

COMP9313: Big Data Management



Lecturer: Xin Cao

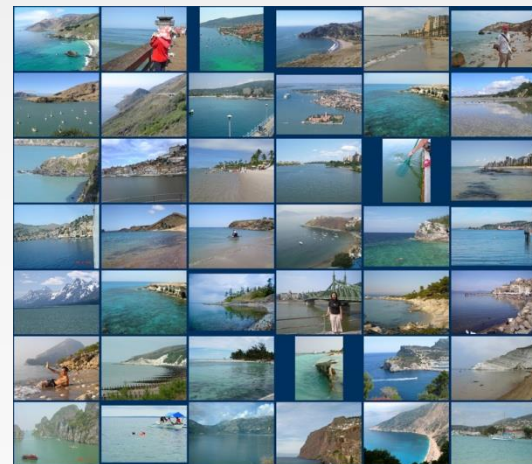
Course web site: <http://www.cse.unsw.edu.au/~cs9313/>

Chapter 8.1: Finding Similar Items

A Common Metaphor

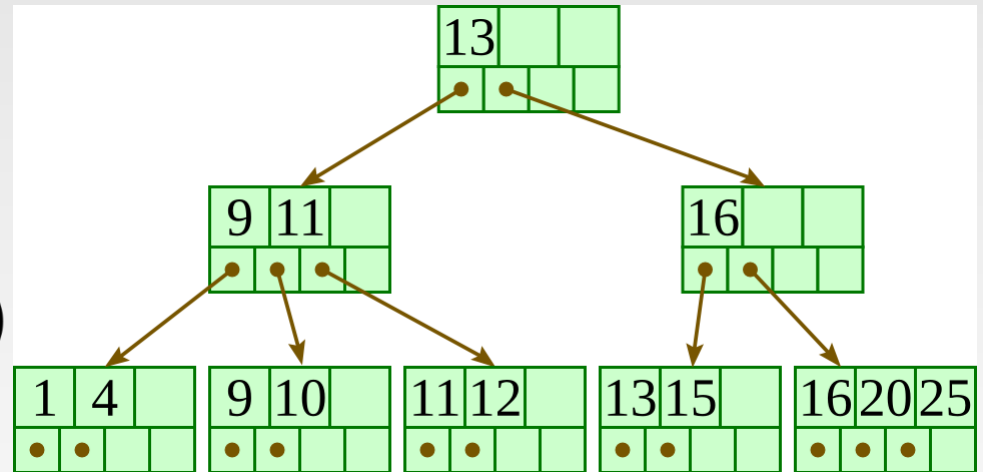
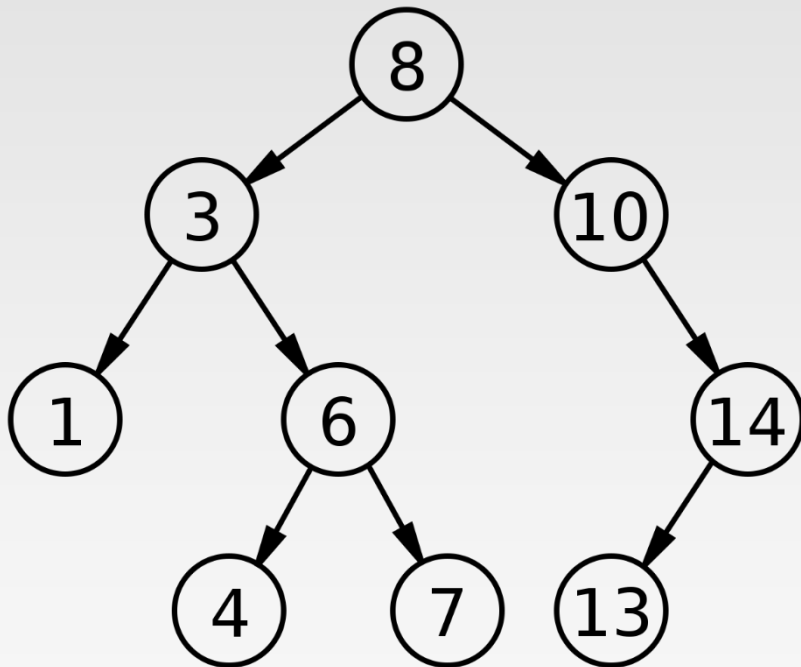
- ❖ Many problems can be expressed as finding “similar” sets:
 - Find near-neighbors in high-dimensional space
- ❖ Examples:
 - Pages with similar words
 - ▶ For duplicate detection, classification by topic
 - Customers who purchased similar products
 - ▶ Products with similar customer sets
 - Images with similar features
 - ▶ Google image search

