Algorithms and Datastructures

Levenshtein distance, Dynamic programming

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science

Algorithms and Datastructures, February 2018

Structure

Introduction

Edit distance

Structure

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

`ejafjatlajökuk` **Search!**

- eyjafjallajökull
- eyjafjallajökull movie
- eyjafjallajälull trailer

Ulrich Latzenhofer; CC BY-SA 2.0

Wikipedia.org:

"Der Eyjafjallajökull ([ˈeɪja,fjatla,jœːkʏtl̥])[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 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:

Introduction

Motivation

A lot of applications where similar string are searched:

- Duplicates in databases:

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

Introduction

Motivation

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

Introduction

Example: Bioinformtics DNA-matching

Search of similar proteins:

Search of similar proteins:

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

Introduction

Example: Bioinformtics DNA-matching

Search of similar proteins:

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

Introduction

Example: Bioinformtics DNA-matching

Search of similar proteins:

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

Introduction

Example: Bioinformtics DNA-matching

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)

Structure

Introduction

Edit distance

Edit distance

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

Edit distance

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

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

Edit distance

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

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

Edit distance

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

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

Edit distance

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

- Let x, y be two strings
- Edit distance $\mathrm{ED}(x, y)$ 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

B L OED

Edit distance

Example

```
12345
DOOF
  ↓     replace(1, B)
BOOF
```

BLOED

Edit distance

Example

$$
\begin{array}{ll}
\text{1 2 3 4 5} & \\
\text{DOOF} & \\
\quad\downarrow & \texttt{replace(1, B)} \\
\text{BOOF} & \\
\quad\downarrow & \texttt{replace(2, L)} \\
\text{BLOF} & \\
\end{array}
$$

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

$$
\begin{array}{ll}
\texttt{12345} & \\
\text{DOOF} & \\
\quad\downarrow & \texttt{replace(1, B)} \\
\text{BOOF} & \\
\quad\downarrow & \texttt{replace(2, L)} \\
\text{BLOF} & \\
\quad\downarrow & \texttt{insert(4, E)} \\
\text{BLOEF} & \\
\quad\downarrow & \texttt{replace(5, D)} \\
\text{BLOED} & \\
 & \underbrace{\hspace{4cm}}_{\text{ED}=4}
\end{array}
$$

Edit distance

Example

```
12345
DOOF
  ↓     replace(1, B)
BOOF
  ↓     replace(2, L)
BLOF
  ↓     insert(4, E)
BLOEF
  ↓     replace(5, D)
BLOED
```

$\underbrace{\hspace{6em}}_{\text{ED}=4}$

12345
BLOED

Edit distance

Example

1 2 3 4 5
DOOF

$\downarrow$ `replace(1, B)`

BOOF

$\downarrow$ `replace(2, L)`

BLOF

$\downarrow$ `insert(4, E)`

BLOEF

$\downarrow$ `replace(5, D)`

BLOED

$\underbrace{\texttt{replace(1, B)}\ \texttt{replace(2, L)}\ \texttt{insert(4, E)}\ \texttt{replace(5, D)}}_{\text{ED}=4}$

1 2 3 4 5
B LOED

DOOF

Edit distance

Example

1 2 3 4 5	
DOOF	
$\downarrow$	`replace(1, B)`
BOOF	
$\downarrow$	`replace(2, L)`
BLOF	
$\downarrow$	`insert(4, E)`
BLOEF	
$\downarrow$	`replace(5, D)`
BLOED	

$\underbrace{\hphantom{replace(5, D)}}_{\text{ED}=4}$

1 2 3 4 5	
BLOED	
$\downarrow$	`replace(5, F)`
BLOEF	

DOOF

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

```
1 2 3 4 5
B LOED
  ↓      replace(5, F)
B LOEF
  ↓      delete(4)
B LOF
```

DOOF

Edit distance

Example

```
1 2 3 4 5
D O O F
  ↓        replace(1, B)
B O O F
  ↓        replace(2, L)
B L O F
  ↓        insert(4, E)
B L O E F
  ↓        replace(5, D)
B L O E D
```

ED=4

```
1 2 3 4 5
B L O E D
  ↓        replace(5, F)
B L O E F
  ↓        delete(4)
B L O F
  ↓        replace(2, O)
B O O F

D O O F
```

Edit distance

Example

$$\begin{array}{ll}
1\,2\,3\,4\,5 & \\
\text{DOOF} & \\
\downarrow & \texttt{replace(1, B)} \\
\text{BOOF} & \\
\downarrow & \texttt{replace(2, L)} \\
\text{BLOF} & \\
\downarrow & \texttt{insert(4, E)} \\
\text{BLOEF} & \\
\downarrow & \texttt{replace(5, D)} \\
\text{BLOED} & \\
 & \underbrace{\qquad\qquad\qquad\qquad}_{\text{ED}=4}
\end{array}$$

$$\begin{array}{ll}
1\,2\,3\,4\,5 & \\
\text{B LOED} & \\
\downarrow & \texttt{replace(5, F)} \\
\text{B LOEF} & \\
\downarrow & \texttt{delete(4)} \\
\text{B LOF} & \\
\downarrow & \texttt{replace(2, O)} \\
\text{B OOF} & \\
\downarrow & \texttt{replace(1, D)} \\
\text{DOOF} &
\end{array}$$

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

1 2 3 4 5
BLOED
↓ `replace(5, F)`
BLOEF
↓ `delete(4)`
BLOF
↓ `replace(2, O)`
BOOF
↓ `replace(1, D)`
DOOF

ED=4

Edit distance

Notation:

Edit distance

Notation:

- ε is the empty string

Edit distance

Notation:

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

Edit distance

Notation:

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

Edit distance

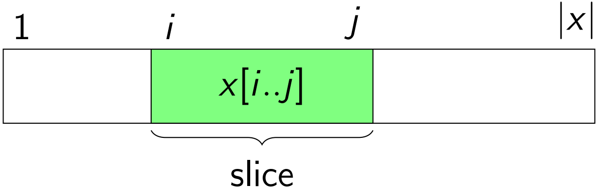
Notation:

- ε 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:

- ε 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:

- $\mathrm{ED}(x, y) = \mathrm{ED}(y, x)$

Edit distance

Trivial facts:

- $\mathrm{ED}(x, y) = \mathrm{ED}(y, x)$
- $\mathrm{ED}(x, \epsilon) = |x|$

Edit distance

Trivial facts:

- $\mathrm{ED}(x, y) = \mathrm{ED}(y, x)$
- $\mathrm{ED}(x, \epsilon) = |x|$
- $\mathrm{ED}(x, y) \geq \mathrm{abs}(|x| - |y|)$

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

Edit distance

Trivial facts:

- $\mathrm{ED}(x, y) = \mathrm{ED}(y, x)$
- $\mathrm{ED}(x, \epsilon) = |x|$
- $\mathrm{ED}(x, y) \geq \mathrm{abs}(|x| - |y|)$ $\qquad \mathrm{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{else} \end{cases}$
- $\mathrm{ED}(x, y) \leq \mathrm{ED}(x[1..n-1], y[1..m-1]) + 1 \;\; n = |x|, \;\; m = |y|$

Solutions based on examples:

Edit distance

Solving examples

Solutions based on examples:

- From `VERIEN` to `FERIEN`?

