Datenstrukturen und Algorithmen

Vorlesung 20: Dynamische Programmierung (K15)

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://www-i2.informatik.rwth-aachen.de/i2/dsal12/

3. Juli 2012

Übersicht

Übersicht

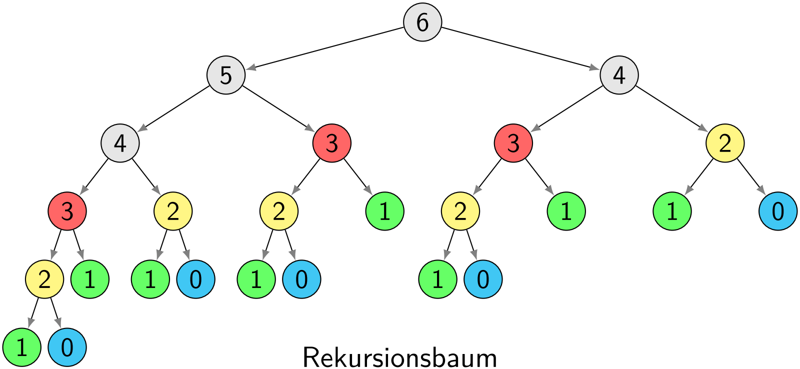
1. Motivation
2. Dynamische Programmierung
   - Rekursionsgleichungen
3. Anwendungen
   - Ketten von Matrixmultiplikationen
   - Das Rucksackproblem
   - Longest Common Subsequence (LCS)

Erinnerung: Fibonacci

$$Fib(0) = 0, \qquad Fib(1) = 1, \qquad Fib(n) = Fib(n-1) + Fib(n-2)$$

Erinnerung: Fibonacci

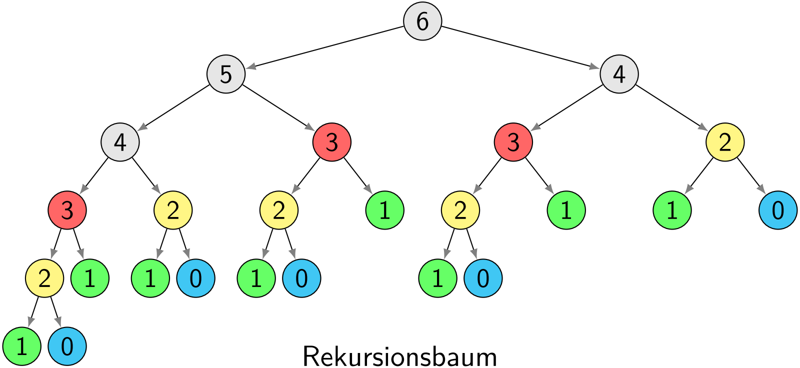
$$Fib(0) = 0, \qquad Fib(1) = 1, \qquad Fib(n) = Fib(n-1) + Fib(n-2)$$



Rekursionsbaum

Erinnerung: Fibonacci

$$Fib(0) = 0, \qquad Fib(1) = 1, \qquad Fib(n) = Fib(n-1) + Fib(n-2)$$

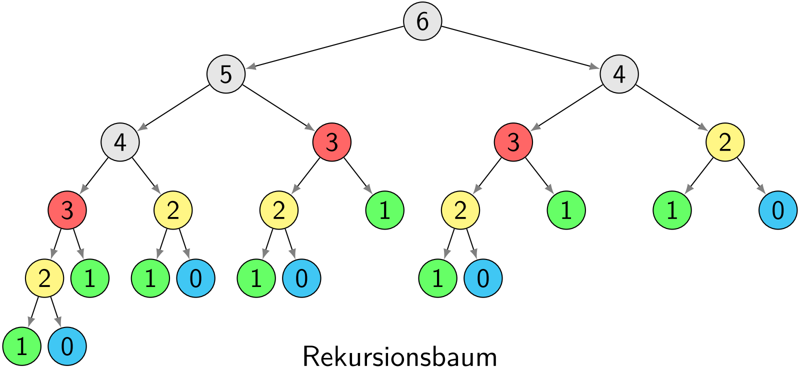


Rekursionsbaum

- Wir wollen z. B. $Fib(3)$ nicht ständig neu berechnen.

Erinnerung: Fibonacci

$$Fib(0) = 0, \qquad Fib(1) = 1, \qquad Fib(n) = Fib(n-1) + Fib(n-2)$$



Rekursionsbaum

- Wir wollen z. B. $Fib(3)$ nicht ständig neu berechnen.
- *Idee: Speichere einmal berechnete Werte.*

Memoization

Memoization

- Bei jedem Funktionsaufruf überprüfe, ob das Ergebnis bereits berechnet wurde (im Cache ist).

Memoization

Memoization

- Bei jedem Funktionsaufruf überprüfe, ob das Ergebnis bereits berechnet wurde (im Cache ist).
- Ist das nicht der Fall, berechne den Wert und speichere zusätzlich das Ergebnis.

Memoization

Memoization

- Bei jedem Funktionsaufruf überprüfe, ob das Ergebnis bereits berechnet wurde (im Cache ist).
- Ist das nicht der Fall, berechne den Wert und speichere zusätzlich das Ergebnis.

Beispiel

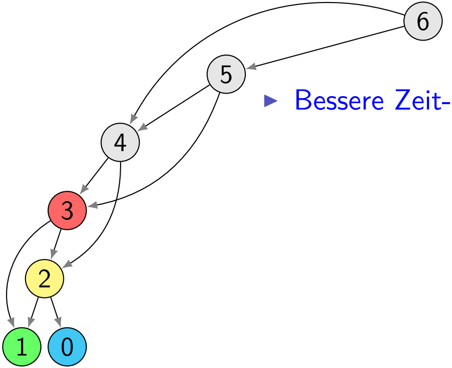
```
int fibDP(int n) {
  if (n < 2) return n;
  int f1 = getCache(n-1), f2 = getCache(n-2);
  if (f1 == -1) f1 = fibDP(n-1); // nicht gefunden
  if (f2 == -1) f2 = fibDP(n-2);
  int fib = f1 + f2;
  setCache(n, fib);
  return fib;
}
```

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.

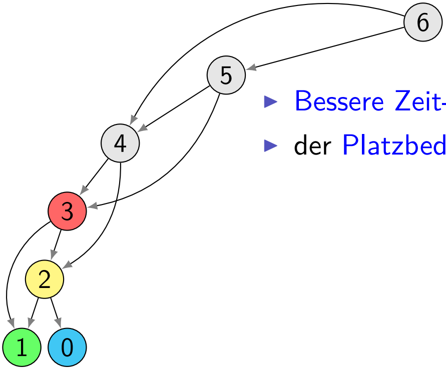


Abhängigkeitsgraph

- Bessere Zeit-Komplexität: $\Theta(n)$ statt $\Theta(2^n)$,

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.

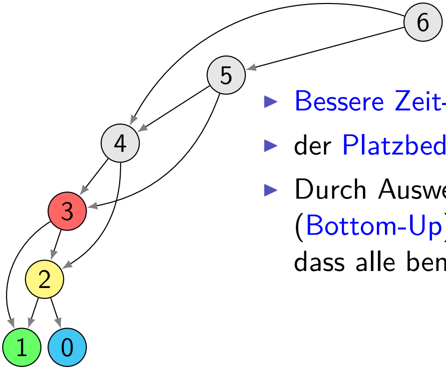


Abhängigkeitsgraph

- Bessere Zeit-Komplexität: $\Theta(n)$ statt $\Theta(2^n)$, aber
- der Platzbedarf wächst dabei auf $\Theta(n)$.

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.

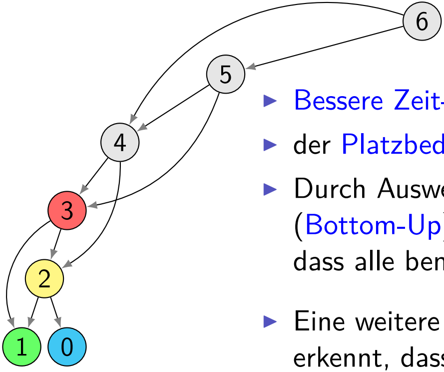


Abhängigkeitsgraph

- Bessere Zeit-Komplexität: $\Theta(n)$ statt $\Theta(2^n)$, aber
- der Platzbedarf wächst dabei auf $\Theta(n)$.
- Durch Auswertung von unten-nach-oben (Bottom-Up) kann sogar sichergestellt werden, dass alle benötigten Werte bereits berechnet sind.

