Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2013 / 2014

Vorlesung 14, Donnerstag, 6. Februar 2014
(Editierdistanz, dynamische Programmierung)

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

[basiert auf der letztjährigen Veranstaltung von Prof. Dr. Hannah Bast, Lehrstuhl für Algorithmen und Datenstrukturen]

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- 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

Offizielle Evaluation der Vorlesung

UNI FREIBURG

- Bitte den Fragebogen **bis Sonntag, 9.2.** ausfüllen
  - Sie haben letzten Donnerstag eine Mail mit dem Link bekommen
  - Sie bekommen heute eine Erinnerungsmail ☺
  - Ich werde das Feedback dann nächste Woche (= letzte Vorlesung) zusammenfassen und besprechen !
  - Nehmen Sie sich bitte genug Zeit für das Ausfüllen
  - Die Freitextkommentare sind für uns am interessantesten
  - Seien Sie bitte ehrlich und möglichst konkret
  - **Abgabe bitte bis Sonntag diese Woche (9. Februar)**

Motivation

UNI FREIBURG

- 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 http://www.google.de/

    eyjaföllajaküll

    uniwersität verien 2014
  - Bioinformatik:
    Ähnlichkeit von DNA-Sequenzen

Editierdistanz 1/10

UNI FREIBURG

- 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

Editierdistanz 2/10

UNI FREIBURG

- 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 $1 \le i \le j \le |x|$
- Ein paar einfache Eigenschaften
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \ge abs(|x| - |y|)$ $\quad abs(x) = x > 0 \; ? \; x : -x$
  - $ED(x, y) \le ED(x[1..n-1], y[1..m-1]) + 1$ $\quad n = |x|, m = |y|$

