying an index

`0123456789`

$T =$ `CGGCATCATG`

P = ATC

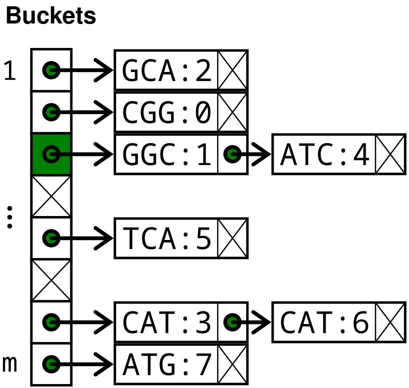


Querying a *k*-mer Index: Hash Map

Example: Querying an index

$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \\ \mathbf{P} = & \mathbf{ATC} \end{array}$$

$h(\texttt{ATC})$

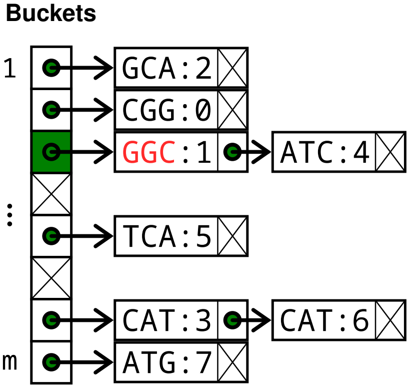


Querying a *k*-mer Index: Hash Map

Example: Querying an index

$$\begin{array}{rl} & \texttt{0123456789} \\ T = & \texttt{CGGCATCATG} \\ \mathbf{P} = & \mathbf{ATC} \end{array}$$

$h(\texttt{ATC})$

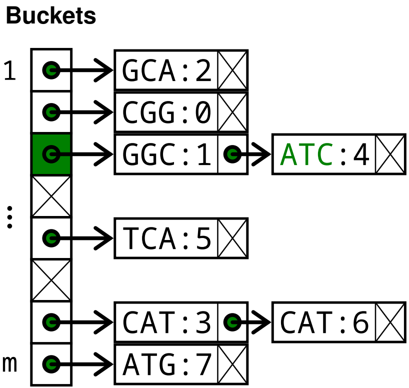


Querying a *k*-mer Index: Hash Map

Example: Querying an index

$$\begin{array}{r} \texttt{0123456789} \\ T = \texttt{CGGCATCATG} \\ \mathbf{P = ATC} \end{array}$$

$h(\texttt{ATC})$



Querying a *k*-mer Index: Hash Map

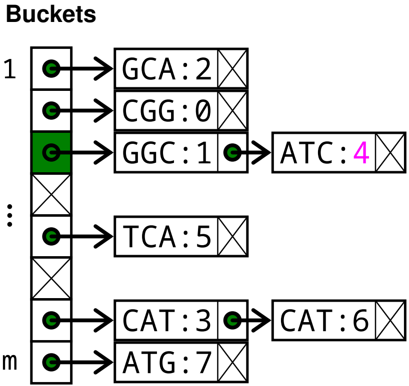
Example: Querying an index

`0123456789`

$T =$ `CGGCATCATG`

P = ATC

$h(\texttt{ATC})$



Encoding vs. Hashing

Encoding

- Assign a unique integer in $[0, 4^k[$ to each k-mer (e.g., base-4 encoding).
- Bijective map $\Sigma_{\mathrm{DNA}} \to \{0, \ldots, 4^k - 1\}$.
- Useful if $n = \Theta(4^k)$: Size of `pos`: n; size of `start`: 4^k.

Encoding vs. Hashing

Encoding

- Assign a unique integer in $[0, 4^k[$ to each k-mer (e.g., base-4 encoding).
- Bijective map $\Sigma_{\text{DNA}} \to \{0, \ldots, 4^k - 1\}$.
- Useful if $n = \Theta(4^k)$: Size of `pos`: n; size of `start`: 4^k.

Example: $f_a(x) := (a \cdot enc(x)) \bmod 4^k$, with a odd, enc the base-4 encoding

Encoding vs. Hashing

Encoding

- Assign a unique integer in $[0, 4^k[$ to each k-mer (e.g., base-4 encoding).
- Bijective map $\Sigma_{\text{DNA}} \to \{0, \dots, 4^k - 1\}$.
- Useful if $n = \Theta(4^k)$: Size of `pos`: n; size of `start`: 4^k.

