

Datenstrukturen und Algorithmen

Vorlesung 19: Dynamische Programmierung (K15)

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

<https://moves.rwth-aachen.de/teaching/ss-18/dsal/>

2. Juli 2018

Übersicht

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

Übersicht

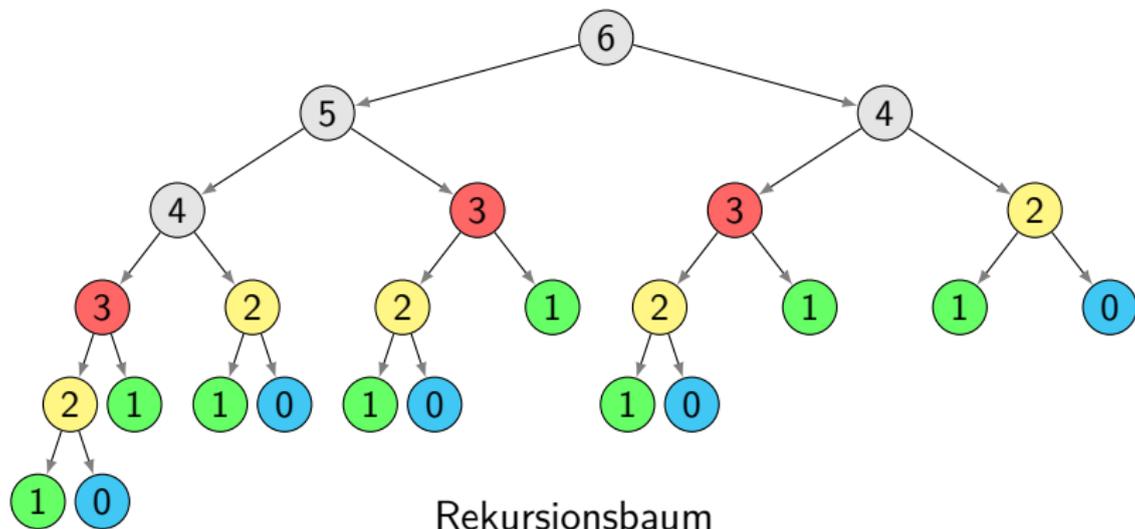
- 1 Motivation
- 2 Dynamische Programmierung
 - Rekursionsgleichungen
- 3 Anwendungen
 - Ketten von Matrixmultiplikationen
 - Das Rucksackproblem
 - Longest Common Subsequence (LCS)

Erinnerung: Fibonacci

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

Erinnerung: Fibonacci

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



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

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

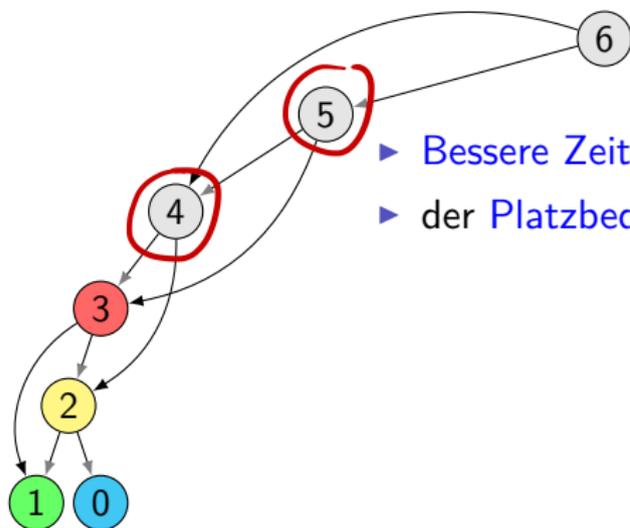
```
1 int fibDP(int n) {
2   if (n < 2) return n;
3   int f1 = getCache(n-1), f2 = getCache(n-2);
4   if (f1 == -1) f1 = fibDP(n-1); // nicht gefunden
5   if (f2 == -1) f2 = fibDP(n-2);
6   int fib = f1 + f2;
7   setCache(n, fib);
8   return fib;
9 }
```

Memoization – Dynamische Programmierung

- ▶ Memoization hilft, wenn die **Teilprobleme überlappen**.

Memoization – Dynamische Programmierung

- ▶ Memoization hilft, wenn die **Teilprobleme überlappen**.

