Algorithmns and Datastructures

Levenshtein distance, Dynamic programming

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science

Algorithmns and Datastructures, February 2017

Structure

Structure

Introduction

Edit distance

Edit distance:

Edit distance:

- Measurement for similarity of two words / strings

Introduction

Edit distance:

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

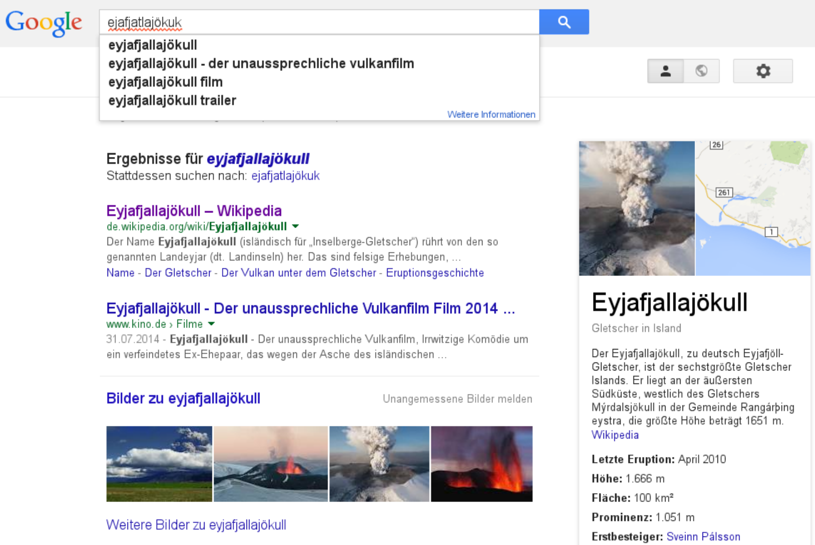
Introduction

Edit distance:

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

Introduction

Motivation: Error tolerant string comparison



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

UNI FREIBURG

Search of similar proteins:

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

Google-Scholar entry:

[HTML] Gapped **BLAST** and PSI-**BLAST**: a new generation of protein database search programs
SF Altschul, TL Madden, AA Schäffer… - Nucleic acids …, 1997 - Oxford Univ Press
Abstract The **BLAST** programs are widely used tools for searching protein and DNA databases for sequence similarities. For protein comparisons, a variety of definitional, algorithmic and statistical refinements described here permits the execution time of the **...**
Zitiert von: 58805 Ähnliche Artikel Alle 135 Versionen Zitieren Speichern

Structure

Introduction

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

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

1 2 3 4 5
DOOF

B L OED

1 2 3 4 5
DOOF
$\downarrow$ `replace(1, B)`
BOOF

BLOED

1 2 3 4 5
DOOF
↓ `replace(1, B)`
BOOF
↓ `replace(2, L)`
BLOF

BLOED

```
1 2 3 4 5
DOOF
   ↓      replace(1, B)
BOOF
   ↓      replace(2, L)
BLOF
   ↓      insert(4, E)
BLOEF

BLOED
```

```
12345
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

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

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

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

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

```
1 2 3 4 5
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

$\underbrace{\hspace{4cm}}_{\text{ED=4}}$

1 2 3 4 5
B LOED

DOOF

Edit distance
Example

1 2 3 4 5 DOOF		1 2 3 4 5 B LOED	
↓	`replace(1, B)`	↓	`replace(5, F)`
BOOF		B LOEF	
↓	`replace(2, L)`		
BLOF			
↓	`insert(4, E)`		
BLOEF			
↓	`replace(5, D)`		
BLOED		DOOF	

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

Edit distance
Example

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

$\underbrace{\hspace{5em}}_{\text{ED=4}}$

12345
BLOED
↓ `replace(5, F)`
BLOEF
↓ `delete(4)`
BLOF

DOOF

Edit distance
Example

1 2 3 4 5		1 2 3 4 5	
DOOF		B LOED	
↓	`replace(1, B)`	↓	`replace(5, F)`
BOOF		B LOEF	
↓	`replace(2, L)`	↓	`delete(4)`
B LOF		B LOF	
↓	`insert(4, E)`	↓	`replace(2, O)`
B LOEF		B OOF	
↓	`replace(5, D)`		
B LOED		DOOF	

ED=4

Edit distance
Example

1 2 3 4 5	
DOOF	
$\downarrow$	`replace(1, B)`
BOOF	
$\downarrow$	`replace(2, L)`
B L O F	
$\downarrow$	`insert(4, E)`
B L O E F	
$\downarrow$	`replace(5, D)`
B L O E D	
	ED=4

1 2 3 4 5	
B L O E D	
$\downarrow$	`replace(5, F)`
B L O E F	
$\downarrow$	`delete(4)`
B L O F	
$\downarrow$	`replace(2, O)`
B O O F	
$\downarrow$	`replace(1, D)`
DOOF	

Edit distance
Example

$$\begin{array}{ll}
\text{1 2 3 4 5} & \\
\text{DOOF} & \\
\downarrow & \texttt{replace(1, B)} \\
\text{BOOF} & \\
\downarrow & \texttt{replace(2, L)} \\
\text{BLOF} & \\
\downarrow & \texttt{insert(4, E)} \\
\text{BLOEF} & \\
\downarrow & \texttt{replace(5, D)} \\
\text{BLOED} & \\
& \underbrace{\hspace{6em}}_{\text{ED=4}}
\end{array}
\qquad
\begin{array}{ll}
\text{1 2 3 4 5} & \\
\text{BLOED} & \\
\downarrow & \texttt{replace(5, F)} \\
\text{BLOEF} & \\
\downarrow & \texttt{delete(4)} \\
\text{BLOF} & \\
\downarrow & \texttt{replace(2, O)} \\
\text{BOOF} & \\
\downarrow & \texttt{replace(1, D)} \\
\text{DOOF} & \\
& \underbrace{\hspace{6em}}_{\text{ED=4}}
\end{array}$$

Notation:

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

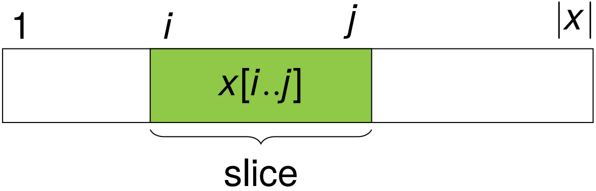
Notation:

- 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:

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

Edit distance

Trivial facts:

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

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

Edit distance

Trivial facts:

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

UNI FREIBURG

Solutions based on examples:

Solutions based on examples:

- From VERIEN to FERIEN?

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 `AAEBEAABEAREEAEBA` to `RBEAAEEBAAAEBBAEAE`?

