Informatik II: Algorithmen und Datenstrukturen SS 2017

Vorlesung 11b, Mittwoch, 12. Juli 2017

(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

- Drumherum
  - Evaluation — **Letzte Erinnerung**
  - Kohlenstoff — Woher kommt er?
- Inhalt
  - Rekursives Programm — Fortsetzung von gestern
  - Iteratives Programm — Algorithmus + Beispiel
  - Verfeinerung — bessere Laufzeit + Platz
  - Dynamische Programmierung — allgemeines Prinzip
  - ÜB11, Aufgabe 2: beschreiben Sie einen Algorithmus, der Laufzeit $O(\min\{|x|, |y|\} \cdot ED(x, y))$ erzielt + Begründung

Kohlenstoff 1/5

- Woher kommt er? Ihre Antworten
  - "Wenn sich zwei Sterne ganz arg lieb haben"
  - "Mama hat immer gesagt, dass ich aus Sternenstaub bin"
  - "Man sollte sich allerdings nichts darauf einbilden, so ziemlich alles ist aus Sternenstaub"
  - "Vom Großhandel"
  - "Wahrscheinlich aus China"
  - "Durch den Kapitalismus der USA bekommen wir täglich viel frischen Kohlen(Dollar)stoff"
  - "Einem geschenkten Gaul schaut man nichts ins Maul"
  - Quark-Gluonen Plasma, Tri-Alpha-Prozess, …

Kohlenstoff 2/5

- Beginn des Universums … heiß, sehr heiß 
  - Ganz am Anfang (falls es einen gab), war da erstmal nur eine Suppe aus kleinsten Teilchen (Quark-Gluonen Plasma)

    Die Anzahl der Quarks ist seitdem vermutlich fix
  - Nach ca. 1 Mikrosekunde haben sich Protonen (**P**) und Neutronen (**N**) gebildet … Elektronen (**E**) waren schon da

    Proton: 2 Up + 1 Down Quark, Neutron: 1 Up + 2 Down
  - Nach ca. 3 Minuten haben sich die ersten **Atomkerne** gebildet, aber erstmal fast nur ganz kleine:

    Wasserstoff-Kern: 1 P, Helium-4-Kern: 2P, 2N

    Auch ein paar wenige Isotope (Deuterium-Kern: 1P, 1N oder Helium-3-Kern: 2P, 1 N) und noch größere Kerne

Kohlenstoff 3/5

- Recombination
  - Solange es noch sehr heiß war, waren die Elektronen einfach zu schnell, um von Atomkernen eingefangen zu werden
  - Ca. 380.000 Jahre nach dem Big Bang dann kühl genug
  - Jetzt konnten sich halbwegs stabil die ersten Atome bilden, aber auch erstmal nur die ganz leichten

    Wasserstoff: 1 P + 1 E (knapp 75%)

    Helium-4: 2 P + 2 N + 2 E (knapp 25%)
  - Auch ein paar Isotope (Deuterium, Helium-3) und sehr sehr wenige schwerere Atome wie Lithium (3P) o. Beryllium (4P)

    Auch schon Kohlenstoff (6P), aber nur klitzeklitzekleine Mengen

Kohlenstoff 4/5

- Kernfusion in Sternen
  - Falls genügend H auf einem Haufen, kommt es zum Gravitationskollaps … und zwar genau dann wenn:

    Gravitationsenergie > 2 · kinetische Energie
  - Durch den Gravitationsdruck setzt irgendwann Kernfusion ein: je zwei H-Atome fusionieren zu Helium

    Diese "Hauptsequenz" hat bei unserer Sonne vor ca. 4.6 Milliarden Jahre angefangen und dauert noch mal so lange

    Tipp: Sie sollten also in den nächsten 5 Milliarden Jahren mit dem Studium fertig werden

Kohlenstoff 5/5

- Das Ende (von einer / unserer Sonne)
  - Irgendwann (in besagten 5 Milliarden Jahren) ist der ganze Wasserstoff in der Sonne zu Helium fusioniert
  - Das wabert jetzt noch ca. 120 Millionen Jahre vor sich hin und wird dabei immer heißer (i.W. durch den Quantendruck)
  - Erst ab ca. 100 Millionen Grad Celsius ist es dann heiß genug, dass endlich auch die Helium-Kerne miteinander fusionieren
  - Die fusionieren dann im Tri-Alpha-Prozess zu **Kohlenstoff**

    He + He + He → C … auch He + He → Be, aber instabiler

    Passiert blitzartig ("Helium-Flash") in gerade mal **3 Minuten**
  - Das ist, wo der meiste Kohlenstoff herkommt