Editierdistanz 3/10

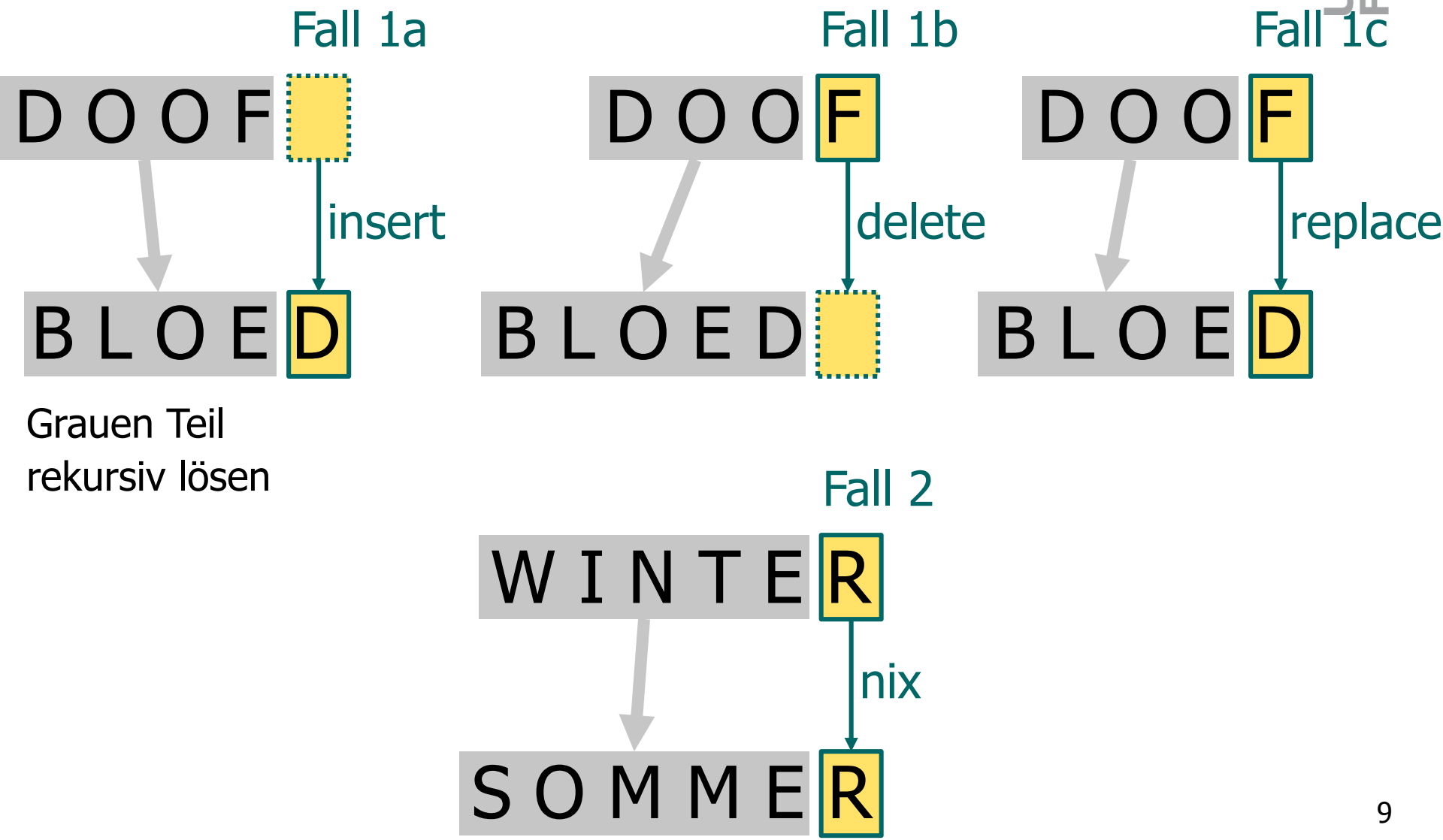
UNI FREIBURG

- Lösungsideen anhand von Beispielen
  - Für VERIEN → FERIEN ? ED = 1
  - Für MEXIKO → AMERIKA ? ED = 3
  - Für AAEBEAABEAREEAEBA → 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(GRAU, RAUM) = 2 aber ED(GR, RA) + ED(AU, UM) = 4
  - Auf ein "kleineres" Problem zurückführen?

    Das probieren wir jetzt !

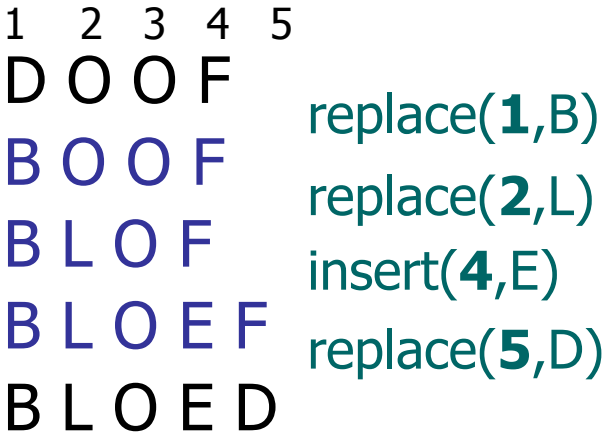
Rekursive Lösung: Letzte Operation betrachten

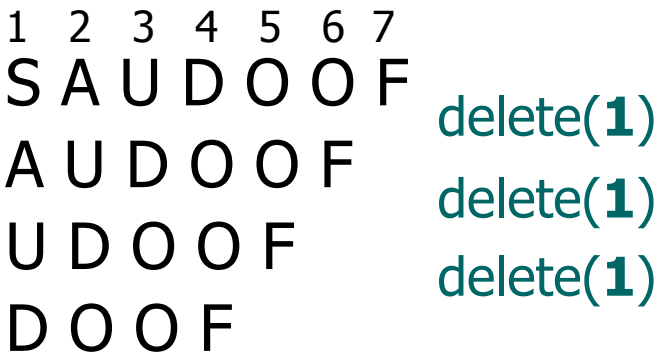


Editierdistanz 4/10

- Terminologie
  - Seien x und y unsere beiden Zeichenketten
  - Seien $\sigma_1, \ldots, \sigma_k$ 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 ≥ die Position von σ_i, wobei = nur dann erlaubt ist, wenn beides delete Operationen sind