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

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 `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 $\mathrm{ED}(GR, RA) + \mathrm{ED}(AU, UM) = 4$

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:

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 posititíon of σ_{i+1} is $\geq$ the position of σ_i where we only allow the positions to be equal on a `delete` operation

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

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 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

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	

Terminology:

Terminology:

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

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)

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

Edit distance
Recursive approach

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

Figure: Case 2

Display of solution:

- Alignment
- Example:

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

Dynamic programming:

UNI FREIBURG

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

UNI FREIBURG

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

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:

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:

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

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:

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

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

Edit distance

Case analysis:

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

UNI FREIBURG

Case analysis:

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: $\sigma_k = \texttt{insert}(m'+1, y[m])$ [then $m' = m-1$]

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 $m' = m+1$]

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 $m' = m$]

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:

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 $\sigma_1, \ldots, \sigma_{k-1}$: $x[1..n-1] \to y[1..m-1]$ and $x[n] = y[m]$

Edit distance

Case analysis:

Edit distance

Case analysis:

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

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

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

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

UNI FREIBURG

Case analysis:

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

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 $\mathrm{ED}(x,y) = |y|$
- For $|y| = 0$ is $\mathrm{ED}(x,y) = |x|$

```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:

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

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

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:

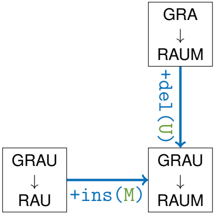
- Operations always refer to the last position (indices are omitted)
- We also display the replaced character on a `replace` operation to visualize operations without costs
  $\Rightarrow$ `repl(A, A)`

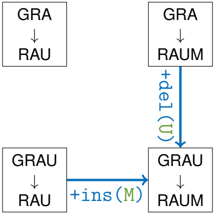
GRAU

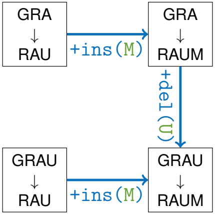
$\downarrow$

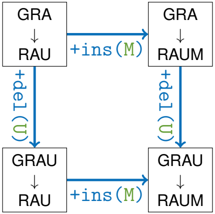
RAUM

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


GRA
↓
RAUM
+del(U)
GRAU
↓
RAU
+ins(M)
GRAU
↓
RAUM

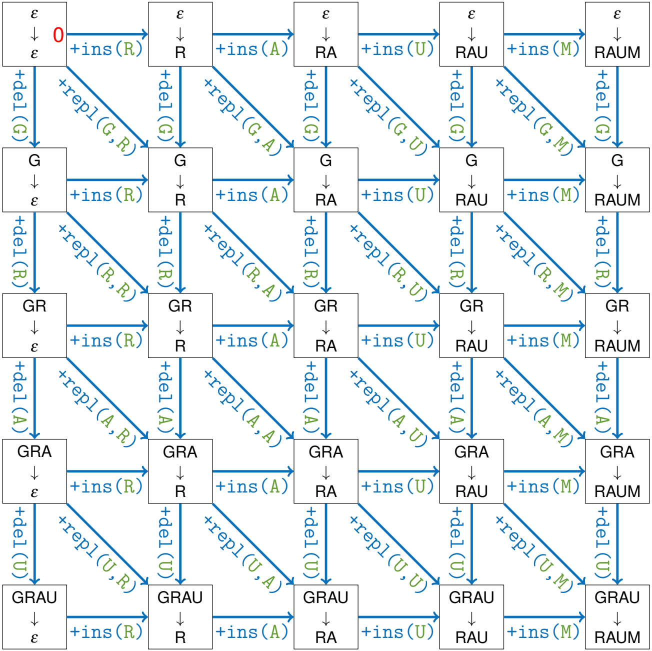

GRA
↓
RAU
GRA
↓
RAUM
+del(U)
GRAU
↓
RAU
+ins(M)
GRAU
↓
RAUM

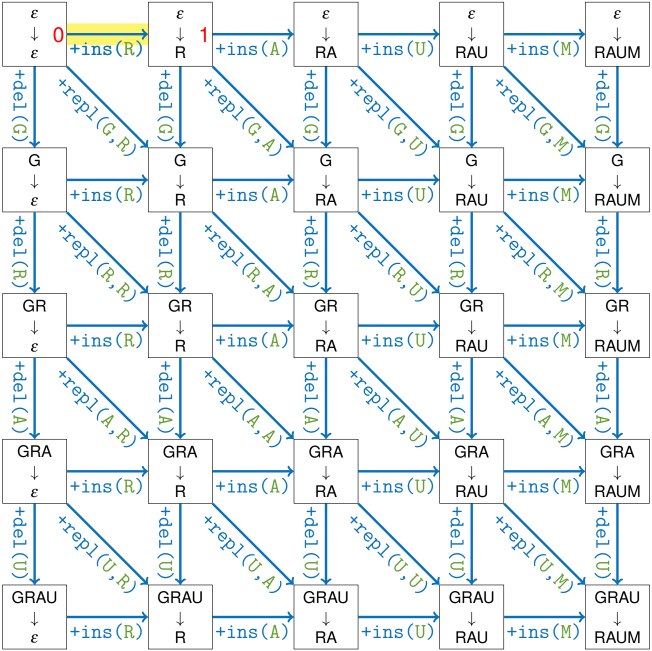

GRA
↓
RAU
+ins(M)
GRA
↓
RAUM
+del(U)
GRAU
↓
RAU
+ins(M)
GRAU
↓
RAUM

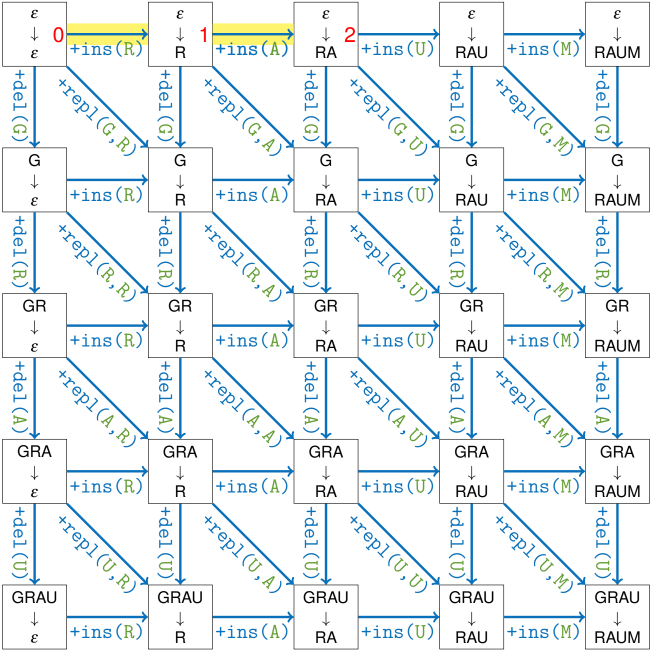

GRA
↓
RAU
+ins(M)
GRA
↓
RAUM
+del(U)
+del(U)
GRAU
↓
RAU
+ins(M)
GRAU
↓
RAUM

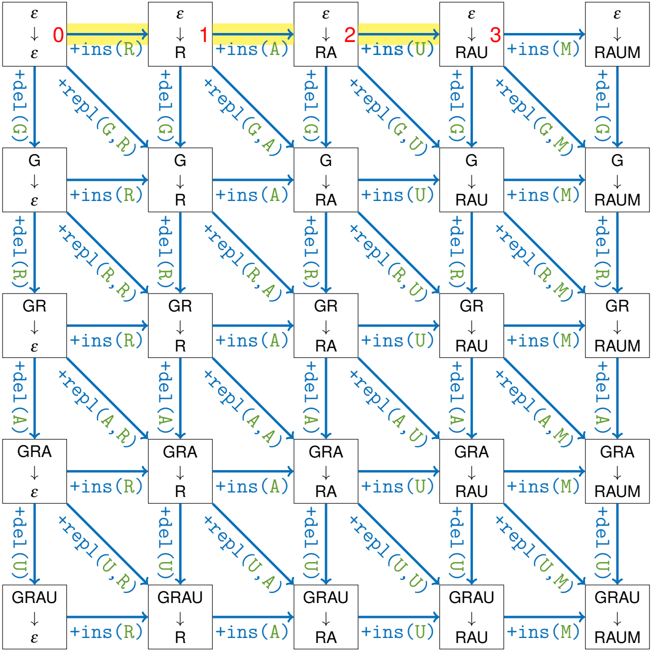
$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\begin{array}{c}\varepsilon\\\downarrow\\\varepsilon\end{array}} \arrow[r, "\texttt{+ins(R)}"'] \arrow[d, "\texttt{+del(G)}"'] & \boxed{\begin{array}{c}\varepsilon\\\downarrow\\\mathrm{R}\end{array}} \arrow[r, "\texttt{+ins(A)}"'] \arrow[d, "\texttt{+del(G)}"'] & \boxed{\begin{array}{c}\varepsilon\\\downarrow\\\mathrm{RA}\end{array}} \arrow[r, "\texttt{+ins(U)}"'] \arrow[d, "\texttt{+del(G)}"'] & \boxed{\begin{array}{c}\varepsilon\\\downarrow\\\mathrm{RAU}\end{array}} \arrow[r, "\texttt{+ins(M)}"'] \arrow[d, "\texttt{+del(G)}"'] & \boxed{\begin{array}{c}\varepsilon\\\downarrow\\\mathrm{RAUM}\end{array}} \arrow[d, "\texttt{+del(G)}"'] \\
