

Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von
Algorithmen (IEMS)
WS 2014 / 2015

Vorlesung 14, Donnerstag, 5. Februar 2015
(Editierdistanz, dynamische Programmierung)

Junior-Prof. Dr. Olaf Ronneberger
Image Analysis Lab
Institut für Informatik
Universität Freiburg

Blick über die Vorlesung heute

■ Organisatorisches

- Offizielle **Evaluation** dieser Vorlesung

■ Editierdistanz

- Maß für Ähnlichkeit zwischen zwei Wörtern / Zeichenketten
- Algorithmus zur effizienten Berechnung
- Allgemeines Prinzip: dynamische Programmierung
- **Übungsblatt 14:** Implementierung des Algorithmus + damit ähnliche Suchanfragen im [AOL Query Log](#) finden

Feedback zum Ü13 „Dijkstra's Algorithmus“

- Anspruchsvoll aber hat Spaß gemacht
- Hat viel Zeit gekostet
- „Aufgabe hat mir bisher am besten gefallen.“
- zu Aufgabe 1: „Umfrage gemacht. (Werden die Fragen in der Klausur ähnlich? :))“

Motivation: Fehlertoleranter Textvergleich



[ejafjallajökull](#)
eyjafjallajökull
 eyjafjallajökull - der unaussprechliche vulkanfilm
 eyjafjallajökull film
 eyjafjallajökull trailer

[Weitere Informationen](#)

Ergebnisse für *eyjafjallajökull*
 Stattdessen suchen nach: [ejafjallajökull](#)

Eyjafjallajökull – Wikipedia
de.wikipedia.org/wiki/Eyjafjallajökull

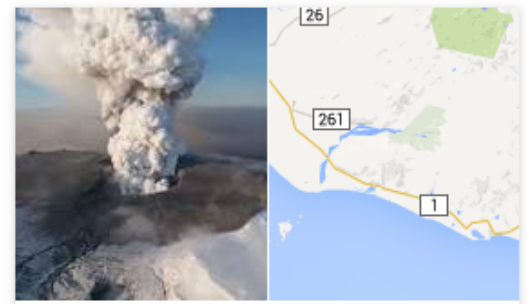
Der Name **Eyjafjallajökull** (isländisch für „Inselberge-Gletscher“) rührt von den so genannten Landeyjar (dt. Landinseln) her. Das sind felsige Erhebungen, ...
 Name - Der Gletscher - Der Vulkan unter dem Gletscher - Eruptionsgeschichte

Eyjafjallajökull - Der unaussprechliche Vulkanfilm Film 2014 ...
www.kino.de > Filme
 31.07.2014 - **Eyjafjallajökull** - Der unaussprechliche Vulkanfilm, Irrwitzige Komödie um ein verfeindetes Ex-Ehepaar, das wegen der Asche des isländischen ...

Bilder zu eyjafjallajökull Unangemessene Bilder melden



[Weitere Bilder zu eyjafjallajökull](#)



Eyjafjallajökull

Gletscher in Island

Der Eyjafjallajökull, zu deutsch Eyjaföll-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.
[Wikipedia](#)

- Letzte Eruption:** April 2010
- Höhe:** 1.666 m
- Fläche:** 100 km²
- Prominenz:** 1.051 m
- Erstbesteiger:** Sveinn Pálsson

Motivation

- Viele Anwendungen, wo man ähnliche Strings sucht
 - Dubletten in Adressdatenbanken
 - Hein Blöd, 27568 Bremerhaven
 - Hein Bloed, 27568 Bremerhafen
 - Hein Doof, 27478 Cuxhaven
 - Produktsuche
 - Memory Stik
 - Websuche
 - eyjaföllajaküll
 - uniwersität verien 2014
 - Bioinformatik:
 - Ähnlichkeit von DNA-Sequenzen

- Definition **Editierdistanz**, auch **Levenshtein-Distanz**
 - Gegeben zwei Zeichenketten (strings) x und y
 - $ED(x, y)$ = Editierdistanz (edit distance) von x und y = die minimale Anzahl **Operationen** um x in y zu transformieren:
 - **Einfügen** eines Buchstabens (**insert**)
 - **Ersetzen** eines Buchstabens durch einen anderen (**replace**)
 - **Löschen** eines Buchstabens (**delete**)
 - Die **Position** einer Operation ist ... siehe Beispiel:

Beispiel

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

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

replace(1,D)