Editierdistanz 5/10

UNI FREIBURG

- 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

Editierdistanz 6/10

1a: DOOF → SAUBLOE → SAUBLOED
1b: DOOF → SAUBLOEDF → SAUBLOED
1c: DOOF → SAUBLOEF → SAUBLOED

2: Sommer → Winter → Winter

- Fallunterscheidung
  - Wir betrachten die letzte Operation σ_k
    - $\sigma_1, \ldots, \sigma_{k-1} : x \rightarrow z$ und $\sigma_k : z \rightarrow y$
      (z.B. x = DOOF, z = SAUBLOEF, y = SAUBLOED)
    - Seien $n = |x|$ (4) und $m = |y|$ (8) und $m' = |z|$ (8)
    - 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: $\sigma_k = insert(m' + 1, y[m])$ [dann ist $m' = m - 1$]
    - Fall 1b: $\sigma_k = delete(m')$ [dann ist $m' = m + 1$]
    - Fall 1c: $\sigma_k = replace(m', y[m])$ [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
      $\sigma_1, \ldots, \sigma_k : x[1..n-1] \rightarrow y[1..m-1]$ und $x[n] = y[m]$

Editierdistanz 7/10

- Wir haben also einen dieser vier Fälle
  - Fall 1a (insert am Ende): $\sigma_1, \ldots, \sigma_{k-1} : x \rightarrow y[1..m-1]$
  - Fall 1b (delete am Ende): $\sigma_1, \ldots, \sigma_{k-1} : x[1..n-1] \rightarrow y$
  - Fall 1c (replace am Ende): $\sigma_1, \ldots, \sigma_{k-1} : x[1..n-1] \rightarrow y[1..m-1]$
  - Fall 2 (nichts am Ende): $\sigma_1, \ldots, \sigma_k : x[1..n-1] \rightarrow y[1..m-1]$
- 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

UNI FREIBURG

```
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"));
  }
}
```

EditDistanceMain.java

UNI FREIBURG

```
/**
 * 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));
  }
}
```

EditDistance.java

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

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

Editierdistanz 8/10

- Rekursives Programm
  - Für die Laufzeit würde folgende rekursive Formel gelten