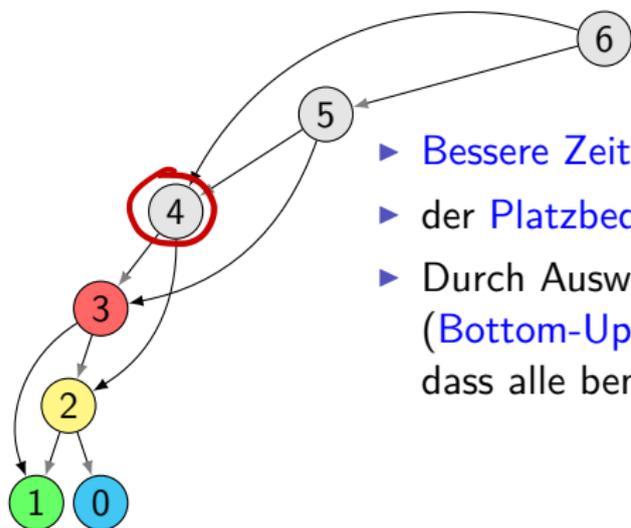


Abhängigkeitsgraph

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

Memoization – Dynamische Programmierung

- ▶ Memoization hilft, wenn die **Teilprobleme überlappen**.

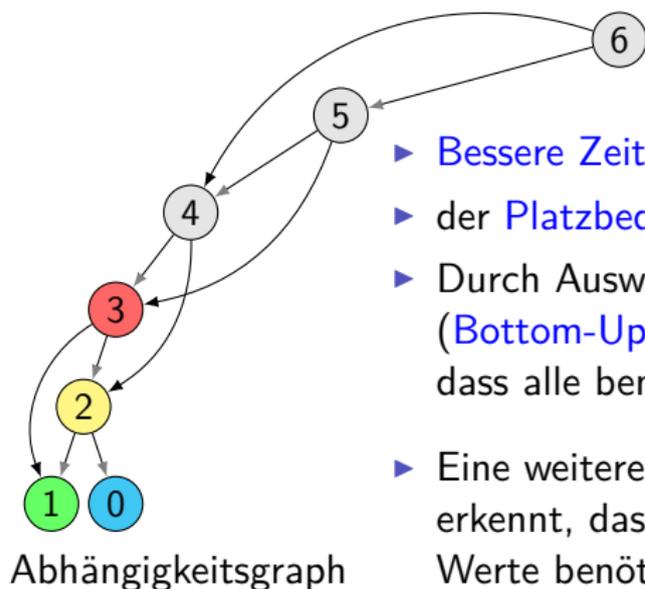


Abhängigkeitsgraph

- ▶ **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**.

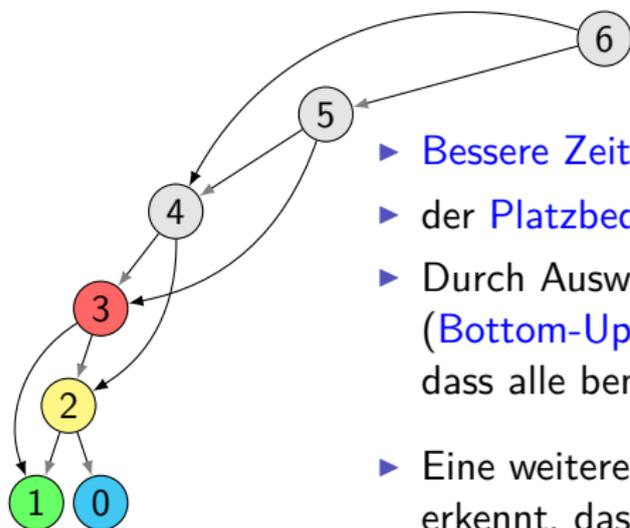


- ▶ **Bessere Zeit-Komplexität:** $\Theta(n)$ statt $\Theta(2^n)$, aber
- ▶ der **Platzbedarf wächst** dabei auf $\Theta(n)$.
- ▶ Durch Auswertung von unten-nach-oben (**Bottom-Up**) kann sogar sichergestellt werden, dass alle benötigten Werte bereits berechnet sind.
- ▶ Eine weitere Verbesserung ergibt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

$\Theta(1)$

Memoization – Dynamische Programmierung

- ▶ Memoization hilft, wenn die **Teilprobleme überlappen**.