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.

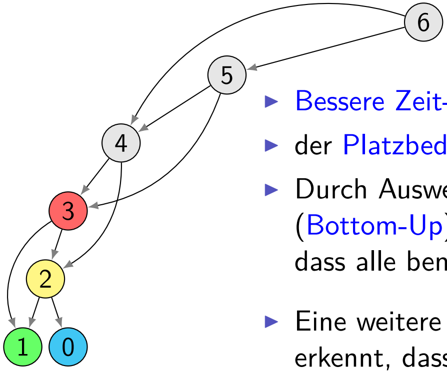


Abhängigkeitsgraph

- Bessere Zeit-Komplexität: $\Theta(n)$ statt $\Theta(2^n)$, aber
- der Platzbedarf wächst dabei auf $\Theta(n)$.
- Durch Auswertung von unten-nach-oben (Bottom-Up) kann sogar sichergestellt werden, dass alle benötigten Werte bereits berechnet sind.
- Eine weitere Verbesserung ergibt sich, indem man erkennt, dass hier jeweils nur die zwei letzten Werte benötigt werden (in-place).

Memoization – Dynamische Programmierung

- Memoization hilft, wenn die Teilprobleme überlappen.



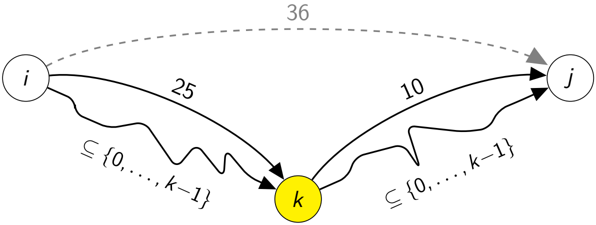
Abhängigkeitsgraph

- Bessere Zeit-Komplexität: $\Theta(n)$ statt $\Theta(2^n)$, aber
- der Platzbedarf wächst dabei auf $\Theta(n)$.
- Durch Auswertung von unten-nach-oben (Bottom-Up) kann sogar sichergestellt werden, dass alle benötigten Werte bereits berechnet sind.
- Eine weitere Verbesserung ergibt sich, indem man erkennt, dass hier jeweils nur die zwei letzten Werte benötigt werden (in-place).

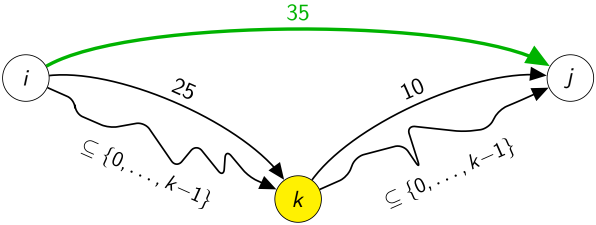
$\Rightarrow$ Auf diesen Grundideen basiert die $\boxed{\text{Dynamische Programmierung}}$.

Übersicht

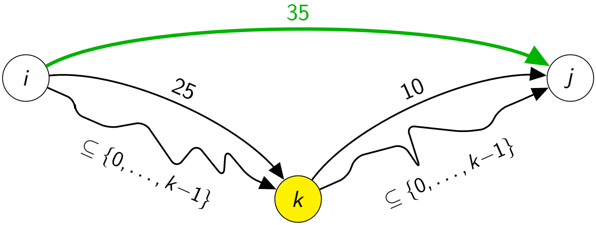
Rekursionsgleichungen: Floyd-Warshall



Rekursionsgleichungen: Floyd-Warshall

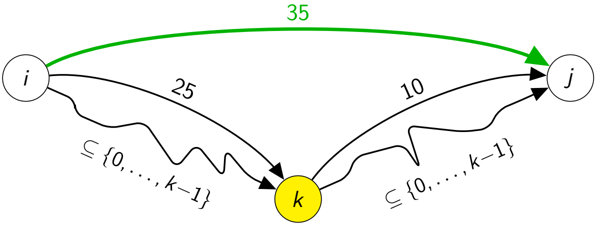


Rekursionsgleichungen: Floyd-Warshall



Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \ldots, k\}$.

Rekursionsgleichungen: Floyd-Warshall

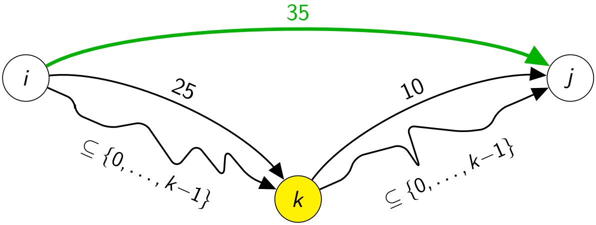


Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \ldots, k\}$.

Rekursionsgleichung die dem Algorithmus zur Grunde liegt:

$$d_{ij}^{(k)} = \begin{cases} W(i,j) & \text{für } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{für } k > 0 \end{cases}$$

Rekursionsgleichungen: Floyd-Warshall

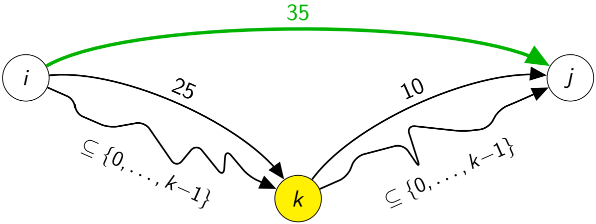


Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \ldots, k\}$.
Rekursionsgleichung die dem Algorithmus zur Grunde liegt:

$$d_{ij}^{(k)} = \begin{cases} W(i,j) & \text{für } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{für } k > 0 \end{cases}$$

- Top-down Abhängigkeit: $d_{ij}^{(k)}$ hängt von $d_{ij}^{(k-1)}$ ab.

Rekursionsgleichungen: Floyd-Warshall



Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \ldots, k\}$.

Rekursionsgleichung die dem Algorithmus zur Grunde liegt:

$$d_{ij}^{(k)} = \begin{cases} W(i, j) & \text{für } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{für } k > 0 \end{cases}$$

- Top-down Abhängigkeit: $d_{ij}^{(k)}$ hängt von $d_{ij}^{(k-1)}$ ab.
- Bottom-up Berechnung: $d_{ij}^{(0)}, d_{ij}^{(1)}, d_{ij}^{(2)}, \ldots$ usw.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.
   - Teilprobleme sind (teilweise) überlappend
   - Rekursive Abhängigkeit zwischen den Teilproblemen.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.
   - Teilprobleme sind (teilweise) überlappend
   - Rekursive Abhängigkeit zwischen den Teilproblemen.
2. Stelle die Rekursionsgleichung (top-down) für den Wert der Lösung auf.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.
   - Teilprobleme sind (teilweise) überlappend
   - Rekursive Abhängigkeit zwischen den Teilproblemen.
2. Stelle die Rekursionsgleichung (top-down) für den Wert der Lösung auf.
3. Löse Rekursionsgleichung bottom-up.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.
   - Teilprobleme sind (teilweise) überlappend
   - Rekursive Abhängigkeit zwischen den Teilproblemen.
2. Stelle die Rekursionsgleichung (top-down) für den Wert der Lösung auf.
3. Löse Rekursionsgleichung bottom-up.
4. Bestimme aus dem Wert der Lösung die Argumente der Lösung. Rekonstruiere die Lösung.

Dynamische Programmierung

- Die meisten DP-Probleme sind Optimierungsprobleme, d. h. es gibt verschiedene Lösungen, die mit einer Kostenfunktion bewertet werden. Gesucht ist jeweils eine Lösung mit minimalen Kosten (bzw. maximalem Wert).

Dynamischer Programmierung kann man i. d. R. in vier Teile gliedern:

1. Charakterisiere die Struktur einer optimalen Lösung, und stelle fest ob diese DP ermöglicht.
   - Teilprobleme sind (teilweise) überlappend
   - Rekursive Abhängigkeit zwischen den Teilproblemen.
2. Stelle die Rekursionsgleichung (top-down) für den Wert der Lösung auf.
3. Löse Rekursionsgleichung bottom-up.
4. Bestimme aus dem Wert der Lösung die Argumente der Lösung. Rekonstruiere die Lösung.

Weitere Bsp: Fibonacci, CYK (Cock-Younger-Kasami)-Algorithmus (VL FOSAP), Floyd(-Warshall), . . .

Übersicht

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

- Die Anzahl der Spalten in Matrix A muss dabei gleich der Anzahl der Zeilen in B sein.