D O O F

■ Etwas Notation

- Mit ε bezeichnen wir das leere Wort
- Mit $|x|$ bezeichnen wir die Länge von x (= Anzahl Zeichen)
- Mit $x[i..j]$ bezeichnen wir die Teilfolge der Zeichen i bis j der Zeichenkette x , wobei $1 \leq i \leq j \leq |x|$

■ Ein paar einfache Eigenschaften

- $ED(x, y) = ED(y, x)$
- $ED(x, \varepsilon) = |x|$
- $ED(x, y) \geq \text{abs}(|x| - |y|)$ $\text{abs}(x) = x > 0 ? x : -x$
- $ED(x, y) \leq ED(x[1..n-1], y[1..m-1]) + 1$ $n = |x|, m = |y|$

■ Lösungsideen anhand von Beispielen

- Für VERIEN → FERIEN ?
- Für MEXIKO → AMERIKA ?
- Für AAEBEAABEAREEEAEB → RBEAAEEBAAAEBBAEAE ?
- **Beobachtung:** möglichst große gemeinsame Teilstrings zu finden klappt manchmal, aber nicht immer

■ Rekursiver Ansatz

- In zwei Teile / Hälften teilen? Keine gute Idee, z.B.
 $ED(\text{GRAU}, \text{RAUM}) = 2$ aber $ED(\text{GR}, \text{RA}) + ED(\text{AU}, \text{UM}) = 4$
- Auf ein "kleineres" Problem zurückführen?
Das probieren wir jetzt !

■ Terminologie

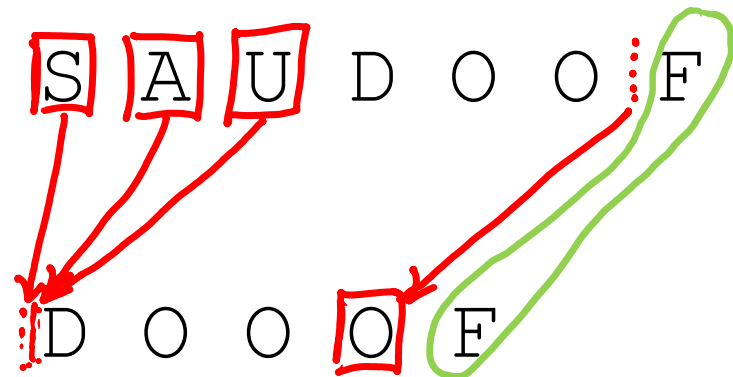
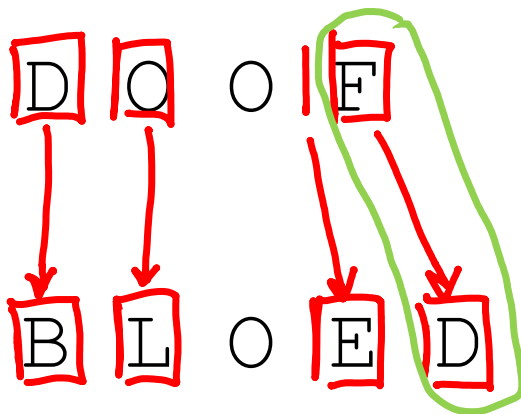
- Seien x und y unsere beiden Zeichenketten
- Seien $\sigma_1, \dots, \sigma_k$ eine Folge von k Operationen, wobei $k = ED(x, y)$ für $x \rightarrow y$, das heißt um x in y zu überführen (Wir nehmen im Folgenden nicht an, dass wir die Folge schon kennen, sondern nur, dass es so eine gibt)
- Wir betrachten im Folgenden nur **monotone** Op.-Folgen, d.h. die Position von σ_{i+1} ist \geq die Position von σ_i , wobei = nur dann erlaubt ist, wenn beides **delete** Operationen sind