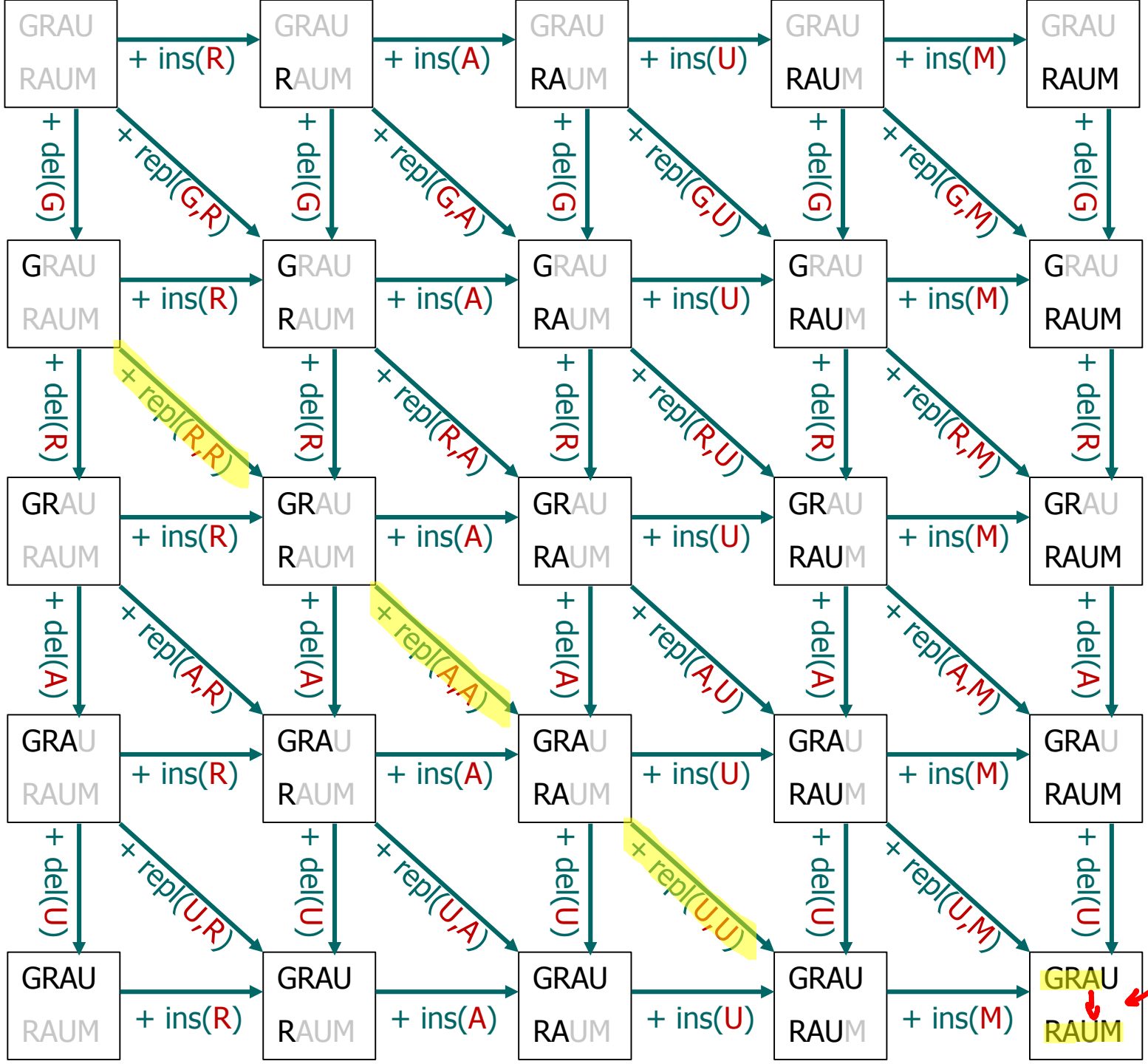
$$T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1$$
$$\geq T(n-1, m-1) + T(n-1, m-1) + T(n-1, m-1)$$
$$= 3 \cdot T(n-1, m-1)$$

