Algorithms and Data Structures
Levenshtein distance, Dynamic programming

Prof. Dr. Rolf Backofen
Bioinformatics Group / Department of Computer Science
Algorithms and Data Structures, January 2019
Structure

Introduction

Edit distance
Introduction

Edit distance
Introduction

Edit distance:
Introduction

Edit distance:
- Measurement for similarity of two words / strings
Introduction

Edit distance:

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

Edit distance:
- Measurement for similarity of two words / strings
- Algorithm for efficient calculation
- General principle: dynamic programming
Introduction
Motivation: Error tolerant string comparison

BioInfSearch

ejafjalajökuk
eyjafjallajökull
eyjafjallajökull movie
eyjafjallajälull trailer

Wikipedia.org:
"Der Eyjafjallajökull ([ˈɛɪjaˌfjatlajœktl̥] [3], auf Deutsch Eyjafjöll-Gletscher, ist der sechstgrößte Gletscher Islands.

Er liegt an der äußersten Südküste, westlich des Gletschers Mýrdalsjökull in der Gemeinde Rangárþing eystra, die größte Höhe beträgt 1651 m. Unter dem Gletscher befindet sich der Vulkan Eyjafjöll mit eigener Magma kammer, 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 Bremerhaven
  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 Bremerhaven
  - Hein Doof 27478 Cuxhaven

- Product search:
  - memory stik

- Websearch:
  - eyjaföllajaküll
  - universität 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
  - universität verien 2017

- Bioinformatics: Similarity of DNA-sequences
Example: Bioinformatics DNA-matching

Search of similar proteins:
Search of similar proteins:

- BLAST (Basic Local Alignment Search Tool)
Search of similar proteins:

- **BLAST (Basic Local Alignment Search Tool)**
- Alignment $\hat{=} \text{Edit distance}$
Introduction
Example: Bioinformatics DNA-matching

Search of similar proteins:

- **BLAST** *(Basic Local Alignment Search Tool)*
- Alignment ≈ Edit distance
- Changed life-science completely
Introduction

Example: Bioinformatics DNA-matching

Search of similar proteins:

- **BLAST** (Basic Local Alignment Search Tool)
- Alignment \( \hat{=} \) Edit distance
- Changed life-science completely
- Cited 63437 times on Google Scholar (Sep. 2017)
Structure

Introduction

Edit distance
Definition of edit distance: \textit{(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
↓
BOOF

replace(1, B)

BLOED
```
Edit distance

Example

1 2 3 4 5
DOOF
↓
BOOF
↓
BLOF

replace(1, B)

replace(2, L)

BLOED
Edit distance

Example

1 2 3 4 5
DOOF
↓
BOOF
↓
BLOF
↓
BLOEF
BLOED
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)
**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
↓ replace(1, B) 1 2 3 4 5
BOOF
↓ replace(2, L) BLOED
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)

1 2 3 4 5
BLOED

replace(2, L)

insert(4, E)

replace(5, D)

DOOF

ED=4

ED = 4
Edit distance

Example

\[
\begin{array}{c}
\text{1 2 3 4 5} \\
\text{DOOF} \\
\downarrow \\
\text{BOOF} \\
\downarrow \\
\text{BLOF} \\
\downarrow \\
\text{BLOEF} \\
\downarrow \\
\text{BLOED} \\
\end{array}
\]

\[\text{replace(1, B)}\]
\[\text{replace(2, L)}\]
\[\text{insert(4, E)}\]
\[\text{replace(5, D)}\]

\[
\begin{array}{c}
\text{1 2 3 4 5} \\
\text{BLOED} \\
\downarrow \\
\text{BLOEF} \\
\downarrow \\
\text{DOOF} \\
\end{array}
\]

\[\text{ED}=4\]
**Edit distance**

**Example**

```
12345
DOOF
↓
BOOF
↓
BLOF
↓
BLOEF
↓
BLOED

ED=4
```

```
12345
BLOED
↓
BLOEF
↓
delete(4)

ED=4
```
Edit distance

Example

1 2 3 4 5
DOOF
↓
BOOF
↓
BLOF
↓
BLOEF
↓
BLOED

1 2 3 4 5
BLOED
↓
BLOEF
↓
BLOF
↓
BOOF

replace(1, B)
replace(2, L)
insert(4, E)
replace(5, D)

replace(5, F)
delete(4)
replace(2, 0)

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
BLOED
↓
BLOEF
↓
BLOF
↓
BOOF
↓
DOOF

replace(5, F)

delete(4)

replace(2, O)

replace(1, D)
```
Edit distance