Iterative Implementierung 1/10

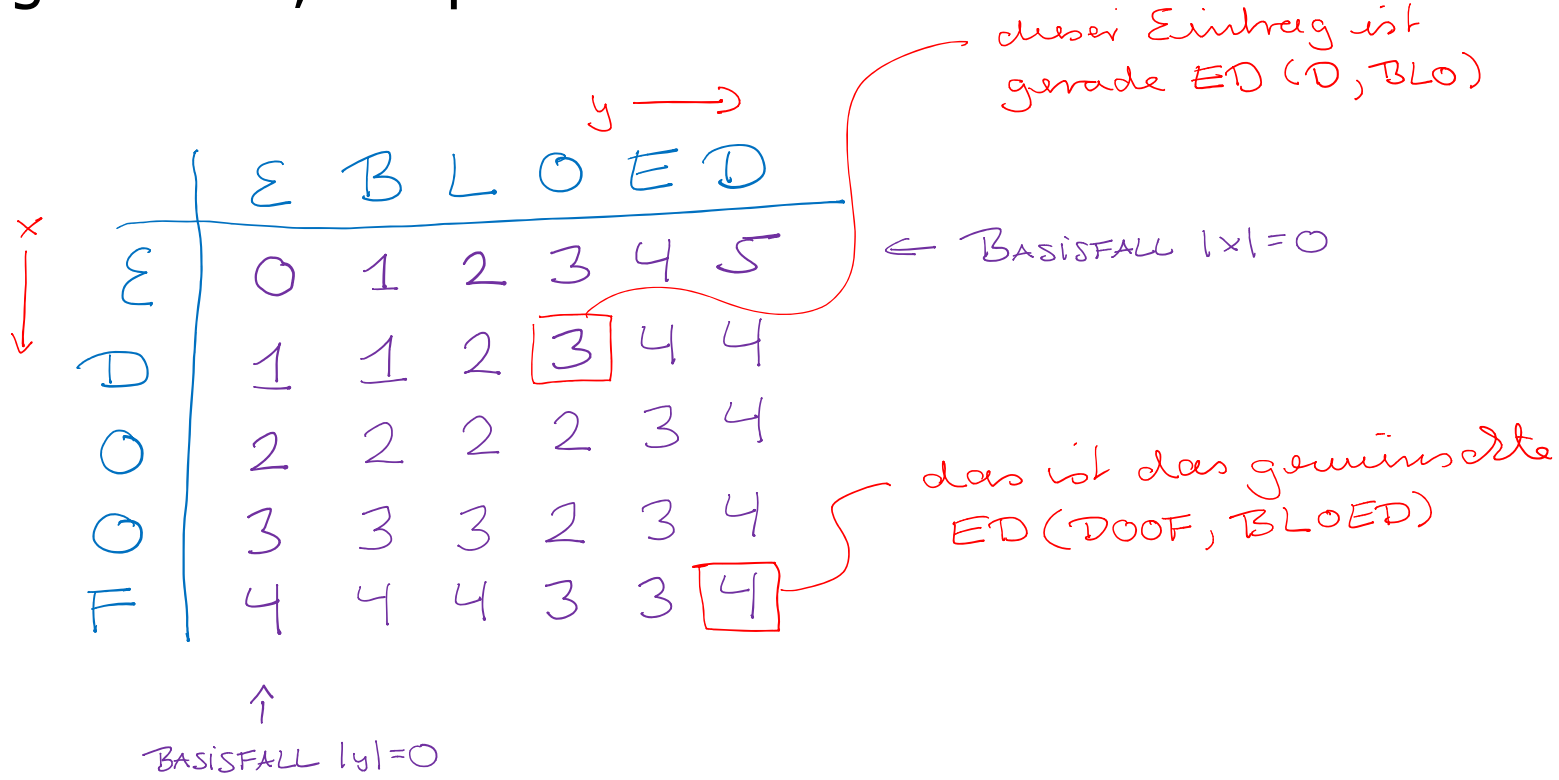
UNI FREIBURG

- 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

    Es gibt |x| + 1 Präfixe von x, nicht nur |x| … das + 1 ist für das leere Wort … dasselbe für y
  - Wenn wir die Präfix-EDs **in der richtigen Reihenfolge** berechnen, haben wir immer schon alle, die wir gemäß der rekursiven Formel brauchen

Iterative Implementierung 2/10

- Algorithmus, Beispiel



		E	B	L	O	E	D
	E	0	1	2	3	4	5
x	D	1	1	2	3	4	4
	O	2	2	2	2	3	4
	O	3	3	3	2	3	4
	F	4	4	4	3	3	4

Iterative Implementierung 3/10

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

    Für Zeile oder Spalte 0 ist das 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 und ÜB11, Aufgabe 2