Images with Similar Features



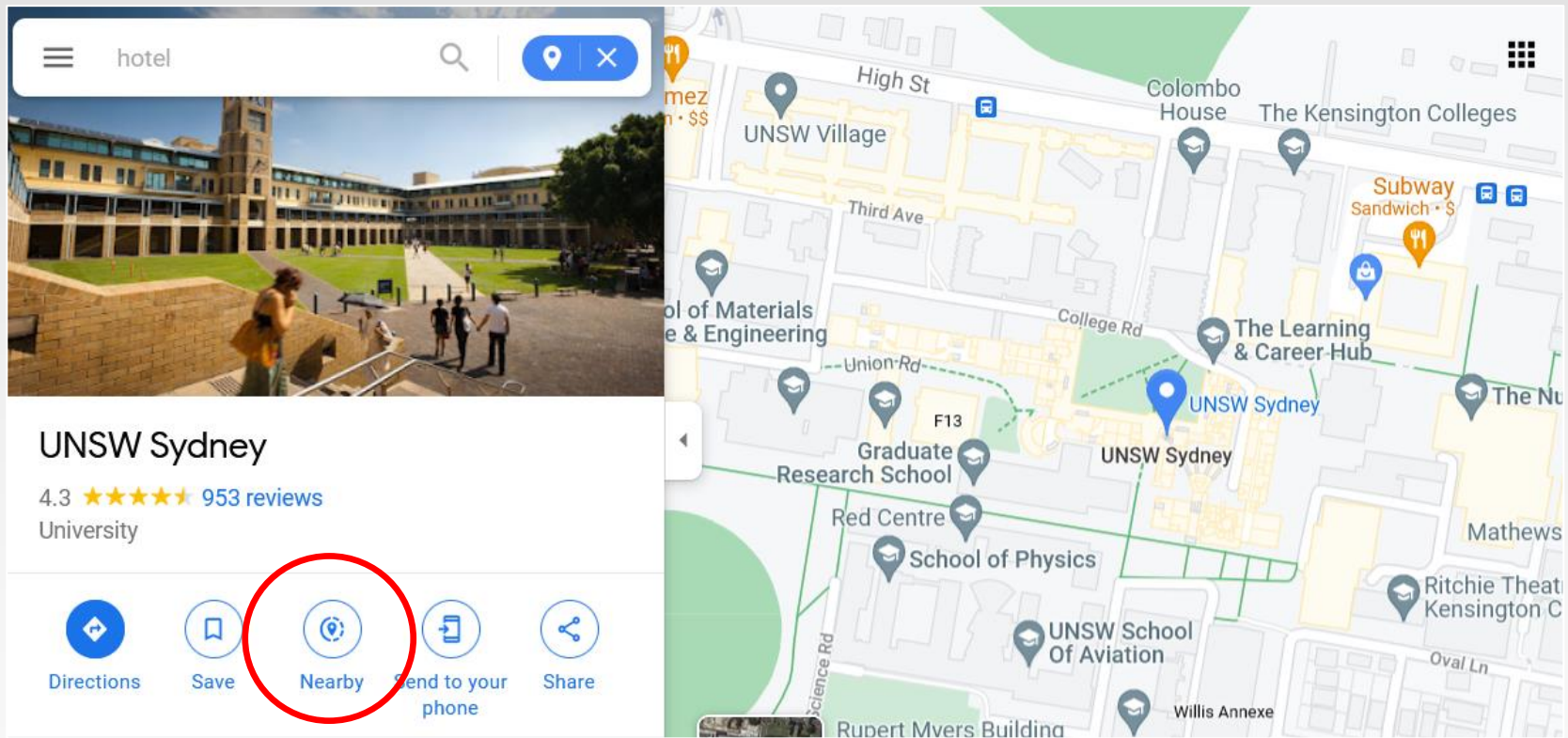
Similarity Search in One Dimensional Space

- ❖ Just numbers, use binary search, binary search tree, B+-Tree...
- ❖ The essential idea behind: objects can be sorted



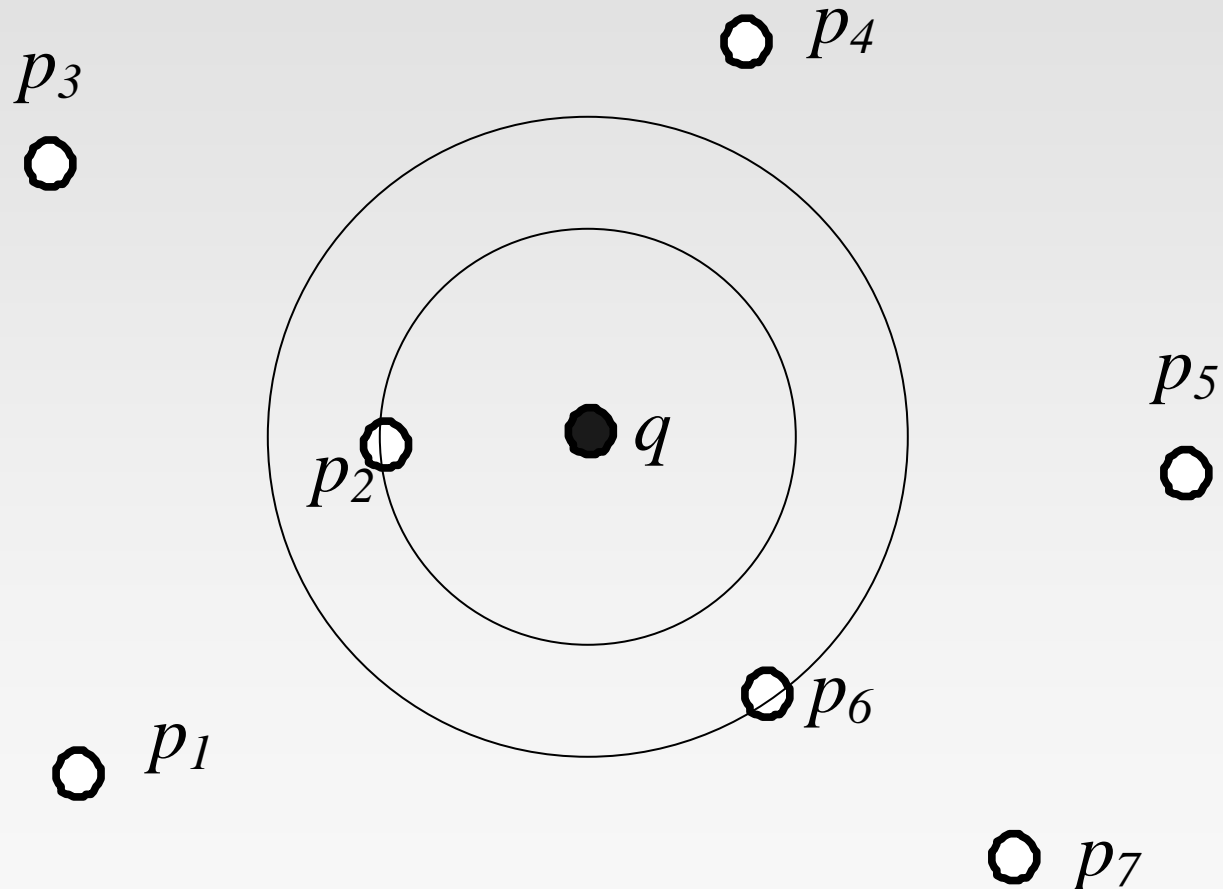
Similarity Search in 2D Space

- ❖ k nearest neighbour (k NN) query: find the top- k nearest spatial object to the query location
- ❖ E.g., find the top-5 closest restaurants to UNSW