Example

\[
\begin{array}{c}
12345 \\
DOOF \\
\downarrow \\
BOOF \\
\downarrow \\
BLOF \\
\downarrow \\
BLOEF \\
\downarrow \\
BLOED \\
\hline
ED=4
\end{array}
\]

\[
\begin{array}{c}
12345 \\
BLOED \\
\downarrow \\
BLOEF \\
\downarrow \\
BLOF \\
\downarrow \\
BOOF \\
\hline
ED=4
\end{array}
\]

- replace(1, B)
- replace(2, L)
- insert(4, E)
- replace(5, D)
- replace(5, F)
- delete(4)
- replace(2, 0)
- replace(1, D)
Edit distance

Notation:
Edit distance

Notation:
- $\varepsilon$ is the empty string
Edit distance

Notation:

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

Notation:

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

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 $1 \leq i \leq j \leq |x|$
Edit distance

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 $1 \leq i \leq j \leq |x|$
Edit distance

Trivial facts:
Edit distance

Trivial facts:

- $\text{ED}(x, y) = \text{ED}(y, x)$
Edit distance

Trivial facts:

- \( ED(x, y) = ED(y, x) \)
- \( ED(x, \varepsilon) = |x| \)
Edit distance

Trivial facts:

- ED(x, y) = ED(y, x)
- ED(x, ε) = |x|
- ED(x, y) ≥ abs(|x| − |y|)
Edit distance

**Trivial facts:**

- $ED(x, y) = ED(y, x)$
- $ED(x, \varepsilon) = |x|$
- $ED(x, y) \geq \text{abs}(|x| - |y|)$
  \[
  \text{abs}(x) = \begin{cases} 
  x & \text{if } x \geq 0 \\
  -x & \text{else}
  \end{cases}
  \]
- $ED(x, y) \leq ED(x[1..n-1], y[1..m-1]) + 1$
  \[
  n = |x|, \ m = |y|
  \]
Solutions based on examples:

From VERIEN to FERIEN?

From MEXIKO to AMERIKA?

From AAEBEAABEAREEAEBA 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(GRAU, RAUM) = 2 but ED(GR, RA) + ED(AU, UM) = 4

Finding "smaller" sub problems?
Let’s try it!
Solutions based on examples:

- From VERIEN to FERIEN?

Searching biggest substrings can yield the solution but doesn’t have to.

Recursive approach:

Dividing in two halves? Not a good idea:

$$ED(GRAU, RAUM) = 2$$ but $$ED(GR, RA) + ED(AU, UM) = 4$$

Finding “smaller” sub problems?

Let’s try it!
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEEAEEBA to RBEAAEEBAAAEBBBAEAE?
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEAEBA to RBEAAEBAAABBAEAEE?
- Searching biggest substrings can yield the solution but doesn’t have to
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
- From AAEBEAABEAREEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn’t have to

Recursive approach:
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
- From AAEBAABEAREEAEBA to RBEAAEEBAAAEBBAEAE?
- Searching biggest substrings can yield the solution but doesn’t have to

Recursive approach:

- Dividing in two halves? Not a good idea:

  \[ ED(GRAU, RAUM) = 2 \quad \text{but} \quad ED(GR, RA) + ED(AU, UM) = 4 \]
Solutions based on examples:

- From VERIEN to FERIEN?
- From MEXIKO to AMERIKA?
- From AAEBEAAABEABEAREEAEBAAEBBAEAE to RBEAAEEBAEEBBAEAE?
- Searching biggest substrings can yield the solution but doesn’t have to

Recursive approach:

- Dividing in two halves? Not a good idea:
  \[ \text{ED}(\text{GRAU}, \text{RAUM}) = 2 \quad \text{but} \quad \text{ED}(\text{GR}, \text{RA}) + \text{ED}(\text{AU}, \text{UM}) = 4 \]
- Finding “smaller” sub problems?
  Let’s try it!
Edit distance

Terminology:
Edit distance

**Terminology:**
- Let $x, y$ be two strings
Edit distance

Terminology:

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

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
↓
BOOF
↓
BLOOF
↓
BLOEF
↓
BLOED

1 2 3 4 5 6 7
SAUDOOOF
↓
delete(1)
AUDOOOF
↓
delete(1)
UDOOF
↓
delete(1)
DOOF
↓
insert(4, O)
DOOOF
↓
insert(4, 0)
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
↓
BOOF
↓
BLOOF
↓
BLOEF
↓
BLOED
```

```
1 2 3 4 5 6 7
SAUDOOF
↓
AUDOOF
↓
UDOOF
↓
DOOF
↓
DOOOF
```

```
1 2 3 4 5 6 7
SAUDOOF
↓
AUDOOF
↓
UDOOF
↓
DOOF
↓
DOOOF
```

```
1 2 3 4 5 6 7
SAUDOOF
↓
AUDOOF
↓
UDOOF
↓
DOOF
↓
DOOOF
```

```
1 2 3 4 5 6 7
SAUDOOF
↓
AUDOOF
↓
UDOOF
↓
DOOF
↓
DOOOF
```

```
1 2 3 4 5 6 7
SAUDOOF
↓
AUDOOF
↓
UDOOF
↓
DOOF
↓
DOOOF
```
Edit distance

Terminology:

For any $x$ and $y$ with $k = ED(x, y)$ 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 $k = ED(x, y)$ 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)
Edit distance

Terminology:

- **Lemma**: For any $x$ and $y$ with $k = ED(x, y)$ 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)
Consider the last operation:
Consider the last operation:

- Solve **blue** part recursively
Consider the last operation:

- Solve **blue** part recursively

