Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 12, Montag, 24. Januar 2011
(Editierdistanz, Dynamische Programmierung)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü11 (Dijkstra / A*)
  - Übrigens 1: das ist heute die drittletzte Vorlesung
  - Übrigens 2: die Klausur ist am 30. März 14:00 – 16:30 Uhr
    - im HS 026 (hier)
    - Teilnahmevoraussetzung: 60% der Punkte in den Ü1-Ü14
    - Möglicher Bonus durch Üs, siehe Folien zur ersten Vorlesung
- Dynamische Programmierung

  ... am Beispiel der Berechnung der Editierdistanz
  - Idee + Verfahren mit Korrektheitsbeweis + Beispiel
  - **Übungsaufgabe (Ü11):** Programm dazu + Teil vom Beweis

Ihre Erfahrungen mit dem Ü11 (A*)

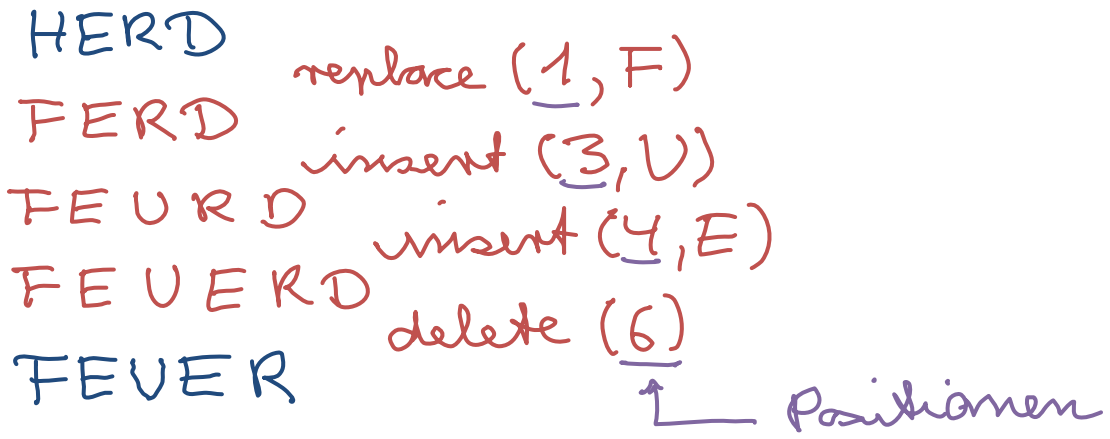
UNI FREIBURG

- Zusammenfassung von Ihrem Feedback **[24.1 16:09]**
  - Die Programmänderung fanden die meisten einfach
  - "Der Beweis war etwas sehr komisch"
  - Ohne Bonuspunkt gab es nur 9 Punkte → ist korrigiert
  - Einige scheinen gerade Zeitstress zu haben und nur das Nötigste zu machen

Editierdistanz — Definition

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:



Editierdistanz — Eigenschaften

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

Editierdistanz — Lösungsideen

UNI FREIBURG

- Wie würden wir als Menschen das Problem lösen
  - für BAUM → MAUER ?
  - für MAUER → AMERIKA ?
  - für AAEBEAABEAREEAEBA → RBEAAEEBAAAEBBAEAE ?
  - → möglichst große gemeinsame Teilstrings zu finden
    klappt manchmal aber nicht immer
- Rekursiver Ansatz
  - In zwei Teile / Hälften teilen?
    - keine gute Idee, z.B. ED(GAUL, AULA) = 2,
      aber ED(GA, AU) + ED(UL, LA) = 4
  - Auf ein "kleineres" Problem zurückführen?
    - das probieren wir jetzt

Editierdistanz — Rekursiver Ansatz 1/3

UNI FREIBURG

- Erst noch etwas Terminologie
  - Seien x und y unsere beiden Zeichenketten
  - Seien $\sigma_1, \ldots, \sigma_k$ eine Folge von $k = ED(x, y)$ Operationen 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
  - **Ü12 Aufgabe 2:** Für beliebige x und y gibt es eine monotone Folge von $ED(x, y)$ Operationen für $x \rightarrow y$

Editierdistanz — Rekursiver Ansatz 2/3

- Fallunterscheidung
  - Wir betrachten die letzte Operation σ_k = delete (6)
    - $\sigma_1, \ldots, \sigma_{k-1} : x \to z$ und $\sigma_k : z \to y$
    - 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: σ_k = insert(m' + 1, y[m]) [dann ist $m' = m - 1$]
    - Fall 1b: σ_k = delete(m') [dann ist $m' = m + 1$]
    - Fall 1c: σ_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

