

# Algorithms and Datastructures

Levenshtein distance, Dynamic programming

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science  
Algorithms and Datastructures, February 2018

Introduction

Edit distance

Introduction

Edit distance

**Edit distance:**

## **Edit distance:**

- Measurement for similarity of two words / strings

## **Edit distance:**

- Measurement for similarity of two words / strings
- Algorithm for efficient calculation

## **Edit distance:**

- Measurement for similarity of two words / strings
- Algorithm for efficient calculation
- General principle: dynamic programming

## BioInfSearch

ejafjatljökuk
eyjafjallajökull
eyjafjallajökull movie
eyjafjallajölull trailer

Search!



Ulrich Latzenhofer; CC BY-SA 2.0

### Wikipedia.org:

"Der Eyjafjallajökull ([ˈeɪjaˌfjatlaˌjœːkʏtʃ]](3), auf Deutsch Eyjafjöll-Gletscher, ist der sechstgrößte Gletscher Islands.

Er liegt an der äußersten Südküste, westlich des Gletschers Mýrdalsjökull in der Gemeinde Rangárpíng eystra, die größte Höhe beträgt 1651 m. Unter dem Gletscher befindet sich der Vulkan Eyjafjöll mit eigener Magmakammer, der seit der Besiedelung von Island in den Jahren 920, 1612 (oder 1613), 1821 bis 1823 und zuletzt im Jahr 2010 aktiv war."





**A lot of applications where similar string are searched:**

### A lot of applications where similar string are searched:

- Duplicates in databases:

Hein Blöd	27568	Bremerhaven
Hein Bloed	27568	Bremerhafen
Hein Doof	27478	Cuxhaven

### **A lot of applications where similar string are searched:**

- Duplicates in databases:

Hein Blöd     27568   Bremerhaven

Hein Bloed   27568   Bremerhafen

Hein Doof    27478   Cuxhaven

- Product search:

memory stik

### A lot of applications where similar string are searched:

- Duplicates in databases:

Hein Blöd     27568   Bremerhaven

Hein Bloed   27568   Bremerhafen

Hein Doof    27478   Cuxhaven

- Product search:

memory stik

- Websearch:

eyjaföllajaküll

uniwersität verien 2017

### A lot of applications where similar string are searched:

- Duplicates in databases:

Hein Blöd     27568 Bremerhaven

Hein Bloed    27568 Bremerhafen

Hein Doof     27478 Cuxhaven

- Product search:

memory stik

- Websearch:

eyjaföllajaküll

uniwersität verien 2017

- Bioinformatics: Similarity of DNA-sequences

### **Search of similar proteins:**

### Search of similar proteins:

- BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool)

### Search of similar proteins:

- BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool)
- Alignment  $\hat{=}$  Edit distance



### Search of similar proteins:

- BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool)
- Alignment  $\hat{=}$  Edit distance
- Changed life-science completely

### Search of similar proteins:

- BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool)
- Alignment  $\hat{=}$  Edit distance
- Changed life-science completely
- Cited 63437 times on Google Scholar (Sep. 2017)

Introduction

Edit distance

**Definition of edit distance:** (*Levenshtein-distance*)

## Definition of edit distance: (*Levenshtein-distance*)

- Let  $x$ ,  $y$  be two strings
- Edit distance  $ED(x, y)$  of  $x$  and  $y$ :  
The minimal number of operations to transform  $x$  into  $y$

## Definition of edit distance: (*Levenshtein-distance*)

- Let  $x$ ,  $y$  be two strings
- Edit distance  $ED(x, y)$  of  $x$  and  $y$ :  
The minimal number of operations to transform  $x$  into  $y$ 
  - Insert a character

## Definition of edit distance: (*Levenshtein-distance*)

- Let  $x, y$  be two strings
- Edit distance  $ED(x, y)$  of  $x$  and  $y$ :  
The minimal number of operations to transform  $x$  into  $y$ 
  - Insert a character
  - Replace a character with another