Edit distance

Solving examples

Solutions based on examples:

- From `VERIEN` to `FERIEN`?
- From `MEXIKO` to `AMERIKA`?

Edit distance

Solving examples

Solutions based on examples:

- From `VERIEN` to `FERIEN`?
- From `MEXIKO` to `AMERIKA`?
- From `AAEBEAABEAREEAEBA` to `RBEAAEEBAAAEBBAEAE`?

Edit distance

Solving examples

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

Edit distance

Solving examples

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:

Edit distance

Solving examples

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:

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

Edit distance

Solving examples

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:

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

- 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 $k = \mathrm{ED}(x, y)$ for $x \to y$ (transform x into y)
  (We do not know this sequence but we assume it exists)

Edit distance

Terminology:

Edit distance

Terminology:

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

Edit distance

Terminology:

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

1 2 3 4 5		1 2 3 4 5 6 7	
DOOF		SAUDOOF	
$\downarrow$	`replace(1, B)`	$\downarrow$	`delete(1)`
BOOF		AUDOOF	
$\downarrow$	`replace(2, L)`	$\downarrow$	`delete(1)`
BLOF		UDOOF	
$\downarrow$	insert(4, E)	$\downarrow$	`delete(1)`
BLOEF		DOOF	
$\downarrow$	`replace(5, D)`	$\downarrow$	`insert(4, O)`
BLOED		DOOOF	

Edit distance

Terminology:

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

1 2 3 4 5 DOOF		1 2 3 4 5 6 7 SAUDOOF	
$\downarrow$	`replace(1, B)`	$\downarrow$	`delete(1)`
BOOF		AUDOOF	
$\downarrow$	`replace(2, L)`	$\downarrow$	`delete(1)`
BLOF		UDOOF	
$\downarrow$	insert(4, E)	$\downarrow$	`delete(1)`
BLOEF		DOOF	
$\downarrow$	`replace(5, D)`	$\downarrow$	`insert(4, O)`
BLOED		DOOOF	

Edit distance

Terminology:

Edit distance

Terminology:

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

Edit distance

Terminology:

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

Edit distance

Recursive approach

Consider the last operation:

- Solve blue part recursively

Edit distance

Recursive approach

Consider the 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:

Edit distance

Recursive approach

Consider the last operation:

- Solve blue part recursively

Edit distance

Recursive approach

Consider the last operation:

- Solve blue part recursively

W I N T E R
↓ ↓ ↓ ↓ ↓ ↓
S OMMER
↓nothing
S OMMER

Figure: Case 2

Display of solution:

- Alignment
- Example:

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

Dynamic programming:

Dynamic programming:

- Instances of Bellman's principle of optimality:

Dynamic programming:

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

Edit distance

Dynamic programming

Dynamic programming:

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

Edit distance

Dynamic programming

Dynamic programming:

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

Figure: Richard Bellman (1920 - 1984)

Edit distance

Dynamic programming

Dynamic programming:

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

Figure: Richard Bellman (1920 - 1984)

Edit distance

Dynamic programming

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

Edit distance

Dynamic programming

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

Dynamic programming

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

Edit distance

Dynamic programming

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

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

Edit distance

Case analysis:

Edit distance

Case analysis:

- We consider the last operation σ_k

Edit distance

Case analysis:

- We consider the last operation σ_k
  - $\sigma_1, \ldots, \sigma_{k-1}$: $x \to z$ and σ_k: $z \to y$
    Example:

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

Edit distance

Case analysis:

- We consider the last operation σ_k
  - $\sigma_1, \ldots, \sigma_{k-1}$: $x \to z$ and σ_k: $z \to y$
    Example:

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

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

Edit distance

Case analysis:

- We consider the last operation σ_k
  - $\sigma_1, \ldots, \sigma_{k-1}$: $x \to z$ and σ_k: $z \to y$
    Example:

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

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

Edit distance

Case analysis:

Edit distance

Case analysis:

- Case 1: σ_k does something at the outer end:

Edit distance

Case analysis:

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

Edit distance

Case analysis:

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

Edit distance

Case analysis:

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

Edit distance

Case analysis:

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

Edit distance

Case analysis:

- Case 1: σ_k does something at the outer end:
  - Case 1a: $\sigma_k = \texttt{insert}(m' + 1, y[m])$ [then $m' = m - 1$]
  - Case 1b: $\sigma_k = \texttt{delete}(m')$ [then $m' = m + 1$]
  - Case 1c: $\sigma_k = \texttt{replace}(m', y[m])$ [then $m' = m$]
- Case 2: σ_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:

Edit distance

Case analysis:

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

Edit distance

Case analysis:

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

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`): $\sigma_1, \ldots, \sigma_{k-1}$: $x[1..n-1] \rightarrow y[1..m-1]$

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`): $\sigma_1, \ldots, \sigma_k$: $x[1..n-1] \rightarrow y[1..m-1]$

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:

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 $\mathrm{ED}(x, y)$ 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 $\mathrm{ED}(x, y)$ the minimum of
  - $\mathrm{ED}(x \quad , y[1..m-1]) + 1$ and

Edit distance

Case analysis:

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

This results in the recursive formula:

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

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 $\mathrm{ED}(x, y)$ the minimum of
  - $\mathrm{ED}(x, y[1..m-1]) + 1$ and
  - $\mathrm{ED}(x[1..n-1], y) + 1$ and
  - $\mathrm{ED}(x[1..n-1], y[1..m-1]) + 1$ 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 $\mathrm{ED}(x, y)$ the minimum of
  - $\mathrm{ED}(x, y[1..m-1]) + 1$ and
  - $\mathrm{ED}(x[1..n-1], y) + 1$ and
  - $\mathrm{ED}(x[1..n-1], y[1..m-1]) + 1$ if $x[n] \neq y[m]$
  - $\mathrm{ED}(x[1..n-1], y[1..m-1]) + 0$ if $x[n] = y[m]$

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

Edit distance

Case analysis:

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

This results in the recursive formula:

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

Edit distance

Implementation - Python

```python
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:

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

Edit distance

Runtime analysis

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

$\Rightarrow$ The runtime is at least exponential

Edit distance

Dynamic programming:

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 $O(n \cdot m)$

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:

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)

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)
- We also display the replaced character on a `replace` operation to visualize operations without costs
  $\Rightarrow$ `repl(A, A)`

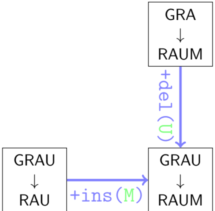
Edit Distance

GRAU
$\downarrow$
RAUM

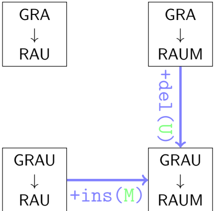
Edit Distance

$$\boxed{\begin{array}{c}\text{GRAU}\\ \downarrow\\ \text{RAU}\end{array}} \xrightarrow[\texttt{+ins(M)}]{} \boxed{\begin{array}{c}\text{GRAU}\\ \downarrow\\ \text{RAUM}\end{array}}$$

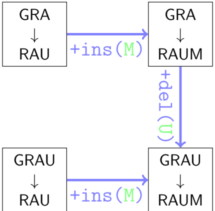
Edit Distance



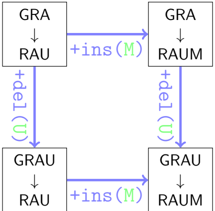
Edit Distance



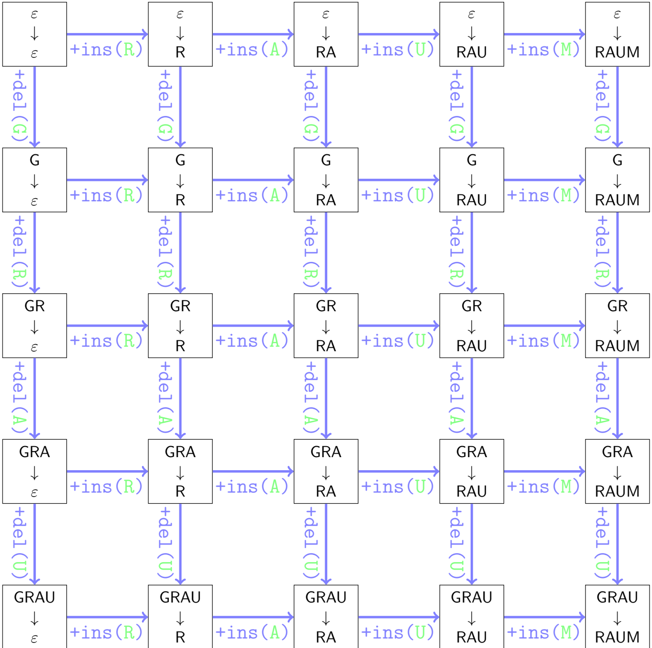
Edit Distance



Edit Distance



Edit Distance

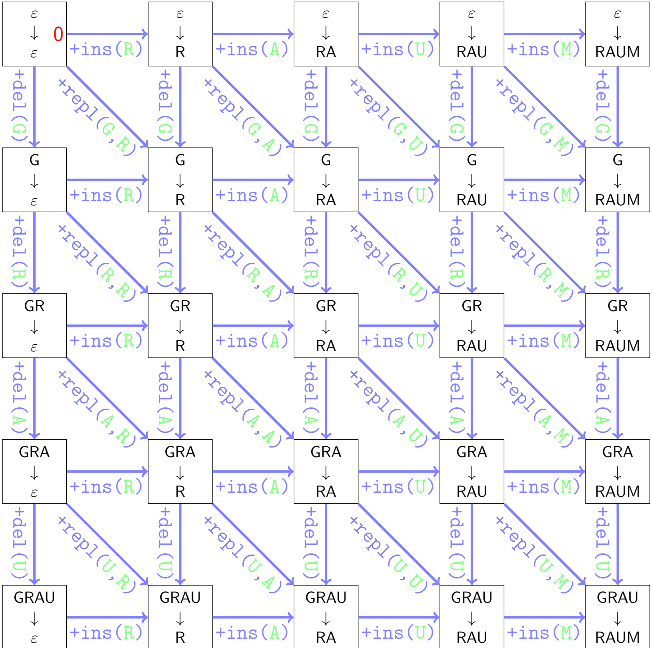


Edit distance

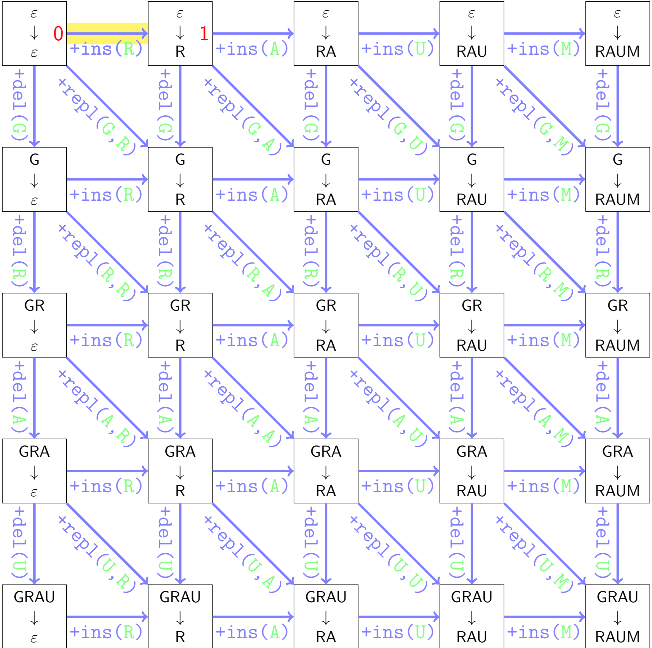
Fast algorithm

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

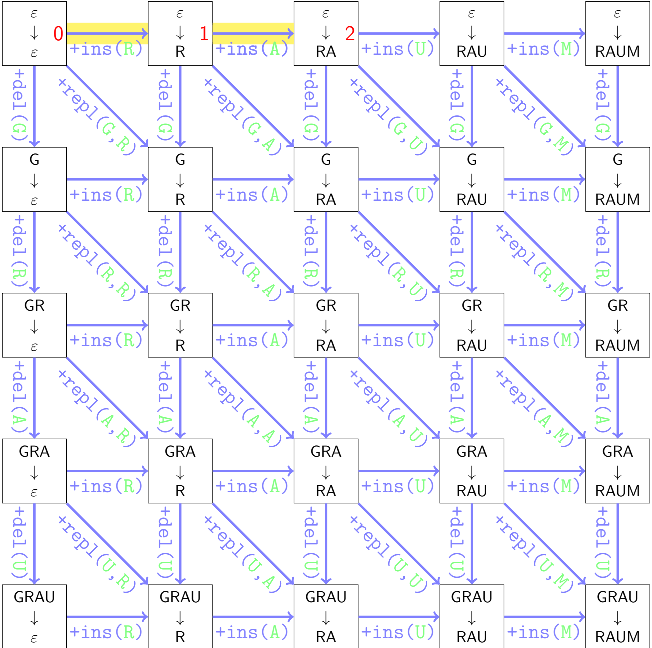
Edit Distance



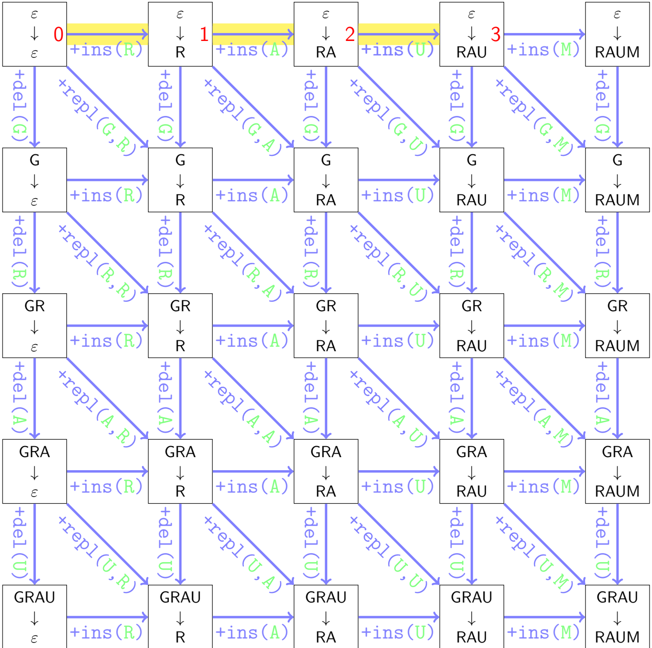
Edit Distance



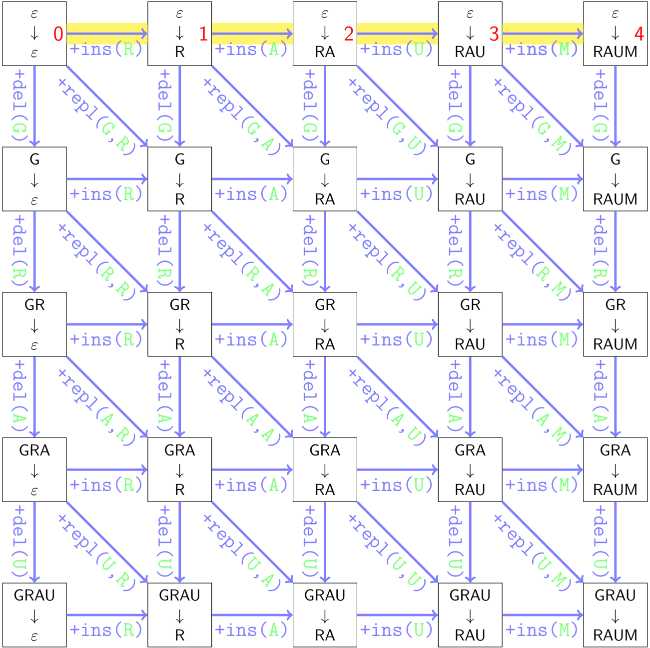
Edit Distance



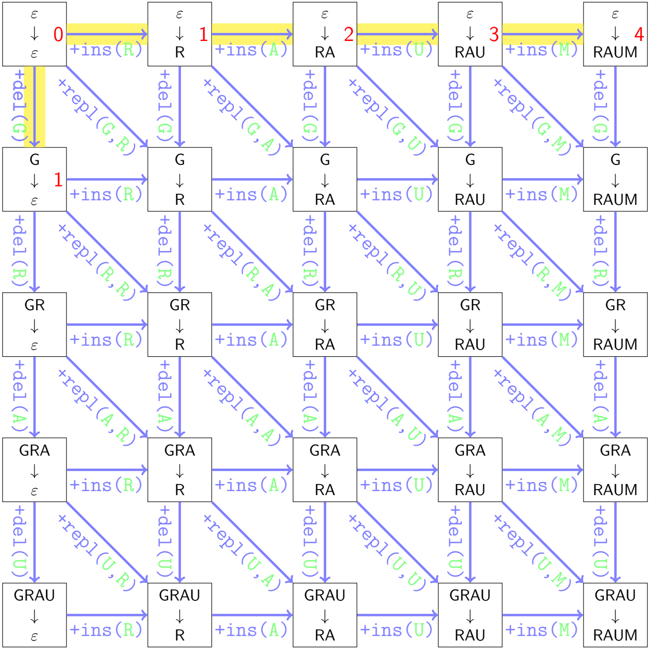
Edit Distance



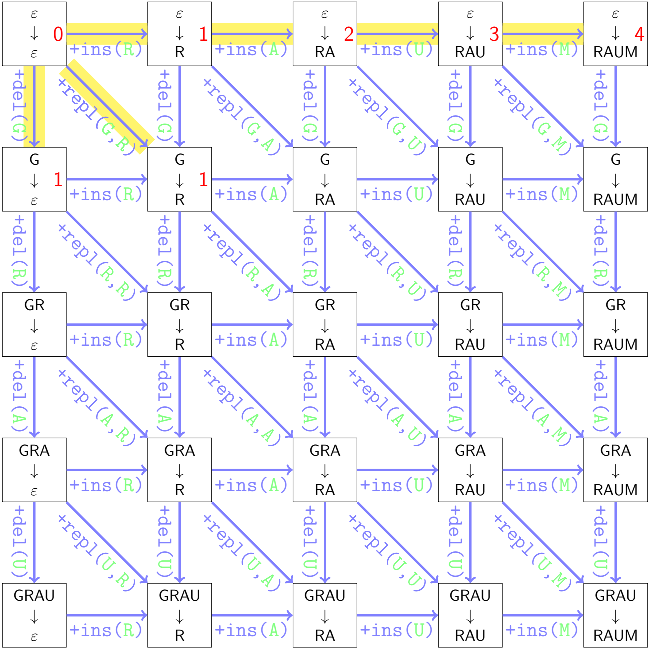
Edit Distance



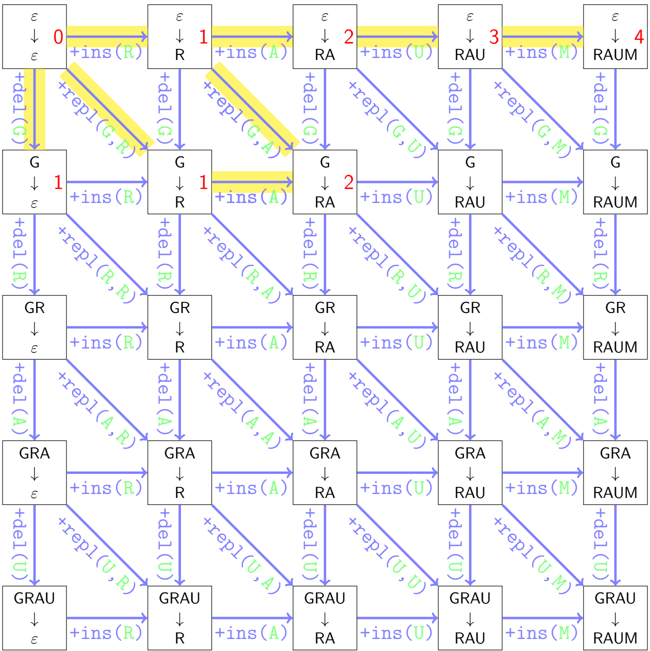
Edit Distance



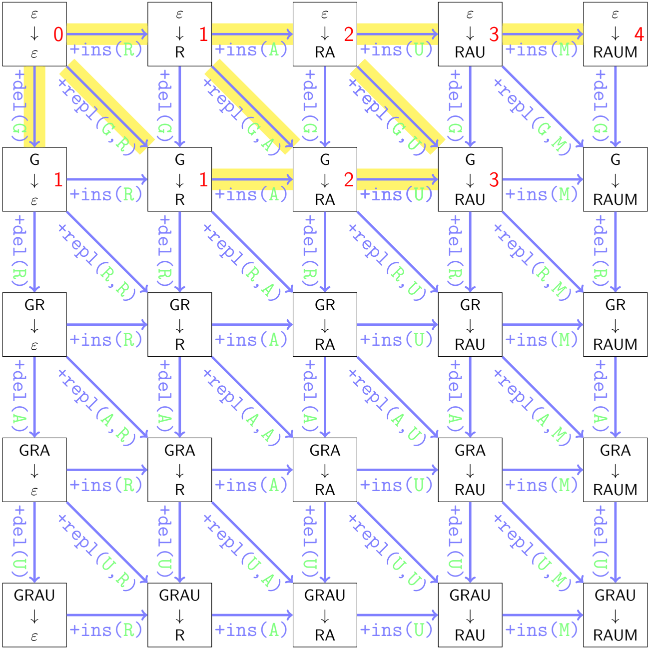
