Informatik II: Algorithmen und Datenstrukturen SS 2015

Vorlesung 11b, Mittwoch, 8. Juli 2015

(Editierdistanz, Teil 2)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Inhalt
  - Editierdistanz, Teil 2

    Rekursives Programm … Fortsetzung von gestern

    Iteratives Programm … Algorithmus, Beispiel, Verfeinerung

    Dynamische Programmierung … allgemeines Prinzip dahinter
  - Ü11, Aufgabe 2: Implementieren des iterativen Algorithmus

Rekursive Implementierung 1/5

UNI FREIBURG

- Rekursive Formel (Wiederholung von gestern)
  - Für $|x| > 0$ und $|y| > 0$ ist:

    $ED(x, y)$ = das Minimum von
    - $ED(x[1..n], y[1..m-1]) + 1$
    - $ED(x[1..n-1], y[1..m]) + 1$
    - $ED(x[1..n-1], y[1..m-1] + \delta$

      wobei $\delta = 1$ falls $x[n] \neq y[m]$, sonst $\delta = 0$
  - Für $|x| = 0$ ist $ED(x, y) = |y|$
  - Für $|y| = 0$ ist $ED(x, y) = |x|$

Rekursive Implementierung 2/5

besser: INS(5,M), INS(6,A), INS(7,N)
INS(8,N)

- Wofür brauchen wir die Monotonie
  - Wir haben uns die Operation angeschaut, die etwas "am Ende" der Zeichenkette ändert (falls überhaupt)
  - Wir haben zudem benutzt, dass die übrigen Operationen den Teil strikt "vor dem Ende" transformieren
  - Unsere Definition von Monotonie ist einfach eine natürliche Art, das formal zu machen

    Die Operation "am Ende" ist dann einfach die letzte

    Die Operationen davor transformieren den Teil davor ...
    dafür wichtig, dass gleiche Position nur bei delete erlaubt

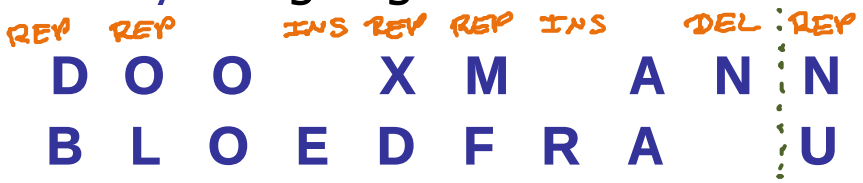
1 2 3 4
DOOF → DOOFN → DOOFNN → DOOFANN → DOOFMANN

INS(5,N) INS(5,N) INS(5,A) INS(5,M)

NICHT monoton in unserem Sinne
(= nur bei DELETE erlaubt)

Rekursive Implementierung 3/5

- Alternative Sichtweise
  - Visualisieren einer Folge von Operationen, indem man x und y mit geeigneten "Lücken" untereinander schreibt



ED = 8

insert = oben leer, darunter ein Buchstabe

delete = unten leer, darüber ein Buchstabe

replace = zwei ungleiche Buchstaben übereinander

Wenn man da jetzt von links nach rechts durchgeht, hat man wieder genau eine monotone Folge

Die rekursive Formel unterscheidet die vier Möglichkeiten in der letzten Spalte (insert, delete, replace, nix)

Rekursive Implementierung 4/5

- Laufzeit
  - Für die Laufzeit gilt folgende rekursive Formel

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

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

    Also **exponentielle** Laufzeit

$n = |x|$, $m = |y|$

$$T(n,m) \geq \underbrace{T(n-1,m)}_{\geq T(n-1,m-1)} + \underbrace{T(n,m-1)}_{T(n-1,m-1)} + T(n-1,m-1) + A$$