## Definition of edit distance: (*Levenshtein-distance*)

- Let  $x$ ,  $y$  be two strings
- Edit distance  $ED(x, y)$  of  $x$  and  $y$ :  
The minimal number of operations to transform  $x$  into  $y$ 
  - Insert a character
  - Replace a character with another
  - Delete a character



# Edit distance

## Example



1 2 3 4 5  
DOOF

BLOED

# Edit distance

## Example



1 2 3 4 5

DOOF



replace(1, B)

BOOF

BLOED

# Edit distance

## Example

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF

BLOED

# Edit distance

## Example

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF



insert(4, E)

BLOEF

BLOED

# Edit distance

## Example

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF



insert(4, E)

BLOEF



replace(5, D)

BLOED

# Edit distance

## Example

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF



insert(4, E)

BLOEF



replace(5, D)

BLOED

⏟  
ED=4

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

1 2 3 4 5

BLOED

⏟  
ED=4

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

ED=4

1 2 3 4 5

B LOED

DOOF



# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

⏟  
ED=4

1 2 3 4 5

B LOED



B LOEF

replace(5, F)

DOOF

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

ED=4

1 2 3 4 5

B LOED



B LOEF



B LOF

replace(5, F)

delete(4)

DOOF

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

ED=4

1 2 3 4 5

B LOED



B LOEF



B LOF



BOOF

DOOF

replace(5, F)

delete(4)

replace(2, O)

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

ED=4

1 2 3 4 5

B LOED



B LOEF



B LOF



BOOF



DOOF

replace(5, F)

delete(4)

replace(2, O)

replace(1, D)

# Edit distance

## Example

1 2 3 4 5

DOOF



BOOF



BLOF



BLOEF



BLOED

replace(1, B)

replace(2, L)

insert(4, E)

replace(5, D)

ED=4

1 2 3 4 5

B LOED



B LOEF



B LOF



BOOF



DOOF

replace(5, F)

delete(4)

replace(2,  $\emptyset$ )

replace(1, D)

ED=4



## Notation:

## Notation:

- $\varepsilon$  is the empty string

## Notation:

- $\varepsilon$  is the empty string
- $|x|$  is the length of the string  $x$  (number of characters)



## Notation:

- $\varepsilon$  is the empty string
- $|x|$  is the length of the string  $x$  (number of characters)

## Notation:

- $\varepsilon$  is the empty string
- $|x|$  is the length of the string  $x$  (number of characters)
- $x[i..j]$  is the slice of  $x$  from  $i$  to  $j$  where  $1 \leq i \leq j \leq |x|$

## Notation:

- $\varepsilon$  is the empty string
- $|x|$  is the length of the string  $x$  (number of characters)
- $x[i..j]$  is the slice of  $x$  from  $i$  to  $j$  where  $1 \leq i \leq j \leq |x|$





## Trivial facts:

## Trivial facts:

- $ED(x, y) = ED(y, x)$

## Trivial facts:

- $ED(x, y) = ED(y, x)$
- $ED(x, \varepsilon) = |x|$

## Trivial facts:

- $ED(x, y) = ED(y, x)$
- $ED(x, \varepsilon) = |x|$
- $ED(x, y) \geq \text{abs}(|x| - |y|)$

$$\text{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{else} \end{cases}$$

## Trivial facts:

■  $ED(x, y) = ED(y, x)$

■  $ED(x, \varepsilon) = |x|$

■  $ED(x, y) \geq \text{abs}(|x| - |y|)$

$$\text{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{else} \end{cases}$$

■  $ED(x, y) \leq ED(x[1..n-1], y[1..m-1]) + 1 \quad n = |x|, m = |y|$





## **Solutions based on examples:**

### **Solutions based on examples:**

- From VERIEN to FERIEEN?



### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?

### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEBA to RBEAAEEBAAAEBBAEAE?

### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn't have to

### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn't have to

### Recursive approach:

### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn't have to

### Recursive approach:

- Dividing in two halves? Not a good idea:

$$ED(\textit{GRAU}, \textit{RAUM}) = 2 \quad \text{but} \quad ED(\textit{GR}, \textit{RA}) + ED(\textit{AU}, \textit{UM}) = 4$$

### Solutions based on examples:

- From VERIEN to FERIEEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn't have to

### Recursive approach:

- Dividing in two halves? Not a good idea:

$$ED(\text{GRAU}, \text{RAUM}) = 2 \quad \text{but} \quad ED(\text{GR}, \text{RA}) + ED(\text{AU}, \text{UM}) = 4$$

- Finding “smaller” sub problems?  
Let's try it!



## Terminology:

## Terminology:

- Let  $x$ ,  $y$  be two strings

## Terminology:

- Let  $x, y$  be two strings
- Let  $\sigma_1, \dots, \sigma_k$  be a sequence of  $k$  operations where  $k = \text{ED}(x, y)$  for  $x \rightarrow y$  (transform  $x$  into  $y$ )  
(We do not know this sequence but we assume it exists)



## Terminology:

## Terminology:

- We only consider **monotonous** sequences:  
The position of  $\sigma_{i+1}$  is  $\geq$  the position of  $\sigma_i$  where we only allow the positions to be equal on a delete operation

## Terminology:

- We only consider **monotonous** sequences:

The position of  $\sigma_{i+1}$  is  $\geq$  the position of  $\sigma_i$  where we only allow the positions to be equal on a delete operation

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF



insert(4, E)

BLOEF



replace(5, D)

BLOED

1 2 3 4 5 6 7

SAUDOOF



delete(1)

AUDOOF



delete(1)

UDOOF



delete(1)

DOOF



insert(4, O)

DOOOF

## Terminology:

- We only consider **monotonous** sequences:

The position of  $\sigma_{i+1}$  is  $\geq$  the position of  $\sigma_i$  where we only allow the positions to be equal on a delete operation

1 2 3 4 5

DOOF



replace(1, B)

BOOF



replace(2, L)

BLOF



insert(4, E)

BLOEF



replace(5, D)

BLOED

1 2 3 4 5 6 7

SAUDOOF



delete(1)

AUDOOF



delete(1)

UDOOF



delete(1)

DOOF



insert(4, O)

DOOOF



## Terminology:



## Terminology:

- **Lemma:** For any  $x$  and  $y$  with  $k = \text{ED}(x, y)$  exists a **monotonous** sequence of  $k$  operations for  $x \rightarrow y$

## Terminology:

- **Lemma:** For any  $x$  and  $y$  with  $k = \text{ED}(x, y)$  exists a **monotonous** sequence of  $k$  operations for  $x \rightarrow y$
- **Intuition:** The order of our sequence is not relevant (Therefore we can also sort them monotonously)

## Terminology:

- **Lemma:** For any  $x$  and  $y$  with  $k = \text{ED}(x, y)$  exists a **monotonous** sequence of  $k$  operations for  $x \rightarrow y$
- **Intuition:** The order of our sequence is not relevant (Therefore we can also sort them monotonously)

1	2	3	4	5
D	O	O	F	

B L O E D

1	2	3	4	5	6	7
S	A	U	D	O	O	F

D O O O F



**Consider the last operation:**

### Consider the last operation:

- Solve **blue** part recursively

### Consider the last operation:

- Solve **blue** part recursively

DOOF

↓↓↓↓

BLOE

↓ insert

BLOED

Figure: Case 1a

DOOF

↓↓↓↓↓

BLOEDF

↓ delete

BLOED

Figure: Case 1b

DOOF

↓↓↓↓↓

BLOEF

↓ replace

BLOED

Figure: Case 1c



**Consider the last operation:**

### Consider the last operation:

- Solve **blue** part recursively



### Consider the last operation:

- Solve **blue** part recursively

W I N T E R



S O M M E R

↓ nothing

S O M M E R

### Display of solution:

- Alignment

- Example:

<u>S</u>	<u>A</u>	<u>U</u>	B	L	O	E	D
S	A	U	B	L	O	E	D

**Figure:** Case 2



## Dynamic programming:

### Dynamic programming:

- Instances of Bellman's principle of optimality:

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance



**Figure:** Richard Bellman  
(1920 - 1984)

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance



**Figure:** Richard Bellman  
(1920 - 1984)

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance



**Figure:** Richard Bellman  
(1920 - 1984)

- Optimal solutions consist of optimal partial solutions



### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance



**Figure:** Richard Bellman  
(1920 - 1984)

- Optimal solutions consist of optimal partial solutions
  - Shortest paths: Each partial path has to be optimal

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance



**Figure:** Richard Bellman  
(1920 - 1984)

- Optimal solutions consist of optimal partial solutions
  - Shortest paths: Each partial path has to be optimal
  - Edit distance: Each partial alignment has to be optimal

### Dynamic programming:

- Instances of Bellman's principle of optimality:
  - Shortest paths
  - Edit distance
- Optimal solutions consist of optimal partial solutions
  - Shortest paths: Each partial path has to be optimal
  - Edit distance: Each partial alignment has to be optimal
- Always solvable through dynamic programming  
(Caching of optimal partial solutions)



**Figure:** Richard Bellman  
(1920 - 1984)



## Case analysis:

## Case analysis:

- We consider the last operation  $\sigma_k$

## Case analysis:

- We consider the last operation  $\sigma_k$ 
  - $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow z$  and  $\sigma_k: z \rightarrow y$

Example:

$x = \text{DOOF}, z = \text{SAUBLOEF}, y = \text{SAUBLOED}$

## Case analysis:

- We consider the last operation  $\sigma_k$ 
  - $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow z$  and  $\sigma_k: z \rightarrow y$

Example:

$x = \text{DOOF}, z = \text{SAUBLOEF}, y = \text{SAUBLOED}$

- Let  $n = |x|, m = |y|, m' = |z|$

## Case analysis:

- We consider the last operation  $\sigma_k$ 
  - $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow z$  and  $\sigma_k: z \rightarrow y$   
Example:

$x = \text{DOOF}, z = \text{SAUBLOEF}, y = \text{SAUBLOED}$

- Let  $n = |x|, m = |y|, m' = |z|$
- We note  $m' \in \{m-1, m, m+1\}$       why?





## Case analysis:

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:
  - Case 1a:  $\sigma_k = \text{insert}(m' + 1, y[m])$  [then  $m' = m - 1$ ]

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:
  - Case 1a:  $\sigma_k = \text{insert}(m' + 1, y[m])$  [then  $m' = m - 1$ ]
  - Case 1b:  $\sigma_k = \text{delete}(m')$  [then  $m' = m + 1$ ]

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:
  - Case 1a:  $\sigma_k = \text{insert}(m' + 1, y[m])$  [then  $m' = m - 1$ ]
  - Case 1b:  $\sigma_k = \text{delete}(m')$  [then  $m' = m + 1$ ]
  - Case 1c:  $\sigma_k = \text{replace}(m', y[m])$  [then  $m' = m$ ]

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:
  - Case 1a:  $\sigma_k = \text{insert}(m' + 1, y[m])$  [then  $m' = m - 1$ ]
  - Case 1b:  $\sigma_k = \text{delete}(m')$  [then  $m' = m + 1$ ]
  - Case 1c:  $\sigma_k = \text{replace}(m', y[m])$  [then  $m' = m$ ]
- Case 2:  $\sigma_k$  does nothing at the outer end:

## Case analysis:

- Case 1:  $\sigma_k$  does something at the outer end:
  - Case 1a:  $\sigma_k = \text{insert}(m' + 1, y[m])$  [then  $m' = m - 1$ ]
  - Case 1b:  $\sigma_k = \text{delete}(m')$  [then  $m' = m + 1$ ]
  - Case 1c:  $\sigma_k = \text{replace}(m', y[m])$  [then  $m' = m$ ]
- Case 2:  $\sigma_k$  does nothing at the outer end:
  - Then  $z[m'] = y[m]$  and  $x[n'] = z[m']$  and with that  
 $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$  and  $x[n] = y[m]$



## Case analysis:



## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

**This results in the recursive formula:**

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and
  - $ED(x[1..n-1], y) + 1$  and



## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and
  - $ED(x[1..n-1], y) + 1$  and
  - $ED(x[1..n-1], y[1..m-1]) + 1$  if  $x[n] \neq y[m]$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and
  - $ED(x[1..n-1], y) + 1$  and
  - $ED(x[1..n-1], y[1..m-1]) + 1$  if  $x[n] \neq y[m]$
  - $ED(x[1..n-1], y[1..m-1]) + 0$  if  $x[n] = y[m]$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and
  - $ED(x[1..n-1], y) + 1$  and
  - $ED(x[1..n-1], y[1..m-1]) + 1$  if  $x[n] \neq y[m]$
  - $ED(x[1..n-1], y[1..m-1]) + 0$  if  $x[n] = y[m]$
- For  $|x| = 0$  is  $ED(x, y) = |y|$

## Case analysis:

- Case 1a (insert):  $\sigma_1, \dots, \sigma_{k-1}: x \rightarrow y[1..m-1]$
- Case 1b (delete):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y$
- Case 1c (replace):  $\sigma_1, \dots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$
- Case 2 (nothing):  $\sigma_1, \dots, \sigma_k: x[1..n-1] \rightarrow y[1..m-1]$

## This results in the recursive formula:

- For  $|x| > 0$  and  $|y| > 0$  is  $ED(x, y)$  the minimum of
  - $ED(x, y[1..m-1]) + 1$  and
  - $ED(x[1..n-1], y) + 1$  and
  - $ED(x[1..n-1], y[1..m-1]) + 1$  if  $x[n] \neq y[m]$
  - $ED(x[1..n-1], y[1..m-1]) + 0$  if  $x[n] = y[m]$
- For  $|x| = 0$  is  $ED(x, y) = |y|$
- For  $|y| = 0$  is  $ED(x, y) = |x|$

```
def edit_distance(x, y):  
    if len(x) == 0:  
        return len(y)  
    if len(y) == 0:  
        return len(x)  
  
    ed1 = edit_distance(x, y[:-1]) + 1  
    ed2 = edit_distance(x[:-1], y) + 1  
    ed3 = edit_distance(x[:-1], y[:-1])  
    if x[-1] != y[-1]:  
        ed3 += 1  
  
    return min(ed1, ed2, ed3)
```



## Recursive program:

### Recursive program:

- The algorithm results in the following recursive formular:

$$\begin{aligned}T(n, m) &= T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1 \\&\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1) \\&= 3 \cdot T(n-1, m-1)\end{aligned}$$

### Recursive program:

- The algorithm results in the following recursive formular:

$$\begin{aligned}T(n, m) &= T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1 \\&\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1) \\&= 3 \cdot T(n-1, m-1)\end{aligned}$$

- This results in  $T(n, n) \geq 3^n$



### Recursive program:

- The algorithm results in the following recursive formular:

$$\begin{aligned}T(n, m) &= T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1 \\&\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1) \\&= 3 \cdot T(n-1, m-1)\end{aligned}$$

- This results in  $T(n, n) \geq 3^n$

⇒ The runtime is at least exponential



## Dynamic programming:

## Dynamic programming:

- We create a table with all possible combination of substrings and save calculated entries
- This results in a runtime and space consumption of  $O(n \cdot m)$

## Dynamic programming:

- We create a table with all possible combination of substrings and save calculated entries
- This results in a runtime and space consumption of  $O(n \cdot m)$

## Visualization on the next slide:

## Dynamic programming:

- We create a table with all possible combination of substrings and save calculated entries
- This results in a runtime and space consumption of  $O(n \cdot m)$

## Visualization on the next slide:

- Operations always refer to the last position (indices are omitted)

## Dynamic programming:

- We create a table with all possible combination of substrings and save calculated entries
- This results in a runtime and space consumption of  $O(n \cdot m)$

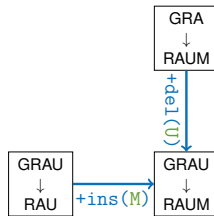
## Visualization on the next slide:

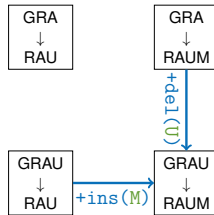
- Operations always refer to the last position (indices are omitted)
- We also display the replaced character on a `replace` operation to visualize operations without costs  
 $\Rightarrow \text{repl}(\text{A}, \text{A})$

















### Fast algorithm:

We can determine the **edit distance** for all combination of partial strings from the top left to bottom right.