\boxed{\begin{array}{c}\mathrm{G}\\\downarrow\\\varepsilon\end{array}} \arrow[r, "\texttt{+ins(R)}"'] \arrow[d, "\texttt{+del(R)}"'] & \boxed{\begin{array}{c}\mathrm{G}\\\downarrow\\\mathrm{R}\end{array}} \arrow[r, "\texttt{+ins(A)}"'] \arrow[d, "\texttt{+del(R)}"'] & \boxed{\begin{array}{c}\mathrm{G}\\\downarrow\\\mathrm{RA}\end{array}} \arrow[r, "\texttt{+ins(U)}"'] \arrow[d, "\texttt{+del(R)}"'] & \boxed{\begin{array}{c}\mathrm{G}\\\downarrow\\\mathrm{RAU}\end{array}} \arrow[r, "\texttt{+ins(M)}"'] \arrow[d, "\texttt{+del(R)}"'] & \boxed{\begin{array}{c}\mathrm{G}\\\downarrow\\\mathrm{RAUM}\end{array}} \arrow[d, "\texttt{+del(R)}"'] \\
\boxed{\begin{array}{c}\mathrm{GR}\\\downarrow\\\varepsilon\end{array}} \arrow[r, "\texttt{+ins(R)}"'] \arrow[d, "\texttt{+del(A)}"'] & \boxed{\begin{array}{c}\mathrm{GR}\\\downarrow\\\mathrm{R}\end{array}} \arrow[r, "\texttt{+ins(A)}"'] \arrow[d, "\texttt{+del(A)}"'] & \boxed{\begin{array}{c}\mathrm{GR}\\\downarrow\\\mathrm{RA}\end{array}} \arrow[r, "\texttt{+ins(U)}"'] \arrow[d, "\texttt{+del(A)}"'] & \boxed{\begin{array}{c}\mathrm{GR}\\\downarrow\\\mathrm{RAU}\end{array}} \arrow[r, "\texttt{+ins(M)}"'] \arrow[d, "\texttt{+del(A)}"'] & \boxed{\begin{array}{c}\mathrm{GR}\\\downarrow\\\mathrm{RAUM}\end{array}} \arrow[d, "\texttt{+del(A)}"'] \\
\boxed{\begin{array}{c}\mathrm{GRA}\\\downarrow\\\varepsilon\end{array}} \arrow[r, "\texttt{+ins(R)}"'] \arrow[d, "\texttt{+del(U)}"'] & \boxed{\begin{array}{c}\mathrm{GRA}\\\downarrow\\\mathrm{R}\end{array}} \arrow[r, "\texttt{+ins(A)}"'] \arrow[d, "\texttt{+del(U)}"'] & \boxed{\begin{array}{c}\mathrm{GRA}\\\downarrow\\\mathrm{RA}\end{array}} \arrow[r, "\texttt{+ins(U)}"'] \arrow[d, "\texttt{+del(U)}"'] & \boxed{\begin{array}{c}\mathrm{GRA}\\\downarrow\\\mathrm{RAU}\end{array}} \arrow[r, "\texttt{+ins(M)}"'] \arrow[d, "\texttt{+del(U)}"'] & \boxed{\begin{array}{c}\mathrm{GRA}\\\downarrow\\\mathrm{RAUM}\end{array}} \arrow[d, "\texttt{+del(U)}"'] \\