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

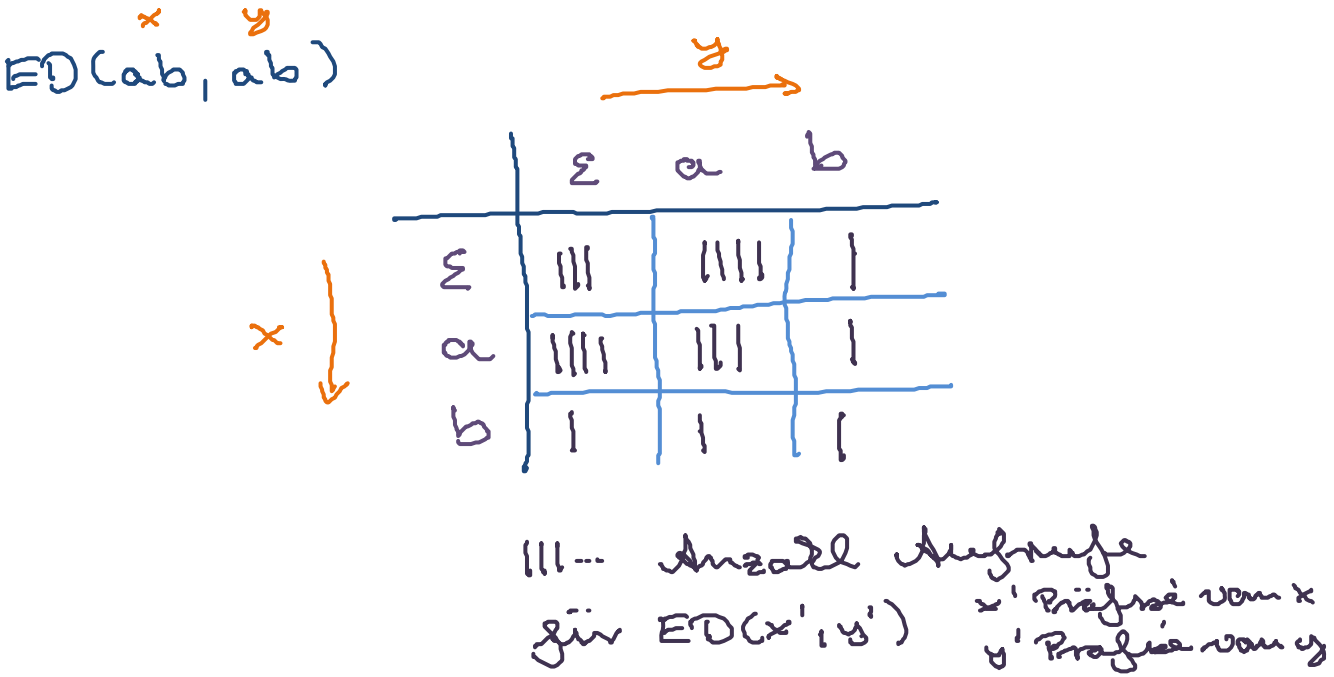
$$\geq 3 \cdot [3 \cdot T(n-2,n-2) + A] + A$$

$$\geq 3^n \cdot T(0,0)$$

Rekursive Implementierung 5/5

- Problem
  - Das rekursive Programm berechnet die gleichen ED Werte immer und immer wieder

    Hatten wir schon mal, in Vorlesung 1b beim rekursiven Programm zur Berechnung der Fibonacci Zahlen