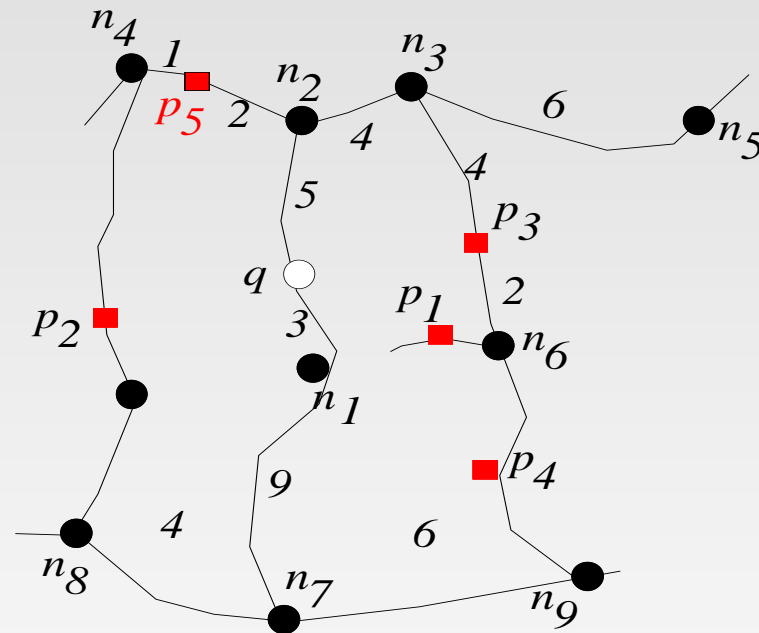
Similarity Search in 2D Space

❖ In Euclidean Space



Similarity Search in 2D Space

- ❖ In road networks: Distance is computed based on the network distance (such as the length of the shortest path)

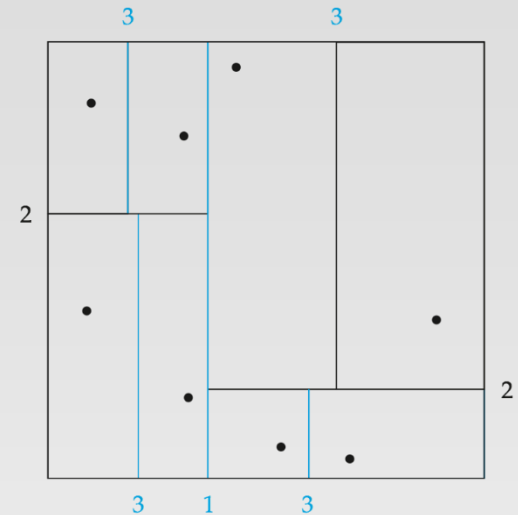


p_5 is the closest in the spatial network setting
 p_1 is the closest in the Euclidean space

The Problem in 2D Space

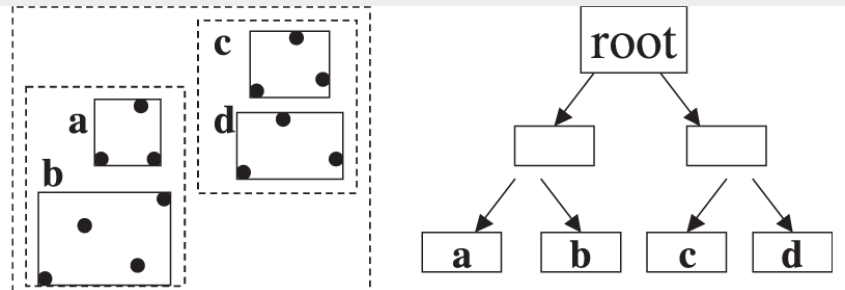
❖ Euclidean space

- Grid index
- Quad-tree
- *k-d* tree
- R-tree (R+-tree, R*-tree, etc.)
- m-tree, x-tree,
- Space filing curves: Z-order, Hilbert order,