\boxed{\begin{array}{c}\mathrm{GRAU}\\\downarrow\\\varepsilon\end{array}} \arrow[r, "\texttt{+ins(R)}"'] & \boxed{\begin{array}{c}\mathrm{GRAU}\\\downarrow\\\mathrm{R}\end{array}} \arrow[r, "\texttt{+ins(A)}"'] & \boxed{\begin{array}{c}\mathrm{GRAU}\\\downarrow\\\mathrm{RA}\end{array}} \arrow[r, "\texttt{+ins(U)}"'] & \boxed{\begin{array}{c}\mathrm{GRAU}\\\downarrow\\\mathrm{RAU}\end{array}} \arrow[r, "\texttt{+ins(M)}"'] & \boxed{\begin{array}{c}\mathrm{GRAU}\\\downarrow\\\mathrm{RAUM}\end{array}}
\end{tikzcd}
$$

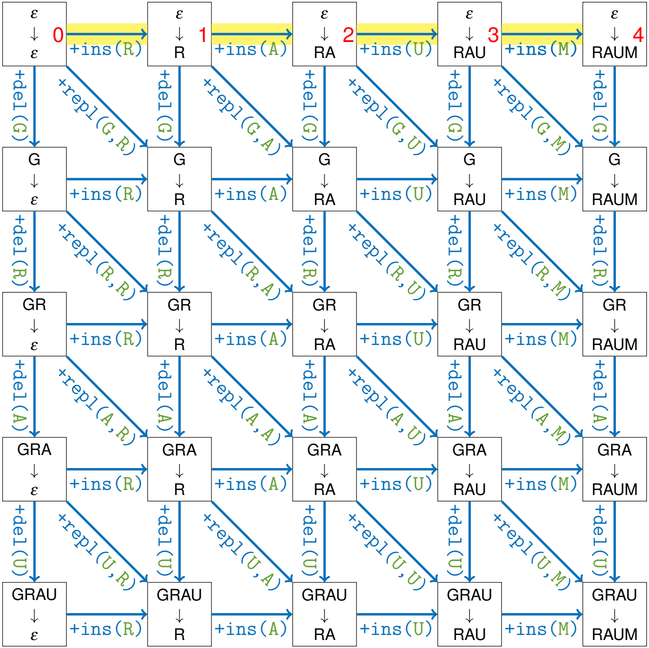
Fast algorithm:
We can determine the edit distance for all combination of partial strings from the top left to bottom right.


0
ε ↓ ε
ε ↓ R
ε ↓ RA
ε ↓ RAU
ε ↓ RAUM
G ↓ ε
G ↓ R
G ↓ RA
G ↓ RAU
G ↓ RAUM
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

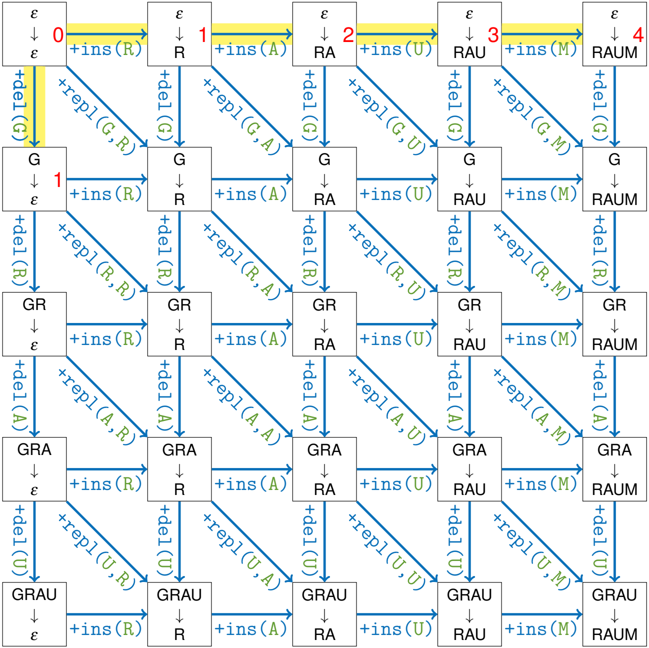

ε ↓ ε
0
+ins(R)
1
ε ↓ R
+ins(A)
ε ↓ RA
+ins(U)
ε ↓ RAU
+ins(M)
ε ↓ RAUM
+del(G)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
G ↓ ε
G ↓ R
G ↓ RA
G ↓ RAU
G ↓ RAUM
+del(R)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
+del(A)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
+del(U)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM

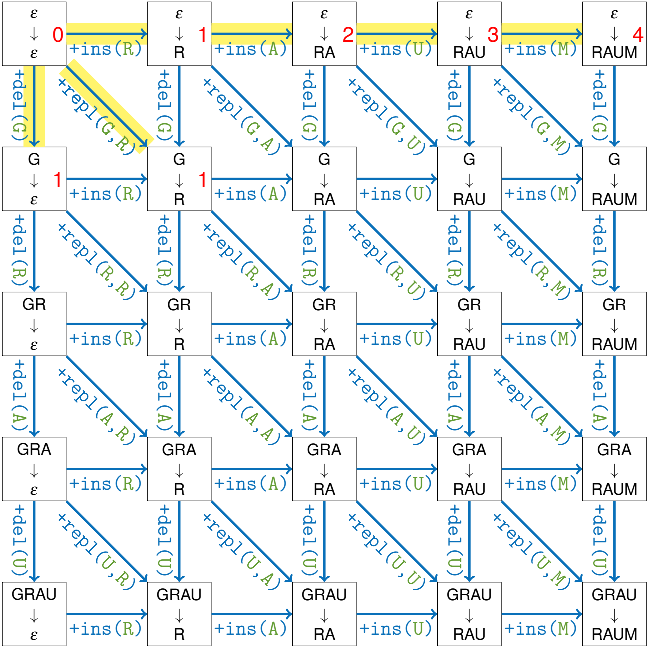

ε → ε
ε → R
ε → RA
ε → RAU
ε → RAUM
0
1
2
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
G → ε
G → R
G → RA
G → RAU
G → RAUM
GR → ε
GR → R
GR → RA
GR → RAU
GR → RAUM
GRA → ε
GRA → R
GRA → RA
GRA → RAU
GRA → RAUM
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM

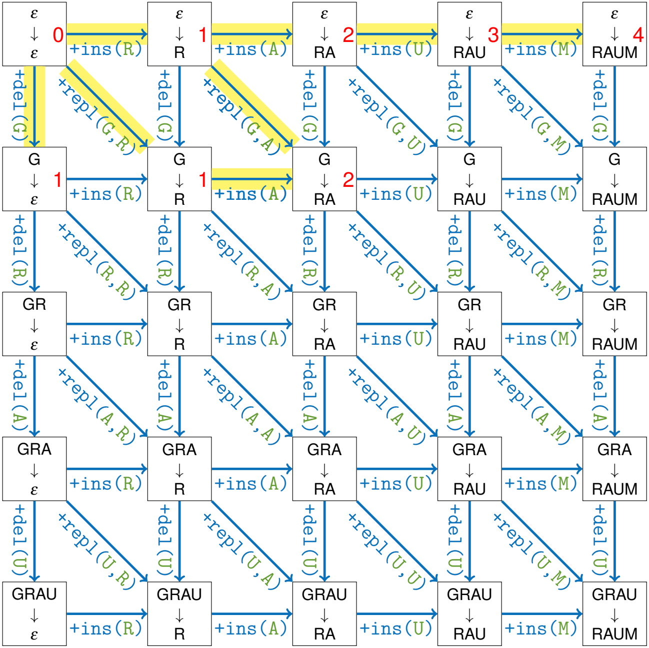

ε ↓ ε
ε ↓ R
ε ↓ RA
ε ↓ RAU
ε ↓ RAUM
0
1
2
3
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
G ↓ ε
G ↓ R
G ↓ RA
G ↓ RAU
G ↓ RAUM
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM

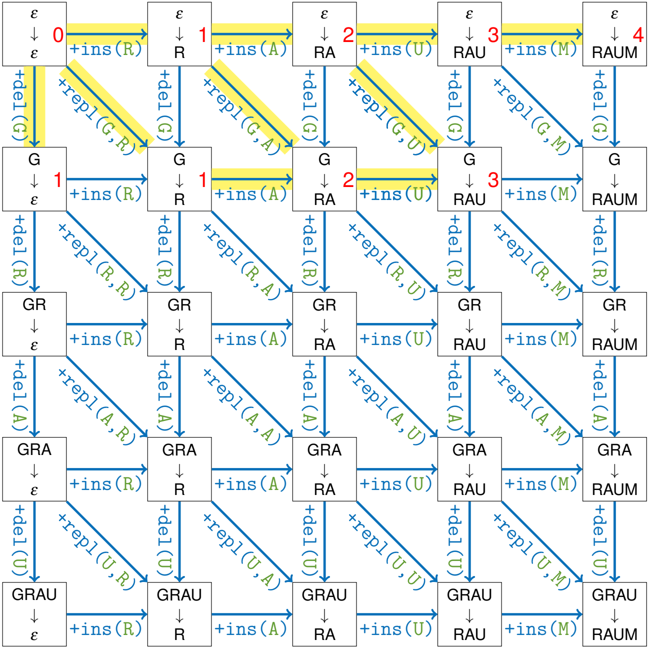


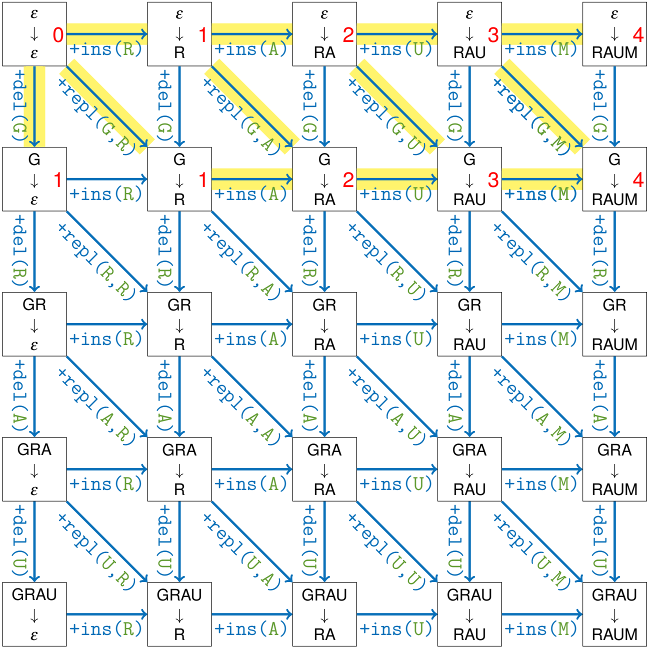
ε ↓ ε
0
+ins(R)
ε ↓ R
1
+ins(A)
ε ↓ RA
2
+ins(U)
ε ↓ RAU
3
+ins(M)
ε ↓ RAUM
4