Example: $f_a(x) := (a \cdot enc(x)) \bmod 4^k$, with a odd, enc the base-4 encoding

Hashing

- Any (non-bijective) function $\Sigma^k \to \{0, \dots, p - 1\}$ for integer p (**address space**).
- Useful if $n \approx p \ll 4^k$ (large k).

Encoding vs. Hashing

Encoding

- Assign a unique integer in $[0, 4^k[$ to each k-mer (e.g., base-4 encoding).
- Bijective map $\Sigma_{\text{DNA}} \to \{0, \ldots, 4^k - 1\}$.
- Useful if $n = \Theta(4^k)$: Size of `pos`: n; size of `start`: 4^k.

Example: $f_a(x) := (a \cdot enc(x)) \bmod 4^k$, with a odd, enc the base-4 encoding

Hashing

- Any (non-bijective) function $\Sigma^k \to \{0, \ldots, p - 1\}$ for integer p (**address space**).
- Useful if $n \approx p \ll 4^k$ (large k).
- **Disadvantages:**
  - storage of k-mers in hash buckets
  - **collisions**
  - below 100% load (empty buckets)

Part II: How to Hash

Collision Resolution Strategies

- Chaining (shown): use linked lists at each address
- Open addressing with linear probing:
  relocate colliding keys to following addresses
- Open addressing with non-linear (quadratic) probing:
  relocate colliding keys to other addresses, non-linearly
- Double hashing: relocate colliding keys by linear probing
  with different step sizes for each key
- Cuckoo hashing: use two hash functions, move keys around
- Robin Hood hashing: use linear probing,
  move keys around such that each key stays close to its hash address

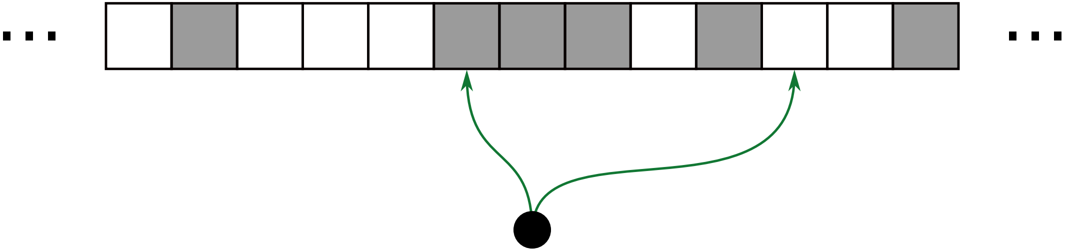
Collision Resolution Strategies

- Chaining (shown): use linked lists at each address
- Open addressing with linear probing: relocate colliding keys to following addresses
- Open addressing with non-linear (quadratic) probing: relocate colliding keys to other addresses, non-linearly
- Double hashing: relocate colliding keys by linear probing with different step sizes for each key
- Cuckoo hashing: use two hash functions, move keys around
- Robin Hood hashing: use linear probing, move keys around such that each key stays close to its hash address

Requirements for Genome Analysis

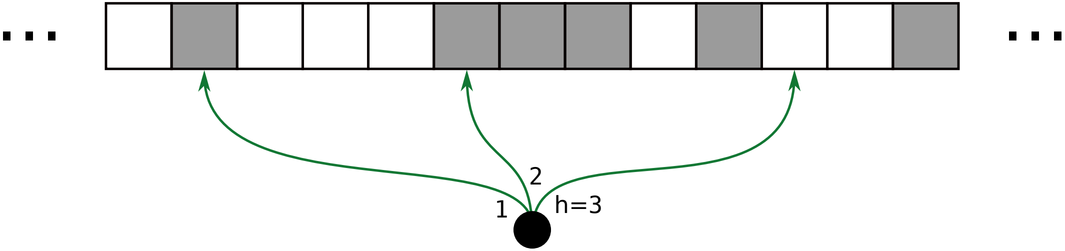
- Very fast lookup, i.e., very few random memory accesses.
- Small size, i.e., high load factor, almost no empty space