Iterative Implementierung 1/10

UNI FREIBURG

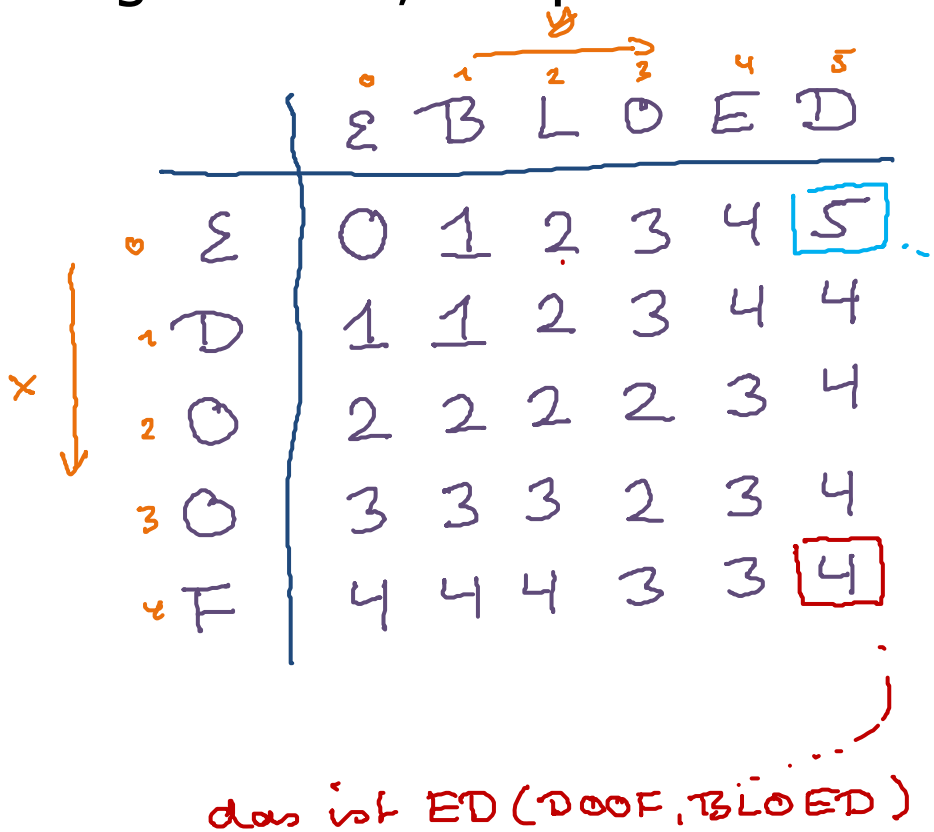
- Algorithmus, Idee
  - Wir berechnen jedes ED(x', y'), wobei x' Präfix von x und y' Präfix von y, nur **einmal** und merken es uns

    Das sind insgesamt $(|x| + 1) \cdot (|y| + 1)$ Werte

  - Wenn wir die in der richtigen Reihenfolge berechnen, haben wir immer alle schon, die wir laut Rekursion brauchen

Iterative Implementierung 2/10

- Algorithmus, Beispiel



Der Eintrag an Stelle i, j ist gerade ED(x', y')

$x' = x[0..i]$

$y' = y[0..j]$

das ist z.B. ED(Σ, BLOED)

Die erste Spalte/Zeile ist der Basisfall unserer Rekursion

Iterative Implementierung 3/10

UNI FREIBURG

- Laufzeitanalyse
  - Jeder Eintrag kann in Zeit $O(1)$ berechnet werden

    Für Zeile oder Spalte 0 trivial

    Sonst Minimum aus den drei benachbarten Einträgen
  - Es gibt genau $(|x| + 1) \cdot (|y| + 1)$ Einträge
  - Also insgesamt $\Theta(|x| \cdot |y|)$ Zeit
  - Und ebenso $\Theta(|x| \cdot |y|)$ Platz

    Es geht aber auch noch schneller und mit weniger Platz, siehe Folien 13 – 17

Iterative Implementierung 4/10

UNI FREIBURG

- Wie kommt man zu der Folge der Operationen
  - Man merkt sich bei der Berechnung des Minimums der drei benachbarten Einträge, über welche das Minimum erzielt wurde