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	

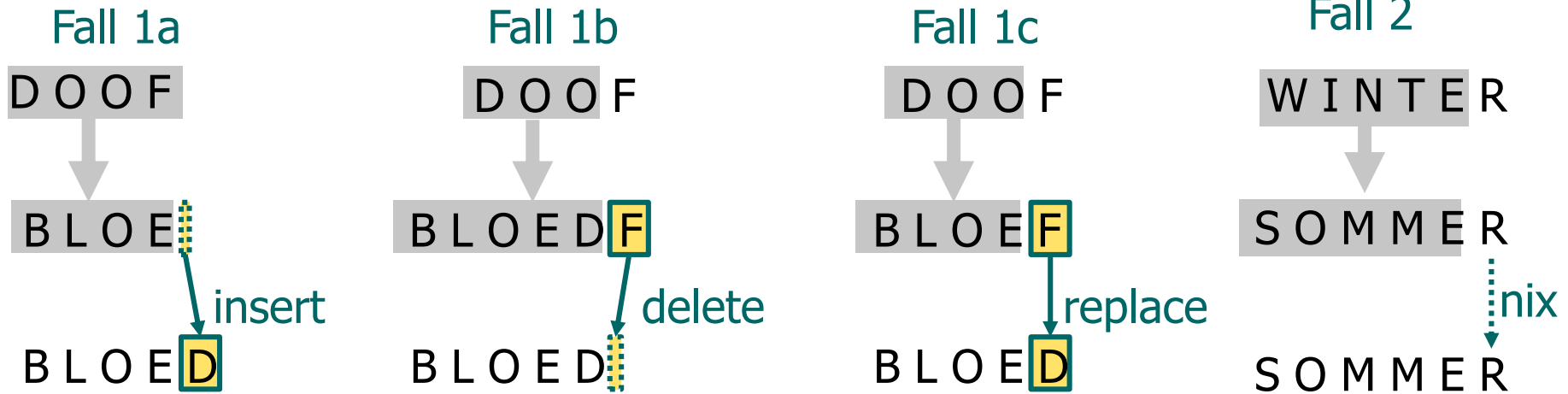
1	2	3	4	5	6	7	
S	A	U	D	O	O	F	
							delete(1)
A	U	D	O	O	F		
							delete(1)
U	D	O	O	F			
							delete(1)
D	O	O	F				
							insert(4 ,O)
D	O	O	O	F			10

■ Terminologie

- **Lemma:** Für beliebige x und y mit $k = ED(x, y)$ gibt es eine **monotone** Folge von k Operationen für $x \rightarrow y$
- **Beweisintuition:** die Reihenfolge der Operationen ist im Grunde egal, also kann man sie auch monoton anordnen



Rekursive Lösung: Letzte Operation betrachten



Grauen Teil
rekursiv lösen

■ Fallunterscheidung

- Wir betrachten die letzte Operation σ_k
 - $\sigma_1, \dots, \sigma_{k-1} : x \rightarrow z$ und $\sigma_k : z \rightarrow y$
(z.B. $x = \text{DOOF}$, $z = \text{SAUBLOEF}$, $y = \text{SAUBLOED}$)
 - Seien $n = |x|$ und $m = |y|$ und $m' = |z|$
 - Man beachte, dass $m' \in \{m - 1, m, m + 1\}$ wieso?
- Fall 1: σ_k macht etwas "ganz am Ende" von z , d.h. eins von:
 - Fall 1a: $\sigma_k = \text{insert}(m' + 1, y[m])$ [dann ist $m' = m - 1$]
 - Fall 1b: $\sigma_k = \text{delete}(m')$ [dann ist $m' = m + 1$]
 - Fall 1c: $\sigma_k = \text{replace}(m', y[m])$ [dann ist $m' = m$]
- Fall 2: σ_k macht nichts "ganz am Ende" von z
 - dann $z[m'] = y[m]$ und $x[n] = z[m']$ und damit
 $\sigma_1, \dots, \sigma_k : x[1..n-1] \rightarrow y[1..m-1]$ und $x[n] = y[m]$

■ Wir haben also einen dieser vier Fälle

- Fall 1a (insert am Ende): $\sigma_1, \dots, \sigma_{k-1} : x \rightarrow y[1..m-1]$
- Fall 1b (delete am Ende): $\sigma_1, \dots, \sigma_{k-1} : x[1..n-1] \rightarrow y$
- Fall 1c (replace am Ende): $\sigma_1, \dots, \sigma_{k-1} : x[1..n-1] \rightarrow y[1..m-1]$
- Fall 2 (nichts am Ende): $\sigma_1, \dots, \sigma_k : x[1..n-1] \rightarrow y[1..m-1]$

■ Daraus folgt die rekursive Formel

- Für $|x| > 0$ und $|y| > 0$ ist $ED(x, y) =$ das Minimum von
 - $ED(x, y[1..m-1]) + 1$, und
 - $ED(x[1..n-1], y) + 1$, und
 - $ED(x[1..n-1], y[1..m-1]) + 1$, falls $x[n] \neq y[m]$
 - $ED(x[1..n-1], y[1..m-1]) + 0$, falls $x[n] = y[m]$
- Für $|x| = 0$ ist $ED(x, y) = |y|$,
- Für $|y| = 0$ ist $ED(x, y) = |x|$