(h, b) Cuckoo hashing



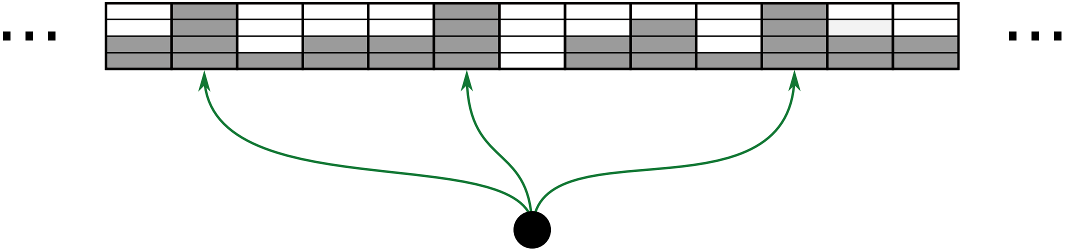
(h, b) Cuckoo hashing

- Use h hash functions



(h, b) Cuckoo hashing

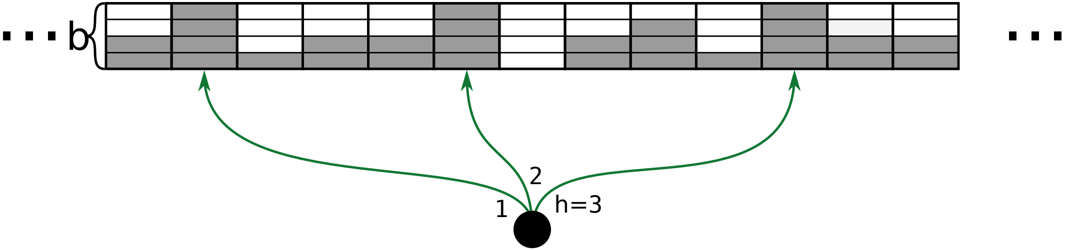
- Use h hash functions
- Store up to b elements at each position



(*h*, *b*) Cuckoo hashing

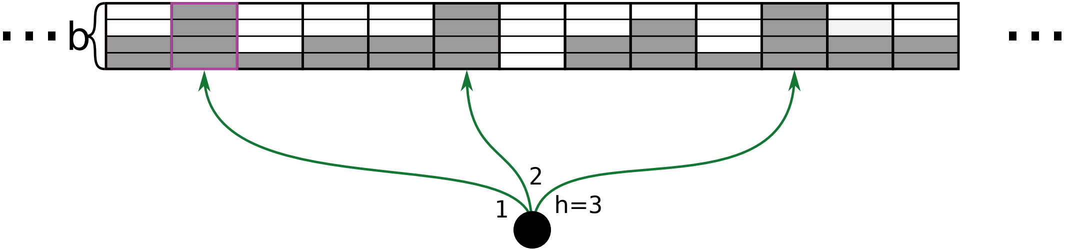
- Use *h* hash functions
- Store up to *b* elements at each position

`load factor =` $$\frac{\#\blacksquare}{\#\square + \#\blacksquare}$$



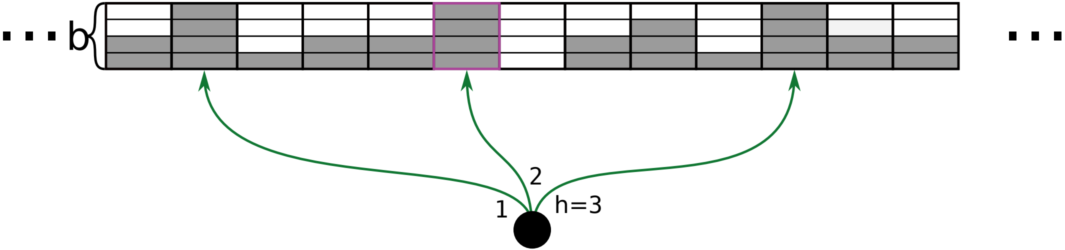
(h, b) Cuckoo Hashing: Insertion by Random Walk