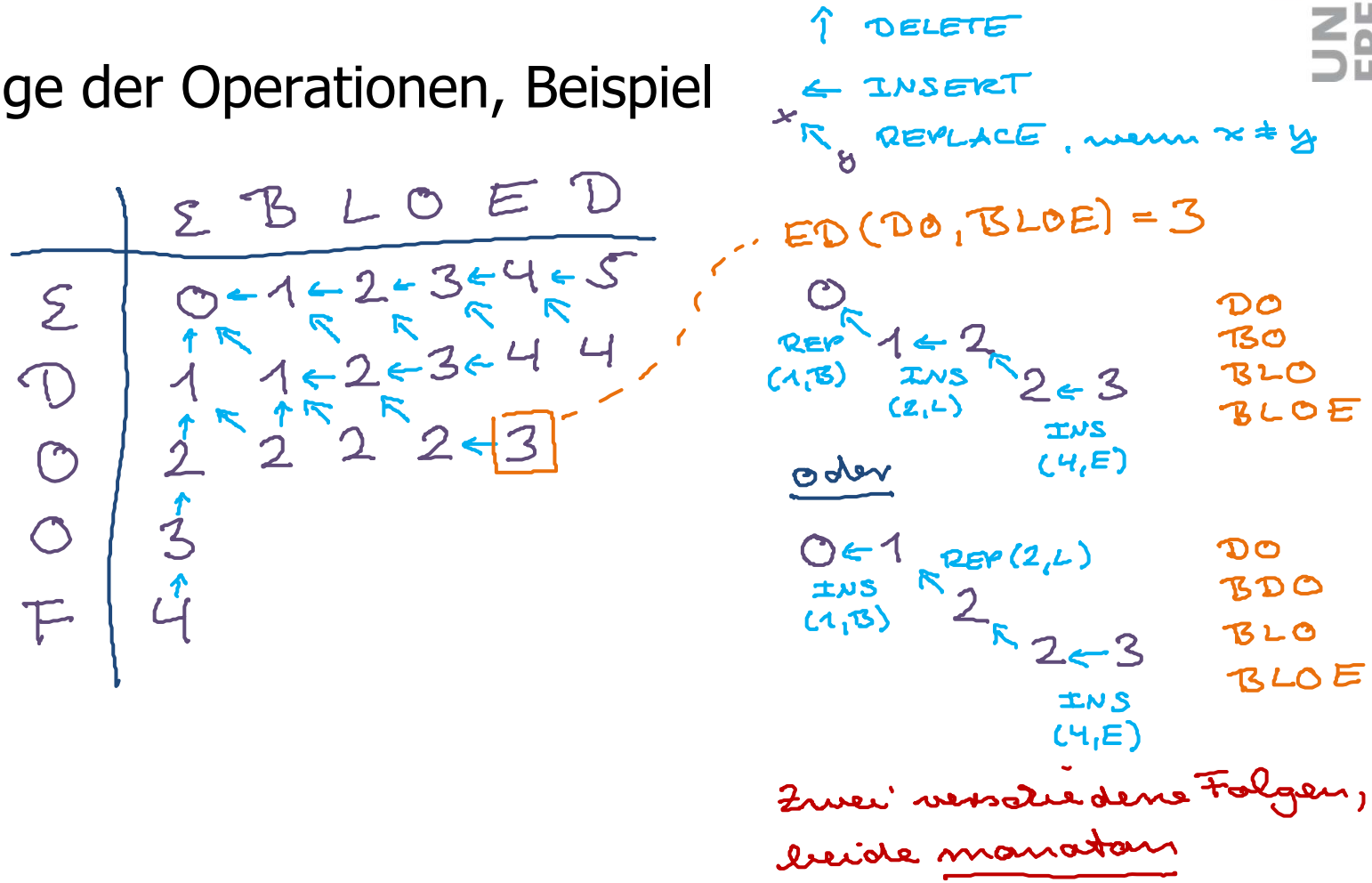
    Das können mehrere sein (eins, zwei oder drei)
  - Jeder Pfad von "unten rechts" (Eintrag bei |x|, |y|) nach "oben links" (Eintrag bei 0, 0) gibt einem dann eine mögliche Folge von Operationen
  - Man erhält so **alle** optimalen **monotonen** Folgen

    Das schauen wir uns jetzt anhand unseres Beispiels an

    Berechnung der Op-Folge ist eine der Zusatzaufgaben

Iterative Implementierung 5/10

- Folge der Operationen, Beispiel



Iterative Implementierung 6/10

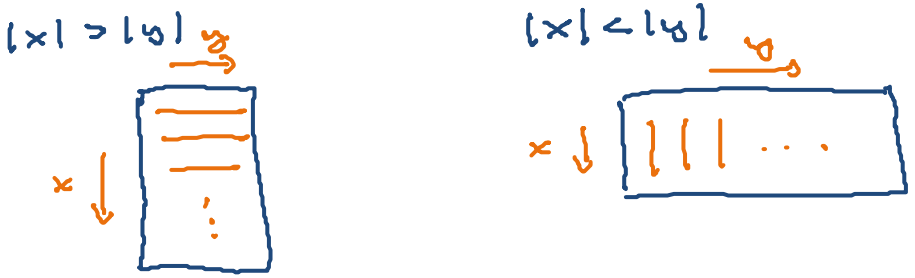
UNI FREIBURG

- Weniger Platz
  - Beobachtung: am Ende interessiert einen nur der Eintrag "unten rechts" (an Stelle |x|, |y|)
  - Um den zu berechnen kann man auch Zeile für Zeile vorgehen, und sich nur die jeweils letzte Zeile merken

    Oder analog Spalte für Spalte berechnen, und sich dabei immer nur die jeweils letzte Spalte merken
  - Das benötigt dann nur Platz O(min(|x|, |y|))