Rekursiver Algorithmus

- Diese rekursiven Algorithmus wollen wir jetzt implementieren

EditDistanceTest.java

```
import org.junit.Test;
import org.junit.Assert;

/**
 * Test class for edit distance computation.
 */
public class EditDistanceTest {

    /**
     * Test the recursive method.
     */
    @Test
    public void testComputeEditDistanceWithRecursion() {
        Assert.assertEquals(0, EditDistance.computeRecursively("", ""));
        Assert.assertEquals(1, EditDistance.computeRecursively("ferien", "verien"));
        Assert.assertEquals(6, EditDistance.computeRecursively("ferien", ""));
        Assert.assertEquals(6, EditDistance.computeRecursively("", "ferien"));
    }
}
```


EditDistance.java

```
/**
 * Class for computing the edit distance.
 */
public class EditDistance {

    /**
     * Compute the edit distance recursively.
     */
    static int computeRecursively(String x, String y) {
    }
}
```

EditDistanceMain.java

```
/**
 * Class for main programm computing the edit distance for two given strings.
 */
public class EditDistanceMain {

    /**
     * Main method.
     */
    public static void main(String[] args) {
        // Parse command line arguments.
        if (args.length != 2) {
            System.out.println("Usage: java -jar EditDistanceMain.jar <str1> <str2>");
            System.out.println("Computes the edit distance between the two strings.");
            System.exit(1);
        }
        String x = args[0];
        String y = args[1];

        // Compute the edit distance.
        System.out.println("Input string 1 = \"" + x + "\"");
        System.out.println("Input string 2 = \"" + y + "\"");
        System.out.println("Edit distance (via recursion) = "
            + EditDistance.computeRecursively(x, y));
    }
}
```