- Use h hash functions
- Store up to b elements at each position



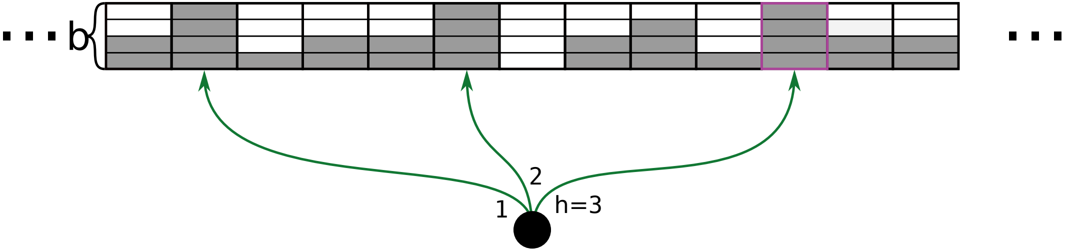
(h, b) Cuckoo Hashing: Insertion by Random Walk

- Use h hash functions
- Store up to b elements at each position



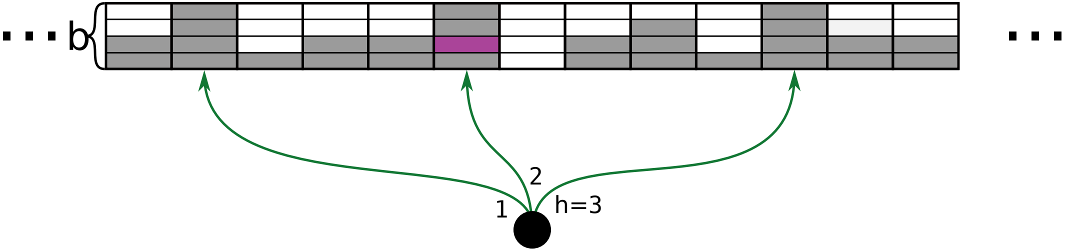
(h, b) Cuckoo Hashing: Insertion by Random Walk

- Use h hash functions
- Store up to b elements at each position



(h, b) Cuckoo Hashing: Insertion by Random Walk

- Use h hash functions
- Store up to b elements at each position



Properties of (h, b) Cuckoo Hashing

- Small number of hash functions: $h = 2, 3, 4$
- Number h limits random memory accesses (cache misses) per lookup
- Higher h allow higher loads with same bucket size b
- Bucket size b limits number of comparisons per bucket (fast anyway)
- Higher bucket size allows higher loads
- Use $(2, 6)$ or $(3, 4)$ in practice
- Low loads: Many keys found at first choice (fast)
- High loads: Frequently have to check 2nd/3rd choice (slower), but less wasted space
- Good load factors in practice: 0.85 to 0.95

Part III: Similarities and Distances

Similarity vs. Distance

Relationship

Similarity measures are usually the inverse of distance metrics and vice versa.

Similarity vs. Distance

Relationship

Similarity measures are usually the inverse of distance metrics and vice versa.

Hamming distance

$P_1 = \texttt{ABCDE}$
$P_2 = \texttt{ABDDE}$
$d(P_1, P_2) = 1$
$\hat{d}(P_1, P_2) = \frac{d(P_1,P_2)}{\ell} = \frac{1}{5}$,
with $\ell = \|P_1\| = \|P_2\|$

Similarity vs. Distance

Relationship

Similarity measures are usually the inverse of distance metrics and vice versa.

Hamming distance

$P_1 = \texttt{ABCDE}$

$P_2 = \texttt{ABDDE}$

$d(P_1, P_2) = 1$

$\hat{d}(P_1, P_2) = \frac{d(P_1,P_2)}{\ell} = \frac{1}{5}$,

with $\ell = \|P_1\| = \|P_2\|$

Hamming similarity

$P_1 = \texttt{ABCDE}$

$P_2 = \texttt{ABDDE}$

$S(P_1, P_2) = 4$

$\hat{S}(P_1, P_2) = \frac{S(P_1,P_2)}{\ell} = \frac{4}{5}$,

with $\ell = \|P_1\| = \|P_2\|$

Similarity vs. Distance

Relationship

Similarity measures are usually the inverse of distance metrics and vice versa.