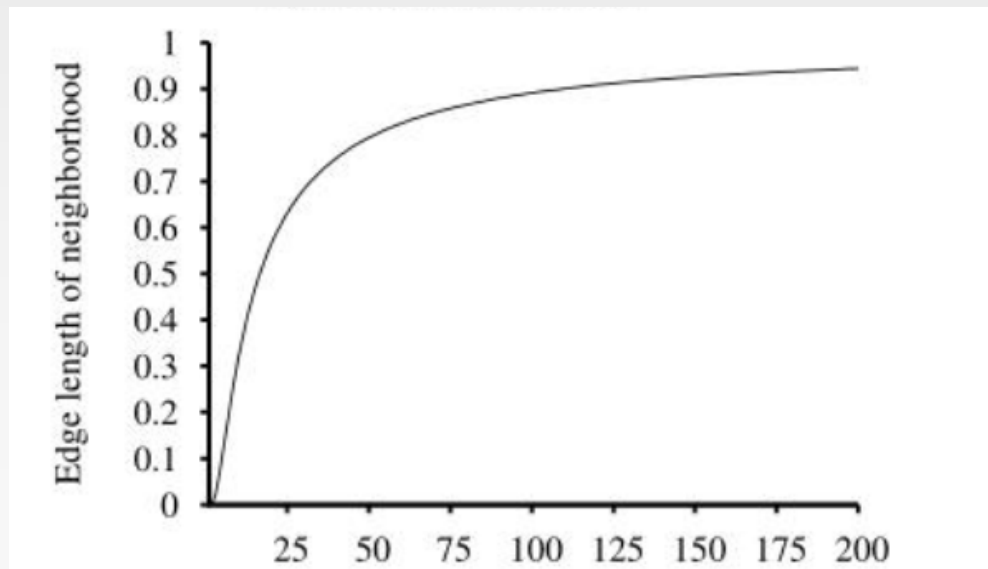
❖ Road Networks

- G-tree
- Contraction Hierarchy
- 2-hop labeling
-



Curse of Dimensionality

- ❖ Refers to various phenomena that arise in high dimensional spaces that do not occur in low dimensional settings.
- ❖ Specifically, refers to the decrease in performance of similarity search query processing when the dimensionality increases.
- ❖ In high dimensional space, almost all points are far away from each other.
 - To find the top-10 nearest neighbors, what is the length of the average neighborhood cube?



Problem

❖ **Given:** High dimensional data points x_1, x_2, \dots

➤ **For example:** Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1 \ 2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 1 \ 0]$$

❖ **And some distance function** $d(x_1, x_2)$

➤ Which quantifies the “distance” between x_1 and x_2

❖ **Goal:** Find **all pairs of data points** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$

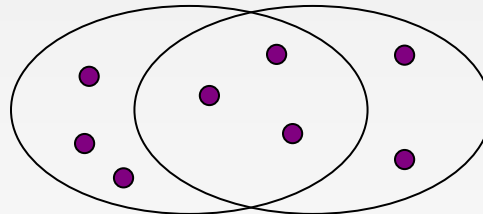
❖ **Note:** Naïve solution would take $O(N^2)$ ☹

where N is the number of data points

❖ **MAGIC:** This can be done in $O(N)!!$
How?

Distance Measures

- ❖ **Goal:** Find near-neighbors in high-dim. space
 - We formally define “near neighbors” as points that are a “small distance” apart
- ❖ For each application, we first need to define what “**distance**” means
- ❖ **Today: Jaccard distance/similarity**
 - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:
$$\text{sim}(\mathbf{C}_1, \mathbf{C}_2) = |\mathbf{C}_1 \cap \mathbf{C}_2| / |\mathbf{C}_1 \cup \mathbf{C}_2|$$
 - **Jaccard distance:** $d(\mathbf{C}_1, \mathbf{C}_2) = 1 - |\mathbf{C}_1 \cap \mathbf{C}_2| / |\mathbf{C}_1 \cup \mathbf{C}_2|$