+del(G)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
G ↓ ε
G ↓ R
G ↓ RA
G ↓ RAU
G ↓ RAUM
+del(R)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
+del(A)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
+del(U)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM

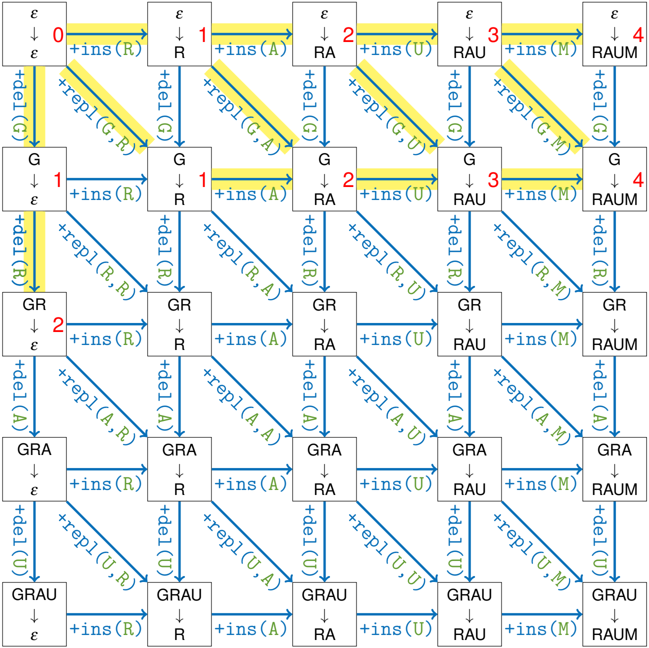

ε ↓ ε
ε ↓ R
ε ↓ RA
ε ↓ RAU
ε ↓ RAUM
0
1
2
3
4
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
G ↓ ε
G ↓ R
G ↓ RA
G ↓ RAU
G ↓ RAUM
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM

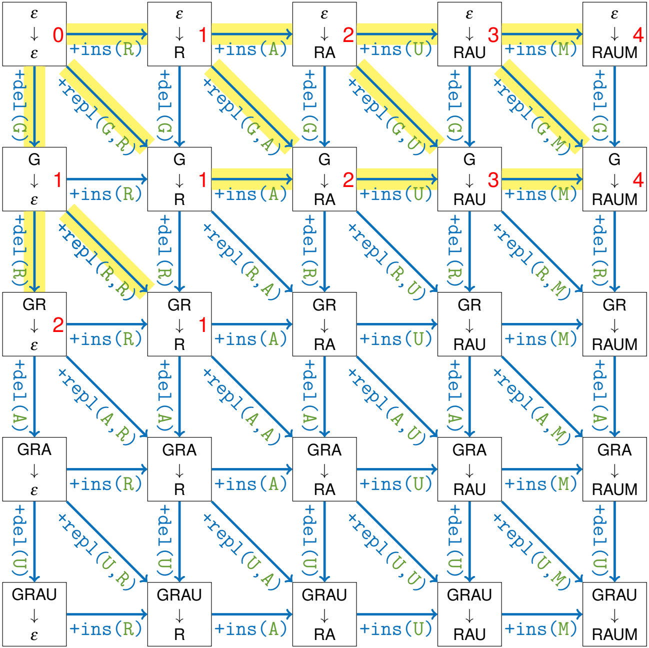

ε ↓ ε
0
ε ↓ R
1
ε ↓ RA
2
ε ↓ RAU
3
ε ↓ RAUM
4
G ↓ ε
1
G ↓ R
1
G ↓ RA
G ↓ RAU
G ↓ RAUM
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε → ε 0
+ins(R)
ε → R 1
+ins(A)
ε → RA 2
+ins(U)
ε → RAU 3
+ins(M)
ε → RAUM 4
+del(G)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
G → ε 1
G → R 1
G → RA 2
G → RAU
G → RAUM
+del(R)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
GR → ε
GR → R
GR → RA
GR → RAU
GR → RAUM
+del(A)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
GRA → ε
GRA → R
GRA → RA
GRA → RAU
GRA → RAUM
+del(U)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM

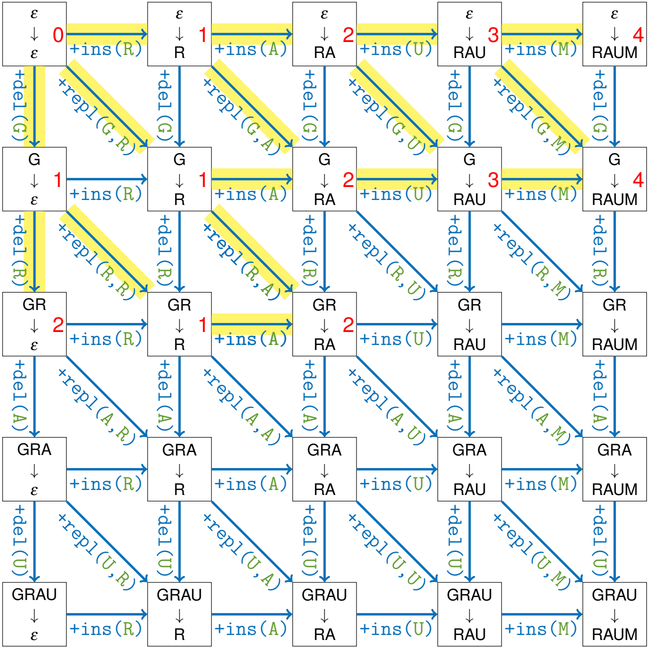

ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM
GR ↓ ε
GR ↓ R
GR ↓ RA
GR ↓ RAU
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε → ε 0
ε → R 1
ε → RA 2
ε → RAU 3
ε → RAUM 4
G → ε 1
G → R 1
G → RA 2
G → RAU 3
G → RAUM 4
GR → ε
GR → R
GR → RA
GR → RAU
GR → RAUM
GRA → ε
GRA → R
GRA → RA
GRA → RAU
GRA → RAUM
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

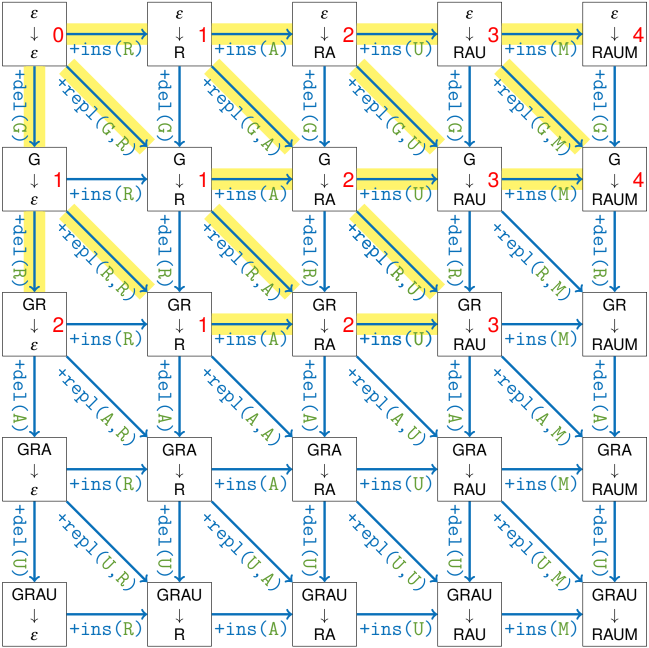

ε → ε 0
ε → R 1
ε → RA 2
ε → RAU 3
ε → RAUM 4
G → ε 1
G → R 1
G → RA 2
G → RAU 3
G → RAUM 4
GR → ε 2
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)



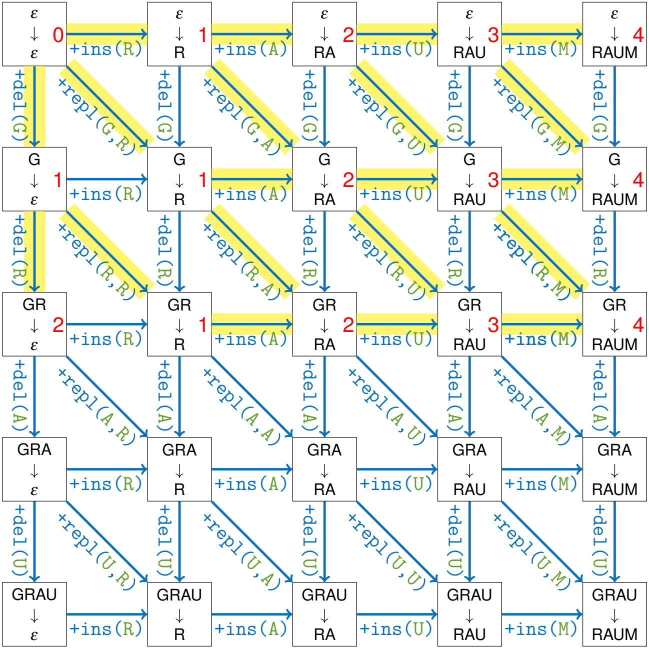
ε → ε 0
ε → R 1
ε → RA 2
ε → RAU 3
ε → RAUM 4
G → ε 1
G → R 1
G → RA 2
G → RAU 3
G → RAUM 4
GR → ε 2
GR → R 1
GR → RA
GR → RAU
GR → RAUM
GRA → ε
GRA → R
GRA → RA
GRA → RAU
GRA → RAUM
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

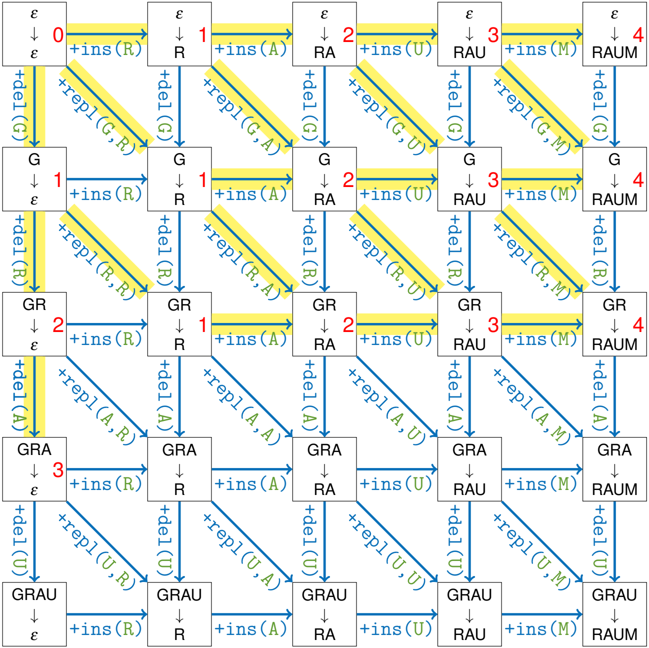