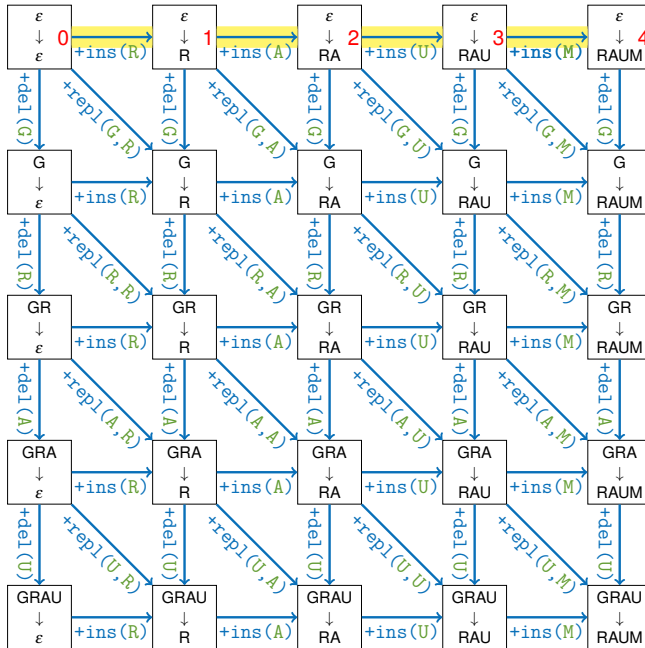














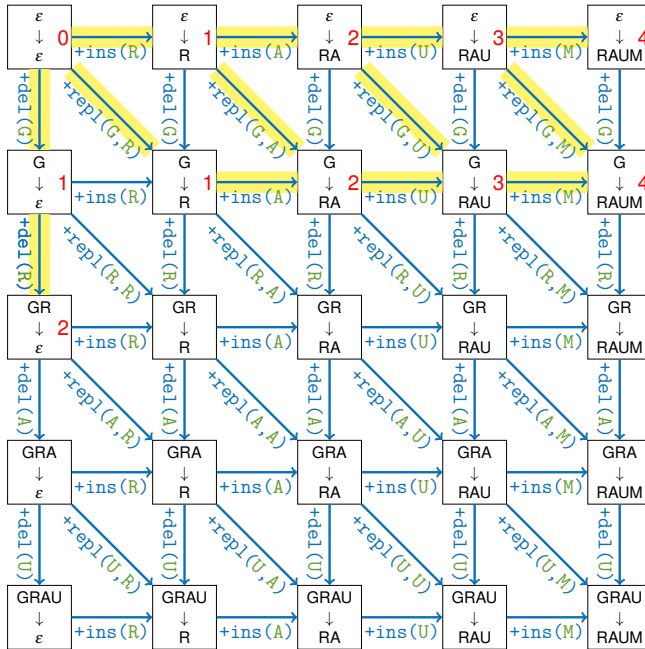


















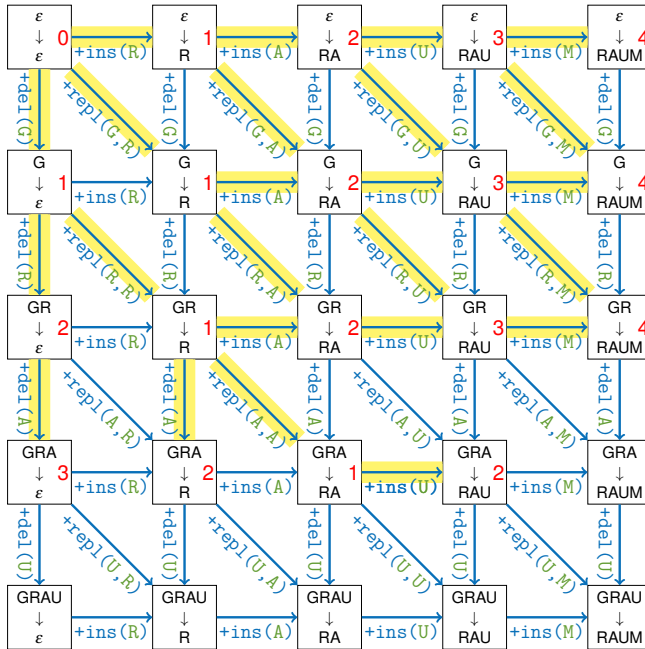










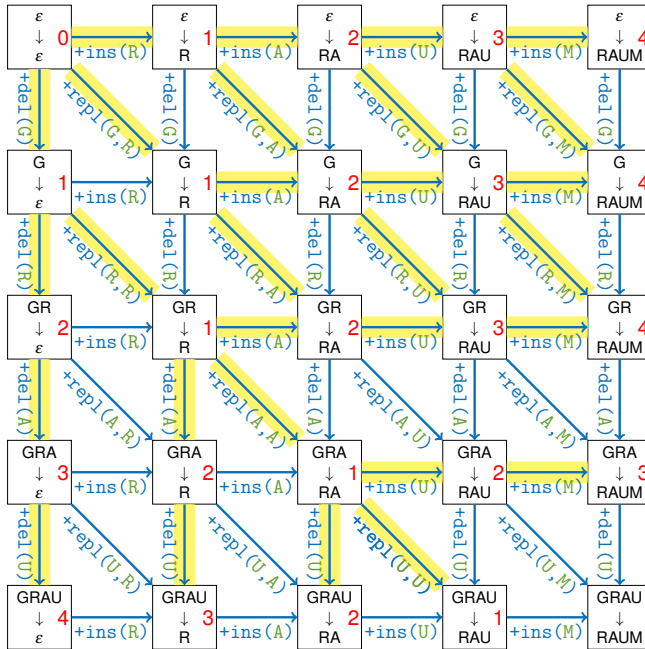














**How to get the sequence of operations?**



## How to get the sequence of operations?