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

- Die Anzahl der Spalten in Matrix A muss dabei gleich der Anzahl der Zeilen in B sein.
- Komplexität: $i \cdot j \cdot k$ Fließkomma-Multiplikationen.

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

- Die Anzahl der Spalten in Matrix A muss dabei gleich der Anzahl der Zeilen in B sein.
- Komplexität: $i \cdot j \cdot k$ Fließkomma-Multiplikationen.

- Betrachte nun die Multiplikation mehrerer Matrizen:
  $M = A_1 \cdot A_2 \cdot \ldots \cdot A_n$

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

- Die Anzahl der Spalten in Matrix A muss dabei gleich der Anzahl der Zeilen in B sein.
- Komplexität: $i \cdot j \cdot k$ Fließkomma-Multiplikationen.

- Betrachte nun die Multiplikation mehrerer Matrizen: $M = A_1 \cdot A_2 \cdot \ldots \cdot A_n$
- Wir gehen davon aus, dass die Matrizen jeweils miteinander kompatibel sind.

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

$C = A \cdot B$ mit $A \in \mathbb{R}^{i \times j}$, $B \in \mathbb{R}^{j \times k}$, $C \in \mathbb{R}^{i \times k}$.

- Die Anzahl der Spalten in Matrix A muss dabei gleich der Anzahl der Zeilen in B sein.
- Komplexität: $i \cdot j \cdot k$ Fließkomma-Multiplikationen.

- Betrachte nun die Multiplikation mehrerer Matrizen:
  $M = A_1 \cdot A_2 \cdot \ldots \cdot A_n$
- Wir gehen davon aus, dass die Matrizen jeweils miteinander kompatibel sind.
- Solche Ketten lassen sich wegen der Assoziativität der Matrixmultiplikation in beliebiger Reihenfolge berechnen / klammern.

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10 \times 100}$, $A_2 \in \mathbb{R}^{100 \times 5}$, $A_3 \in \mathbb{R}^{5 \times 50}$.

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10 \times 100}$, $A_2 \in \mathbb{R}^{100 \times 5}$, $A_3 \in \mathbb{R}^{5 \times 50}$.

- Berechnen wir $A_1 \cdot (A_2 \cdot A_3)$,

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10\times 100}$, $A_2 \in \mathbb{R}^{100\times 5}$, $A_3 \in \mathbb{R}^{5\times 50}$.

- Berechnen wir $A_1{\cdot}(A_2{\cdot}A_3)$, so muss man $10{\cdot}100{\cdot}50 + 100{\cdot}5{\cdot}50 = 50\,000 + 25\,000 = 75\,000$ mal multiplizieren.

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10\times 100}$, $A_2 \in \mathbb{R}^{100\times 5}$, $A_3 \in \mathbb{R}^{5\times 50}$.

- Berechnen wir $A_1{\cdot}(A_2{\cdot}A_3)$, so muss man $10{\cdot}100{\cdot}50 + 100{\cdot}5{\cdot}50 = 50\,000 + 25\,000 = 75\,000$ mal multiplizieren.
- Berechnen wir aber $(A_1{\cdot}A_2){\cdot}A_3$,

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10\times100}$, $A_2 \in \mathbb{R}^{100\times5}$, $A_3 \in \mathbb{R}^{5\times50}$.

- Berechnen wir $A_1 \cdot (A_2 \cdot A_3)$, so muss man $10 \cdot 100 \cdot 50 + 100 \cdot 5 \cdot 50 = 50\,000 + 25\,000 = 75\,000$ mal multiplizieren.
- Berechnen wir aber $(A_1 \cdot A_2) \cdot A_3$, dann ergeben sich $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5000 + 2500 = 7500$ Multiplikationen.

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

Sei $A_1 \in \mathbb{R}^{10 \times 100}$, $A_2 \in \mathbb{R}^{100 \times 5}$, $A_3 \in \mathbb{R}^{5 \times 50}$.

- Berechnen wir $A_1 \cdot (A_2 \cdot A_3)$, so muss man $10 \cdot 100 \cdot 50 + 100 \cdot 5 \cdot 50 = 50\,000 + 25\,000 = 75\,000$ mal multiplizieren.
- Berechnen wir aber $(A_1 \cdot A_2) \cdot A_3$, dann ergeben sich $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5000 + 2500 = 7500$ Multiplikationen.

Problem

Finde für eine Kette von Matrizen $A_1, A_2, \ldots, A_n$, mit Dimensionen $d_0 \times d_1$, $d_1 \times d_2$, $\ldots$, $d_{n-1} \times d_n$, eine Klammerung, so dass die Anzahl der Fließkomma-Multiplikationen minimal ist.

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Damit erhält man die Rekursionsgleichung

$$P(n) = \sum_{k=1}^{n-1} P(k) \cdot P(n-k), \qquad P(1) = 1.$$

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Damit erhält man die Rekursionsgleichung

$$P(n) = \sum_{k=1}^{n-1} P(k) \cdot P(n-k), \qquad P(1) = 1.$$

- Deren Lösung liegt in $\Omega(2^n)$.

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Damit erhält man die Rekursionsgleichung

$$P(n) = \sum_{k=1}^{n-1} P(k) \cdot P(n-k), \qquad P(1) = 1.$$

- Deren Lösung liegt in $\Omega(2^n)$.
- Einfach alle Möglichkeiten auszuprobieren ist keine Option.

Ketten von Matrixmultiplikationen

Sei $P(n)$ die Anzahl der möglichen Klammerungen für n Matrizen:

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Damit erhält man die Rekursionsgleichung

$$P(n) = \sum_{k=1}^{n-1} P(k) \cdot P(n-k), \qquad P(1) = 1.$$

- Deren Lösung liegt in $\Omega(2^n)$.
- Einfach alle Möglichkeiten auszuprobieren ist keine Option.

Idee: Stelle nach dem selben Prinzip eine Rekursionsgleichung für die minimale Anzahl an Multiplikationen auf.

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

$m[i, j]$ sei die minimale Anzahl Multiplikationen für die Teilkette $A_i \cdot \ldots \cdot A_j$.

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k\text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k\text{ Matrizen}}$$

$m[i,j]$ sei die minimale Anzahl Multiplikationen für die Teilkette $A_i \cdot \ldots \cdot A_j$.

- Offenbar ist $m[i,i] = 0$ für alle $0 < i \leqslant n$.

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

$m[i, j]$ sei die minimale Anzahl Multiplikationen für die Teilkette $A_i \cdot \ldots \cdot A_j$.

- Offenbar ist $m[i, i] = 0$ für alle $0 < i \leqslant n$.
- Die Dimension einer Teilkette ist $d_{i-1} \times d_j$.

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

$m[i, j]$ sei die minimale Anzahl Multiplikationen für die Teilkette $A_i \cdot \ldots \cdot A_j$.

- Offenbar ist $m[i, i] = 0$ für alle $0 < i \leqslant n$.
- Die Dimension einer Teilkette ist $d_{i-1} \times d_j$.
- Teilen bei Position k ergibt: $m[i, j] = m[i, k] + d_{i-1} \cdot d_k \cdot d_j + m[k+1, j]$.

Rekursionsgleichung

$$\underbrace{(A_1 \cdot \ldots \cdot A_k)}_{k \text{ Matrizen}} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_n)}_{n-k \text{ Matrizen}}$$

$m[i, j]$ sei die minimale Anzahl Multiplikationen für die Teilkette $A_i \cdot \ldots \cdot A_j$.

- Offenbar ist $m[i, i] = 0$ für alle $0 < i \leqslant n$.
- Die Dimension einer Teilkette ist $d_{i-1} \times d_j$.
- Teilen bei Position k ergibt: $m[i, j] = m[i, k] + d_{i-1} \cdot d_k \cdot d_j + m[k+1, j]$.
- Wir suchen dabei das optimale k, also:

$$m[i, j] = \begin{cases} 0 & \text{für } i = j, \\ \min_{i \leqslant k < j}(m[i, k] + m[k+1, j] + d_{i-1} \cdot d_k \cdot d_j) & \text{für } i < j. \end{cases}$$

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (I)

$$m[i,j] = \begin{cases} 0 & \text{für } i = j, \\ \min_{i \leqslant k < j}(m[i,k] + m[k{+}1,j] + d_{i-1} \cdot d_k \cdot d_j) & \text{für } i < j. \end{cases}$$

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (I)

$$m[i,j] = \begin{cases} 0 & \text{für } i = j, \\ \min_{i \leqslant k < j}(m[i,k] + m[k{+}1,j] + d_{i-1}{\cdot}d_k{\cdot}d_j) & \text{für } i < j. \end{cases}$$

- Wie bei Fibonacci wird z. B. das Teilproblem $m[0,1]$ mehrfach verwendet: von $m[0,2], m[0,3], \ldots, m[0,n]$.