Hamming distance

$P_1 = \mathtt{ABCDE}$
$P_2 = \mathtt{ABDDE}$
$d(P_1, P_2) = 1$
$\hat{d}(P_1, P_2) = \frac{d(P_1,P_2)}{\ell} = \frac{1}{5}$,
with $\ell = \|P_1\| = \|P_2\|$

Hamming similarity

$P_1 = \mathtt{ABCDE}$
$P_2 = \mathtt{ABDDE}$
$S(P_1, P_2) = 4$
$\hat{S}(P_1, P_2) = \frac{S(P_1,P_2)}{\ell} = \frac{4}{5}$,
with $\ell = \|P_1\| = \|P_2\|$

$$\hat{S}(P_1, P_2) = 1 - \hat{d}(P_1, P_2)$$

Distance Measures: ℓ_p Distances

Definition

In an ***n*-dimensional real vector space**, points are vectors of n real numbers. For any constant $p \geq 1$, we define the **ℓ_p distance** by

$$d_p([x_1, \cdots, x_n], [y_1, \cdots, y_n]) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

$$d_2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

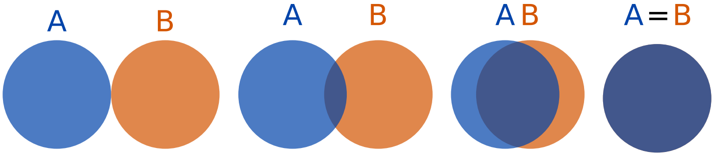
$$d_\infty(x, y) = \max_{i=1,\ldots,n} |x_i - y_i|$$

Distance Measure: Jaccard Similarity

Definition

Given two sets A and B, the **Jaccard index** or **Jaccard similarity** is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$



$$0 \leq J(A,B) \leq 1$$

Distance Measure: Jaccard Similarity

Definition

Given two sets A and B, the **Jaccard index** or **Jaccard similarity** is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \,.$$

Jaccard Distance

We define the **Jaccard distance** as

$$d_J(A, B) := 1 - J(A, B) \,,$$

which is a metric.

Reminder: Hamming Distance

Setting

Compare two strings $s, t \in \Sigma^n$ of same length.

Definition

Given $s, t \in \Sigma^n$, we define the **Hamming distance** $d_H(s,t)$ as the number of positions where s and t differ. In other words,

$$d_H(s,t) := \left|\left\{i \in \{0, \ldots, n-1\} : s[i] \neq t[i]\right\}\right|$$

Example

$$\begin{matrix} s = \\ t = \end{matrix}\ \begin{matrix} \texttt{C} \\ \texttt{C} \end{matrix}\ \boxed{\begin{matrix} \texttt{T} \\ \texttt{A} \end{matrix}}\ \begin{matrix} \texttt{G} \\ \texttt{G} \end{matrix}\ \begin{matrix} \texttt{T} \\ \texttt{T} \end{matrix}\ \boxed{\begin{matrix} \texttt{A} \\ \texttt{C} \end{matrix}}\ \begin{matrix} \texttt{A} \\ \texttt{A} \end{matrix}\ \begin{matrix} \texttt{T} \\ \texttt{T} \end{matrix}\ \begin{matrix} \texttt{A} \\ \texttt{A} \end{matrix}\ \begin{matrix} \texttt{C} \\ \texttt{C} \end{matrix}$$

$\Rightarrow$ Hamming distance 2

Reminder: Edit Distance

Edit Operations

- Replacement:

  C T G T A A T A C

  C A G T A A T A C

- Insertion:

  C T G T A A T A C

  C T G T C A A T A C

- Deletion:

  C T G T A A T A C

  C T G T A T A C

Definition: edit distance

The **edit distance** of s and t is defined as the **minimum** number of **edit operations** needed to turn s into t.

Problem: Finding Similar Items

Setting

Suppose $\mathcal{U}$ is a universe (set) of objects and d is a metric on $\mathcal{U}$.

Problem

Given a set $A \subseteq \mathcal{U}$, an item x, and $\epsilon > 0$.
Find all items similar to x in A, i.e., all items $e \in A$ such that $d(e, x) < \epsilon$.

Problem: Finding Similar Items

Setting

Suppose $\mathcal{U}$ is a universe (set) of objects and d is a metric on $\mathcal{U}$.