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

Editierdistanz 9/10

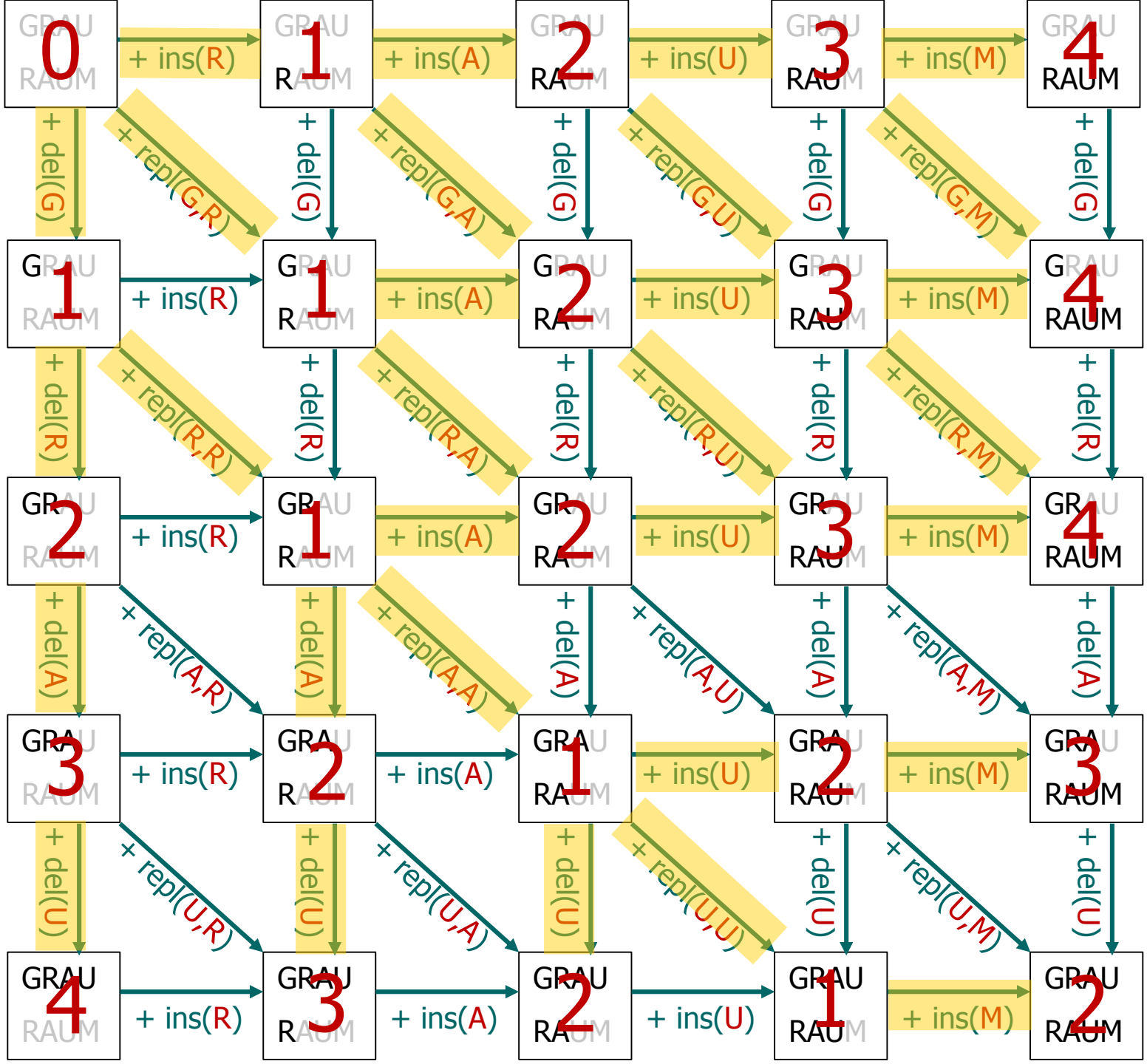
UNI FREIBURG

- 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 Operatinen ohne Kosten (z.B: repl(A,A)) einfacher erkennt.


GRAU RAUM
+ ins(R)
+ ins(A)
+ ins(U)
+ ins(M)
+ del(G)
+ repl(G,R)
+ repl(G,A)
+ repl(G,U)
+ repl(G,M)
GRAU RAUM
+ del(R)
+ repl(R,R)
+ repl(R,A)
+ repl(R,U)
+ repl(R,M)
GRAU RAUM
+ del(A)
+ repl(A,R)
+ repl(A,A)
+ repl(A,U)
+ repl(A,M)
GRAU RAUM
+ del(U)
+ repl(U,R)
+ repl(U,A)
+ repl(U,U)
+ repl(U,M)
GRAU RAUM
RAUMU
del(U)
RAUM

Schnellerer Algorithmus

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


0
+ ins(R)
1
+ ins(A)
2
+ ins(U)
3
+ ins(M)
4
+ del(G)
+ repl(G,R)
+ repl(G,A)
+ repl(G,U)
+ repl(G,M)
1
2
3
4
+ del(R)
+ repl(R,R)
+ repl(R,A)
+ repl(R,U)
+ repl(R,M)
2
3
4
+ del(A)
+ repl(A,R)
+ repl(A,A)
+ repl(A,U)
+ repl(A,M)
3
4
+ del(U)
+ repl(U,R)
+ repl(U,A)
+ repl(U,U)
+ repl(U,M)
4