- We save at each recursion the most efficient previous entry (the **highlighted arrows** in our image)

## How to get the sequence of operations?

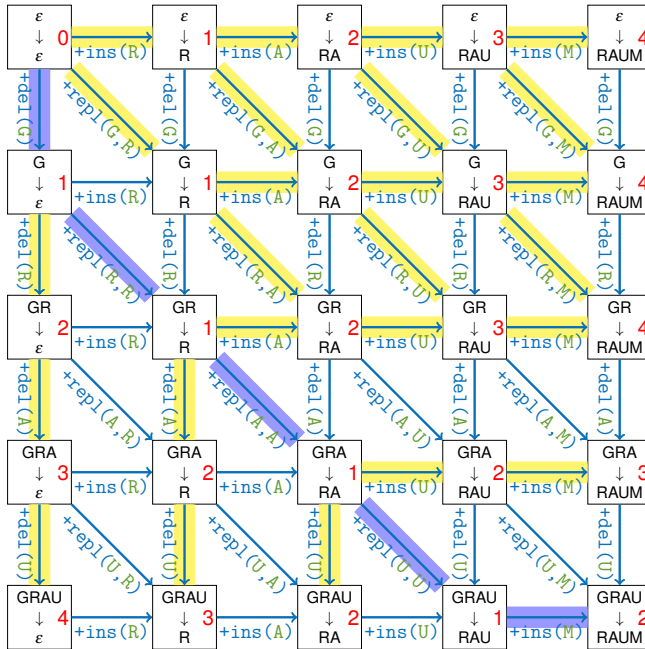
- We save at each recursion the most efficient previous entry (the **highlighted arrows** in our image)
- There can be **more than one** arrows to the three previous entries

