Agenda

- Vector space model
- Cosine similarity

The Vector Space Model

- Assume t distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These "orthogonal" terms form a vector space.

  $$\text{Dimension} = t = |\text{vocabulary}|$$

- Each term, i, in a document or query, j, is given a real-valued weight, w_{ij}.
- Both documents and queries are expressed as t-dimensional vectors:

  $$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

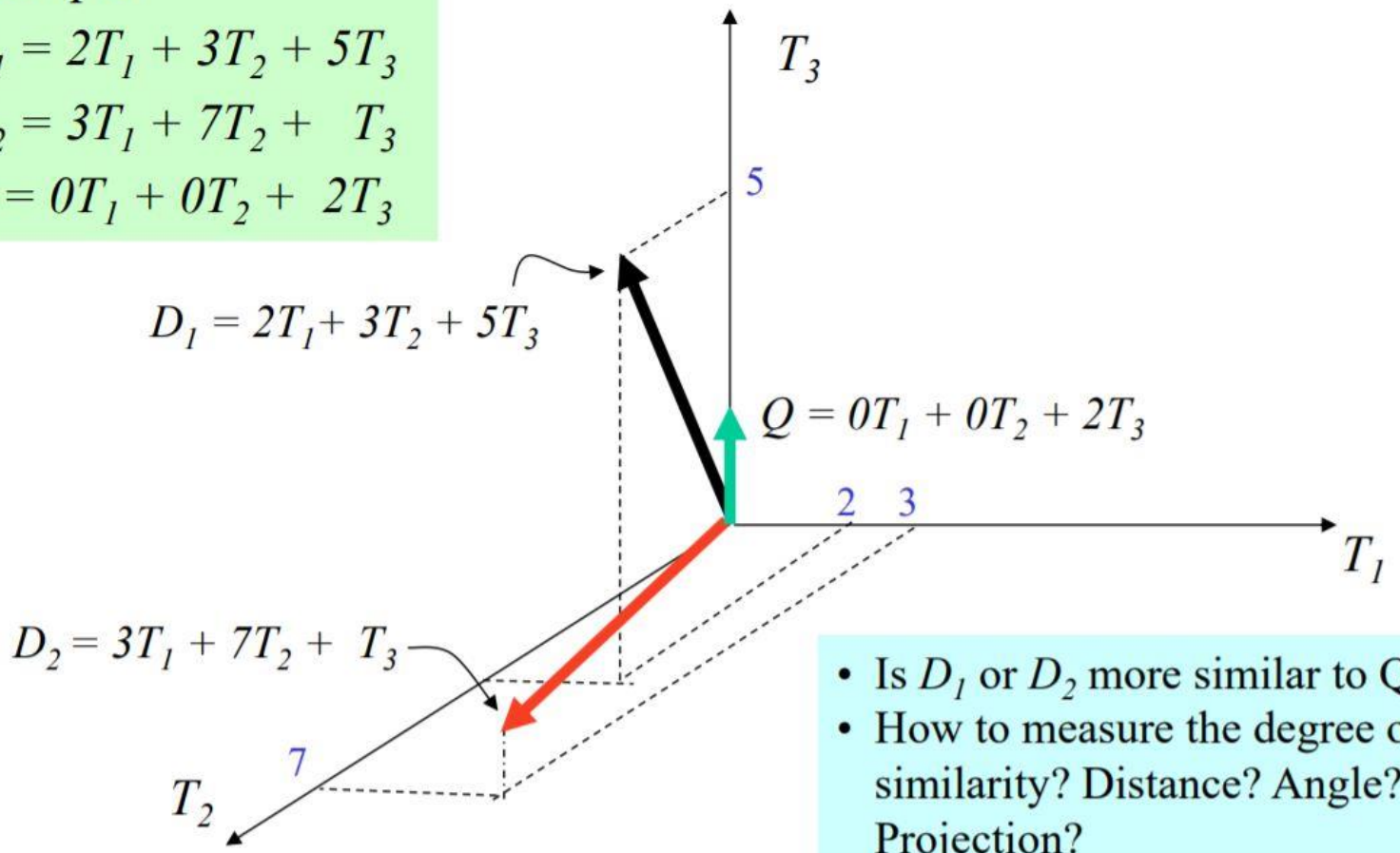
Graphical representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