    Falls |x| > |y| zeilenweise, sonst spaltenweise

dann hat man allerdings (ohne Weiteres) keine Pfade mehr

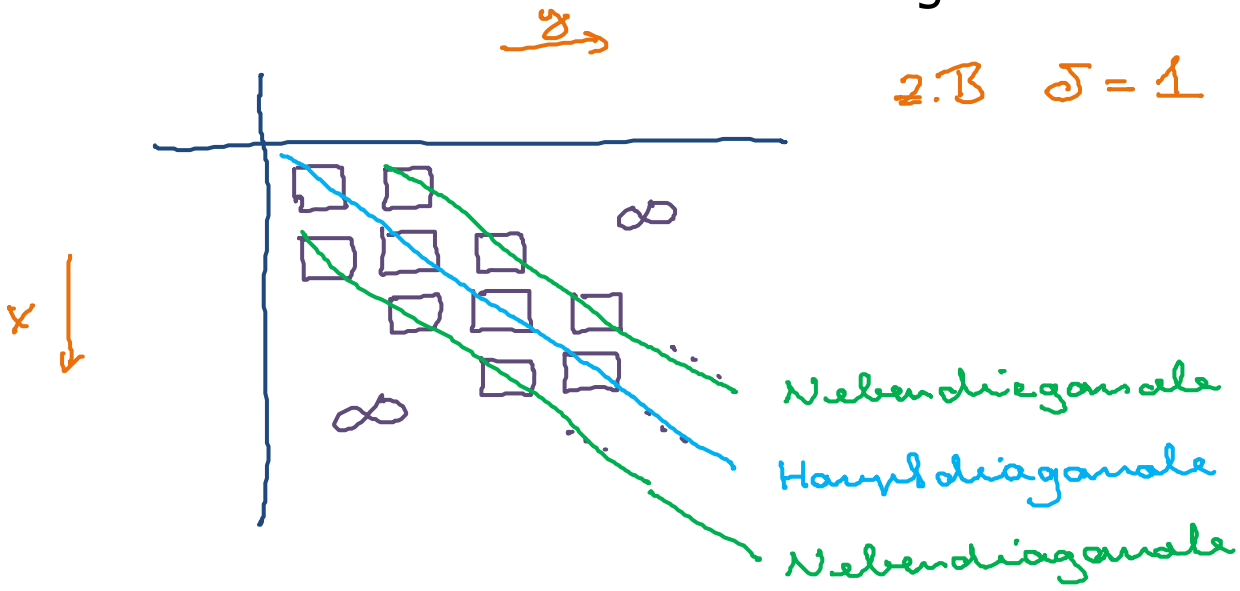


Iterative Implementierung 7/10

UNI FREIBURG

- Bessere Laufzeit, Idee
  - Nehmen wir erstmal an, wir wüssten, dass ED(x, y) ≤ δ
  - Dann reicht es, die Einträge auf der Hauptdiagonalen und den δ Nebendiagonalen darüber und darunter zu berechnen

    Alle anderen Werte können auf ∞ gesetzt werden



Iterative Implementierung 8/10

UNI FREIBURG

- Bessere Laufzeit, Korrektheit
  - Alle Pfade aus dem Ursprungstableau, die ganz innerhalb dieses "Streifens" bleiben, finden wir auch weiterhin
  - Nehmen wir an, es gibt einen optimalen Pfad, der diesen Streifen verlässt

    Dann geht er > δ mal nach oben (je ein delete) oder > δ mal nach links (je ein insert)

    In beiden Fällen wäre die ED > δ, aber wir hatten ja gerade angenommen, dass ED ≤ δ

Iterative Implementierung 9/10

UNI FREIBURG

- Bessere Laufzeit, Analyse
  - Die Laufzeit ist dann $\Theta(\min\{|x|, |y|\} \cdot \delta)$

    Die Länge der Hauptdiagonale ist gerade $\min\{|x|, |y|\}$

    Die ∞ außerhalb des Streifens brauchen wir ja nicht explizit hinzuschreiben
  - Man bekommt auch leicht Platz $\Theta(\min\{|x|, |y|\} \cdot \delta)$