## How to get the sequence of operations?

- We save at each recursion the most efficient previous entry (the **highlighted arrows** in our image)
- There can be **more than one** arrows to the three previous entries
- If we follow the highlighted path from  $(n, m)$  to  $(1, 1)$  we get the optimum operations to transform  $x$  into  $y$

## How to get the sequence of operations?

- We save at each recursion the most efficient previous entry (the **highlighted arrows** in our image)
- There can be **more than one** arrows to the three previous entries
- If we follow the highlighted path from  $(n, m)$  to  $(1, 1)$  we get the optimum operations to transform  $x$  into  $y$ 
  - If we can follow **more than one path** there exist more than one ideal **sequence**



## General principle:

## General principle:

- Recursive computation of ...
  - ... the same reoccurring partial problems
  - ... a limited number of partial problems

## General principle:

- Recursive computation of ...
  - ... the same reoccurring partial problems
  - ... a limited number of partial problems
- Computation of the solutions for all partial problems



## General principle:

- Recursive computation of ...
  - ... the same reoccurring partial problems
  - ... a limited number of partial problems
- Computation of the solutions for all partial problems
- In a order that unsolved partial problems consist of already solved partial problems

## General principle:

- Recursive computation of ...
  - ... the same reoccurring partial problems
  - ... a limited number of partial problems
- Computation of the solutions for all partial problems
- In a order that unsolved partial problems consist of already solved partial problems
- The “path” to our solution normally gets computed while searching the best solution
- Dijkstra algorithm is basically dynamic programming!



## **Additional applications:**

### Additional applications:

- *Edit distance*: global alignment with  $O(n^2)$  space and time consumption

### Additional applications:

- *Edit distance*: global alignment with  $O(n^2)$  space and time consumption
- But: Model for deletion unrealistic

### Additional applications:

- *Edit distance*: global alignment with  $O(n^2)$  space and time consumption
- But: Model for deletion unrealistic
  - In evolution larger pieces are more likely

### Additional applications:

- *Edit distance*: global alignment with  $O(n^2)$  space and time consumption
- But: Model for deletion unrealistic
  - In evolution larger pieces are more likely
  - delete operation: first gap expensive (e.g. 2), remaining are cheaper (e.g. 0.5)

$\bar{S}$	$\bar{A}$	$\bar{U}$	B	L	O	E	D
			B	L	O	E	D

### Additional applications:

- *Edit distance*: global alignment with  $O(n^2)$  space and time consumption
- But: Model for deletion unrealistic
  - In evolution larger pieces are more likely
  - delete operation: first gap expensive (e.g. 2), remaining are cheaper (e.g. 0.5)

			B	L	O	E	D
$\bar{S}$	$\bar{A}$	$\bar{U}$	B	L	O	E	D

- Solution in  $O(n^3)$  time or  $O(n^2)$  affine



$O(n^2)$  space consumption might be problematic:

**Hirschberg algorithm:**

$O(n^2)$  space consumption might be problematic:

### **Hirschberg algorithm:**

- Divide-and-conquer approach

$O(n^2)$  space consumption might be problematic:

### **Hirschberg algorithm:**

- Divide-and-conquer approach
- $O(n)$  space and  $O(n^2)$  time consumption

# Edit distance

## Additional applications (III)





- Sequencing:  $O(n^2)$  is too much



- Sequencing:  $O(n^2)$  is too much
- Index: suffixtree, suffixarray, burrow-wheeler-transform

## ■ General

- [CRL01] Thomas H. Cormen, Ronald L. Rivest, and Charles E. Leiserson.

**Introduction to Algorithms.**

MIT Press, Cambridge, Mass, 2001.

- [MS08] Kurt Mehlhorn and Peter Sanders.

Algorithms and data structures, 2008.

<https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf>.

## ■ **Dynamic programming**

[Wik] [Dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming)

`https:`

`//en.wikipedia.org/wiki/Dynamic_programming`

## ■ **Edit distance**

[Wik] [Levenshtein distance](https://en.wikipedia.org/wiki/Levenshtein_distance)

`https:`

`//en.wikipedia.org/wiki/Levenshtein_distance`