Document collection

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$\begin{array}{c|cccc} & T_1 & T_2 & \ldots & T_t \\ D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\ D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}$$

Term Weights: Term Frequency (Recap)

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/ df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Example (TF-IDF)

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and

document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = log(10000/50) = 5.3; tf-idf = 5.3

B: tf = 2/3; idf = log(10000/1300) = 2.0; tf-idf = 1.3

C: tf = 1/3; idf = log(10000/250) = 3.7; tf-idf = 1.2

Similarity Measure - Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product:

$$\text{sim}(\boldsymbol{d}_j, \boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

Inner Product - Examples

Binary:

retrieval, database, architecture, computer, text, management, information

- D = 1, 1, 1, 0, 1, 1, 0
- Q = 1, 0, 1, 0, 0, 1, 1

Size of vector = size of vocabulary = 7
0 means corresponding term not found in document or query

$sim(D, Q) = 3$

Weighted:

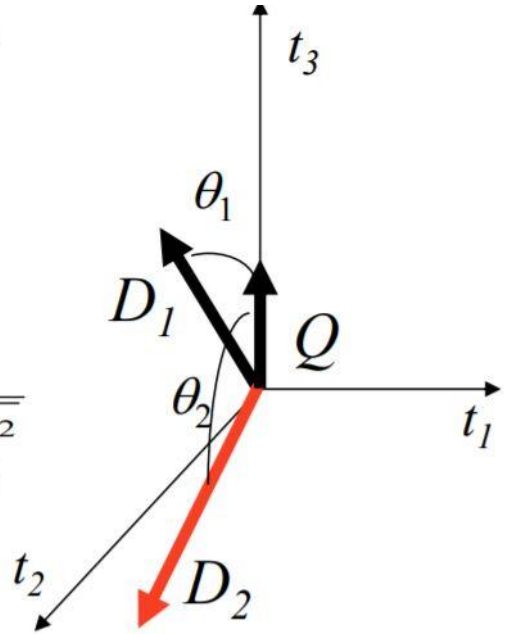
$D_1 = 2T_1 + 3T_2 + 5T_3 \qquad D_2 = 3T_1 + 7T_2 + 1T_3$
$Q = 0T_1 + 0T_2 + 2T_3$

$sim(D_1, Q) = 2*0 + 3*0 + 5*2 = 10$
$sim(D_2, Q) = 3*0 + 7*0 + 1*2 = 2$

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\boldsymbol{d_j}, \boldsymbol{q}) = \frac{\vec{d_j} \cdot \vec{q}}{\left|\vec{d_j}\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^{\ 2} \cdot \sum_{i=1}^{t} w_{iq}^{\ 2}}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \text{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$

$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \text{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$

$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Exercise

- A user is browsing 4 websites, he wants to find the similarity between each website content. After pre-processing is finished, the content of the 4 websites is as follows:
  - $\mathbf{W_1}$ = information media science
  - $\mathbf{W_2}$ = media science lebanese
  - $\mathbf{W_3}$ = media lebanese science information science
  - $\mathbf{W_4}$ = lebanese media lebanese science
- Write the binary and term-weighted matrix
- Calculate the similarity of W_1 with W_2 and W_3
- Calculate the similarity of W_1 with W_4
- What is the most similar document to W_1
- Draw the vectors in a 2 dimentional (2D) space, with the correct angle between each vector.