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (I)

$$m[i,j] = \begin{cases} 0 & \text{für } i = j, \\ \min_{i \leqslant k < j}(m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j) & \text{für } i < j. \end{cases}$$

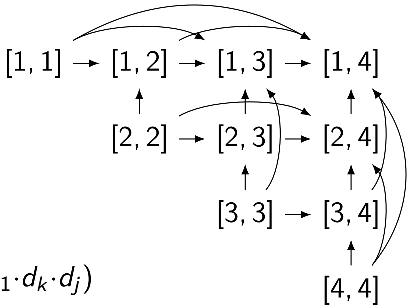
- Wie bei Fibonacci wird z. B. das Teilproblem $m[0,1]$ mehrfach verwendet: von $m[0,2], m[0,3], \ldots, m[0,n]$.
- Es gibt für alle $1 \leqslant i < j \leqslant n$ ein Teilproblem, also insgesamt nur $\binom{n}{2} \in \Theta(n^2)$ Teilprobleme.

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (I)

$$m[i,j] = \begin{cases} 0 & \text{für } i = j, \\ \min_{i \leqslant k < j}(m[i,k] + m[k{+}1,j] + d_{i-1} \cdot d_k \cdot d_j) & \text{für } i < j. \end{cases}$$

- Wie bei Fibonacci wird z. B. das Teilproblem $m[0,1]$ mehrfach verwendet: von $m[0,2], m[0,3], \ldots, m[0,n]$.
- Es gibt für alle $1 \leqslant i < j \leqslant n$ ein Teilproblem, also insgesamt nur $\binom{n}{2} \in \Theta(n^2)$ Teilprobleme.
- Wählen wir eine geschickte Berechnungsreihenfolge und speichern alle $m[i,j]$, dann lässt sich $m[i,j]$ in $\Theta(n)$ berechnen, da die Werte $m[i,k]$ und $m[k{+}1,j]$ bereits bekannt sind.

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)



$$m[i,j] = \min_{i \leqslant k < j} \left(m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j \right)$$

- Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.
- Erinnerung: Die naive Variante hätte $\Omega(2^n)$ Zeit, allerdings bei nur $\Theta(1)$ Platzbedarf, benötigt (*Time-Memory-Tradeoff*).
- In der Regel ist es nun eine große Zeitersparnis, zunächst die optimale Klammerung zu finden, statt unüberlegt zu multiplizieren.

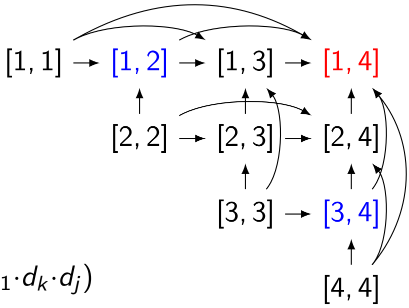
Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)

$$
\begin{tikzcd}
{[1,1]} \arrow[r] \arrow[rr, bend left] \arrow[rrr, bend left] & {[1,2]} \arrow[r] \arrow[rr, bend left] & {[1,3]} \arrow[r] & {[1,4]} \\
 & {[2,2]} \arrow[u] \arrow[r] \arrow[rr, bend left] & {[2,3]} \arrow[u] \arrow[r] & {[2,4]} \arrow[u] \\
 & & {[3,3]} \arrow[u] \arrow[r] \arrow[uu, bend right] & {[3,4]} \arrow[u] \arrow[uu, bend right] \\
 & & & {[4,4]} \arrow[u] \arrow[uu, bend right] \arrow[uuu, bend right]
\end{tikzcd}
$$

$$m[i,j] = \min_{i \leqslant k < j} (m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j)$$

- Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.
- Erinnerung: Die naive Variante hätte $\Omega(2^n)$ Zeit, allerdings bei nur $\Theta(1)$ Platzbedarf, benötigt (*Time-Memory-Tradeoff*).
- In der Regel ist es nun eine große Zeitersparnis, zunächst die optimale Klammerung zu finden, statt unüberlegt zu multiplizieren.

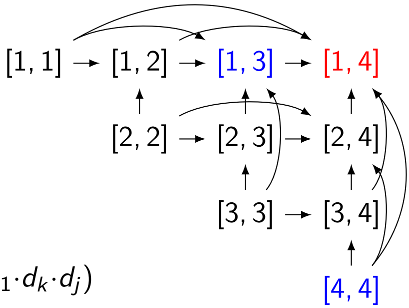
Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)



$$m[i,j] = \min_{i \leqslant k < j} (m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j)$$

- Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.
- Erinnerung: Die naive Variante hätte $\Omega(2^n)$ Zeit, allerdings bei nur $\Theta(1)$ Platzbedarf, benötigt (*Time-Memory-Tradeoff*).
- In der Regel ist es nun eine große Zeitersparnis, zunächst die optimale Klammerung zu finden, statt unüberlegt zu multiplizieren.

Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)



$$m[i, j] = \min_{i \leqslant k < j} (m[i, k] + m[k+1, j] + d_{i-1} \cdot d_k \cdot d_j)$$

- Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.
- Erinnerung: Die naive Variante hätte $\Omega(2^n)$ Zeit, allerdings bei nur $\Theta(1)$ Platzbedarf, benötigt (*Time-Memory-Tradeoff*).
- In der Regel ist es nun eine große Zeitersparnis, zunächst die optimale Klammerung zu finden, statt unüberlegt zu multiplizieren.

Ketten von Matrixmultiplikationen – Algorithmus

```
// Eingabe: Die Dimensionen der Matrizen: dim[i]=d_i für i=0...n
int matMultOrder(int dim[n+1], int n) {
  int m[n,n]; // hier 0-basiert!
  for (int i = 0; i < n; i++)
    m[i,i] = 0; // Diagonale
  for (int i = n-1; i >= 0; i--) // Zeilen
    for (int j = i+1; j < n; j++) { // Spalten
      int curMin = +inf;
      for (int k = i; k < j-1; k++) {
        curMin = min(curMin,
                     m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
      }
      m[i,j] = curMin;
    }
  return m[0,n-1];
}
```

Ketten von Matrixmultiplikationen – Algorithmus

```
// Eingabe: Die Dimensionen der Matrizen: dim[i]=d_i für i=0...n
int matMultOrder(int dim[n+1], int n) {
  int m[n,n]; // hier 0-basiert!
  for (int i = 0; i < n; i++)
    m[i,i] = 0; // Diagonale
  for (int i = n-1; i >= 0; i--) // Zeilen
    for (int j = i+1; j < n; j++) { // Spalten
      int curMin = +inf;
      for (int k = i; k < j-1; k++) {
        curMin = min(curMin,
                     m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
      }
      m[i,j] = curMin;
    }
  return m[0,n-1];
}
```

- Zur einfacheren Rekonstruktion der Lösung werden wir in einer zweiten Matrix jeweils den Index k mit dem Minimum speichern.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$\begin{array}{cc} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \left[\begin{array}{cccc} \quad & \quad & \quad & \quad \\ & & & \\ & & & \\ & & & \end{array}\right] \end{array} \left[\begin{array}{c} \quad \\ \\ \\ \\ \end{array}\right]$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$
\begin{array}{c c}
 & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{bmatrix}
\end{array}
\quad
\begin{bmatrix} & \\ & \\ & \\ & \end{bmatrix}
$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & & & \\
1 & & 0 & & \\
2 & & & 0 & 10\,000 \\
3 & & & & 0
\end{array}
\quad
\begin{bmatrix} & & & \\ & & & \\ & & & 2 \\ & & & \end{bmatrix}
\quad
\begin{array}{c} i = 2,\ j = 3 \\ A_2 A_3 \end{array}
$$

$$0 + 0 + 40{\cdot}10{\cdot}25$$
$$(k{=}2)\colon\ A_2 \cdot A_3$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}
\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & & & \\ & 0 & \color{blue}{400} & \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix} \end{array}
\begin{bmatrix} & & & \\ & & \color{blue}{1} & \\ & & & 2 \\ & & & \end{bmatrix}
\begin{array}{c} i = 1,\ j = 2 \\ A_1 A_2 \end{array}$$

$$0 + 0 + 1{\cdot}40{\cdot}10$$
$$(k{=}1)\text{: } A_1 \cdot A_2$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

$$\begin{array}{c|c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \left[\begin{array}{cccc} 0 & & & \\ & 0 & 400 & \color{blue}{650} \\ & & 0 & 10\,000 \\ & & & 0 \end{array}\right] \end{array} \left[\begin{array}{ccc} & & \\ & 1 & \color{blue}{2} \\ & & 2 \\ & & \end{array}\right] \begin{array}{c} i = 1,\ j = 3 \\ A_1 A_2 A_3 \end{array}$$

$$\begin{array}{cc} 0 + 10\,000 + 1{\cdot}40{\cdot}25, & 400 + 0 + 1{\cdot}10{\cdot}25 \\ (k{=}1)\text{: } A_1{\cdot}(A_2A_3) & (k{=}2)\text{: } (A_1A_2){\cdot}A_3 \end{array}$$