Edit Distance



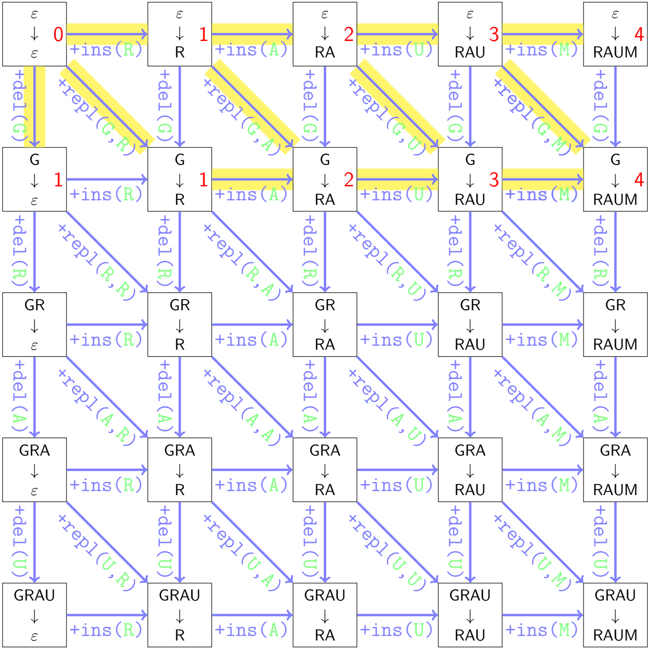
Edit Distance



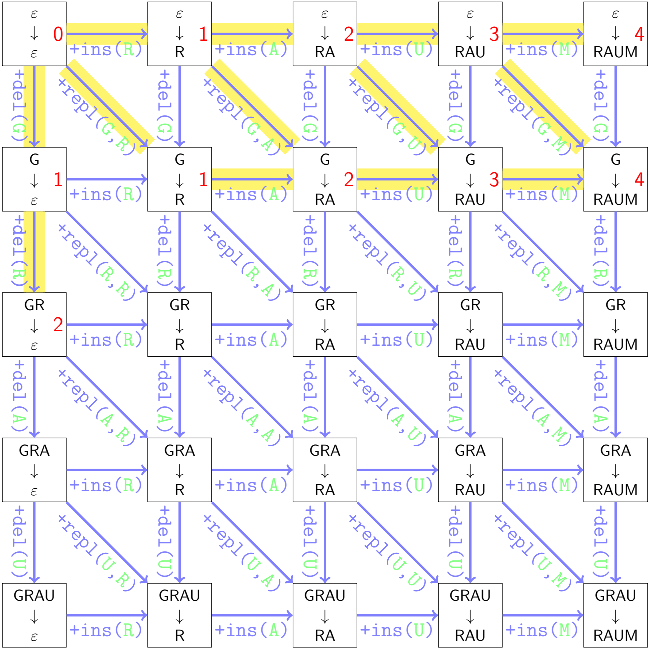
Edit Distance



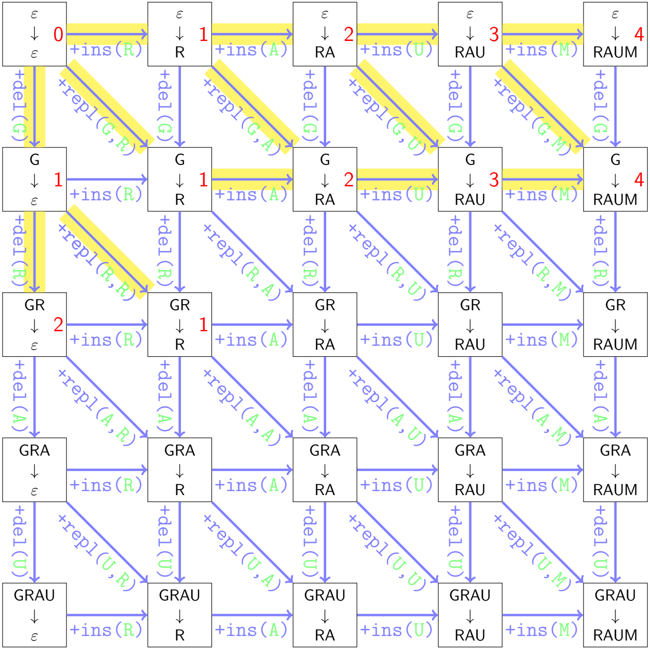
Edit Distance



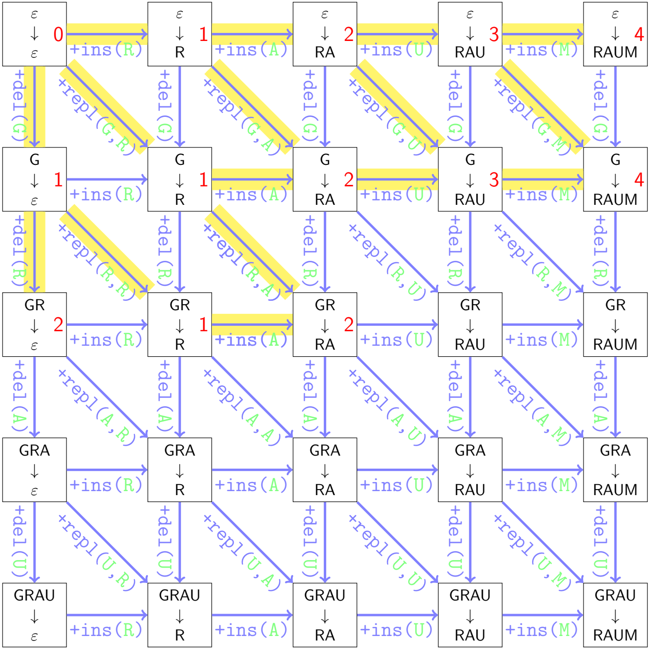
Edit Distance



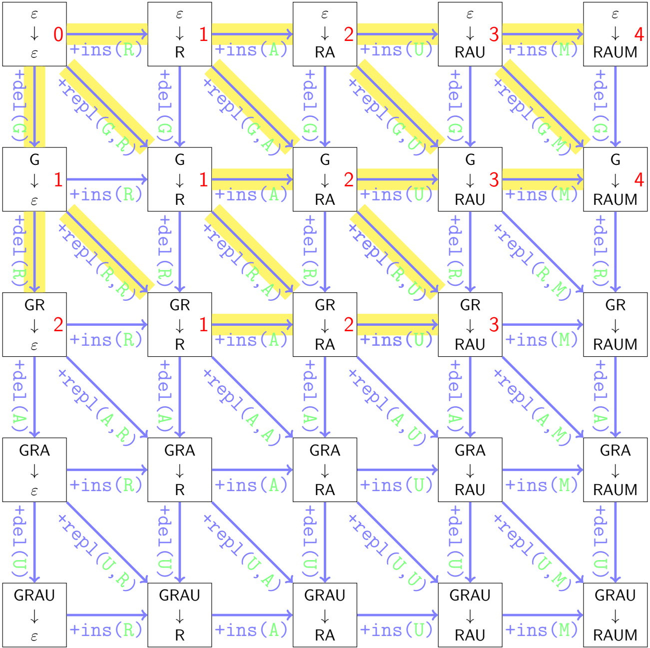
Edit Distance



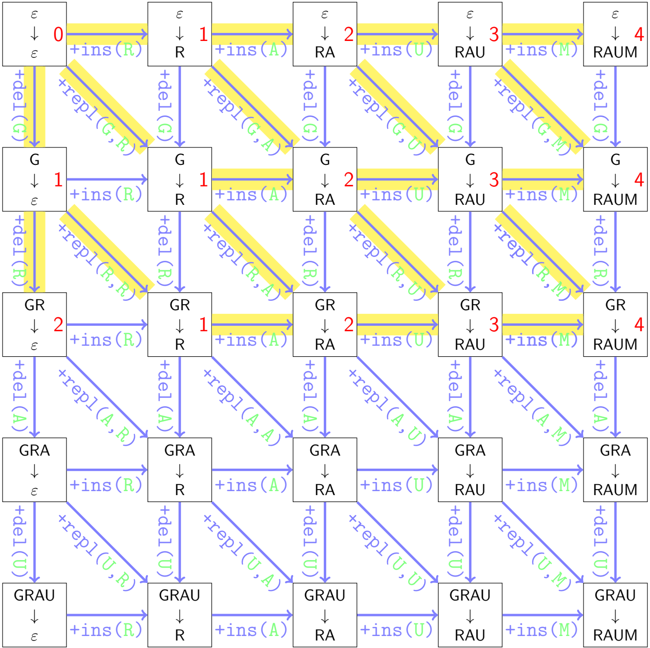
Edit Distance



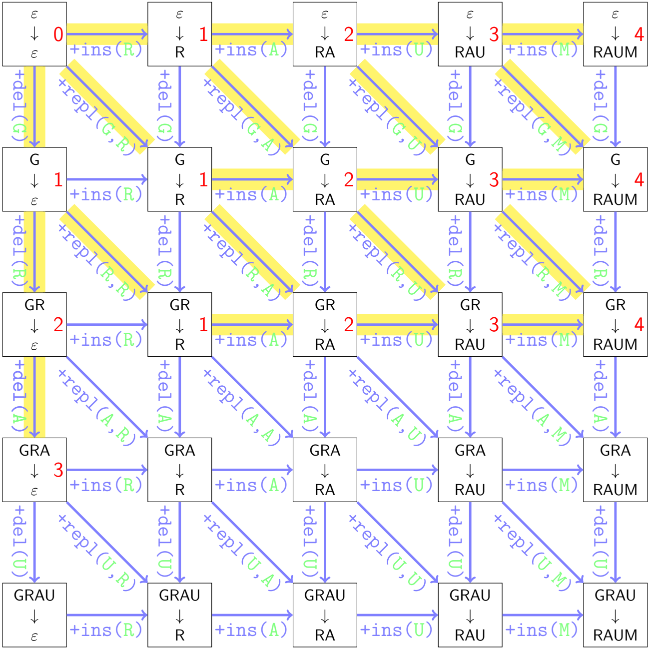
Edit Distance



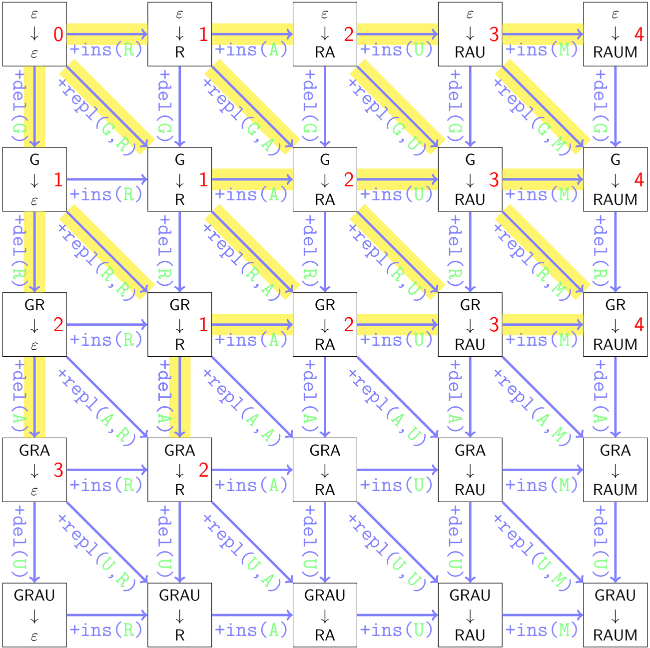
Edit Distance



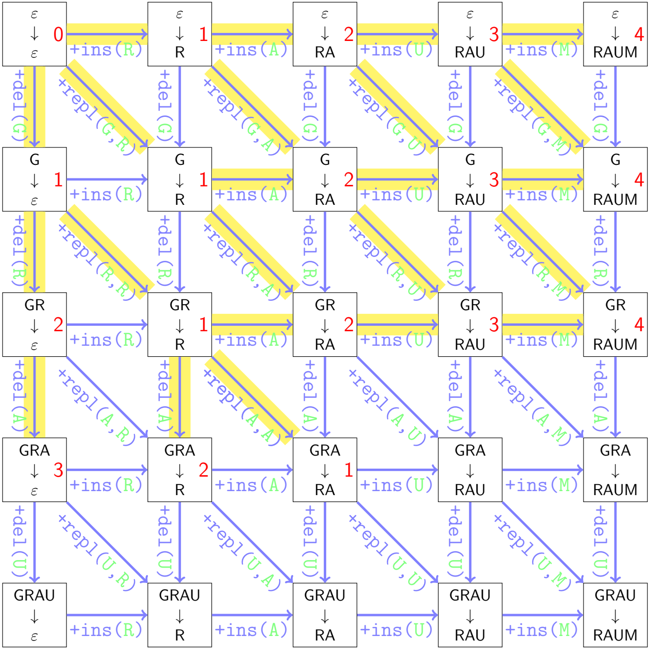
Edit Distance



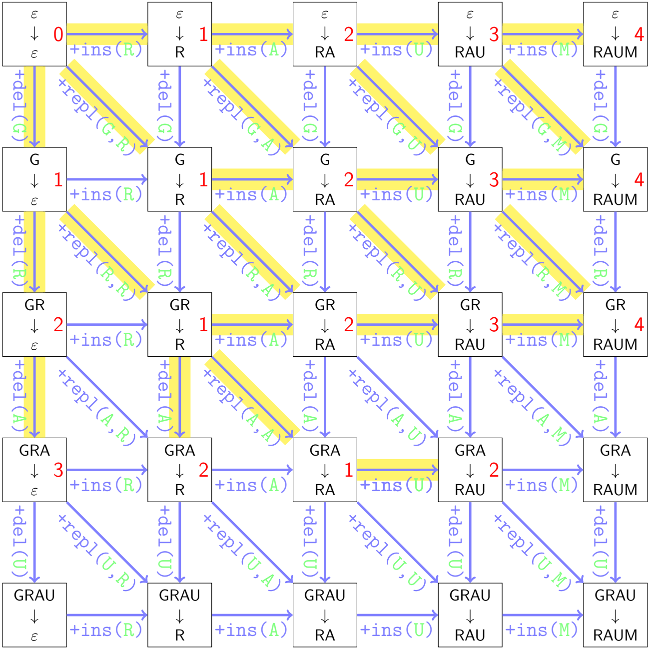
Edit Distance



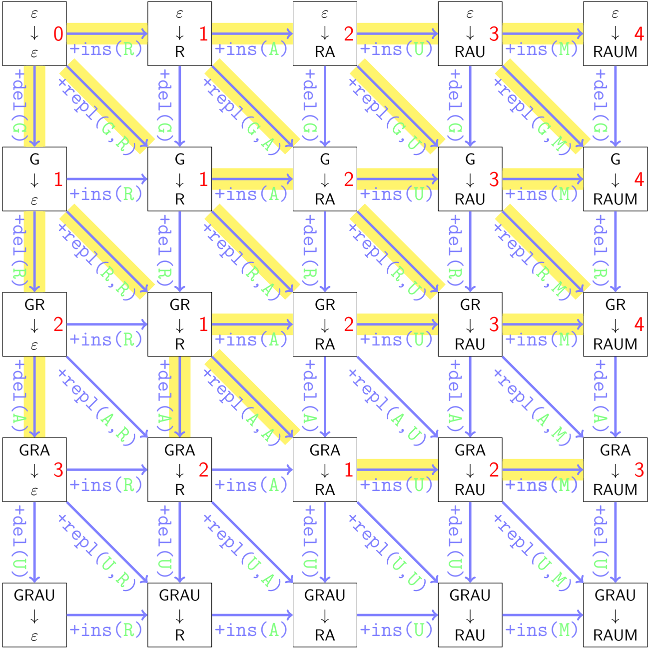
Edit Distance



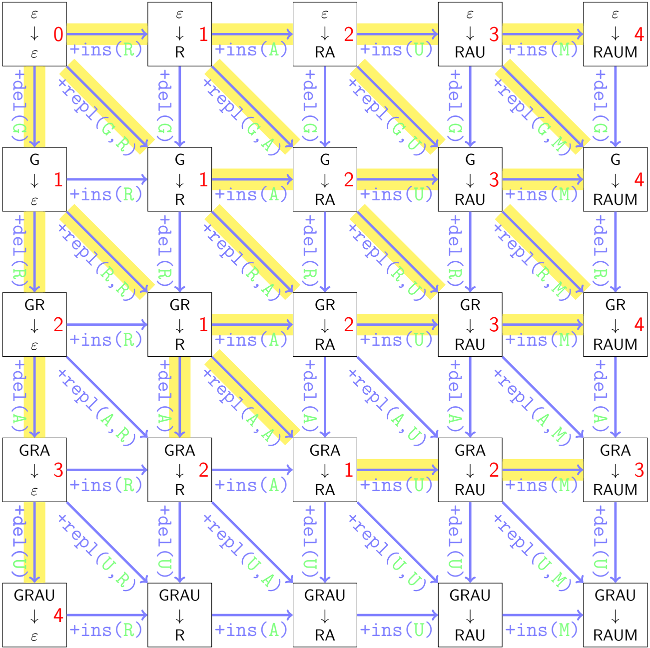
Edit Distance



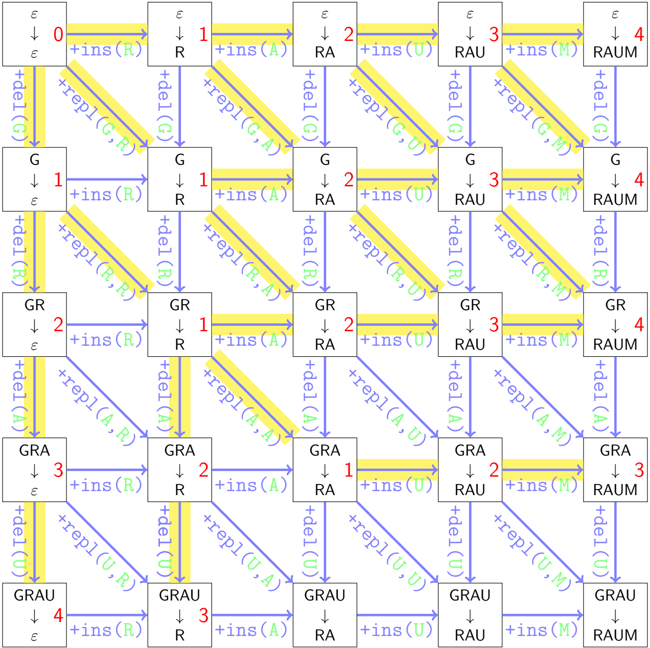
Edit Distance



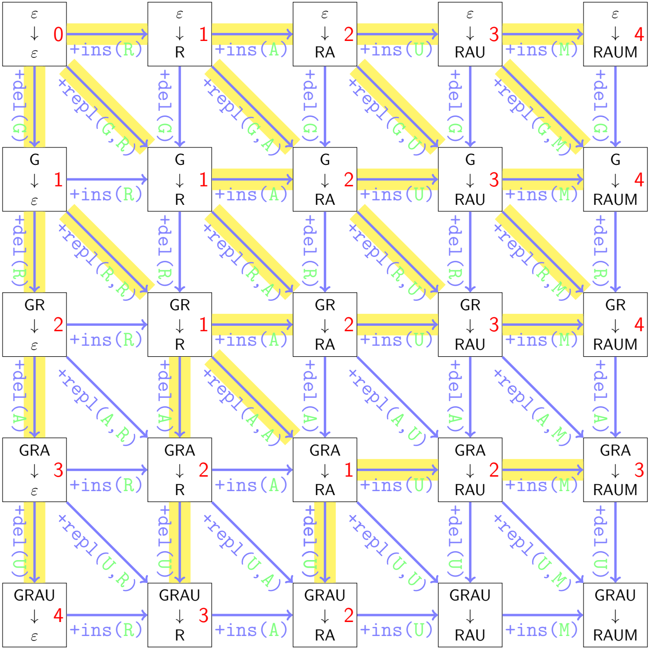
Edit Distance



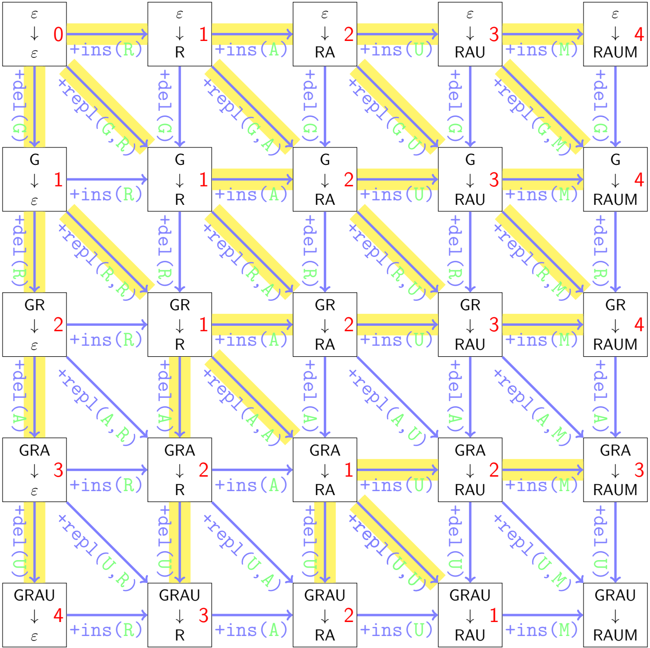
Edit Distance



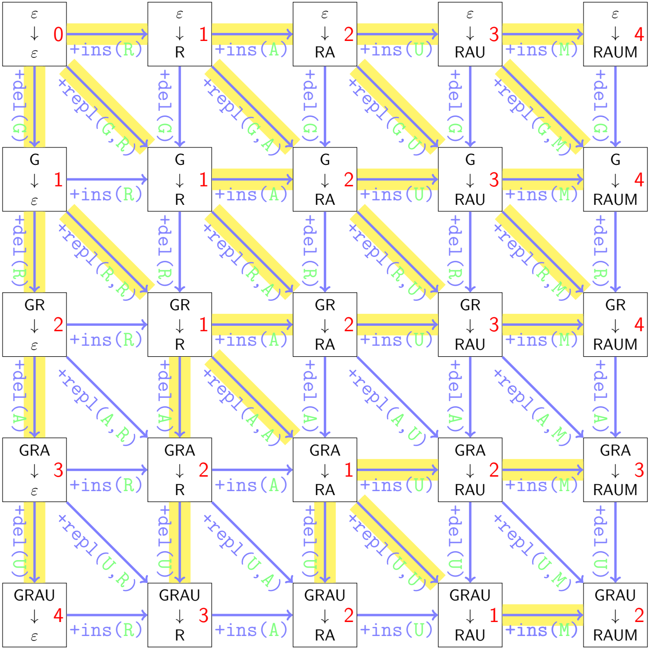