Iterative Implementierung 4/10

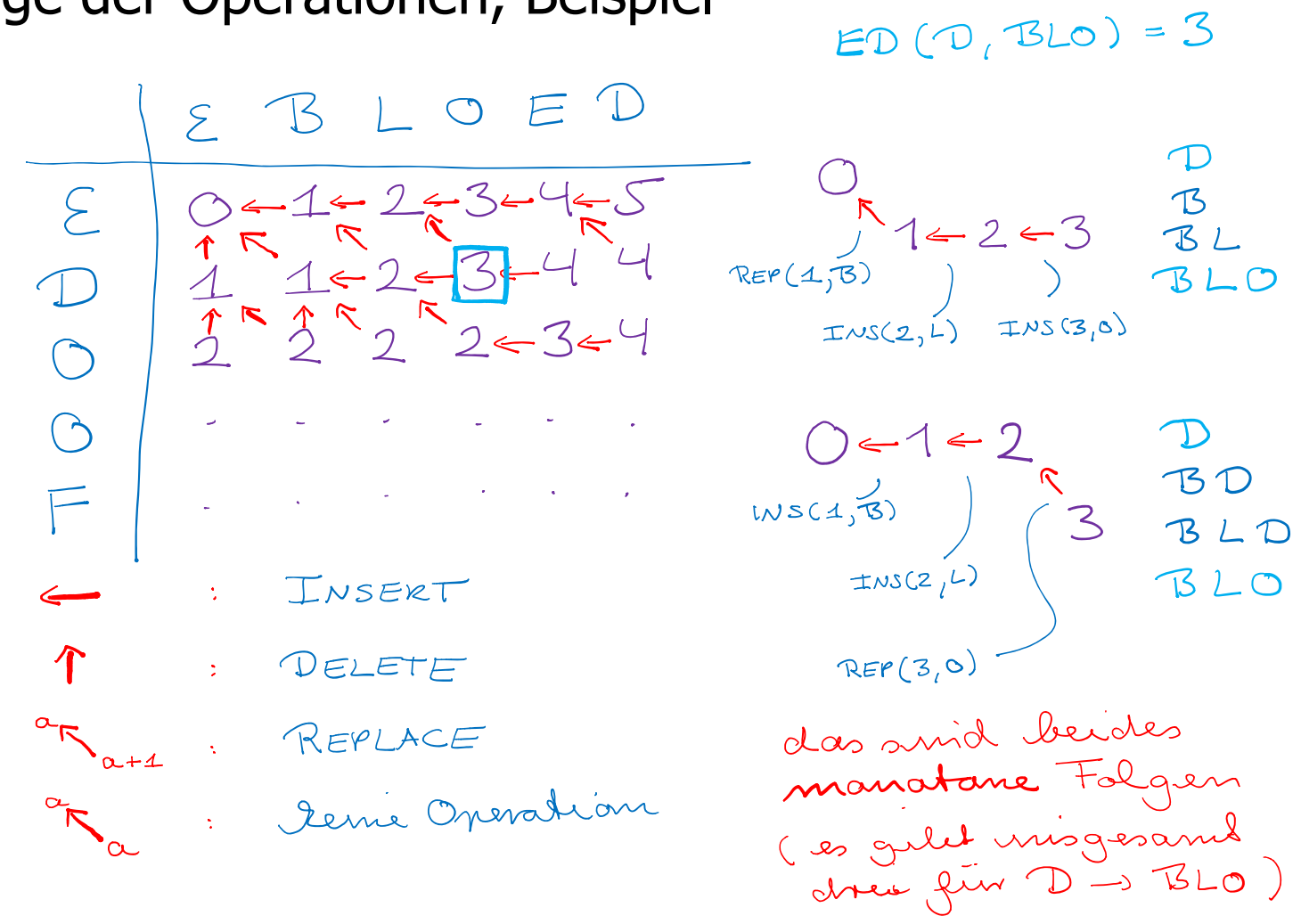
- Wie kommt man zu der Folge der Operationen
  - Man merkt sich bei der Berechnung des Minimums der drei benachbarten Einträge, über welche das Minimum erzielt wurde