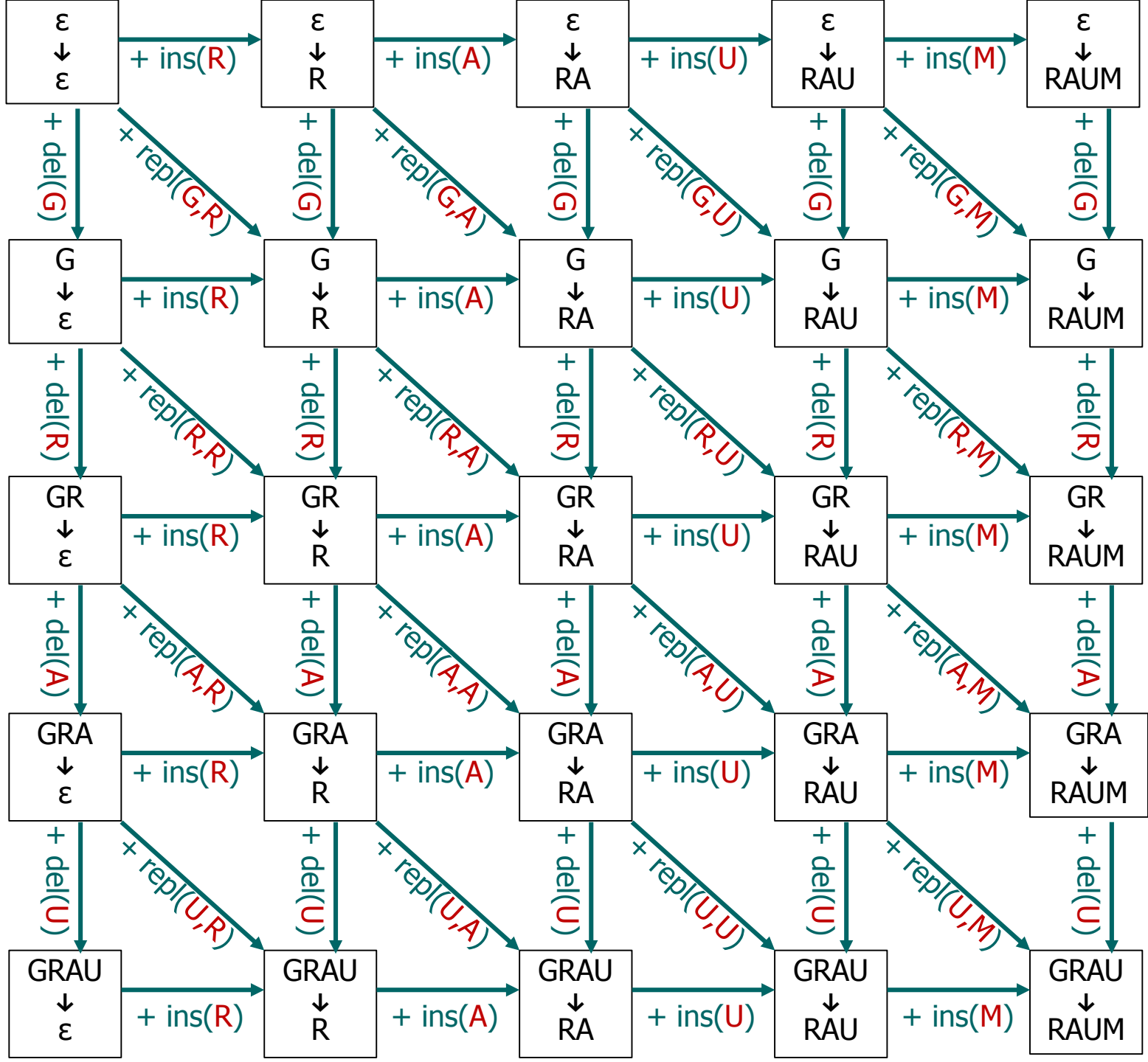
■ Rekursives Programm

- Für die Laufzeit würde folgende rekursive Formel gelten

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

- Man kann leicht ausrechnen, dass dann $T(n, n) \geq 3^n$
- Das heißt die Laufzeit wäre (mindestens) **exponentiell**

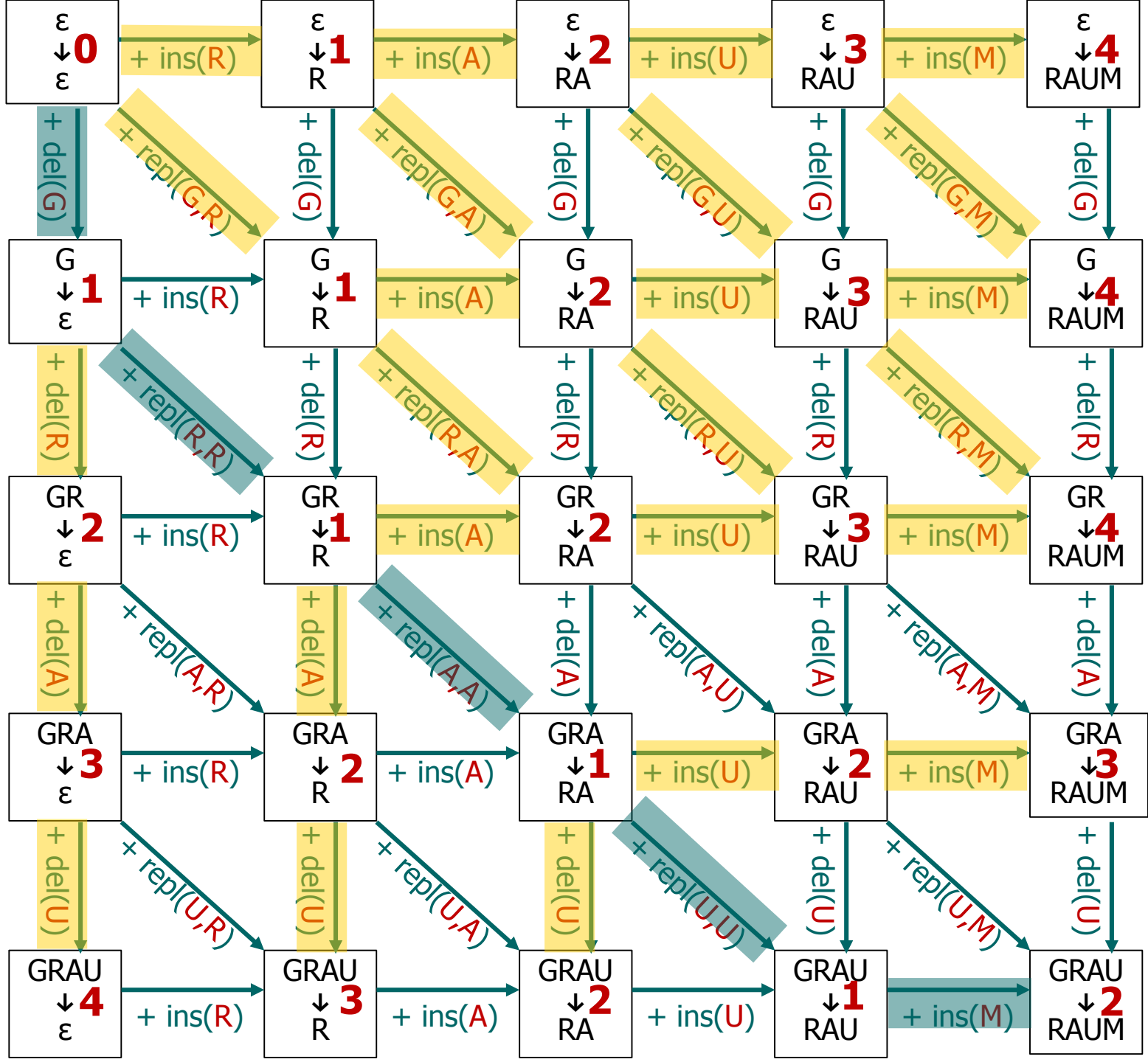
- Dynamische Programmierung
 - Wir legen eine Tabelle an für alle möglichen Kombinationen von Substrings und merken uns alle Einträge, die wir schon berechnet haben
 - Das braucht dann Laufzeit und Speicherplatz $O(n \cdot m)$
- Für die Visualisierung auf der nächsten Folie:
 - Operationen beziehen sich immer auf die letzte Position (Positionsindizes sind weggelassen)
 - Bei replace wird zusätzlich der alte Buchstabe dargestellt, damit man die Operativen ohne Kosten (z.B: `repl(A,A)`) einfacher erkennt.