Edit Distance



Edit Distance



Edit Distance



Edit distance

How to get the sequence of operations?

Edit distance

How to get the sequence of operations?

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

Edit distance

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

Edit distance

How to get the sequence of operations?

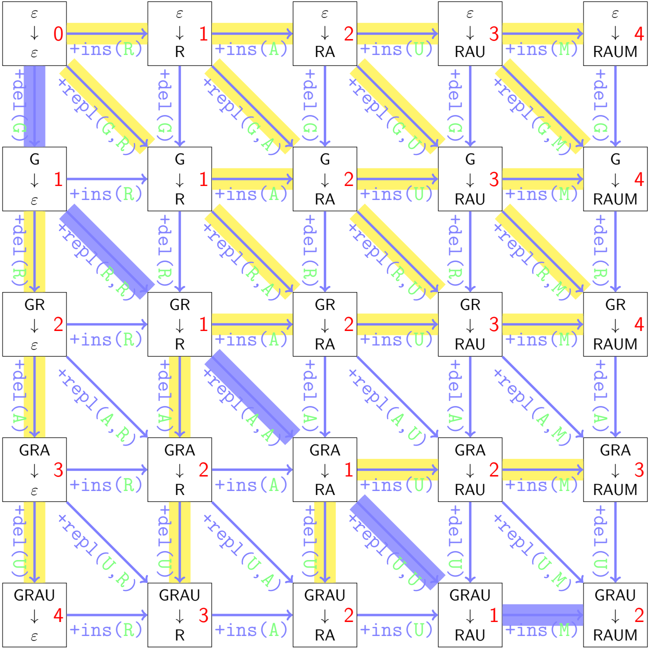
- We save at each recursion the most efficient previous entry (the highlighted arrows in our image)
- There can be more than one arrows to the three previous entries
- If we follow the highlighted path from (n, m) to $(1, 1)$ we get the optimum operations to transform x into y

Edit distance

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



Edit distance

General principle:

Edit distance

General principle:

- Recursive computation of ...

  ... the same reoccuring partial problems