- $((A_0A_1)A_2)A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & \color{blue}{1200} & & \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix} \end{array} \begin{bmatrix} \color{blue}{0} & & \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix} \begin{array}{c} i = 0,\ j = 1 \\ A_0 A_1 \end{array}$$

$$0 + 0 + 30{\cdot}1{\cdot}40$$
$$(k{=}0){:}\ A_0{\cdot}A_1$$

- $((A_0A_1)A_2)A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$\begin{array}{c|c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1200 & {\color{blue}700} & \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix} \end{array} \quad \begin{bmatrix} 0 & {\color{blue}0} & \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix} \begin{array}{c} i = 0,\ j = 2 \\ A_0 A_1 A_2 \end{array}$$

$$\underset{(k=0):\ A_0\cdot(A_1 A_2)}{0 + 400 + 30\cdot 1\cdot 10}, \quad \underset{(k=1):\ (A_0 A_1)\cdot A_2}{1200 + 0 + 30\cdot 40\cdot 10}$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1200 & 700 & \color{red}{1400} \\ 1 & & 0 & 400 & 650 \\ 2 & & & 0 & 10\,000 \\ 3 & & & & 0 \end{array} \quad \begin{bmatrix} 0 & 0 & \color{blue}{0} \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix} \begin{array}{l} i=0,\ j=3 \\ A_0A_1A_2A_3 \end{array}$$

$0 + 650 + 30{\cdot}1{\cdot}25$, $1200 + 10\,000 + 30{\cdot}40{\cdot}25$, $700 + 0 + 30{\cdot}10{\cdot}25$

$(k{=}0)$: $A_0{\cdot}(A_1A_2A_3)$ $(k{=}1)$: $(A_0A_1){\cdot}(A_2A_3)$ $(k{=}2)$: $(A_0A_1A_2){\cdot}A_3$

- $((A_0A_1)A_2)A_3$ benötigt 20 700 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$
\begin{array}{c c}
 & \begin{array}{cccc} \scriptstyle 0 & \scriptstyle 1 & \scriptstyle 2 & \scriptstyle 3 \end{array} \\
\begin{array}{c} \scriptstyle 0 \\ \scriptstyle 1 \\ \scriptstyle 2 \\ \scriptstyle 3 \end{array} &
\begin{bmatrix} 0 & 1200 & 700 & \color{red}{1400} \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix}
\end{array}
\begin{bmatrix} 0 & 0 & 0 \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix}
$$

$$
\begin{array}{ccc}
0 + 650 + 30{\cdot}1{\cdot}25, & 1200 + 10\,000 + 30{\cdot}40{\cdot}25, & 700 + 0 + 30{\cdot}10{\cdot}25 \\
(k{=}0)\text{: } A_0{\cdot}(A_1A_2A_3) & (k{=}1)\text{: } (A_0A_1){\cdot}(A_2A_3) & (k{=}2)\text{: } (A_0A_1A_2){\cdot}A_3
\end{array}
$$

- $((A_0A_1)A_2)A_3$ benötigt 20 700 Multiplikationen.
- Rekonstruktion: $A_0 \quad A_1\ A_2 \quad A_3$ ist optimal – 1400 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

$$\begin{array}{c c}
 & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1200 & 700 & \color{red}{1400} \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix}
\end{array}
\begin{bmatrix} 0 & 0 & \color{blue}{0} \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix}
\begin{array}{l} i = 0,\ j = 3 \\ A_0 A_1 A_2 A_3 \end{array}$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.
- Rekonstruktion: $A_{\color{blue}{0}} \cdot (\ A_1\ A_2\ \ A_3)$ ist optimal – 1400 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,  25};
```

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1200 & 700 & 1400 \\ 1 & & 0 & 400 & 650 \\ 2 & & & 0 & 10\,000 \\ 3 & & & & 0 \end{array} \qquad \begin{bmatrix} 0 & 0 & 0 \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix} \quad \begin{array}{c} i = 1,\ j = 3 \\ A_1 A_2 A_3 \end{array}$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.
- Rekonstruktion: $A_0 \cdot ((A_1\ A_2) \cdot A_3)$ ist optimal – 1400 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

$$\begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1200 & 700 & \color{red}{1400} \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix} \end{array} \begin{bmatrix} 0 & 0 & 0 \\ & \color{blue}{1} & 2 \\ & & 2 \\ & & \end{bmatrix} \begin{array}{c} i = 1,\ j = 2 \\ A_1 A_2 \end{array}$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.
- Rekonstruktion: $A_0 \cdot ((A_{\color{blue}{1}} \cdot A_2) \cdot A_3)$ ist optimal – 1400 Multiplikationen.

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

Sei $A_0 \in \mathbb{R}^{30\times 1}$, $A_1 \in \mathbb{R}^{1\times 40}$, $A_2 \in \mathbb{R}^{40\times 10}$, $A_3 \in \mathbb{R}^{10\times 25}$.