Problem

Given a set $A \subseteq \mathcal{U}$, an item x, and $\epsilon > 0$.
Find all items similar to x in A, i.e., all items $e \in A$ such that $d(e, x) < \epsilon$.

Naïve approach

```
def find_similar(A, x, eps):
    for element in A:
        if d(element, x) < eps:
            yield element
```

Time complexity: $O(|A| \cdot \dim \mathcal{U})$

Part IV: Locality Sensitive Hashing

Hashing

Observation

Conventional hash functions are designed to generate scattered hash values even for **similar** (not identical) items.

Argument

It is necessary:

- Avoid **collisions** as much as possible,
- Preserve **constant time** lookup operation in exact membership query.

Sought

Find a hashing technique to give two similar items an identical hash value.

Hashing

Observation

Conventional hash functions are designed to
generate scattered hash values even for **similar** (not identical) items.

Argument

It is necessary:

- Avoid **collisions** as much as possible,
- Preserve **constant time** lookup operation in exact membership query.

Sought

Find a hashing technique to give two similar items an identical hash value.
Obvious transitivity problem!

Locality Sensitive Hashing

Idea

- Design hash functions that tend to assign **identical hash values** for **similar items** *a* and *b* with high probability.
- **Collision:** items **may** be similar.
- Distinct hash values: items may also be similar ?!

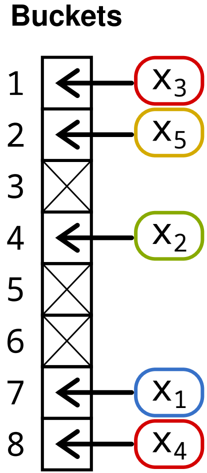
Preprocessing

- Compute hash value for each element in the set.
- Put element in the corresponding bucket.

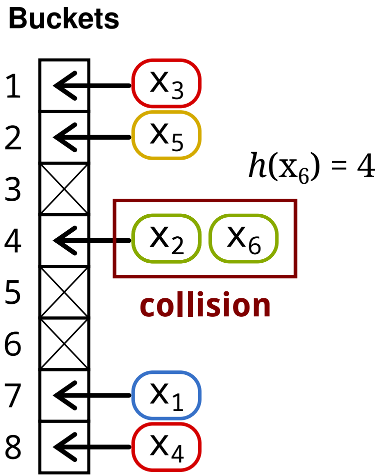
Querying

- Compute the hash value for query item.
- Compare the query item only with items in its corresponding bucket.

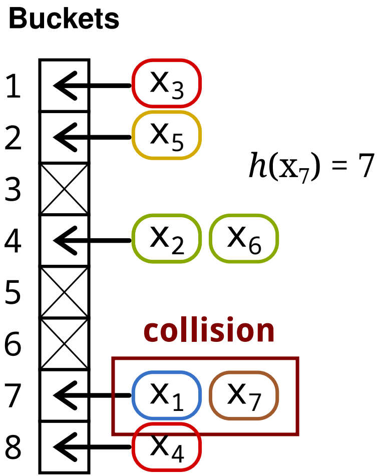
Locality Sensitive Hashing



Locality Sensitive Hashing



Locality Sensitive Hashing



$h(x_7) = 7$

Locality Sensitive Hashing

Definition: Locality Sensitive Hashing

Let $\mathcal{S}$ be a similarity measure on space or universe $\mathcal{U}$.
A set $\mathcal{H}$ of hash functions is **locality sensitive** for $\mathcal{S}$ if

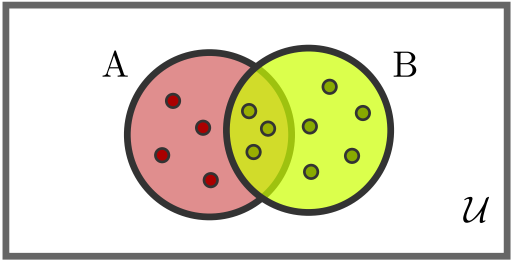
$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \mathcal{S}(x, y) \quad \text{for all } x, y\,,$$

where the probability is taken over the distribution of hash functions.