    Einfach ein Feld der Größe $\min\{|x|, |y|\}$ pro Diagonale

    Die Indices der Nachbarn lassen sich dann immer noch leicht berechnen

Iterative Implementierung 10/10

UNI FREIBURG

- Bessere Laufzeit, noch besser
  - Die beschriebene Algorithmus nimmt an, dass wir ein δ kennen, so dass $ED(x, y) \leq \delta$
  - Schöner wäre Laufzeit und Platz $\Theta(\min\{|x|, |y|\} \cdot ED(x, y))$

    Egal wie klein oder groß ED(x, y) ist

    Das ist eine der Zusatzaufgaben für das Ü11

    Hinweis: probieren Sie eine geeignete Auswahl von Werten für δ aus

Dynamische Programmierung 1/4

UNI FREIBURG

- Allgemeines Prinzip
  - Man hat eine **rekursive** Formel, bei der dieselben Teilprobleme mehrfach vorkommen (in verschiedenen rekursiven Aufrufen)
  - Man berechnet dann **iterativ**, in einer systematischen Reihenfolge, die Lösung aller relevanten Teilprobleme
  - Zusammen mit dem "Wert" der optimalen Lösung erhält man so immer auch den "Weg" dorthin
  - Der Name "dynamische Programmierung" ist ziemlich irreführend und hat wenig mit dem Grundprinzip zu tun
  - Dijkstras Algorithmus war auch ein (verkapptes) Beispiel für dynamische Programmierung ... siehe Folie 21

Dynamische Programmierung 2/4

UNI FREIBURG

- Warum "dynamische Programmierung"
  - Das Prinzip wurde im informatischen Kontext erstmals beschrieben von Richard Bellman in den 1950er Jahren
  - Die Benennung ist **sehr** merkwürdig
  - Aber es gibt einen historischen Grund dafür

    Er hatte beim Militär gearbeitet und sein Chef hatte eine extreme Abneigung gegen Mathematik und Forschung

    Sie haben aber viel Planung gemacht, und die Pläne hießen damals "programs"

    Also nannte Bellman es "dynamic programming", um den mathematischen Charakter zu verschleiern

Dynamische Programmierung 3/4

UNI FREIBURG

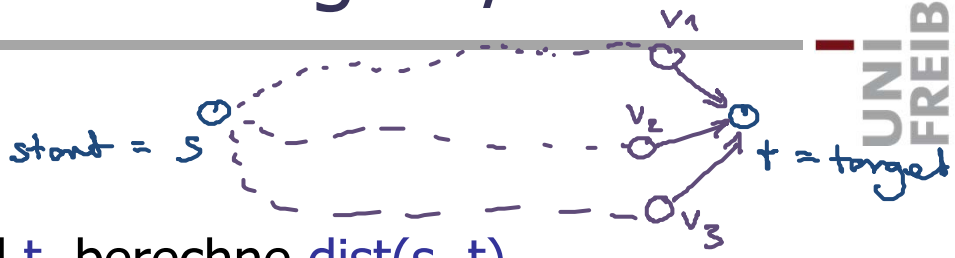
- Beispiele
  - Edi-Tier Distanz haben wir gerade gesehen
  - Die Fibonacci Zahlen kann man auch mit "dynamischer Programmierung" berechnen

    Einfach iterativ der Reihe nach ... und man muss sich dann immer nur die letzten beiden merken
  - Dijkstras Algorith. ist auch dynamische Programmierung

    Warum sehen wir auf der nächsten Folie
  - In der Freiburger Bioinformatik wird sehr viel mit dynamischer Programmierung gemacht

    "Alignment" von DNA/RNA-Sequenzen = Zeichenketten

Dynamische Programmierung 4/4



- Dijkstras Algorithmus
  - **Ziel:** für gegebene s und t, berechne dist(s, t)
  - Rekursive Formel: dist(s, t) = min dist(s, v) + cost(v, t)

    Minimum über alle nach t eingehenden Kanten (v, t)

    Würde man das trivial rekursiv programmieren, hätte man das gleiche Problem wie beim rekursiven Programm für ED
  - Stattdessen macht man es **iterativ**, in der Reihenfolge aufsteigender Entfernung (bezüglich dist Wert) von s

    Die Reihenfolge ist bei Dijkstra allerdings aufwändiger zu realisieren (man braucht eine PW) als bei ED

Literatur / Links

UNI FREIBURG

- 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