Solution

- **Binary matrix**

	Media	Science	Lebanese	Information
W_1	1	1	0	1
W_2	1	1	1	0
W_3	1	1	1	1
W_4	1	1	1	0

- **Term weighted matrix**

	Media	Science	Lebanese	Information
W_1	1	1	0	1
W_2	1	1	1	0
W_3	1	2	1	1
W_4	1	1	2	0

Solution

» **Vectors**

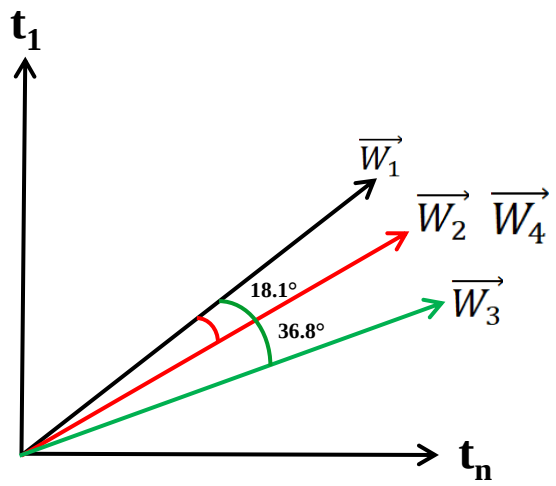
- $W_1 = 1T_1+1T_2+0T_3+1T_4$
- $W_2 = 1T_1+1T_2+0T_3+0T_4$
- $W_3 = 1T_1+2T_2+0T_3+1T_4$
- $W_4 = 1T_1+1T_2+2T_3+0T_4$

Solution

- **Inner products**
  - $sim(W_1, W_2) = (1x1)+(1x1)+(0x1)+(1x0) = 2$
  - $sim(W_1, W_3) = 4$
  - $sim(W_1, W_4) = 2$
- Cosine similarity
  - $cos(W_1, W_2) = \frac{4}{\sqrt{(1+1+1)(1+1+1)}} = \frac{2}{2.4} = 0.8$
  - $cos(W_1, W_3) = 0.95$
  - $cos(W_1, W_4) = 0.8$

Solution

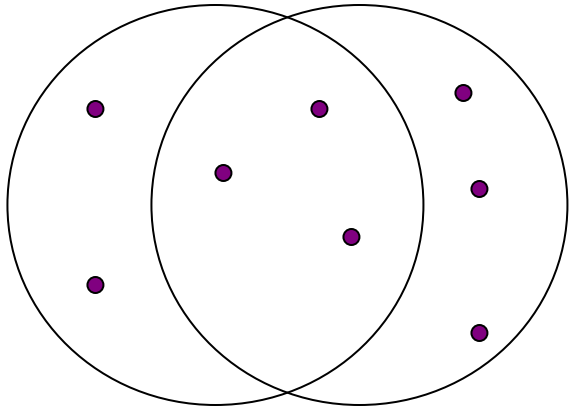
- **Vector space**
  - $\angle(W_1, W_2) = \cos^{-1}(0.8) = 36.8$
  - $\angle(W_1, W_3) = \cos^{-1}(0.95) = 18.1$
  - $\angle(W_1, W_4) = \cos^{-1}(0.8) = 36.8$
- W_2 and W_4 are the most relevant documents to W_1.



Jaccard Similarity

» The Jaccard similarity (Jaccard coefficient) of two sets S_1, S_2 is the size of their intersection divided by the size of their union.

- JSim $(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.