3 in intersection
8 in union
Jaccard similarity = 3/8
Jaccard distance = 5/8

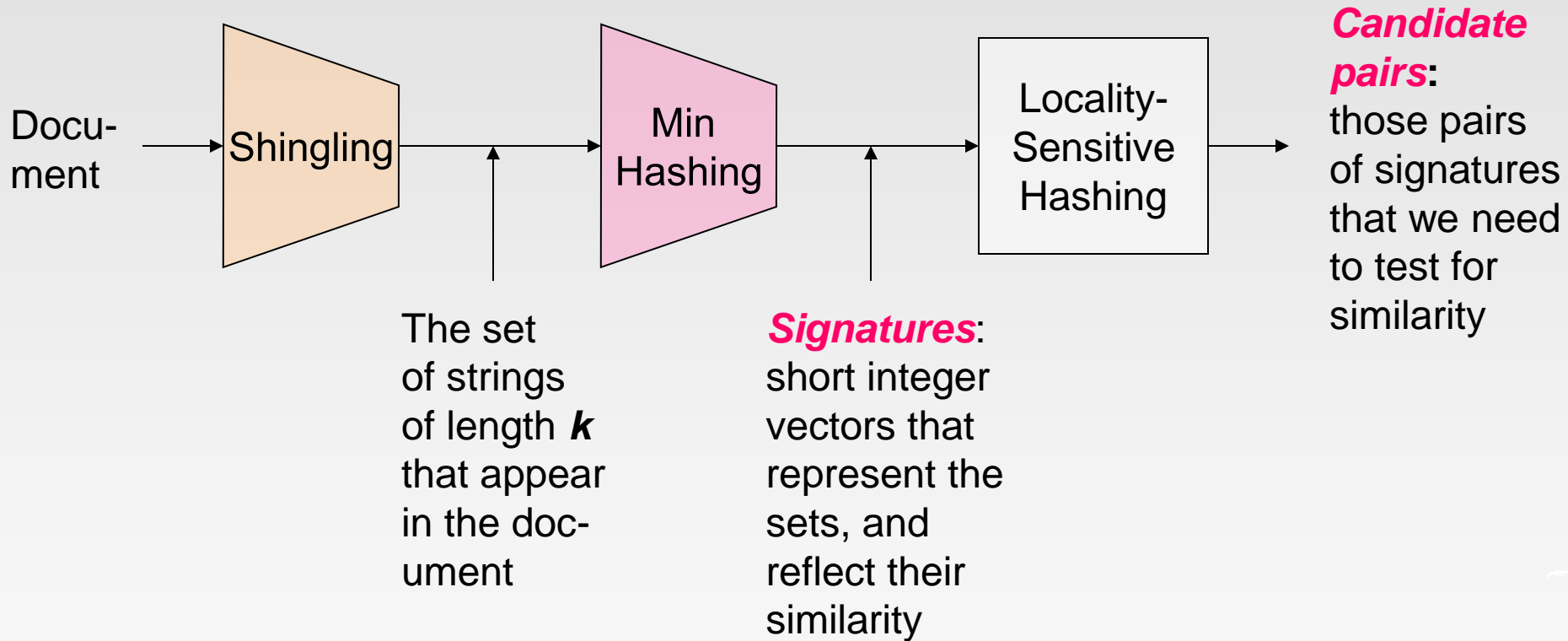
Task: Finding Similar Documents

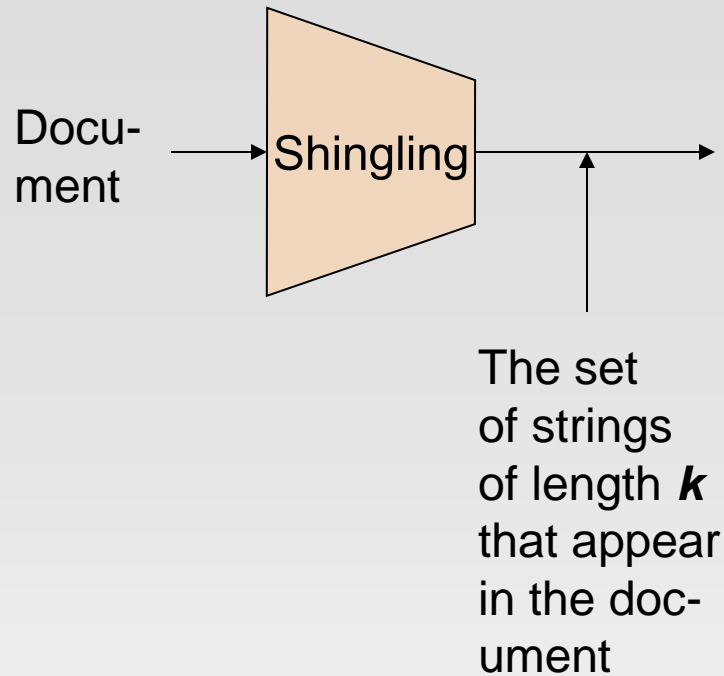
- ❖ **Goal:** Given a large number (N in the millions or billions) of documents, find “near duplicate” pairs.
- ❖ **Applications:**
 - Mirror websites, or approximate mirrors
 - ▶ Don't want to show both in search results
 - Similar news articles at many news sites
 - ▶ Cluster articles by “same story”
- ❖ **Problems:**
 - Many small pieces of one document can appear out of order in another
 - Too many documents to compare all pairs
 - Documents are so large or so many that they cannot fit in main memory

3 Essential Steps for Similar Docs

1. **Shingling:** Convert documents to sets
2. **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
3. **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
 - **Candidate pairs!**

The Big Picture





Step 1: *Shingling*: Convert documents to sets

Documents as High-Dim Data

- ❖ Step 1: **Shingling**: Convert documents to sets
- ❖ Simple approaches:
 - Document = set of words appearing in document
 - Document = set of “important” words
 - Don’t work well for this application. Why?
- ❖ Need to account for ordering of words!
- ❖ A different way: **Shingles**!

Define: Shingles

- ❖ A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
 - Tokens can be *characters*, *words* or something else, depending on the application
 - Assume tokens = characters for examples
- ❖ **Example:** $k=2$; document $D_1 = \text{abcab}$
Set of 2-shingles: $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$
 - **Option:** Shingles as a bag (multiset), count ab twice: $S'(D_1) = \{\text{ab}, \text{bc}, \text{ca}, \text{ab}\}$

Shingles and Similarity

- ❖ Documents that are intuitively similar will have many shingles in common.
- ❖ Changing a word only affects k -shingles within distance $k-1$ from the word.
- ❖ Reordering paragraphs only affects the $2k$ shingles that cross paragraph boundaries.
- ❖ **Example:** $k=3$, “The dog which chased the cat” versus “The dog that chased the cat”.
 - Only 3-shingles replaced are g_w , $_wh$, whi , hic , ich , $ch_$, and h_c .

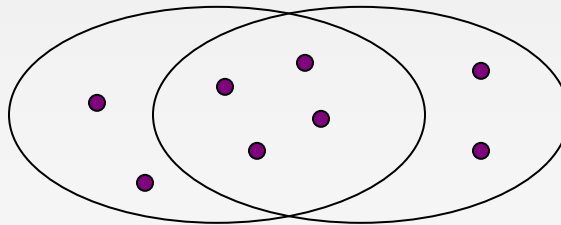
Compressing Shingles

- ❖ To **compress long shingles**, we can **hash** them to (say) 4 bytes
- ❖ **Represent a document by the set of hash values of its k -shingles**
 - **Idea:** Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared
- ❖ **Example:** $k=2$; document $D_1 = \text{abcaab}$
Set of 2-shingles: $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$
Hash the shingles: $h(D_1) = \{1, 5, 7\}$

Similarity Metric for Shingles

- ❖ Document D_1 is a set of its k -shingles $C_1 = S(D_1)$
- ❖ Equivalently, each document is a 0/1 vector in the space of k -shingles
 - Each unique shingle is a dimension
 - Vectors are very sparse
- ❖ A natural similarity measure is the **Jaccard similarity**:

$$\text{sim}(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

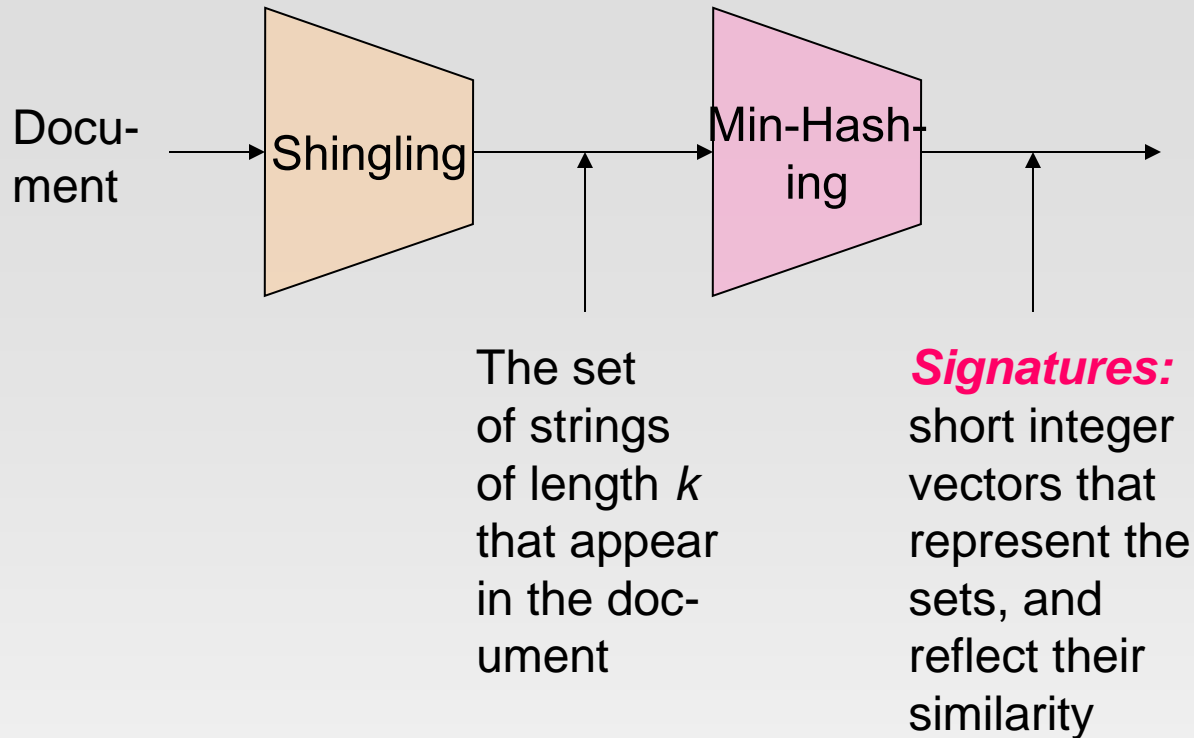


Working Assumption

- ❖ Documents that have lots of shingles in common have similar text, even if the text appears in different order
- ❖ If we pick k too small, then we would expect most sequences of k characters to appear in most documents
 - We could have documents whose shingle-sets had high Jaccard similarity, yet the documents had none of the same sentences or even phrases
 - Extreme case: when we use $k = 1$, almost all Web pages will have high similarity.
- ❖ **Caveat:** You must pick k large enough, or most documents will have most shingles
 - $k = 5$ is OK for short documents
 - $k = 10$ is better for long documents

Motivation for Minhash/LSH

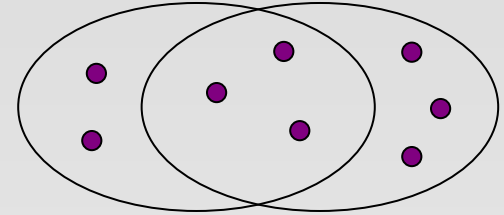
- ❖ Suppose we need to find near-duplicate documents among $N = 1$ million documents
- ❖ Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
 - $N(N - 1)/2 \approx 5 \cdot 10^{11}$ comparisons
 - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- ❖ For $N = 10$ million, it takes more than a year...



Step 2: *Minhashing*: Convert large sets to short signatures, while preserving similarity

Encoding Sets as Bit Vectors

- ❖ Many similarity problems can be formalized as **finding subsets that have significant intersection**



- ❖ **Encode sets using 0/1 (bit, boolean) vectors**
 - One dimension per element in the universal set
- ❖ Interpret **set intersection as bitwise AND**, and **set union as bitwise OR**
- ❖ **Example:** $C_1 = 10111$; $C_2 = 10011$
 - Size of intersection = 3; size of union = 4,
 - **Jaccard similarity** (not distance) = $3/4$
 - **Distance:** $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 1/4$

From Sets to Boolean Matrices

- ❖ **Rows** = elements (shingles)
- ❖ **Columns** = sets (documents)
 - 1 in row **e** and column **s** if and only if **e** is a member of **s**
 - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
 - **Typical matrix is sparse!**
- ❖ **Each document is a column:**

	Documents			
Shingles	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

From Sets to Boolean Matrices

❖ **Example:** $S_1 = \{a, d\}$, $S_2 = \{c\}$, $S_3 = \{b, d, e\}$, and $S_4 = \{a, c, d\}$

<i>Element</i>	S_1	S_2	S_3	S_4
<i>a</i>	1	0	0	1
<i>b</i>	0	0	1	0
<i>c</i>	0	1	0	1
<i>d</i>	1	0	1	1
<i>e</i>	0	0	1	0

➤ **$\text{sim}(S_1, S_3) = ?$**

- ▶ Size of intersection = 1; size of union = 4,
Jaccard similarity (not distance) = $1/4$
- ▶ **$d(S_1, S_3) = 1 - (\text{Jaccard similarity}) = 3/4$**

Outline: Finding Similar Columns

❖ So far:

- Documents → Sets of shingles
- Represent sets as boolean vectors in a matrix

❖ Next goal: Find similar columns while computing small signatures

- Similarity of columns == similarity of signatures

Outline: Finding Similar Columns

- ❖ **Next Goal: Find similar columns, Small signatures**
- ❖ **Naïve approach:**
 - **1) Signatures of columns:** small summaries of columns
 - **2) Examine pairs of signatures** to find similar columns
 - ▶ **Essential:** Similarities of signatures and columns are related
 - **3) Optional:** Check that columns with similar signatures are really similar
- ❖ **Warnings:**
 - Comparing all pairs may take too much time: **Job for LSH**
 - ▶ These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures)