  ... a limited number of partial problems

Edit distance

General principle:

- Recursive computation of ...

  ... the same reoccuring 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 reoccuring 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 reoccuring 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

Edit distance

Additional applications (I)

Additional applications:

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

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
  - `delete` operation: first gap expensive (e.g. 2), remainding are cheaper (e.g. 0.5)

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

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
  - `delete` operation: first gap expensive (e.g. 2), remainding are cheaper (e.g. 0.5)

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

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

Edit distance
Additional applications (II)

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

Hirschberg algorithm:

Edit distance

Additional applications (II)

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

Hirschberg algorithm:

- Divide-and-conquer approach

Edit distance

Additional applications (II)

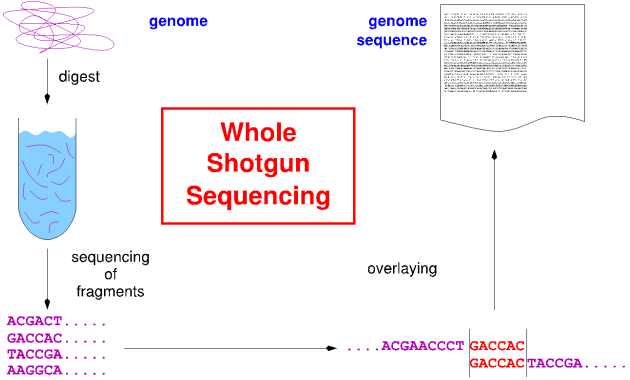
$O(n^2)$ space consumption might be problematic:

Hirschberg algorithm:

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

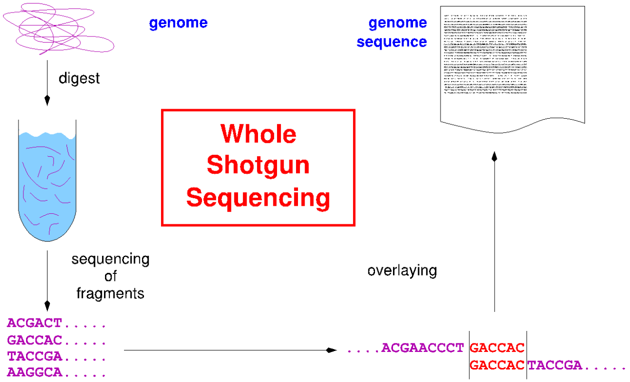
Edit distance

Additional applications (III)



Edit distance

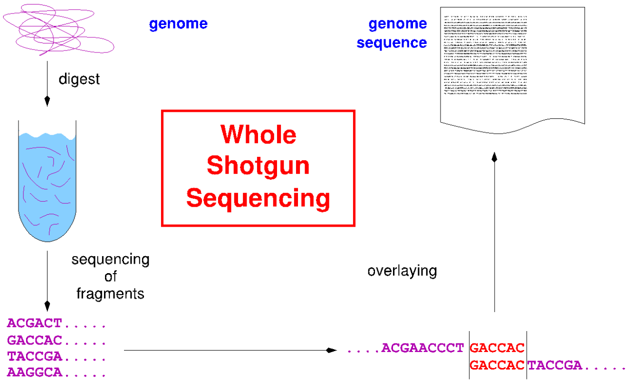
Additional applications (III)



- 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**

[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.

Further Literature

- **Dynamic programming**

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

- **Edit distance**

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