Editierdistanz 10/10

UNI FREIBURG

- 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


0
+ ins(R)
1
+ ins(A)
2
+ ins(U)
3
+ ins(M)
4
+ del(G)
+ repl(G,R)
+ repl(G,A)
+ repl(G,U)
+ repl(G,M)
1
2
3
3
+ del(R)
+ repl(R,R)
+ repl(R,A)
+ repl(R,U)
+ repl(R,M)
2
1
2
3
4
+ del(A)
+ repl(A,R)
+ repl(A,A)
+ repl(A,U)
+ repl(A,M)
3
2
1
2
3
+ del(U)
+ repl(U,R)
+ repl(U,A)
+ repl(U,U)
+ repl(U,M)
4
3
2
1
2
+ ins(R)
+ ins(A)
+ ins(U)
+ ins(M)
+ del(G)

Dynamische Programmierung

UNI FREIBURG

- Das allgemeine Prinzip dazu
  - Rekursive Berechnung, bei der

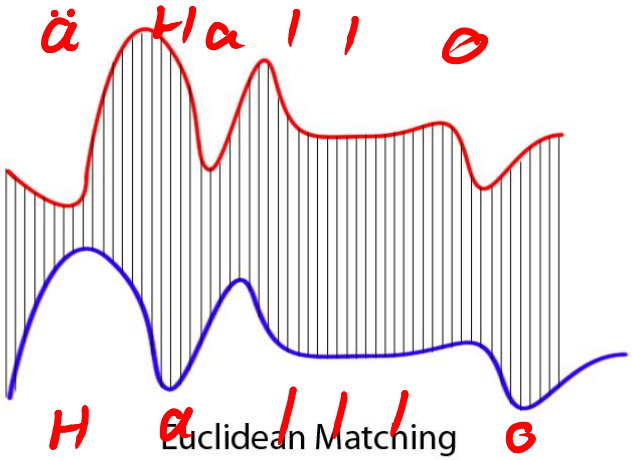
    … dieselben Teilprobleme mehrmals auftauchen

    … die Gesamtzahl von Teilproblemen begrenzt ist
  - Dann Lösung für alle Teilprobleme berechnen
  - Und zwar nach und nach in solch einer Reihenfolge

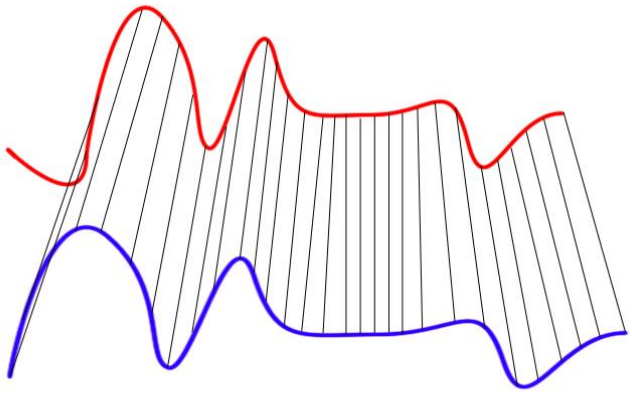
    … dass sich noch nicht berechnete Lösungen aus schon berechneten zusammensetzen lassen
  - Zusammen mit dem "Wert" der optimalen Lösung erhält man so in der Regel auch den "Weg" dorthin
  - Dijkstras Algorithmus war vom Prinzip her auch dynamische Programmierung !