```
m[i,j] = min(m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
dim[] = {30,   1,   40,   10,   25};
```

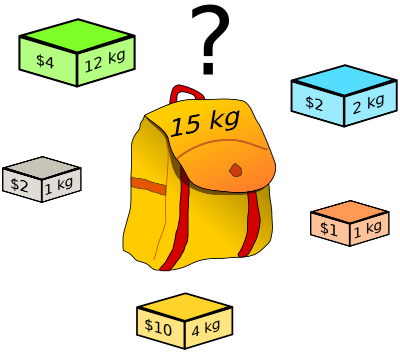
$$\begin{array}{c c}
 & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1200 & 700 & \color{red}{1400} \\ & 0 & 400 & 650 \\ & & 0 & 10\,000 \\ & & & 0 \end{bmatrix}
\end{array}
\begin{bmatrix} 0 & 0 & 0 \\ & 1 & 2 \\ & & 2 \\ & & \end{bmatrix}$$

- $((A_0 A_1) A_2) A_3$ benötigt 20 700 Multiplikationen.
- Rekonstruktion: $A_0 \cdot ((A_1 \cdot A_2) \cdot A_3)$ ist optimal – 1400 Multiplikationen.

Das Rucksackproblem (I)

Das Rucksackproblem (0-1 Knapsack)

Gegeben sei ein Rucksack, mit maximaler Tragkraft M, sowie n Gegenstände, die sowohl ein Gewicht als auch einen Wert haben.
Nehme möglichst viel Wert mit, ohne den Rucksack zu überladen.



Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n{-}1 \}$,

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n{-}1 \}$,
- Gewichte: $w_i \in \mathbb{N}_0$ für $i \in G$,

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n-1 \}$,
- Gewichte: $w_i \in \mathbb{N}_0$ für $i \in G$,
- Wert: $c_i \in \mathbb{N}_0$ für $i \in G$ (bzw. Kosten).

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n-1 \}$,
- Gewichte: $w_i \in \mathbb{N}_0$ für $i \in G$,
- Wert: $c_i \in \mathbb{N}_0$ für $i \in G$ (bzw. Kosten).

Gesucht:

- Der maximale Wert c_{max}.

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n-1 \}$,
- Gewichte: $w_i \in \mathbb{N}_0$ für $i \in G$,
- Wert: $c_i \in \mathbb{N}_0$ für $i \in G$ (bzw. Kosten).

Gesucht:

- Der maximale Wert $c_{\max}$.
- $S \subseteq G$ mit $c_{\max} = \sum\limits_{i \in S} c_i$ unter der Nebenbedingung $\sum\limits_{i \in S} w_i \leqslant M$.

Das Rucksackproblem (II)

Gegeben:

- Maximale Tragkraft M,
- n Gegenstände: $G = \{ 0, \ldots, n-1 \}$,
- Gewichte: $w_i \in \mathbb{N}_0$ für $i \in G$,
- Wert: $c_i \in \mathbb{N}_0$ für $i \in G$ (bzw. Kosten).

Gesucht:

- Der maximale Wert $c_{\max}$.
- $S \subseteq G$ mit $c_{\max} = \sum_{i \in S} c_i$ unter der Nebenbedingung $\sum_{i \in S} w_i \leqslant M$.

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst c_{max}:

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst c_{max}:
- Angenommen wir kennen den maximalen Wert c'_{max} des Rucksacks mit Tragkraft M', bei dem nur die ersten $n-1$ Gegenstände berücksichtigt werden.

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst $c_{\max}$:
- Angenommen wir kennen den maximalen Wert $c'_{\max}$ des Rucksacks mit Tragkraft M', bei dem nur die ersten $n-1$ Gegenstände berücksichtigt werden.
- Für $c_{\max}$ stellt sich also die Frage, ob der n-te Gegenstand (Gewicht: w_{n-1}, Wert: c_{n-1}) mitgenommen wird.

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst $c_{\max}$:
- Angenommen wir kennen den maximalen Wert $c'_{\max}$ des Rucksacks mit Tragkraft M', bei dem nur die ersten $n-1$ Gegenstände berücksichtigt werden.
- Für $c_{\max}$ stellt sich also die Frage, ob der n-te Gegenstand (Gewicht: w_{n-1}, Wert: c_{n-1}) mitgenommen wird.

Betrachten wir beide Fälle:

Ohne: $c_{\max}$ wäre dann gleich $c'_{\max}$ für $M' = M$.

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst $c_{\max}$:
- Angenommen wir kennen den maximalen Wert $c'_{\max}$ des Rucksacks mit Tragkraft M', bei dem nur die ersten $n-1$ Gegenstände berücksichtigt werden.
- Für $c_{\max}$ stellt sich also die Frage, ob der n-te Gegenstand (Gewicht: w_{n-1}, Wert: c_{n-1}) mitgenommen wird.

Betrachten wir beide Fälle:

Ohne: $c_{\max}$ wäre dann gleich $c'_{\max}$ für $M' = M$.

Mit: $c_{\max}$ wäre dann gleich $c'_{\max} + c_{n-1}$ für $M' = M - w_{n-1}$.
Falls $M' < 0$, dann setzen wir $c'_{\max} = -\infty$ („geht nicht“).

Das Rucksackproblem – Rekursionsgleichung (I)

Wir wollen das Problem mittels Dynamischer Programmierung lösen.

- Wir bestimmen zunächst $c_{\max}$:
- Angenommen wir kennen den maximalen Wert $c'_{\max}$ des Rucksacks mit Tragkraft M', bei dem nur die ersten $n-1$ Gegenstände berücksichtigt werden.
- Für $c_{\max}$ stellt sich also die Frage, ob der n-te Gegenstand (Gewicht: w_{n-1}, Wert: c_{n-1}) mitgenommen wird.

Betrachten wir beide Fälle:

Ohne: $c_{\max}$ wäre dann gleich $c'_{\max}$ für $M' = M$.

Mit: $c_{\max}$ wäre dann gleich $c'_{\max} + c_{n-1}$ für $M' = M - w_{n-1}$.
Falls $M' < 0$, dann setzen wir $c'_{\max} = -\infty$ („geht nicht“).

$\Rightarrow$ Wähle den Fall mit dem größeren Wert.

Das Rucksackproblem – Rekursionsgleichung (II)

Sei also $C[i, j]$ der maximale Wert des Rucksacks mit Tragkraft j, wenn man nur die Gegenstände $\{ 0, \ldots, i - 1 \}$ berücksichtigt.

Das Rucksackproblem – Rekursionsgleichung (II)

Sei also $C[i, j]$ der maximale Wert des Rucksacks mit Tragkraft j, wenn man nur die Gegenstände $\{0, \ldots, i-1\}$ berücksichtigt.

Es ergibt sich folgende Rekursionsgleichung:

$$C[i,j] = \begin{cases} \max(C[i-1,j], c_{i-1} + C[i-1, j-w_{i-1}]) & \\ -\infty & \text{für } j < 0 \\ 0 & \text{für } i = 0,\ j \geqslant 0 \end{cases}$$

Das Rucksackproblem – Rekursionsgleichung (II)

Sei also $C[i, j]$ der maximale Wert des Rucksacks mit Tragkraft j, wenn man nur die Gegenstände $\{0, \ldots, i-1\}$ berücksichtigt.

Es ergibt sich folgende Rekursionsgleichung:

$$C[i,j] = \begin{cases} \max(C[i{-}1,j], c_{i-1} + C[i{-}1, j{-}w_{i-1}]) & \\ -\infty & \text{für } j < 0 \\ 0 & \text{für } i = 0,\ j \geqslant 0 \end{cases}$$

- Dann ist $c_{\max} = C[n, M]$.

Das Rucksackproblem – Rekursionsgleichung (II)

Sei also $C[i, j]$ der maximale Wert des Rucksacks mit Tragkraft j, wenn man nur die Gegenstände $\{ 0, \ldots, i - 1 \}$ berücksichtigt.

Es ergibt sich folgende Rekursionsgleichung:

$$C[i, j] = \begin{cases} \max(C[i-1, j], c_{i-1} + C[i-1, j-w_{i-1}]) & \\ -\infty & \text{für } j < 0 \\ 0 & \text{für } i = 0, j \geqslant 0 \end{cases}$$

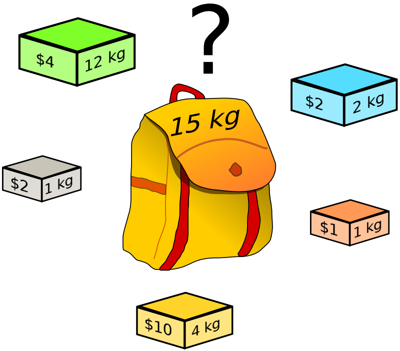
- Dann ist $c_{\max} = C[n, M]$.
- Diese Rekursionsgleichung lösen wir nun bottom-up, indem wir die Rucksäcke mit allen möglichen Gewichten $\{ 0, \ldots, M \}$ berechnen, wenn wir jeweils einen weiteren Gegenstand hinzunehmen.

Das Rucksackproblem – Algorithmus

```
// Eingabe: Gewichte w[i], Werte c[i], Tragkraft M
int knapDP(int w[n], int c[n], int n, int M) {
  int C[n+1,M+1];
  for (int j = 0; j <= M; j++)
    C[0,j] = 0;
  for (int i = 1; i <= n; i++)
    for (int j = 0; j <= M; j++)
      if (w[i-1] <= j) {
        C[i,j] = max(C[i-1,j], c[i-1] + C[i-1,j-w[i-1]]);
      } else {
        C[i,j] = C[i-1,j]; // passt nicht
      }
  return C[n,M];
}
```

- Zeitkomplexität: $\Theta(n \cdot M)$, Platzkomplexität: $\Theta(n \cdot M)$.

Das Rucksackproblem – Beispiel



Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1																
2																
3																
4																
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2																
3																
4																
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3																
4																
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4																
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5																

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i,j] = C[i-1,j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i, j] = C[i-1, j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n, M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i, j] = C[i-1, j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n, M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i, j] = C[i-1, j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n, M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i, j] = C[i-1, j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n, M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i, j] = C[i-1, j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n, M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

`w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Das Rucksackproblem – Rekonstruktion

Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um $c_{\max}$ zu erreichen.

- Falls $C[i,j] = C[i-1,j]$ ist, dann wurde der Gegenstand nicht mitgenommen (auch bei $c_i = 0$).
- Ausgehend von $C[n,M]$ kann man somit (mit Hilfe der w_i) die Menge S rekonstruieren (in $\Theta(n)$).

Beispiel

```
w[] = { 2, 12, 1, 1, 4 }, c[] = { 2, 4, 2, 1, 10 }, M = 15
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Longest Common Subsequence – einige Begriffe

Teilsequenz

Sei $A = (a_1, \ldots, a_m)$ eine Sequenz. Die Sequenz $A_{i_k} = (a_{i_1}, \ldots, a_{i_k})$ ist eine Teilsequenz von A wobei $i_1 < i_2 < \ldots < i_k$ und $i_j \in \{1, \ldots, m\}$.