- ❖ **Key idea:** “hash” each column \mathbf{C} to a small *signature* $h(\mathbf{C})$, such that:
 - (1) $h(\mathbf{C})$ is small enough that the signature fits in RAM
 - (2) $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is the same as the “similarity” of signatures $h(\mathbf{C}_1)$ and $h(\mathbf{C}_2)$
 - ❖ **Goal:** Find a hash function $h(\cdot)$ such that:
 - If $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is high, then with high prob. $h(\mathbf{C}_1) = h(\mathbf{C}_2)$
 - If $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is low, then with high prob. $h(\mathbf{C}_1) \neq h(\mathbf{C}_2)$
- ❖ Hash docs into buckets. Expect that “most” pairs of near duplicate docs hash into the same bucket!

Min-Hashing

- ❖ **Goal: Find a hash function $h(\cdot)$ such that:**
 - if $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is high, then with high prob. $h(\mathbf{C}_1) = h(\mathbf{C}_2)$
 - if $\text{sim}(\mathbf{C}_1, \mathbf{C}_2)$ is low, then with high prob. $h(\mathbf{C}_1) \neq h(\mathbf{C}_2)$
- ❖ **Clearly, the hash function depends on the similarity metric:**
 - Not all similarity metrics have a suitable hash function
- ❖ **There is a suitable hash function for the Jaccard similarity: Min-Hashing**

Min-Hashing

- ❖ Imagine the rows of the boolean matrix permuted under **random permutation** π
- ❖ Define a “**hash**” function $h_{\pi}(\mathbf{C})$ = the index of the **first** (in the permuted order π) row in which column \mathbf{C} has value **1**:

$$h_{\pi}(\mathbf{C}) = \min_{\pi} \pi(\mathbf{C})$$

- ❖ Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Min-Hashing Example

1	0	1	1	0
2	0	0	1	1
3	1	0	0	0
4	0	1	0	1
5	0	0	0	1
6	1	1	0	0
7	0	0	1	0

Input Matrix

3	1	1	2
---	---	---	---

Signature Matrix

Min-Hashing Example

7	1	0	1	1	0
6	2	0	0	1	1
5	3	1	0	0	0
4	4	0	1	0	1
3	5	0	0	0	1
2	6	1	1	0	0
1	7	0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3

Signature Matrix

Min-Hashing Example

6	7	1	0	1	1	0
3	6	2	0	0	1	1
1	5	3	1	0	0	0
7	4	4	0	1	0	1
2	3	5	0	0	0	1
5	2	6	1	1	0	0
4	1	7	0	0	1	0