Weitere Anwendungen

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

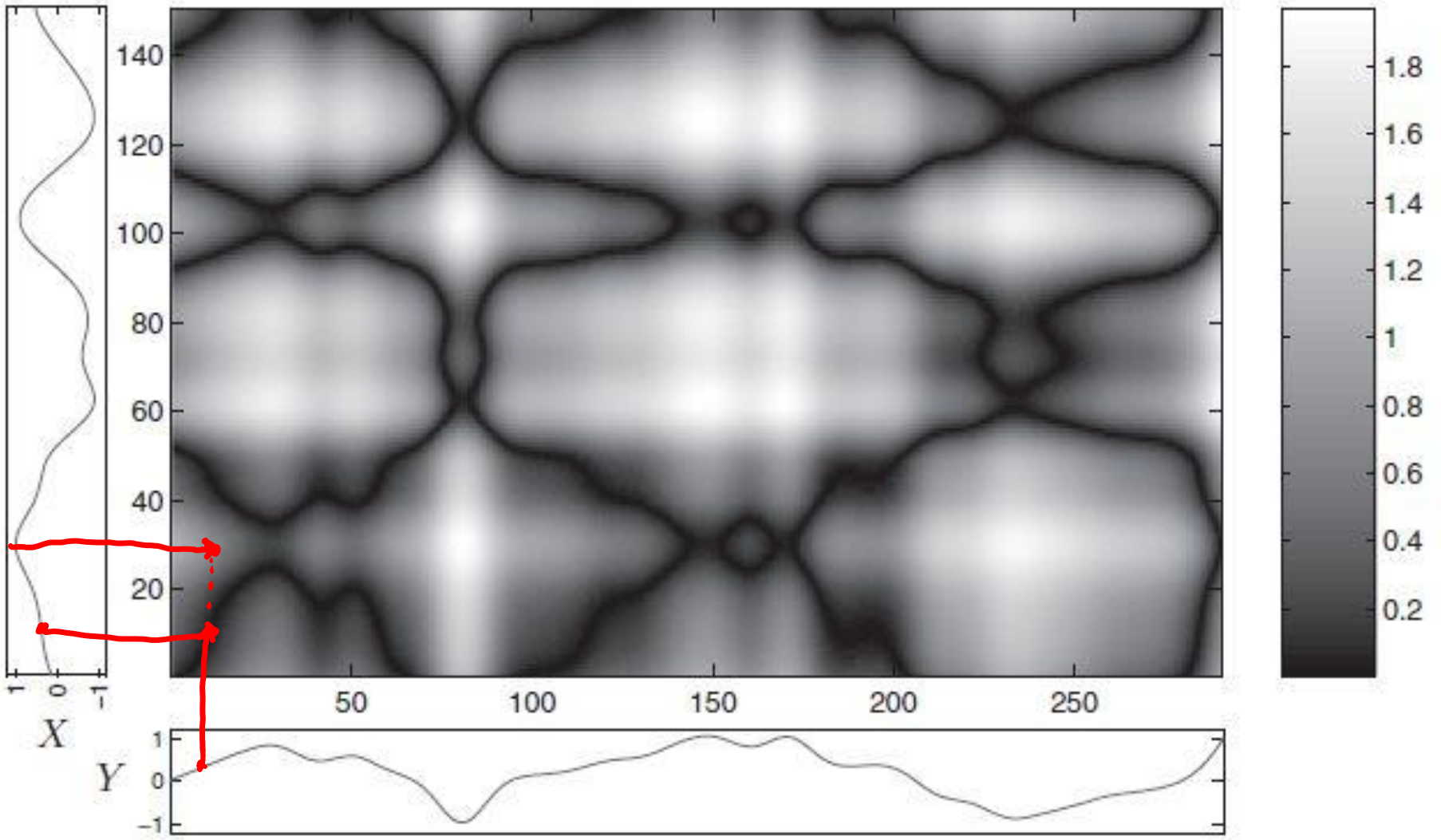


Euclidean Matching


Dynamic Time Warping Matching

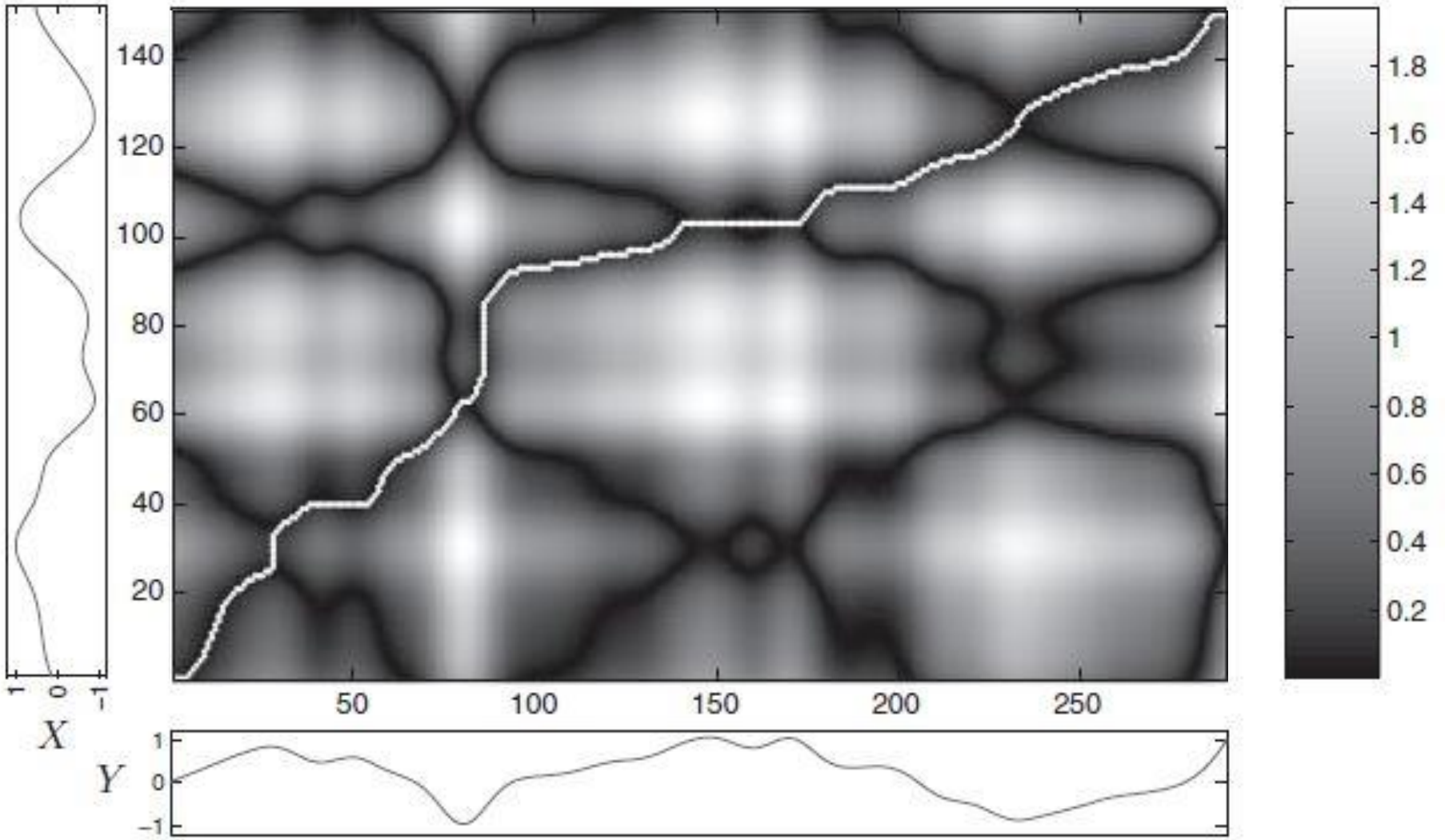
[http://commons.wikimedia.org/wiki/File:Euclidean_vs_DTW.jpg]

Dynamic Time Warping: Cost Matrix



http://commons.wikimedia.org/wiki/File:Cost_matrix.jpg

Dynamic Time Warping: Cheapest Path



http://commons.wikimedia.org/wiki/File:Cost_matrix.jpg

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