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
+del(R)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU
GR ↓ RAUM
+del(A)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
+del(U)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM

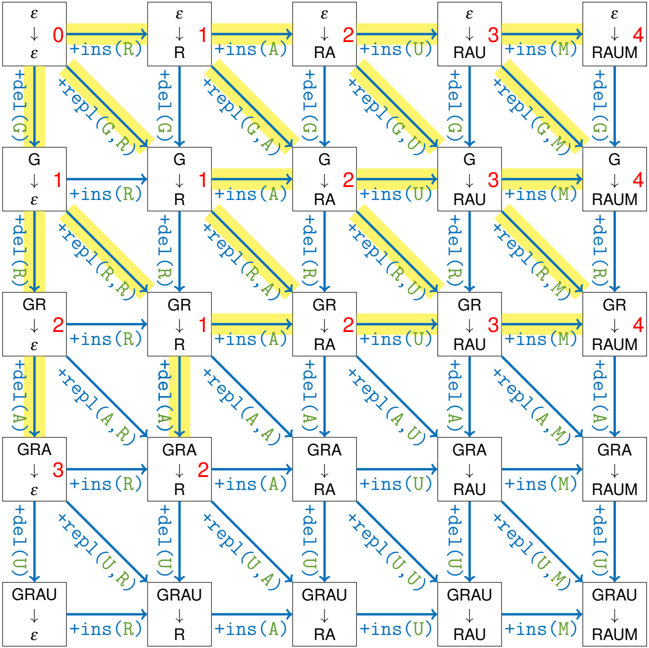


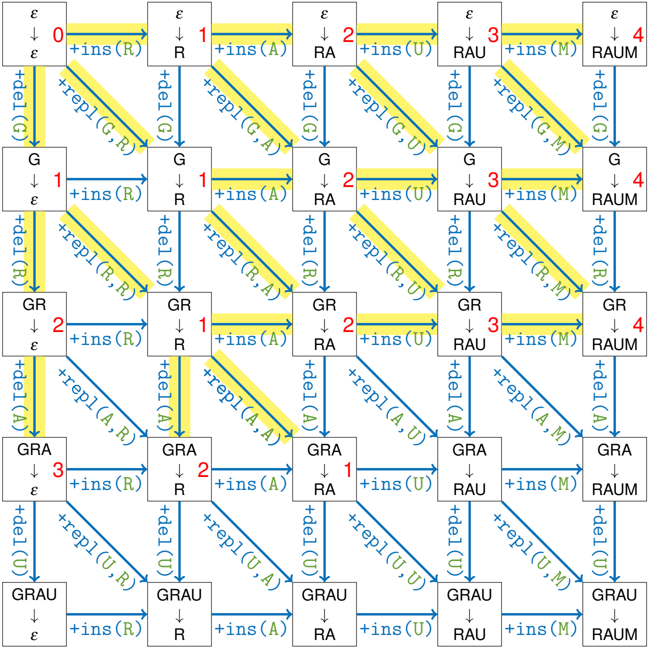
ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

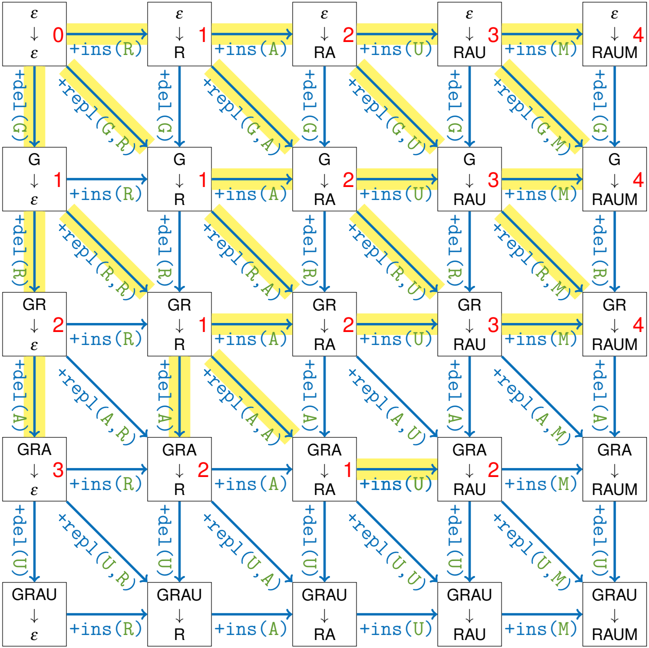


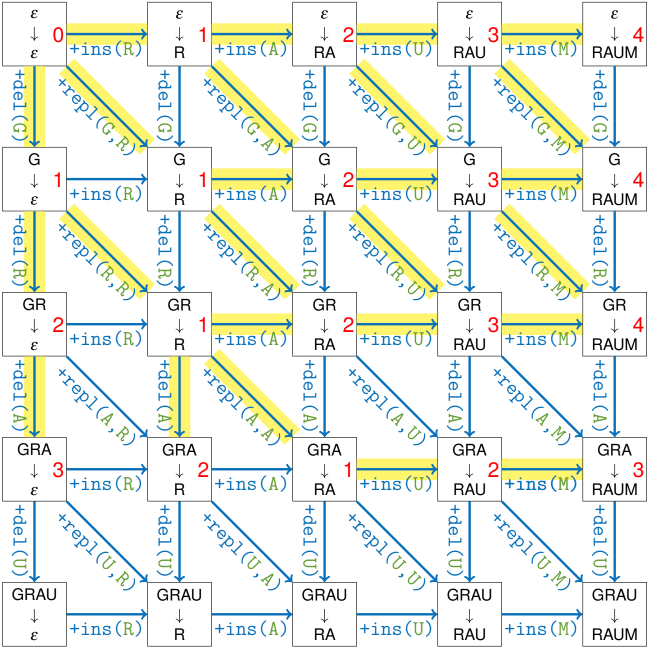
ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

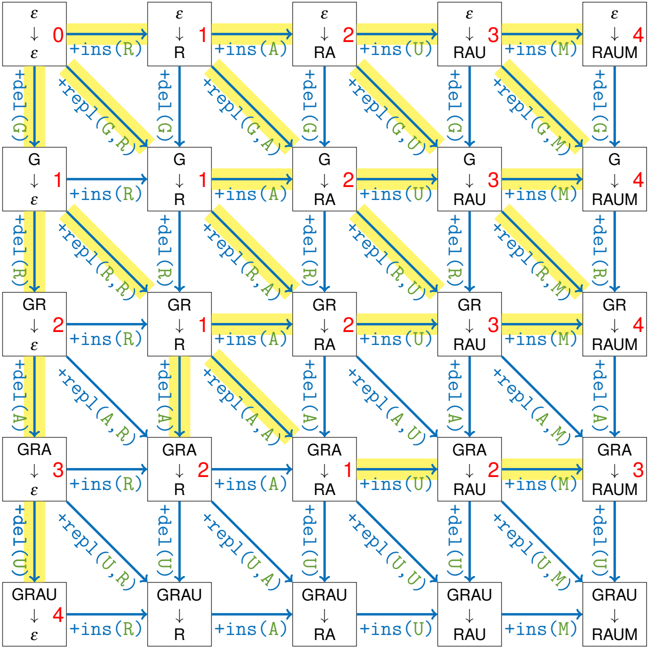


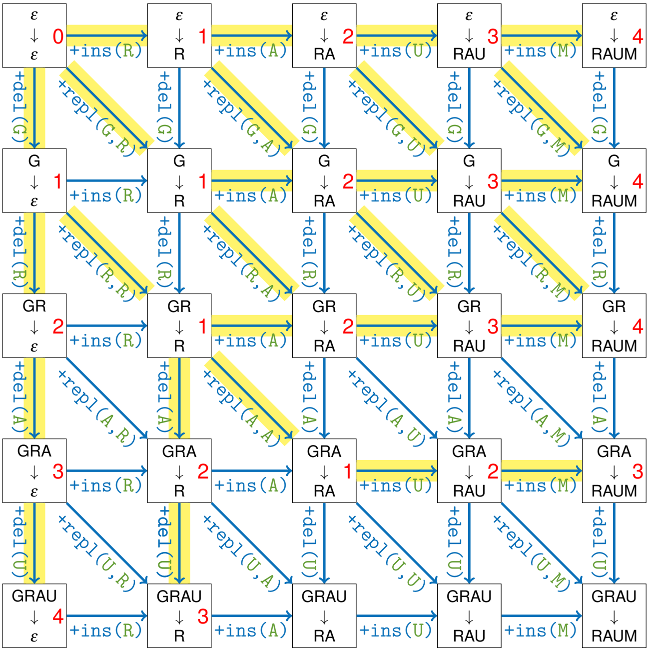
ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU
GRA ↓ RAUM
GRAU ↓ ε
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

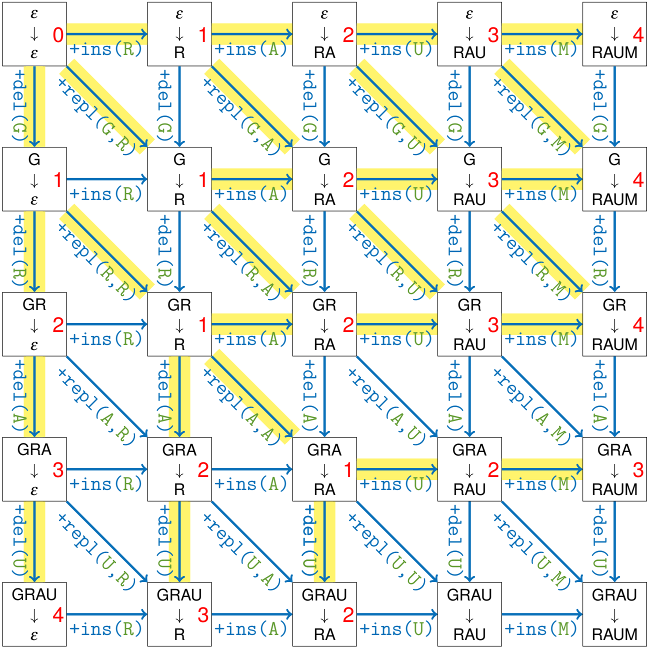

ε → ε 0
ε → R 1
ε → RA 2
ε → RAU 3
ε → RAUM 4
G → ε 1
G → R 1
G → RA 2
G → RAU 3
G → RAUM 4
GR → ε 2
GR → R 1
GR → RA 2
GR → RAU 3
GR → RAUM 4
GRA → ε 3
GRA → R 2
GRA → RA 1
GRA → RAU 2
GRA → RAUM
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)



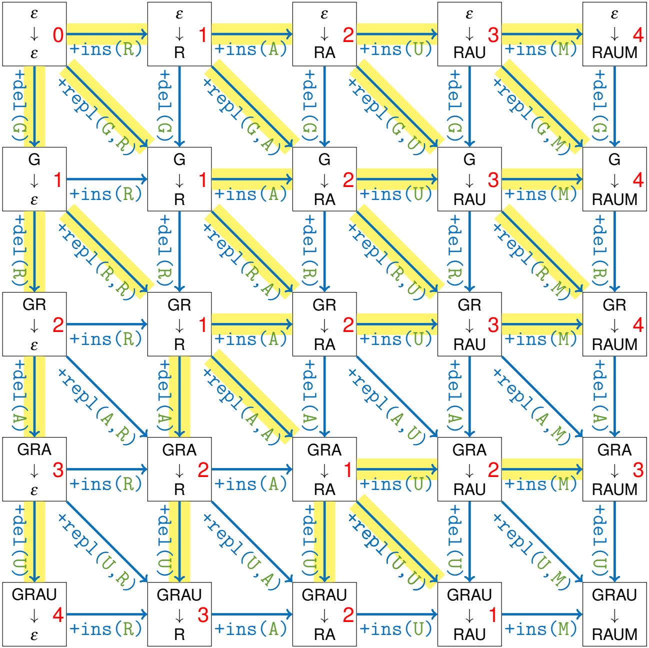
ε → ε 0
ε → R 1
ε → RA 2
ε → RAU 3
ε → RAUM 4
G → ε 1
G → R 1
G → RA 2
G → RAU 3
G → RAUM 4
GR → ε 2
GR → R 1
GR → RA 2
GR → RAU 3
GR → RAUM 4
GRA → ε 3
GRA → R 2
GRA → RA 1
GRA → RAU 2
GRA → RAUM 3
GRAU → ε
GRAU → R
GRAU → RA
GRAU → RAU
GRAU → RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

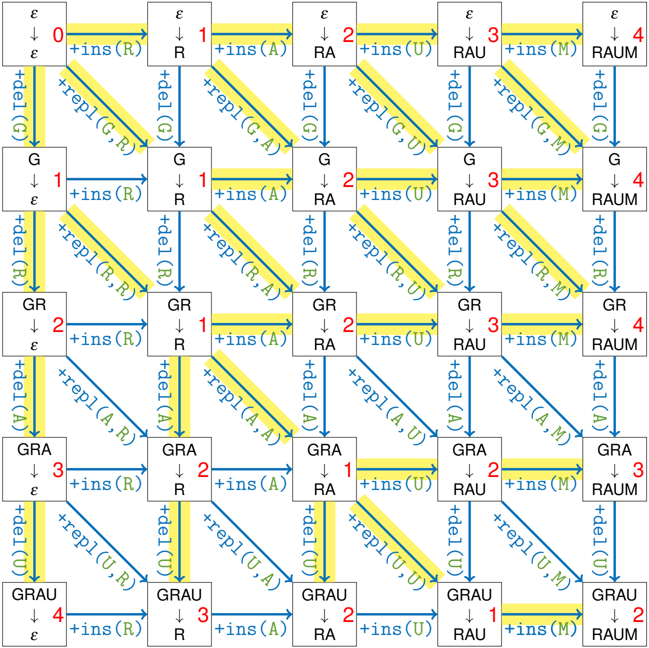

ε ↓ ε 0
+ins(R)
ε ↓ R 1
+ins(A)
ε ↓ RA 2
+ins(U)
ε ↓ RAU 3
+ins(M)
ε ↓ RAUM 4
+del(G)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
+del(R)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
+del(A)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
+del(U)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)
GRAU ↓ ε 4
GRAU ↓ R
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
GRAU ↓ ε 4
GRAU ↓ R 3
GRAU ↓ RA
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)



ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
GRAU ↓ ε 4
GRAU ↓ R 3
GRAU ↓ RA 2
GRAU ↓ RAU
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