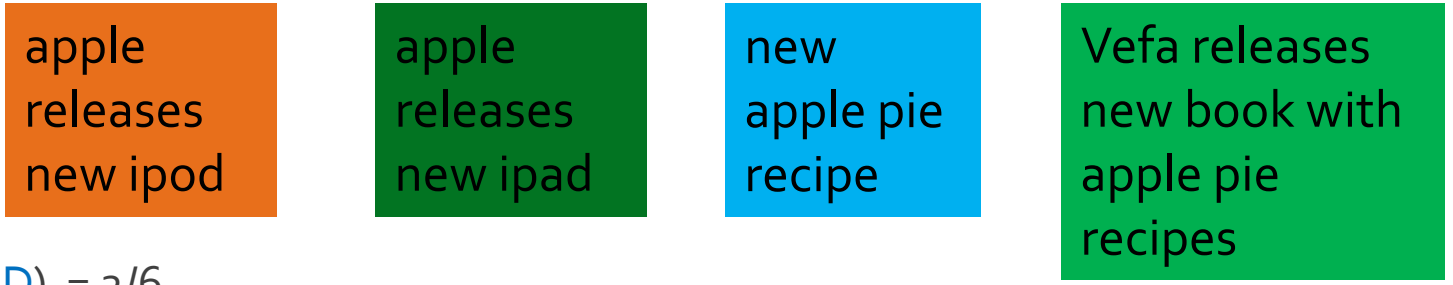


3 in intersection.
8 in union.
Jaccard similarity
= 3/8

- Extreme behavior:
  - Jsim(X,Y) = 1, iff X = Y
  - Jsim(X,Y) = 0 iff X,Y have no elements in common
- JSim is symmetric

Jaccard Similarity between sets

- The distance for the documents



- JSim(D,D) = 3/5
- JSim(D,D) = JSim(D,D) = 2/6
- JSim(D,D) = JSim(D,D) = 3/9

Agenda

- Dynamic programming
- Edit distance
- Applications

1
Dynamic Programming
Introduction

Dynamic Programming

1. General Design and Problem Solving Strategies
2. More about Dynamic Programming
   - Example: Edit Distance

Design Strategies

- Dynamic Programming Design Strategy
  - Solve an “easy” sub-problem
  - Store the solution
  - Use stored solution to solve a more difficult sub-problem.
  - Repeat until you solve the “big” hard problem

Overview

- What is edit distance? Why do we care?
- A few interesting properties
- Some algorithms from the past for finding exact edit distance
- More (complicated) algorithms from today for approximate edit distance
- Embedding edit distance into l_1

What is that?

Suppose we have two strings x,y

e.g. x = kitten

y = sitting

and we want to transform x into y.

We use *edit operations*: 1. insertions

2. deletions

3. substitutions

2
Edit Distance
Introduction

What is that?

A closer look:

k i t t e n

s i t t i n g

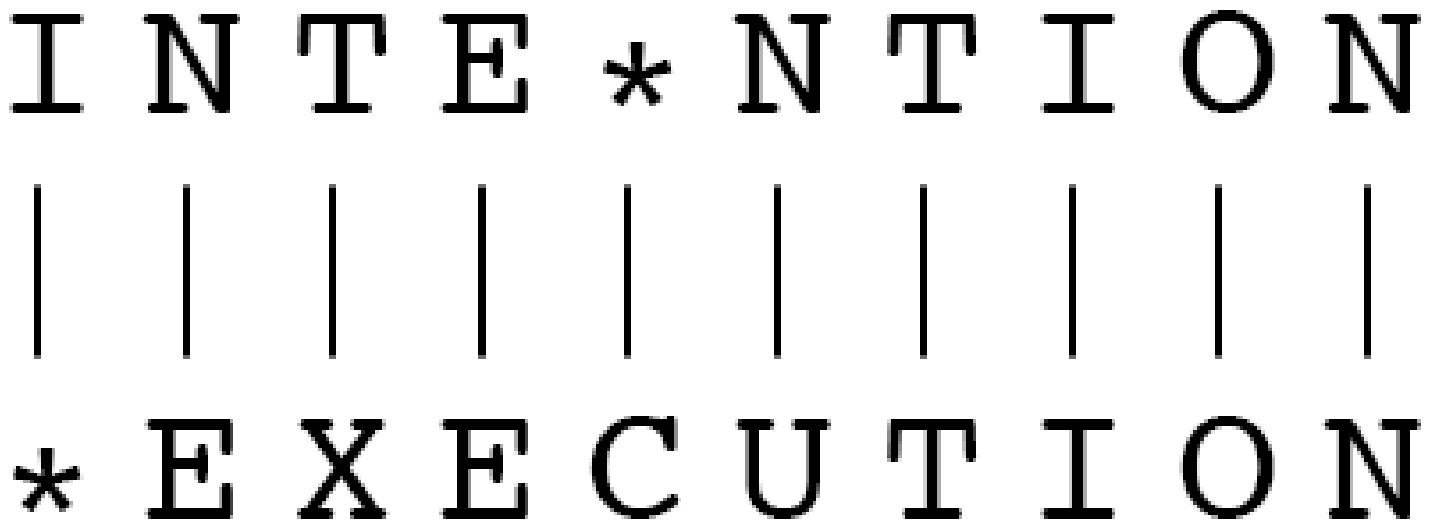
1st step: kitten →sitten (substitution)