```
DOOF  DOOF  DOOF
↓↓↓↓↓↓  ↓↓↓↓↓  ↓↓↓↓↓
BLOE  BLOEDF  BLOEF
  ↓insert  ↓delete  ↓replace
BLOED  BLOED  BLOED
```

**Figure:** Case 1a  **Figure:** Case 1b  **Figure:** Case 1c
Consider the last operation:
Consider the last operation:

- Solve blue part recursively
Consider the last operation:

- Solve blue part recursively

\[
\begin{align*}
\text{WINTER} & \quad \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
SOMMER & \quad \downarrow \text{nothing}
\end{align*}
\]

Display of solution:

- Alignment
- Example:

```
  _ _ _ B L O E D
S A U B L O E D
```

**Figure:** Case 2
Dynamic programming:

Dynamic programming:

- 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)
Dynamic programming:
- Instances of Bellman’s principle of optimality:

Optimal solutions consist of optimal partial solutions.

Always solvable through dynamic programming (Caching of optimal partial solutions).

Figure: Richard Bellman (1920 - 1984)
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

- Optimal solutions consist of optimal partial solutions

Figure: Richard Bellman (1920 - 1984)
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

Figure: Richard Bellman (1920 - 1984)
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

Figure: Richard Bellman (1920 - 1984)
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:

We consider the last operation $\sigma_k, \ldots, \sigma_{k-1}$:

$$x \rightarrow z \quad \text{and} \quad \sigma_k: z \rightarrow y$$

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:
- We consider the last operation $\sigma_k$
Case analysis:

- We consider the last operation $\sigma_k$

  - $\sigma_1, \ldots, \sigma_{k-1}: x \rightarrow z$ and $\sigma_k: z \rightarrow y$

Example:

$$x = \text{DOOF}, \quad z = \text{SAUBLOEF}, \quad y = \text{SAUBLOED}$$
Case analysis:

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

\( \sigma_k \) does something at the outer end:

Case 1a:

\( \sigma_k = \text{insert}(m', y[m]) \) [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 \( \sigma_1, \ldots, \sigma_{k-1} : x[1..n-1] \rightarrow y[1..m-1] \) and \( x[n] = y[m] \)
Case analysis:

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

  - Case 1a: $\sigma_k = \text{insert}(m', y[m])$ [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 $\sigma_1, ..., \sigma_{k-1}$: $x[1..n-1] \rightarrow y[1..m-1]$ and $x[n] = y[m]$. 
Edit distance

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 2: $\sigma_k$ does nothing at the outer end:
  Then $z[m'] = y[m]$ and $x[n'] = z[m']$ and with that $\sigma_1, \ldots, \sigma_{k-1}$: $x[1..n-1] \rightarrow y[1..m-1]$ and $x[n] = y[m]$
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 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 $m' = m$]
Edit distance

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: $\sigma_k = \text{insert}(m' + 1, y[m])$ \hspace{1.5cm} [then $m' = m - 1$]
  - Case 1b: $\sigma_k = \text{delete}(m')$ \hspace{1.5cm} [then $m' = m + 1$]
  - Case 1c: $\sigma_k = \text{replace}(m', y[m])$ \hspace{1.5cm} [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
    $\sigma_1, \ldots, \sigma_{k-1}: x[1..n-1] \rightarrow y[1..m-1]$ and $x[n] = y[m]$
Edit distance

Case analysis:

---

January 2019
Prof. Dr. Rolf Backofen – Bioinformatics - University Freiburg - Germany
Edit distance

Case analysis:

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

Case 1b (delete):

Case 1c (replace):

Case 2 (nothing):

This results in the recursive formula:

For $|x| > 0$ and $|y| > 0$ is

$\text{ED}(x, y)$ the minimum of

$\text{ED}(x, y[1..m-1]) + 1$ and

$\text{ED}(x[1..n-1], y) + 1$ if $x[n] \neq y[m]$

$\text{ED}(x[1..n-1], y[1..m-1]) + 1$

For $|x| = 0$ is

$\text{ED}(x, y) = |y|$

For $|y| = 0$ is

$\text{ED}(x, y) = |x|$
Edit distance

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

This results in the recursive formula:
For \(|x| > 0\) and \(|y| > 0\) is
\[
ED(x, y) = \min \left( ED(x, y[1..m-1]) + 1, ED(x[1..n-1], y) + 1, ED(x[1..n-1], y[1..m-1]) + 1 \right)
\]
if \(x_n \neq y_m\)
\[
ED(x, y[1..m-1]) + 0 \quad \text{if} \quad x_n = y_m
\]

For \(|x| = 0\) is
\[
ED(x, y) = |y|
\]

For \(|y| = 0\) is
\[
ED(x, y) = |x|
\]
Case analysis:

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

Case analysis:

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

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

This results in the recursive formula:
Case analysis:

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

This results in the recursive formula:

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

Case analysis:

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

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