GRAU ↓ ε 4
GRAU ↓ R 3
GRAU ↓ RA 2
GRAU ↓ RAU 1
GRAU ↓ RAUM
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)


ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
GRAU ↓ ε 4
GRAU ↓ R 3
GRAU ↓ RA 2
GRAU ↓ RAU 1
GRAU ↓ RAUM 2
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

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?

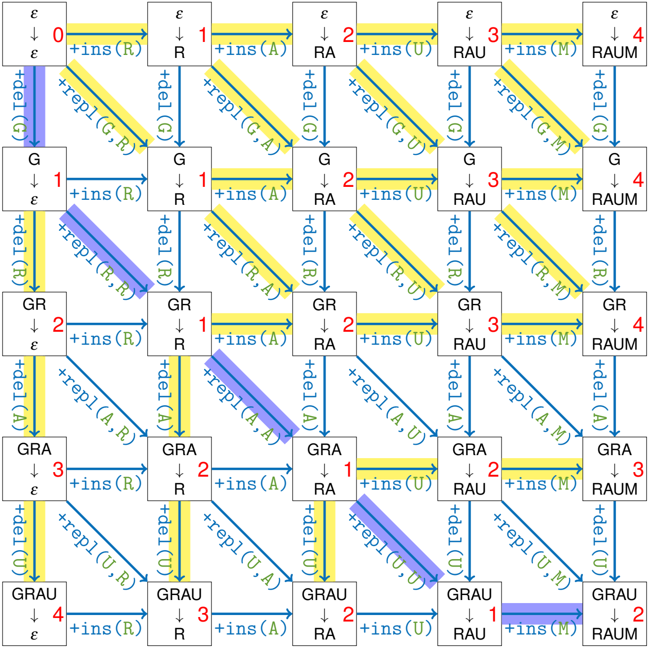
- We save at each recursion the most efficient previous entry (the highlighted arrows in our image)
- There can be more than one arrows to the three previous entries

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



ε ↓ ε 0
ε ↓ R 1
ε ↓ RA 2
ε ↓ RAU 3
ε ↓ RAUM 4
G ↓ ε 1
G ↓ R 1
G ↓ RA 2
G ↓ RAU 3
G ↓ RAUM 4
GR ↓ ε 2
GR ↓ R 1
GR ↓ RA 2
GR ↓ RAU 3
GR ↓ RAUM 4
GRA ↓ ε 3
GRA ↓ R 2
GRA ↓ RA 1
GRA ↓ RAU 2
GRA ↓ RAUM 3
GRAU ↓ ε 4
GRAU ↓ R 3
GRAU ↓ RA 2
GRAU ↓ RAU 1
GRAU ↓ RAUM 2
+ins(R)
+ins(A)
+ins(U)
+ins(M)
+del(G)
+del(R)
+del(A)
+del(U)
+repl(G,R)
+repl(G,A)
+repl(G,U)
+repl(G,M)
+repl(R,R)
+repl(R,A)
+repl(R,U)
+repl(R,M)
+repl(A,R)
+repl(A,A)
+repl(A,U)
+repl(A,M)
+repl(U,R)
+repl(U,A)
+repl(U,U)
+repl(U,M)

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!

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 deletition unrealistic

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 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), remaining 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

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

Hirschberg algorithm:

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

Hirschberg algorithm:

- Divide-and-conquer approach

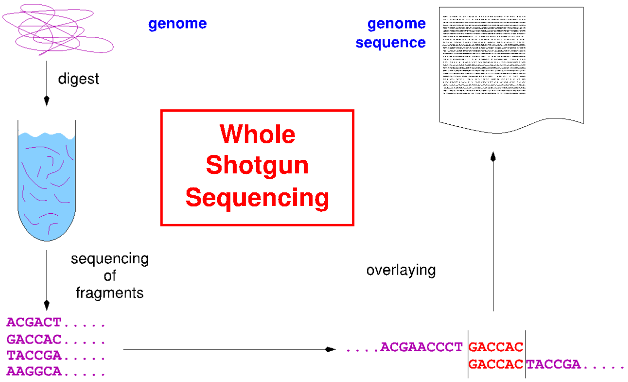
$O(n^2)$ space consumption might be problematic:

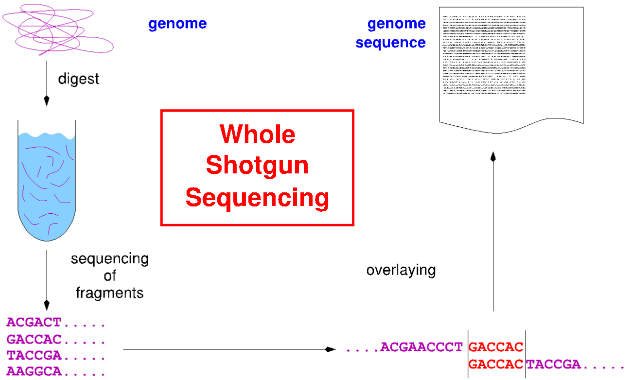
Hirschberg algorithm:

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

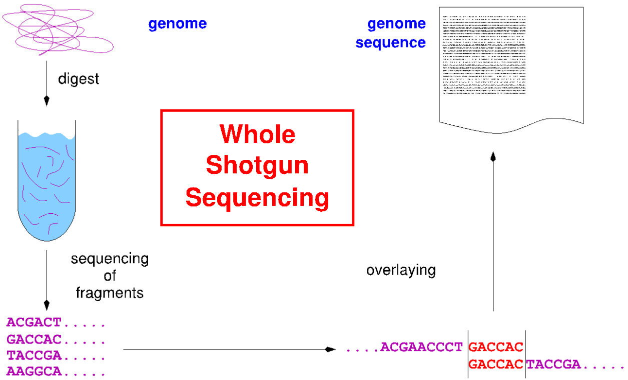
Edit distance

Additional applications (III)





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



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

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