2nd step: sitten→sittin (substitution)

3rd step: sittin→sitting (insertion)

Minimum Edit Distance

- Two strings and their **alignment**:



Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N

d s s   i s
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

What is that?

Can we do better?

Answer here is no (obviously)

What about:

x = darladidirladada

y = marmelladara

Tough…

Why do we care?

A lot of applications depend on the similarity of two strings

» Computational Biology:

…ATGCATACGATCGATT…

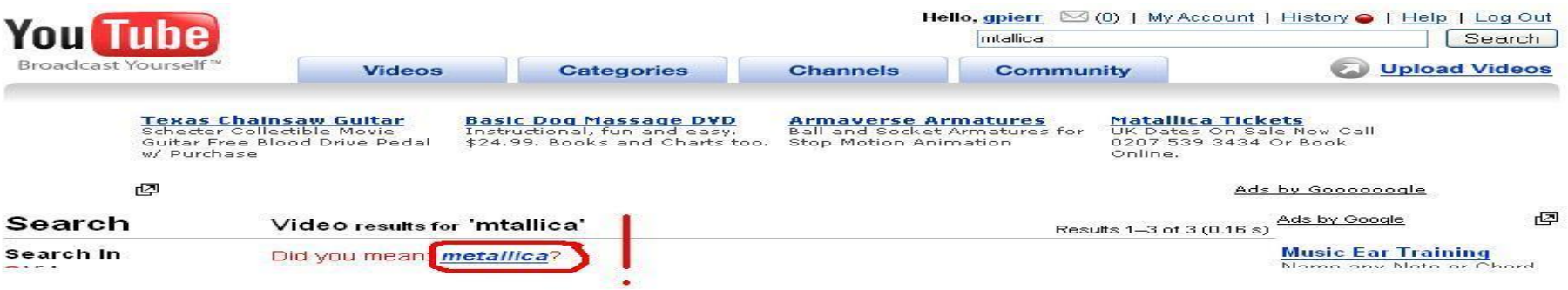
…TGCAATGGCTTAGCTA…

Animal species from the same family are bound to have more similar DNAs

What about evolutionary biology?

Why do we care?

» searching keywords through the net: usually by “mtallica” we mean “metallica”:



Other uses of Edit Distance in NLP

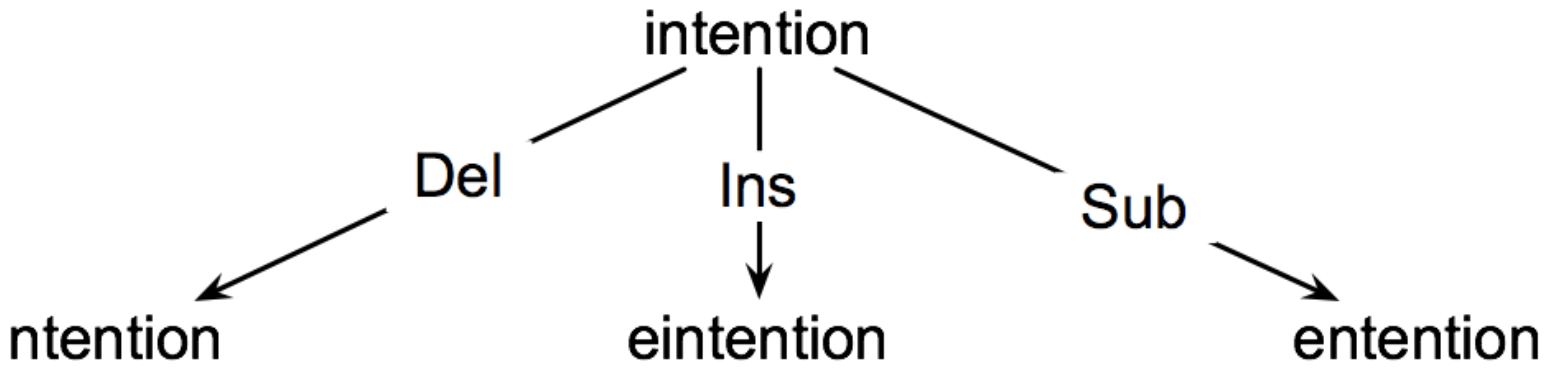
- Evaluating Machine Translation and speech recognition

R	Spokesman	confirms			senior	government	adviser		the	senior		was	shot
H	Spokesman	said								adviser		was	shot dead
		S		I			D						I

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announcedyesterday
  - for Stanford University President John Hennessy

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(*n,m*)
- Solving problems by combining solutions to subproblems.
- Bottom--up
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Defining Min Edit Distance (Levenshtein)

- Initialization $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

For each $i = 1 \ldots M$

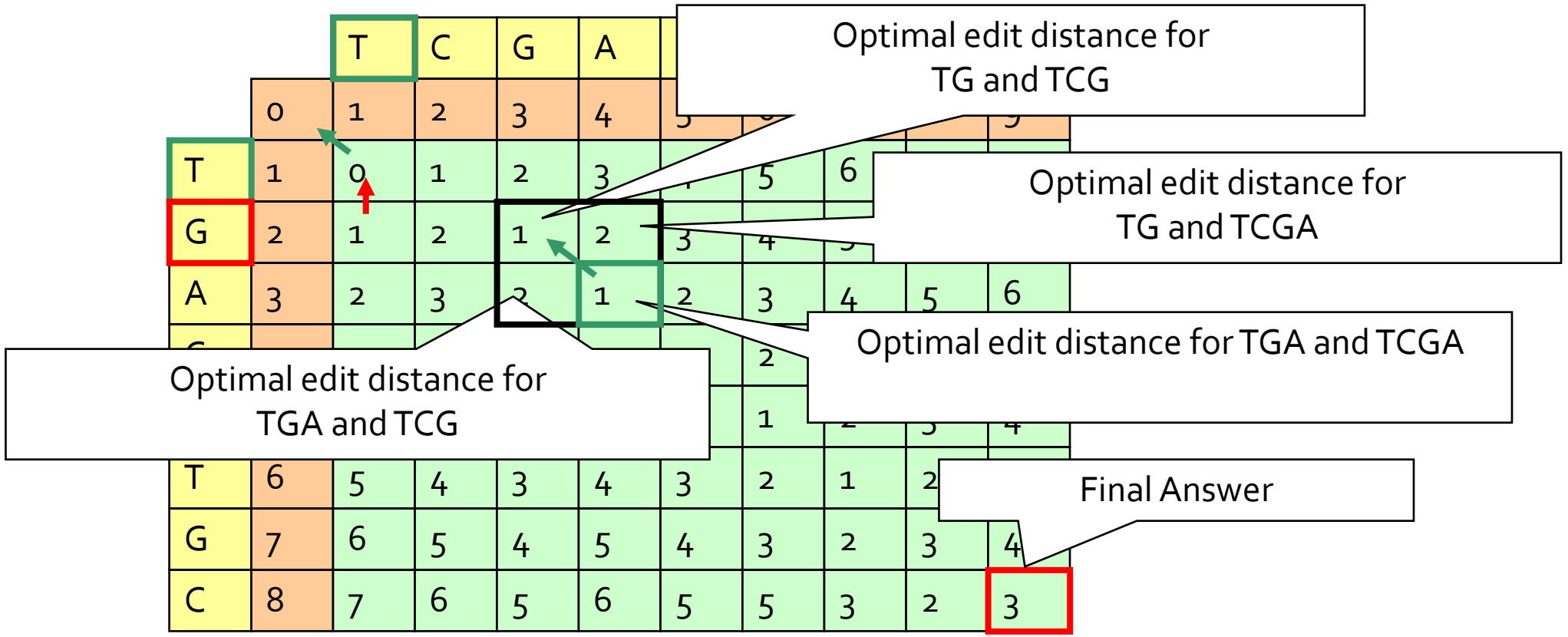
For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance

Edit Distance – Dynamic Programming



Example

I want to find the edit distance between «abcdef» and «azced»

	Null	a	b	c	d	e	f	Exp.
Null	0	1	2	3	4	5	6	Null -> abcdef, costs 8 D(null, abcdef) = 8
a	1	0	1	2 D(a,abc)	3	4	5	D(A , abcdef) = 5
z	2	1	1	2	3	4	5	az -> abcdef
c	3	2	2	1	2	3	4	
e	4	3	3	2	2	2	3	
d	5	4	4	3	2	3	3	D(abcdef, azced) = 3
Exp.	**Null -> azced D(null, azced) = 5**							Transfer f to d Delete d Transfer b to z =3

Computing alignments

- Edit distance isn't sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a "backtrace"
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment

Edit distance in R

```
#load stringdist package
library(stringdist)

#calculate Levenshtein distance between two strings
stringdist("string1", "string2", method = "lv")
```

Java Code

```java
package org.arpit.java2blog;

import java.util.Scanner;

public class EditDistance {

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        String s1 = scn.nextLine();
        String s2 = scn.nextLine();

        System.out.println(editDist(s1, s2));

    }

    public static int editDist(String s1, String s2) {
        int[][] dp = new int[s1.length() + 1][s2.length() +

        /* this is the last column of the matrix which
         *  represents the result of empty string1
         *  and string2 starting from current row.
         */
        for (int row = s2.length(); row >= 0; row--) {
            dp[row][s1.length()] = s2.length() - row;
        }
        /* this is the last row of the matrix which
        * represents the result of empty string2
        *  and string1 starting from current col.
        */
        for (int col = s1.length(); col >= 0; col--) {
            dp[s2.length()][col] = s1.length() - col;
        }

        for (int col = s1.length() - 1; col >= 0; col--) {
            for (int row = s2.length() - 1; row >= 0; row--) {

                /* If characters are same, then the solution will be without
                 * these characters */
                if (s1.charAt(col) == s2.charAt(row)) {
                    dp[row][col] = dp[row + 1][col + 1];
                } else {
                /* else it will be minimum of these three operations
                */
                dp[row][col] = 1 + Math.min(dp[row + 1][col + 1], // replace
                                Math.min(dp[row][col + 1]  //   removal
                                , dp[row + 1][col]));   // insertion
                }
            }
        }

        return dp[0][0];
    }
```

String similarity measures

- Token-based
  - Examples
    - Jaccard
    - TF-IDF Cosine similarities
  - Suitable for large documents
- Character-based
  - Examples:
    - Edit-distance and variants like Levenshtein, Jaro-Winkler
    - Soundex
  - Suitable for short strings with spelling mistakes
- Hybrids