Schnellerer Algorithmus

- Nun können wir von links oben nach rechts unten die Editierdistanz für jede Kombination aus Teilstrings berechnen

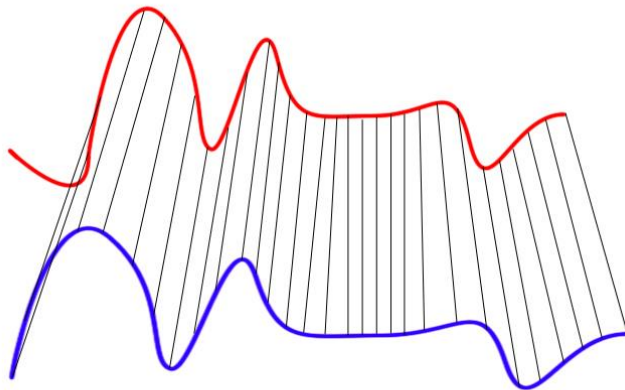
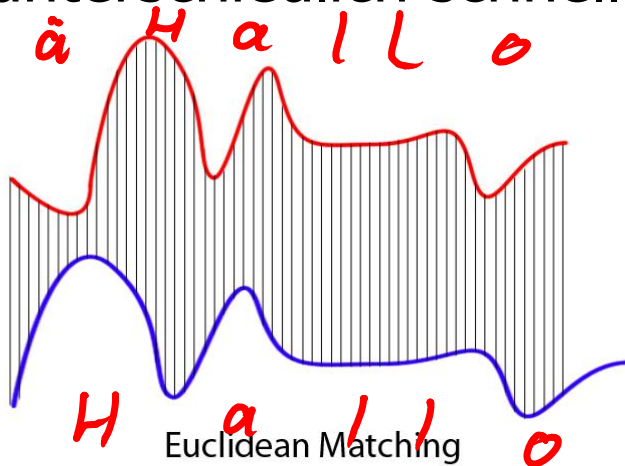
- Wie bekommen wir die Folge von Operationen?
 - Wir merken uns einfach bei jeder Anwendung der Rekursionsformel, welcher der vorherigen Einträge den kleinsten Wert ergeben hat (die hervorgehobenen Pfeile in unserem Bild)
 - Es kann von einem Eintrag mehrere hervorgehobene Pfeile zu den drei Einträgen davor geben
 - Wenn wir den hervorgehobenen Pfeilen von dem Eintrag bei (n, m) bis zum Eintrag für $(1, 1)$ folgen, bekommen wir eine optimale Folge von Operationen
 - Können wir unterwegs mehreren Pfaden folgen, gibt es entsprechend mehrere optimale Folgen
 - Diese Folgen sind nach Konstruktion alle monoton



- Das allgemeine Prinzip dazu
 - Rekursive Berechnung, bei der
 - ... dieselben Teilprobleme mehrmals auftauchen
 - ... die Gesamtzahl von Teilproblemen begrenzt ist
 - Dann Lösung für alle Teilprobleme berechnen
 - Und zwar nach und nach in solch einer Reihenfolge
 - ... dass sich noch nicht berechnete Lösungen aus schon berechneten zusammensetzen lassen
 - Zusammen mit dem "Wert" der optimalen Lösung erhält man so in der Regel auch den "Weg" dorthin
 - Dijkstras Algorithmus war vom Prinzip her auch dynamische Programmierung !