    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

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|))**

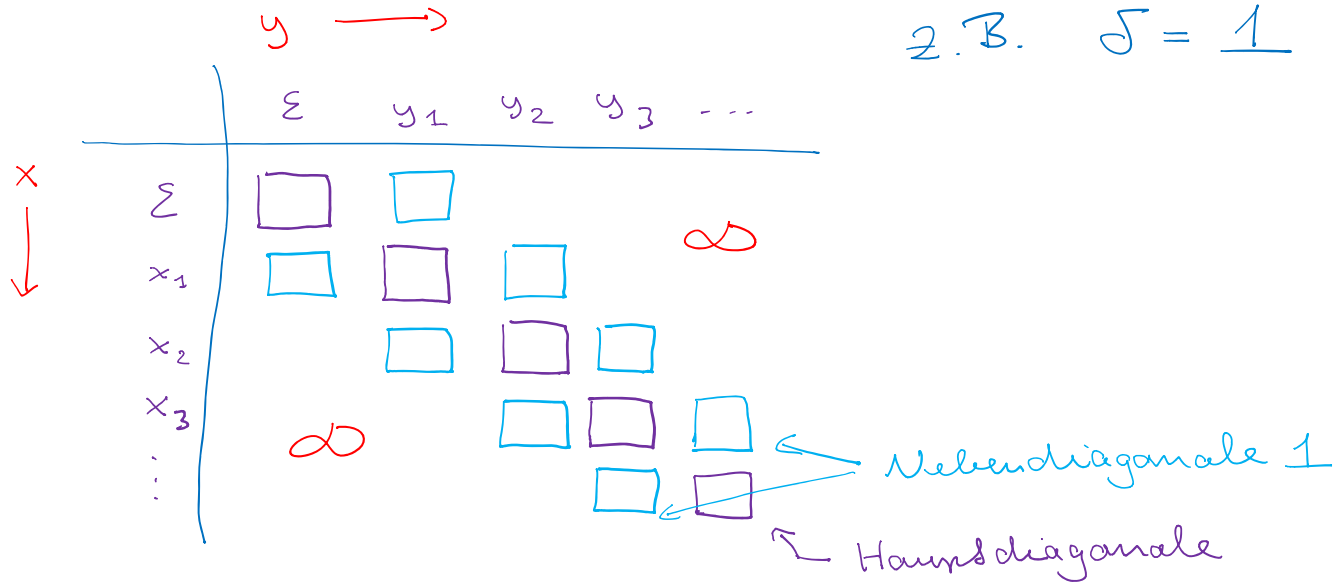
    Falls |x| > |y| zeilenweise, sonst spaltenweise

    Dann hat man allerdings die Pfadinformation nicht mehr

Iterative Implementierung 7/10

- Bessere Laufzeit, Idee
  - Nehmen wir 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

- 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 ED > δ, aber wir hatten ja gerade angenommen, dass ED ≤ δ

Iterative Implementierung 9/10

- 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 Indizes der Nachbarn lassen sich dann immer noch leicht berechnen

Iterative Implementierung 10/10

- Bessere Laufzeit, noch besser
  - Die beschriebene Algorithmus nimmt an, dass wir ein δ kennen, so dass ED(x, y) ≤ δ
  - Schöner wäre Laufzeit und Platz:

    Θ(min{|x|, |y|} · ED(x, y))

    Also proportional zur Editierdistanz, ohne vorher irgendetwas darüber zu wissen, wie groß sie ist

    Das ist Aufgabe 2 vom ÜB11

    Hinweis: probieren Sie eine geeignete Auswahl von Werten für δ aus, aber nicht einfach 1, 2, 3, 4, …

für $x \neq y$
(für $x = y$ ist das Problem trivial und $ED(x,y) = 0$)

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

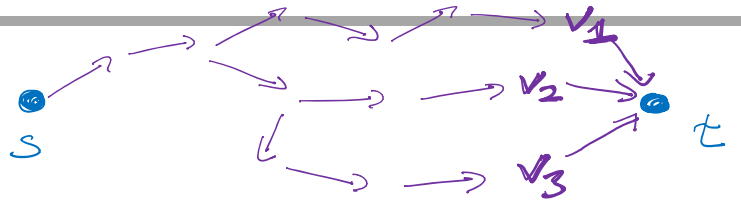
- Beispiele
  - Edi-Tier Distanz haben wir gerade gesehen
  - Die Fibonacci Zahlen kann man auch mit "dynamischer Programmierung" berechnen

    Einfach iterativ der Reihe nach ... und man muss sich dann immer nur die letzten beiden merken
  - Dijkstras Algo. 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)
    START TARGET
  - 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
  - Besonderheit 1: Dijkstra berechnet das Minimum für einen Knoten nicht auf einmal, sondern nach und nach
  - Besonderheit 2: die Reihenfolge der Operationen ist bei Dijkstra aufwändiger zu realisieren: man braucht eine PW

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