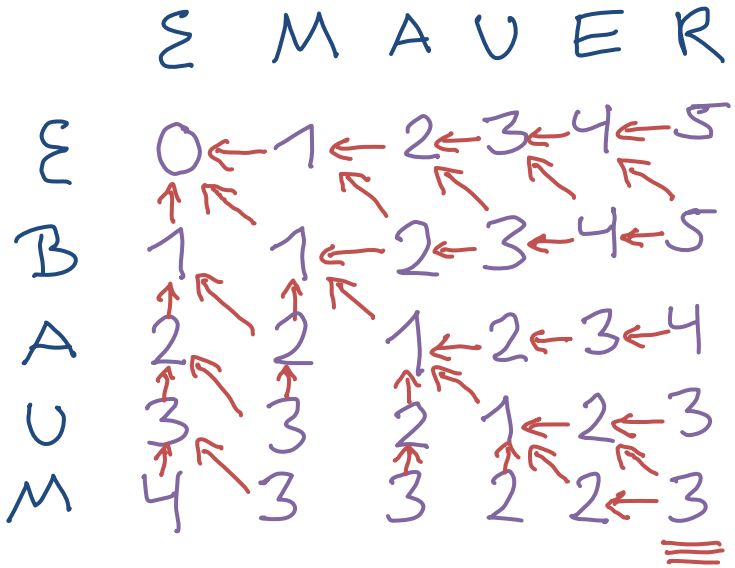
      $\sigma_1, \ldots, \sigma_k : x[1..n-1] \to y[1..m-1]$ und $x[n] = y[m]$

Editierdistanz — Rekursiver Ansatz 3/3

UNI FREIBURG

- Wir haben also einen dieser vier Fälle
  - Fall 1a (insert am Ende): $\sigma_1, \ldots, \sigma_{k-1} : x[1..n] \rightarrow y[1..m-1]$
  - Fall 1b (delete am Ende): $\sigma_1, \ldots, \sigma_{k-1} : x[1..n-1] \rightarrow y[1..m]$
  - 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
    - $ED(x[1..n], y[1..m-1]) + 1$, und
    - $ED(x[1..n-1], y[1..m]) + 1$, und
      - $ED(x[1..n-1], y[1..m-1] + 1$ falls $x[n] \neq y[m]$
      - $ED(x[1..n-1], y[1..m-1]$ falls $x[n] = y[m]$
  - Für $|x| = 0$ ist $ED(x, y) = |y|$, für $|y| = 0$ ist $ED(x, y) = |x|$

Editierdistanz — Beispielberechnung



Zwei Pfade:

BAUM — replace (1, M)
MAUM — replace (4, E)
MAUE — insert (5, R)
MAUER

↑ maximum!

BAUM — replace (1, M)
MAUM — insert (4, E)
MAUEM — replace (5, R)
MAUER

↑ maximum!

Editierdistanz — Folge von Operationen

UNI FREIBURG

- Die Tabelle gibt uns auch eine optimale Folge
  - Wir merken uns einfach bei jeder Anwendung der Rekursionsformel, welcher der vorherigen Einträge den kleinsten Wert ergeben hat (die Pfeile in unserem Bild)
    - es kann vorkommen, dass mehrere der vorherigen Einträge zum Minimum führen
  - Wenn wir den Pfeilen von dem Eintrag bei (n, m) bis zum Eintrag für (0, 0) folgen, bekommen wir eine optimale Folge von Operationen
    - können wir unterwegs mehreren Pfeilen folgen, gibt es entsprechend mehrere optimale Folgen

Editierdistanz — Programm

- Rekursives Programm
  - Es liegt nahe, das rekursiv zu programmieren
  - Für die Laufzeit würde folgende rekursive Formel gelten

    $T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1$
  - Man kann leicht ausrechnen, dass dann $T(n, n) \geq 3^n$
  - Das heißt die Laufzeit wäre (mindestens) exponentiell
- Dynamische Programmierung
  - Wir berechnen die Tabelle einfach Eintrag für Eintrag, so wie wir es in dem Beispiel gemacht haben und merken uns alle Einträge, die wir schon berechnet haben
  - Das braucht dann Laufzeit und Speicherplatz $O(n \cdot m)$

Dynamisches Programmieren

UNI FREIBURG

- Allgemeines Prinzip
  - Reduziere das Problem auf Unterprobleme kleinerer Größe (bei der ED: kleineres $|x| + |y|$)
  - Für eine gegebene Eingabe berechne alle Unterprobleme in der Reihenfolge aufsteigender Größe
  - Die Laufzeit und der Platzverbrauch hängen dann von der Anzahl dieser Unterprobleme ab (bei der ED: $|x| \cdot |y|$)
  - Zusammen mit dem "Wert" der optimalen Lösung erhält man auch leicht einen "Weg" dorthin
- Warum haben wir z.B. QuickSort nicht so realisiert?
  - Da brauchten wir nicht die Lösung aller möglichen Unterprobleme, sondern eine beliebige Aufteilung in zwei Teile war gut genug; bei der ED müssen wir quasi "alles ausprobieren"

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