Longest Common Subsequence – einige Begriffe

Teilsequenz

Sei $A = (a_1, \ldots, a_m)$ eine Sequenz. Die Sequenz $A_{i_k} = (a_{i_1}, \ldots, a_{i_k})$ ist eine Teilsequenz von A wobei $i_1 < i_2 < \ldots < i_k$ und $i_j \in \{1, \ldots, m\}$.

Eine Teilsequenz von A entsteht aus A indem Elemente weggelassen werden.

Longest Common Subsequence – einige Begriffe

Teilsequenz

Sei $A = (a_1, \ldots, a_m)$ eine Sequenz. Die Sequenz $A_{i_k} = (a_{i_1}, \ldots, a_{i_k})$ ist eine Teilsequenz von A wobei $i_1 < i_2 < \ldots < i_k$ und $i_j \in \{1, \ldots, m\}$.

Eine Teilsequenz von A entsteht aus A indem Elemente weggelassen werden.

Beispiel

$bcdb$ und aa sind Teilsequenzen von $A = abcbdab$.

Longest Common Subsequence – einige Begriffe

Teilsequenz

Sei $A = (a_1, \ldots, a_m)$ eine Sequenz. Die Sequenz $A_{i_k} = (a_{i_1}, \ldots, a_{i_k})$ ist eine Teilsequenz von A wobei $i_1 < i_2 < \ldots < i_k$ und $i_j \in \{1, \ldots, m\}$.

Eine Teilsequenz von A entsteht aus A indem Elemente weggelassen werden.

Beispiel

$bcdb$ und aa sind Teilsequenzen von $A = abcbdab$.

Gemeinsame Teilsequenz

Sequenz C ist eine gemeinsame Teilsequenz von A und B wenn C sowohl von A als auch von B eine Teilsequenz ist.

Longest Common Subsequence – einige Begriffe

Teilsequenz

Sei $A = (a_1, \ldots, a_m)$ eine Sequenz. Die Sequenz $A_{i_k} = (a_{i_1}, \ldots, a_{i_k})$ ist eine Teilsequenz von A wobei $i_1 < i_2 < \ldots < i_k$ und $i_j \in \{1, \ldots, m\}$.

Eine Teilsequenz von A entsteht aus A indem Elemente weggelassen werden.

Beispiel

$bcdb$ und aa sind Teilsequenzen von $A = abcbdab$.

Gemeinsame Teilsequenz

Sequenz C ist eine gemeinsame Teilsequenz von A und B wenn C sowohl von A als auch von B eine Teilsequenz ist.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abcbdab$ und $B = bdcaba$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

Gegeben die zwei Sequenzen $A = (a_1, \ldots, a_m)$ und $B = (b_1, \ldots, b_n)$, bestimme deren längsten gemeinsamen Teilsequenz (longest common subsequence), $LCS(A, B)$.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abcbdab$ und $B = bdcaba$, aber keine LCS.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

Gegeben die zwei Sequenzen $A = (a_1, \ldots, a_m)$ und $B = (b_1, \ldots, b_n)$, bestimme deren längsten gemeinsamen Teilsequenz (longest common subsequence), $LCS(A, B)$.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abcbdab$ und $B = bdcaba$, aber keine LCS.

$bcba$ ist eine LCS von $A = abcbdab$ und $B = bdcaba$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

Gegeben die zwei Sequenzen $A = (a_1, \ldots, a_m)$ und $B = (b_1, \ldots, b_n)$, bestimme deren längsten gemeinsamen Teilsequenz (longest common subsequence), $LCS(A, B)$.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abcbdab$ und $B = bdcaba$, aber keine LCS.

$bcba$ ist eine LCS von $A = abcbdab$ und $B = bdcaba$.

$bdab$ ist auch eine LCS von $A = abcbdab$ und $B = bdcaba$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

Gegeben die zwei Sequenzen $A = (a_1, \ldots, a_m)$ und $B = (b_1, \ldots, b_n)$, bestimme deren längsten gemeinsamen Teilsequenz (longest common subsequence), $LCS(A, B)$.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abcbdab$ und $B = bdcaba$, aber keine LCS.

$bcba$ ist eine LCS von $A = abcbdab$ und $B = bdcaba$.

$bdab$ ist auch eine LCS von $A = abcbdab$ und $B = bdcaba$.

Naiver Ansatz: betrachte alle Teilsequenzen von A, und überprüfe, welche auch eine Teilsequenz von B sind, und bewahre die längste gefundene Teilsequenz. Zeitkomplexität $\Theta(2^m)$ da es 2^m Teilsequenzen von A gibt.

Longest Common Subsequence – Eigenschaften (I)

Sei $A_i = (a_1, \ldots, a_i)$ der i-te Präfix von $A = (a_1, \ldots, a_m)$ für $0 \leqslant i \leqslant m$.

Longest Common Subsequence – Eigenschaften (I)

Sei $A_i = (a_1, \ldots, a_i)$ der i-te Präfix von $A = (a_1, \ldots, a_m)$ für $0 \leqslant i \leqslant m$.

Lemma (Optimale Teilstruktur)

1. *Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann ist dieses Zeichen auch Teil der LCS:*

$$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m).$$

Longest Common Subsequence – Eigenschaften (I)

Sei $A_i = (a_1, \ldots, a_i)$ der i-te Präfix von $A = (a_1, \ldots, a_m)$ für $0 \leqslant i \leqslant m$.

Lemma (Optimale Teilstruktur)

1. *Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann ist dieses Zeichen auch Teil der LCS:*

$$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m).$$

2. *Andernfalls gilt entweder*

$$LCS(A_m, B_n) = LCS(A_m, B_{n-1}) \quad \textit{oder}$$
$$LCS(A_m, B_n) = LCS(A_{m-1}, B_n).$$

Longest Common Subsequence – Eigenschaften (I)

Sei $A_i = (a_1, \ldots, a_i)$ der i-te Präfix von $A = (a_1, \ldots, a_m)$ für $0 \leqslant i \leqslant m$.

Lemma (Optimale Teilstruktur)

1. *Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann ist dieses Zeichen auch Teil der LCS:*

$$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m).$$

2. *Andernfalls gilt entweder*

$$LCS(A_m, B_n) = LCS(A_m, B_{n-1}) \quad \textit{oder}$$
$$LCS(A_m, B_n) = LCS(A_{m-1}, B_n).$$

   Insbesondere ist

$$|LCS(A_m, B_n)| = \max(|LCS(A_m, B_{n-1})|, |LCS(A_{m-1}, B_n)|).$$

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann: $LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$. Wir zeigen $C_{k-1} = LCS(A_{m-1}, B_{n-1})$.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$. Wir zeigen $C_{k-1} = LCS(A_{m-1}, B_{n-1})$.
Widerspruchsbeweis.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$. Wir zeigen $C_{k-1} = LCS(A_{m-1}, B_{n-1})$.
Widerspruchsbeweis. Nehme an, es gibt eine gemeinsame Teilsequenz D von A_{m-1} und B_{n-1} mit einer Länge von mindestens k.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$. Wir zeigen $C_{k-1} = LCS(A_{m-1}, B_{n-1})$.
Widerspruchsbeweis. Nehme an, es gibt eine gemeinsame Teilsequenz D von A_{m-1} und B_{n-1} mit einer Länge von mindestens k. Dann würde das Anhängen von $a_m = b_n$ an D zu einer gemeinsamen Teilsequenz von A und B führen, deren Länge größer ist als k.

Longest Common Subsequence – Eigenschaften (II)

Lemma (Optimale Teilstruktur)

Enden zwei Sequenzen mit dem selben Zeichen $a_m = b_n$, dann:
$LCS(A_m, B_n) = (LCS(A_{m-1}, B_{n-1}), a_m)$.

Beweis.

Sei $C = LCS(A_m, B_n)$ der Länge k. Wenn a_m nicht als letztes Zeichen in C vorkommt, dann könnte man a_m an C anhängen, und würde eine Teilsequenz von A_m und B_n erhalten. Dies widerspricht der Annahme, dass C eine LCS ist.