3	1	1	2
2	2	1	3
1	5	3	2

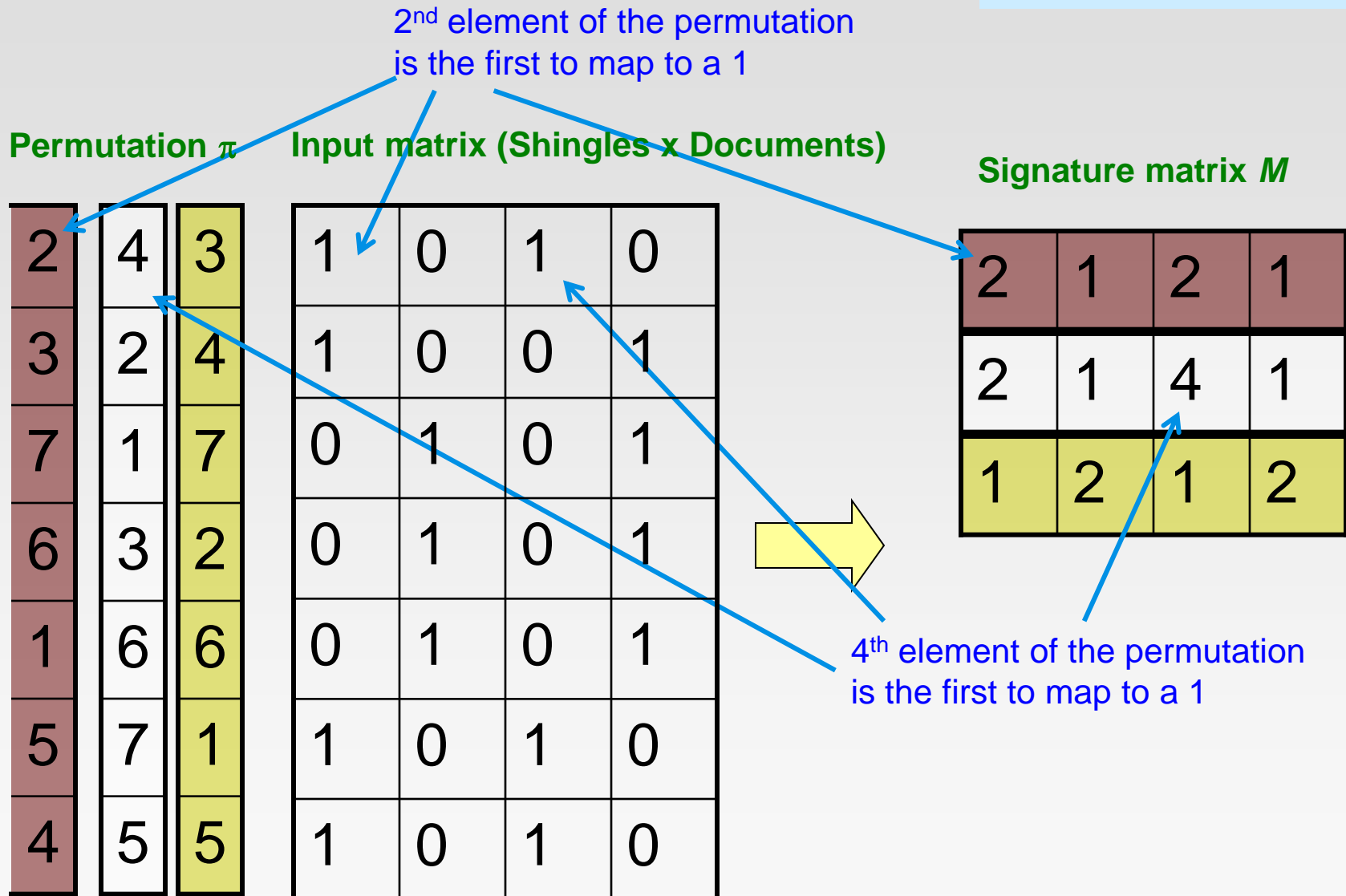
Input Matrix

Signature Matrix

Min-Hashing Exam

Note: Another (equivalent) way is to store row indexes:

1	5	1	5
2	3	1	3
6	4	6	4



The Min-Hash Property

❖ Choose a random permutation π

❖ Claim: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

❖ Why?

- Let X be a doc (set of shingles), $y \in X$ is a shingle
- **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
 - ▶ It is equally likely that any $y \in X$ is mapped to the **min** element
- Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
- **Then either:**
 - $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**
 - $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$
- So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
- $\Pr[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

0	0
0	0
1	1
0	0
0	1
1	0

One of the two
cols had to have
1 at position y

Four Types of Rows

❖ Given cols C_1 and C_2 , rows may be classified as:

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

➤ a = # rows of type A, etc.

❖ **Note:** $\text{sim}(C_1, C_2) = a/(a+b+c)$

❖ **Then:** $\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)$

➤ Look down the cols C_1 and C_2 until we see a 1

➤ If it's a type-A row, then $h(C_1) = h(C_2)$

If a type-B or type-C row, then not

Similarity for Signatures

Permutation π

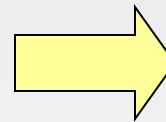
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix M

2	1	2	1
2	1	4	1
1	2	1	2



Similarities:

Col/Col
Sig/Sig

1-3	2-4	1-2	3-4
0.75	0.75	0	0
0.67	1.00	0	0

Similarity for Signatures

- ❖ We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- ❖ Now generalize to multiple hash functions
- ❖ The *similarity of two signatures* is the fraction of the hash functions in which they agree
- ❖ **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hash Signatures

- ❖ Pick $K=100$ random permutations of the rows
- ❖ Think of $\text{sig}(\mathbf{C})$ as a column vector
- ❖ $\text{sig}(\mathbf{C})[i] =$ according to the i -th permutation, the index of the first row that has a 1 in column C

$$\text{sig}(\mathbf{C})[i] = \min (\pi_i(\mathbf{C}))$$

- ❖ **Note:** The sketch (signature) of document C is small ~ 100 bytes!
- ❖ **We achieved our goal!** We “compressed” long bit vectors into short signatures

Implementation Trick

❖ **Permuting rows even once is prohibitive**

❖ **Row hashing!**

- Pick **$K = 100$** hash functions k_i
- Ordering under k_i gives a random row permutation!

❖ **One-pass implementation**

- For each column **C** and hash-func. k_i keep a “slot” for the min-hash value
- Initialize all **$sig(C)[i] = \infty$**
- **Scan rows looking for 1s**
 - ▶ Suppose row j has 1 in column **C**
 - ▶ Then for each k_i :
 - If $k_i(j) < sig(C)[i]$, then **$sig(C)[i] \leftarrow k_i(j)$**

How to pick a random hash function $h(x)$?

Universal hashing:

$$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$$

where:

a, b ... random integers

p ... prime number ($p > N$)

Implementation Example

Row	S_1	S_2	S_3	S_4	$x + 1 \bmod 5$	$3x + 1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

□ 0. Initialize all $\text{sig}(\mathbf{C})[i] = \infty$

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

❖ Row 0: we see that the values of $h_1(0)$ and $h_2(0)$ are both 1, thus $\text{sig}(S_1)[0] = 1$,
 $\text{sig}(S_1)[1] = 1$, $\text{sig}(S_4)[0] = 1$, $\text{sig}(S_4)[1] = 1$,

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

❖ Row 1, we see $h_1(1) = 2$ and $h_2(1) = 4$,
 thus $\text{sig}(S_3)[0] = 2$, $\text{sig}(S_3)[1] = 4$

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Implementation Example

Row	S_1	S_2	S_3	S_4	$x + 1 \bmod 5$	$3x + 1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

- ❖ Row 2: $h_1(2) = 3$ and $h_2(2) = 2$, thus
 $\text{sig}(S_2)[0] = 3$, $\text{sig}(S_2)[1] = 2$, no update for S_4

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

- ❖ Row 3: $h_1(3) = 4$ and $h_2(3) = 0$, update
 $\text{sig}(S_1)[1] = 0$, $\text{sig}(S_3)[1] = 0$, $\text{sig}(S_4)[1] = 0$,

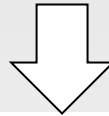
	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

- ❖ Row 4: $h_1(4) = 0$ and $h_2(4) = 3$, update
 $\text{sig}(S_3)[0] = 0$,

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Implementation Example

Row	S_1	S_2	S_3	S_4	$x + 1 \bmod 5$	$3x + 1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3



	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

- ❖ We can estimate the Jaccard similarities of the underlying sets from this signature matrix.
 - Signature matrix: $\text{SIM}(S_1, S_4) = 1.0$
 - Jaccard Similarity: $\text{SIM}(S_1, S_4) = 2/3$

References

- ❖ Chapter 3 of Mining of Massive Datasets.

End of Chapter 7.1