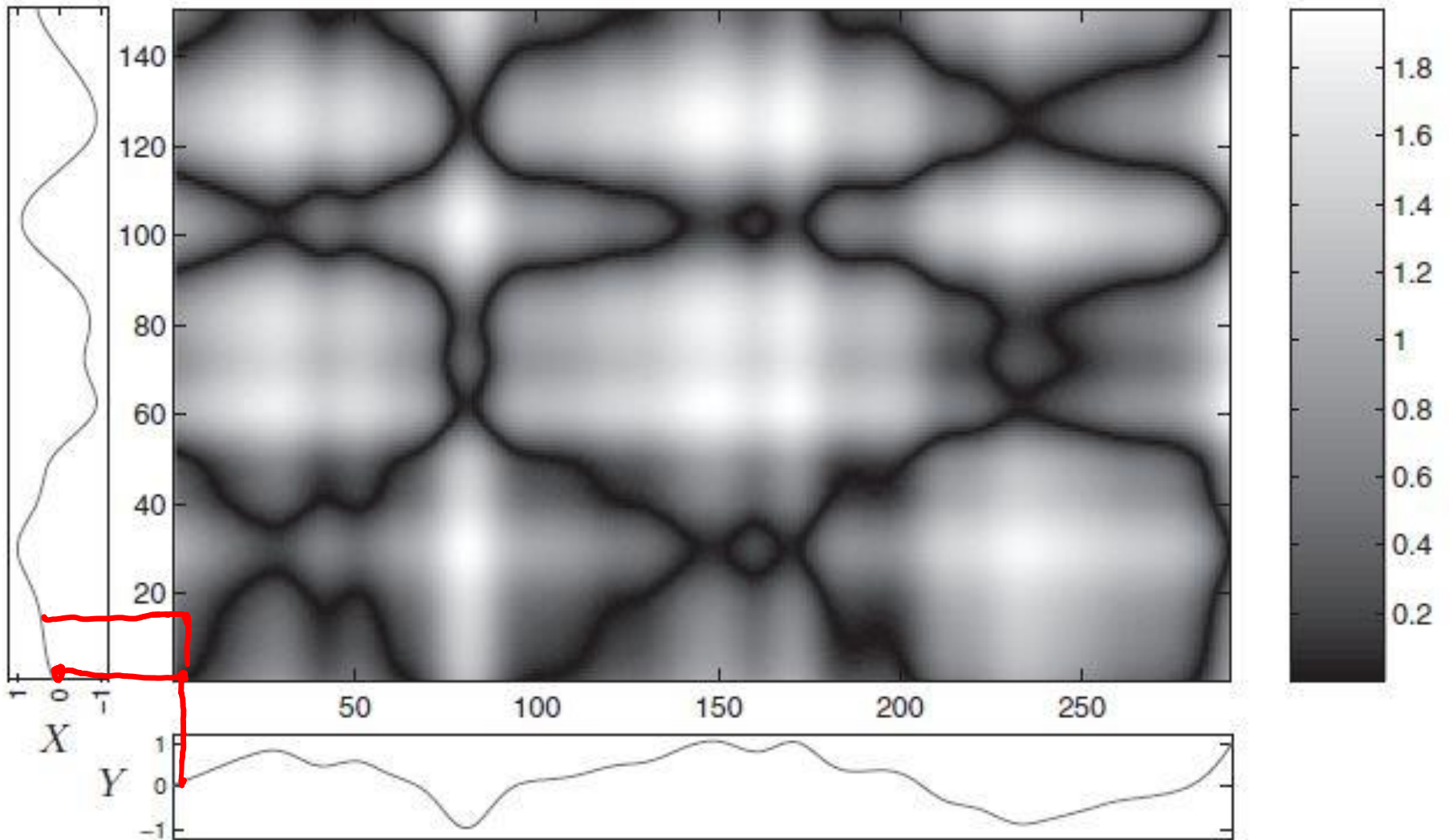
Weitere Anwendungen

- Dynamic Time Warping. Z.B. bei Spracherkennung unterschiedlich schnell gesprochene Worte matchen:

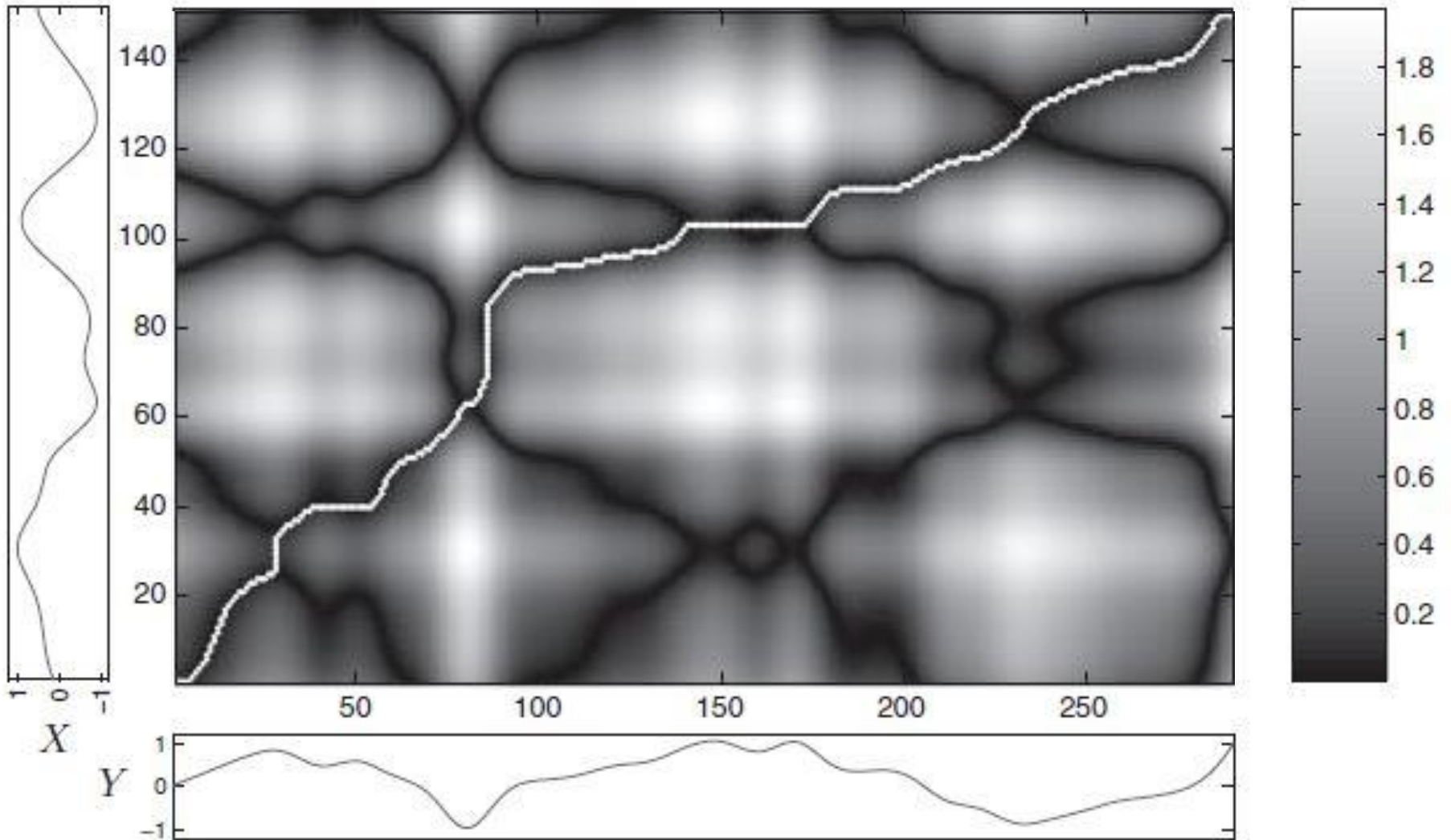


Dynamic Time Warping Matching

Dynamic Time Warping: Cost Matrix



Dynamic Time Warping: Cheapest Path



Literatur / Links

■ Dynamische Programmierung

- In Mehlhorn/Sanders:

12.3 Dynamic Programming

- In Wikipedia

http://en.wikipedia.org/wiki/Dynamic_programming

http://de.wikipedia.org/wiki/Dynamische_Programmierung

■ Editierdistanz

- In Wikipedia

http://en.wikipedia.org/wiki/Levenshtein_distance

<http://de.wikipedia.org/wiki/Levenshtein-Distanz>