Nun ist der Präfix C_{k-1} eine gemeinsame Teilsequenz von A_{m-1} und B_{n-1} der Länge $k-1$. Wir zeigen $C_{k-1} = LCS(A_{m-1}, B_{n-1})$.
Widerspruchsbeweis. Nehme an, es gibt eine gemeinsame Teilsequenz D von A_{m-1} und B_{n-1} mit einer Länge von mindestens k. Dann würde das Anhängen von $a_m = b_n$ an D zu einer gemeinsamen Teilsequenz von A und B führen, deren Länge größer ist als k. Widerspruch. $\square$

Longest Common Subsequence – Rekursiongleichung

Wir können wieder die Rekursiongleichung für den *Wert*, also die Länge der LCS aufstellen: $L[i, j] = |LCS(A_i, B_j)|$

Longest Common Subsequence – Rekursiongleichung

Wir können wieder die Rekursiongleichung für den *Wert*, also die Länge der LCS aufstellen: $L[i,j] = |LCS(A_i, B_j)|$

$$L[i,j] = \begin{cases} 0 & \text{für } i = 0 \text{ oder } j = 0 \\ L[i-1, j-1] + 1 & \text{falls } a_i = b_j,\ i,j > 0 \\ \max(L[i, j-1], L[i-1, j]) & \text{falls } a_i \neq b_j,\ i,j > 0 \end{cases}$$

- Das lässt sich direkt als Algorithmus umsetzen (Hausaufgabe).

Longest Common Subsequence – Rekursiongleichung

Wir können wieder die Rekursiongleichung für den *Wert*, also die Länge der LCS aufstellen: $L[i,j] = |LCS(A_i, B_j)|$

$$L[i,j] = \begin{cases} 0 & \text{für } i = 0 \text{ oder } j = 0 \\ L[i-1, j-1] + 1 & \text{falls } a_i = b_j,\ i,j > 0 \\ \max(L[i, j-1], L[i-1, j]) & \text{falls } a_i \neq b_j,\ i,j > 0 \end{cases}$$

- Das lässt sich direkt als Algorithmus umsetzen (Hausaufgabe).
- Dessen Laufzeit ist $O(|A| \cdot |B|)$, ebenso seine Platzkomplexität.

Longest Common Subsequence – Rekursiongleichung

Wir können wieder die Rekursiongleichung für den *Wert*, also die Länge der LCS aufstellen: $L[i,j] = |LCS(A_i, B_j)|$

$$L[i,j] = \begin{cases} 0 & \text{für } i = 0 \text{ oder } j = 0 \\ L[i-1,j-1]+1 & \text{falls } a_i = b_j,\ i,j > 0 \\ \max(L[i,j-1], L[i-1,j]) & \text{falls } a_i \neq b_j,\ i,j > 0 \end{cases}$$

- Das lässt sich direkt als Algorithmus umsetzen (Hausaufgabe).
- Dessen Laufzeit ist $O(|A| \cdot |B|)$, ebenso seine Platzkomplexität.
- Ähnlich dem Rucksackproblem lässt sich dann die LCS rekonstruieren.

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0											
E	0											
U	0											
T	0											
S	0											
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0											
U	0											
T	0											
S	0											
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0											
T	0											
S	0											
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0											
S	0											
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0											
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow$0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\nwarrow$1	$\leftarrow$1
E	0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\uparrow$1	$\nwarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\nwarrow$2
U	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
T	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
S	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
C	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
H	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
L	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\nwarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3
A	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\nwarrow$4	$\leftarrow$4	$\leftarrow$4	$\leftarrow$4
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
L	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$
A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$
N	0	$\nwarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\nwarrow 5$	$\leftarrow 5$	$\leftarrow 5$
D	0	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\uparrow 5$	$\nwarrow 6$	$\leftarrow 6$

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
L	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$
A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$
N	0	$\nwarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\nwarrow 5$	$\leftarrow 5$	$\leftarrow 5$
D	0	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\uparrow 5$	$\nwarrow 6$	$\leftarrow 6$

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow$0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\nwarrow$1	$\leftarrow$1
E	0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\uparrow$1	$\nwarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\nwarrow$2
U	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
T	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
S	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
C	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
H	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
L	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\nwarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3
A	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\nwarrow$4	$\leftarrow$4	$\leftarrow$4	$\leftarrow$4
N	0	$\nwarrow$1	$\leftarrow$1	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\nwarrow$5	$\leftarrow$5	$\leftarrow$5
D	0	$\uparrow$1	$\uparrow$1	$\uparrow$1	$\nwarrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\uparrow$5	$\nwarrow$6	$\leftarrow$6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$\nwarrow$	bei $a_i = b_j$,	also $L[i,j] = L[i-1, j-1] + 1$
$\uparrow$	bei $a_i \neq b_j$,	für $L[i,j] = L[i-1, j]$
$\leftarrow$	bei $a_i \neq b_j$,	für $L[i,j] = L[i, j-1]$ und nicht $\uparrow$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1, j-1] + 1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1, j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i, j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↖1	←1	←1	←1	←1	←1	↖1	←1
E	0	↑0	↑0	↖1	↑1	↖2	←2	←2	←2	←2	←2	↖2
U	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
T	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
S	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
C	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
H	0	↑0	↑0	↑1	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
L	0	↑0	↑0	↑1	↑1	↑2	↑2	↖3	←3	←3	←3	←3
A	0	↑0	↑0	↑1	↑1	↑2	↑2	↑3	↖4	←4	←4	←4
N	0	↖1	←1	↑1	↑1	↑2	↑2	↑3	↑4	↖5	←5	←5
D	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3	↑4	↑5	↖6	←6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow$0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\nwarrow$1	$\leftarrow$1
E	0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\uparrow$1	$\nwarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\nwarrow$2
U	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
T	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
S	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
C	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
H	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
L	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\nwarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3
A	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\nwarrow$4	$\leftarrow$4	$\leftarrow$4	$\leftarrow$4
N	0	$\nwarrow$1	$\leftarrow$1	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\nwarrow$5	$\leftarrow$5	$\leftarrow$5
D	0	$\uparrow$1	$\uparrow$1	$\uparrow$1	$\nwarrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\uparrow$5	$\nwarrow$6	$\leftarrow$6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
L	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$
A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$
N	0	$\nwarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\nwarrow 5$	$\leftarrow 5$	$\leftarrow 5$
D	0	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\uparrow 5$	$\nwarrow 6$	$\leftarrow 6$

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$
\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}
$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
E	0	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 2$
L	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$
A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$
N	0	$\nwarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\nwarrow 5$	$\leftarrow 5$	$\leftarrow 5$
D	0	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\uparrow 5$	$\nwarrow 6$	$\leftarrow 6$

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll} \nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\ \uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\ \leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow \end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow$0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\nwarrow$1	$\leftarrow$1
E	0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\uparrow$1	$\nwarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\nwarrow$2
U	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
T	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
S	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
C	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
H	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
L	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\nwarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3
A	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\nwarrow$4	$\leftarrow$4	$\leftarrow$4	$\leftarrow$4
N	0	$\nwarrow$1	$\leftarrow$1	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\nwarrow$5	$\leftarrow$5	$\leftarrow$5
D	0	$\uparrow$1	$\uparrow$1	$\uparrow$1	$\nwarrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\uparrow$5	$\nwarrow$6	$\leftarrow$6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

$$\begin{array}{lll}
\nwarrow & \text{bei } a_i = b_j, & \text{also } L[i,j] = L[i-1,j-1]+1 \\
\uparrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i-1,j] \\
\leftarrow & \text{bei } a_i \neq b_j, & \text{für } L[i,j] = L[i,j-1] \text{ und nicht } \uparrow
\end{array}$$

Beispiel

		N	I	E	D	E	R	L	A	N	D	E
	0	0	0	0	0	0	0	0	0	0	0	0
D	0	$\uparrow$0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\leftarrow$1	$\nwarrow$1	$\leftarrow$1
E	0	$\uparrow$0	$\uparrow$0	$\nwarrow$1	$\uparrow$1	$\nwarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\leftarrow$2	$\nwarrow$2
U	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
T	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
S	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
C	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
H	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$2
L	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\nwarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3	$\leftarrow$3
A	0	$\uparrow$0	$\uparrow$0	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\nwarrow$4	$\leftarrow$4	$\leftarrow$4	$\leftarrow$4
N	0	$\nwarrow$1	$\leftarrow$1	$\uparrow$1	$\uparrow$1	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\nwarrow$5	$\leftarrow$5	$\leftarrow$5
D	0	$\uparrow$1	$\uparrow$1	$\uparrow$1	$\nwarrow$2	$\uparrow$2	$\uparrow$2	$\uparrow$3	$\uparrow$4	$\uparrow$5	$\nwarrow$6	$\leftarrow$6