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

- For \( |x| = 0 \) is \( ED(x, y) = |y| \)

- For \( |y| = 0 \) is \( ED(x, y) = |x| \)
Case analysis:

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

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 $x[n] \neq y[m]$
Edit distance

Case analysis:

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

This results in the recursive formula:

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

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

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 $ED(x,y) = |y|$
Edit distance

Case analysis:

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

This results in the recursive formula:

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

- For \(|x| = 0\) is \( \text{ED}(x, y) = |y| \)
- For \(|y| = 0\) is \( \text{ED}(x, y) = |x| \)
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:

The algorithm results in the following recursive formula:

$$T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1 \geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1) = 3 \cdot T(n-1, m-1)$$

This results in:

$$T(n, n) \geq 3^n$$

⇒ The runtime is at least exponential.
Recursive program:

The algorithm results in the following recursive formula:

$$T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1$$

$$\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1)$$

$$= 3 \cdot T(n-1, m-1)$$
Recursive program:

- The algorithm results in the following recursive formula:

\[
T(n, m) = T(n - 1, m) + T(n, m - 1) + T(n - 1, m - 1) + 1 \\
\geq T(n - 1, m - 1) + T(n - 1, m - 1) + T(n - 1, m - 1) \\
= 3 \cdot T(n - 1, m - 1)
\]

- This results in \( T(n, n) \geq 3^n \)
Recursive program:

- The algorithm results in the following recursive formula:

\[
T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1
\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1)
= 3 \cdot T(n-1, m-1)
\]

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

⇒ The runtime is at least \emph{exponential}
Dynamic programming:

We create a table with all possible combinations 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 $\Rightarrow \text{repl}(A, A)$. 
Edit distance

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:
Edit distance

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

⇒ repl(A, A)
Edit Distance

GRAU ↓ RAU

+ins(M)

GRAU ↓ RAUM
Fast algorithm:
We can determine the edit distance for all combination of partial strings from the top left to bottom right.
$\varepsilon \downarrow \varepsilon$

$\varepsilon \downarrow R$

$\varepsilon \downarrow RA$

$\varepsilon \downarrow RAU$

$\varepsilon \downarrow RAUM$

$G \downarrow \varepsilon$

$G \downarrow R$

$G \downarrow RA$

$G \downarrow RAU$

$G \downarrow RAUM$

$GR \downarrow \varepsilon$

$GR \downarrow R$

$GR \downarrow RA$

$GR \downarrow RAU$

$GR \downarrow RAUM$

$GRA \downarrow \varepsilon$

$GRA \downarrow R$

$GRA \downarrow RA$

$GRA \downarrow RAU$

$GRA \downarrow RAUM$

$GRAU \downarrow \varepsilon$

$GRAU \downarrow R$

$GRAU \downarrow RA$

$GRAU \downarrow RAU$

$GRAU \downarrow RAUM$
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.
Edit distance

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
Edit distance

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
Edit distance

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!
Edit distance
Additional applications (I)

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
Edit distance
Additional applications (I)

Additional applications:

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

```
  S A U B L O E D
  _ _ _ B L O E D
```

---

**Solution** in $O(n^3)$ time or $O(n^2)$ affine
Additional applications:

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

- But: Model for deletition unrealistic
  
  - In evolution larger pieces are more likely
  - *delete* operation: first gap expensive (e.g. 2), remaining are cheaper (e.g. 0.5)

    | S | A | U | B | L | O | E | D |
    |---|---|---|---|---|---|---|---|
    |   |   |   | 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)

Whole Shotgun Sequencing

ACGACT
GACCAC
TACCGA
AAGGCA

...ACGAACCT
GACCAC
GACCCTACCGA...
Edit distance

Additional applications (III)

Sequencing: $O(n^2)$ is too much
Edit distance
Additional applications (III)

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

General

Introduction to Algorithms. 

[MS08] Kurt Mehlhorn and Peter Sanders. 
Algorithms and data structures, 2008. 
Further Literature

- **Dynamic programming**
  - [Wiki] Dynamic programming

- **Edit distance**
  - [Wiki] Levenshtein distance
    - https://en.wikipedia.org/wiki/Levenshtein_distance