Example: Hamming similarity

Consider the set of hash functions $\mathcal{H} = \{P_i | i \in 1..n\}$, where $P_i(s_1 s_2 \ldots s_n) := s_i$.
Then $\Pr[h(x) = h(y)] = \mathcal{S}_{\text{Hamming}}(x, y)$.
Therefore $\mathcal{H}$ is a LS set of hash functions for $\mathcal{S}_{\text{Hamming}}$.

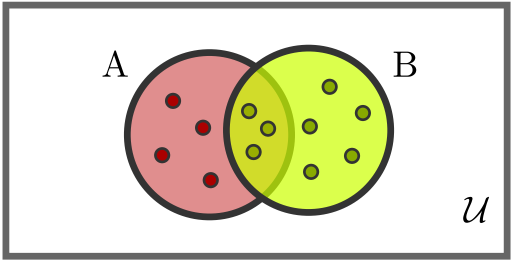
LSH for Jaccard Similarity



$$\mathcal{S}_J = \frac{|A \cap B|}{|A \cup B|}$$

$$\mathcal{S}_J(A, B) = \frac{3}{12} = 0.25$$

LSH for Jaccard Similarity



$$\mathcal{S}_J = \frac{|A \cap B|}{|A \cup B|}$$

$$\mathcal{S}_J(A, B) = \frac{3}{12} = 0.25$$

Idea

A bijective function $\pi : \mathcal{U} \to [0, |\mathcal{U}|[$ is a ranking (ordering) function of $\mathcal{U}$.
The family $\mathcal{H}$ of hash functions

$$h_\pi(A) := \min_{x \in A} \pi(x)\,,$$

where π ranges over **all orderings** of $\mathcal{U}$, is **locality sensitive** for $\mathcal{S}_J$.

Observations on Locality Sensitive Hashing

Observation I: Two different elements can collide

- The same hash values can be assigned to very different elements because of accidental collision.

Observations on Locality Sensitive Hashing

Observation I: Two different elements can collide

- The same hash values can be assigned to very different elements because of accidental collision.

Observation II: Two similar elements can be missed

- Two similar elements can have different hash values.

Observations on Locality Sensitive Hashing

Observation I: Two different elements can collide

- The same hash values can be assigned to very different elements because of accidental collision.

Observation II: Two similar elements can be missed

- Two similar elements can have different hash values.

Idea (amplification)

Use **multiple hash functions**,
hoping that all similar elements get identical value for at least one hash function.

Sketches

Definition

For an element x, a **sketch** for the LSH $\mathcal{H}$ is a vector $[h_1(x), h_2(x), \ldots, h_r(x)]$, where hash functions h_i are selected from $\mathcal{H}$ according to a probability distribution.

Two benefits of sketches

1. Increased chance of finding a similar item
   when searching with more hash functions
2. Estimation of similarity: $|\{i \mid h_i(A) = h_i(B)\}|/r$ is an estimate of $S(A, B)$:
   Using only one hash function gives a **high-variance** estimator.
   Using more hash functions gives higher precision.

Error Rates with Sketches (Several Hash Functions)

- False negative errors decrease exponentially with r
- False positive errors increase slowly linearly with r

Summary

Hashing

- Alternative to classical k-mer index for large k
- Requires a collision resolution strategy
- Good in practice: (h, b) Cuckoo hashing: several hash functions, buckets of size b

Summary

Hashing

- Alternative to classical k-mer index for large k
- Requires a collision resolution strategy
- Good in practice: (h, b) Cuckoo hashing: several hash functions, buckets of size b

Locality Sensitive Hashing

- Different similarity measures
- Probability that hash values of x, y agree $=$ similarity of x, y
- sets of k-mers: Jaccard similarity
- min hashing
- amplification using several hash functions; sketches

Possible exam questions

- How can a k-mer index be implemented?
- What is the disadvantage of hash-based vs. encoding-based implementations?
- How can k-mers be mapped bijectively to the integers $0, \dots, 4^k - 1$?
- What are some common collision resolution strategies when hashing?
- Explain (h, b) Cuckoo hashing
- Why are the advantages and disadvantages of (h, b) Cuckoo hashing?
- When is a set of hash functions "locality sensitive"?
- Why is standard hashing usually not locality sensitive?
- Explain min-hashing.
- Why is min-hashing locality sensitive for the Jaccard similarity?