Abhängigkeitsgraph

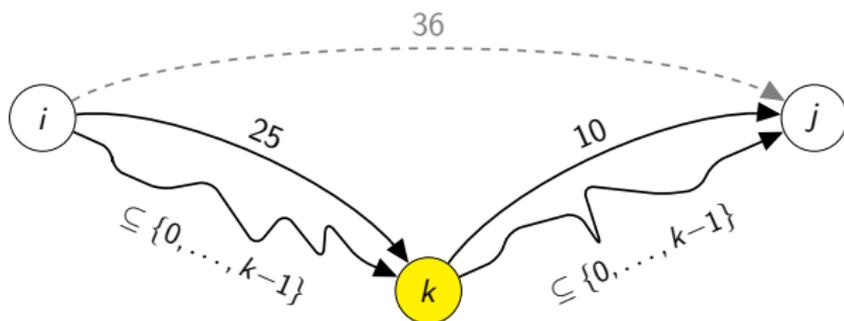
- ▶ **Bessere Zeit-Komplexität:** $\Theta(n)$ statt $\Theta(2^n)$, aber
- ▶ der **Platzbedarf wächst** dabei auf $\Theta(n)$.
- ▶ Durch Auswertung von unten-nach-oben (**Bottom-Up**) kann sogar sichergestellt werden, dass alle benötigten Werte bereits berechnet sind.
- ▶ Eine weitere Verbesserung ergibt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

⇒ Auf diesen Grundideen basiert die Dynamische Programmierung.

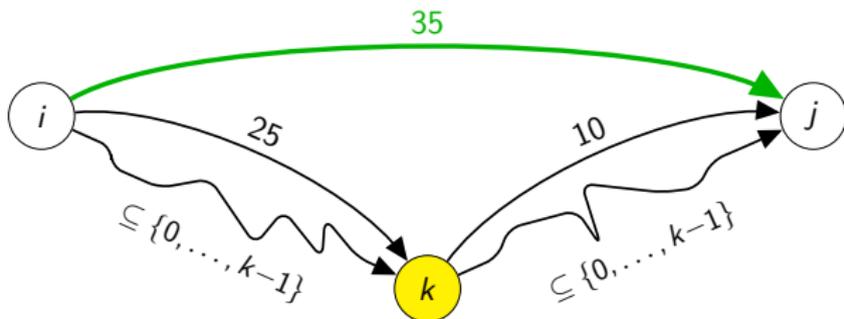
Übersicht

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

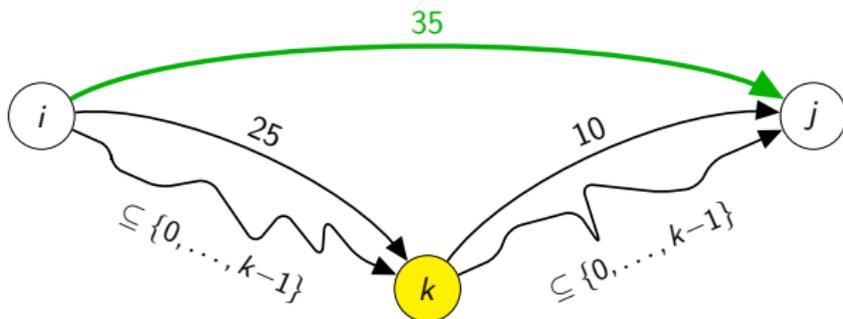
Rekursionsgleichungen: Floyd-Warshall



Rekursionsgleichungen: Floyd-Warshall

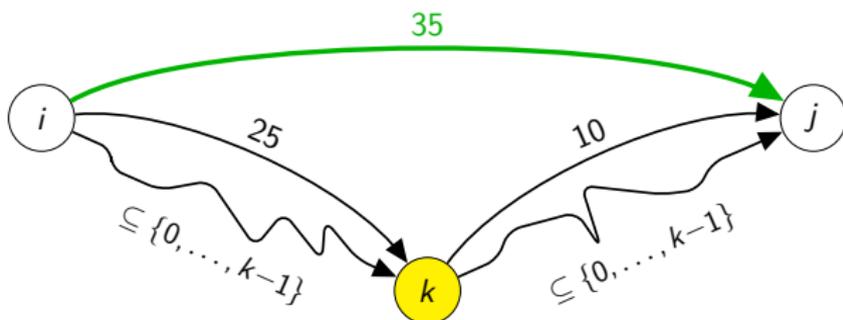


Rekursionsgleichungen: Floyd-Warshall



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

Rekursionsgleichungen: Floyd-Warshall

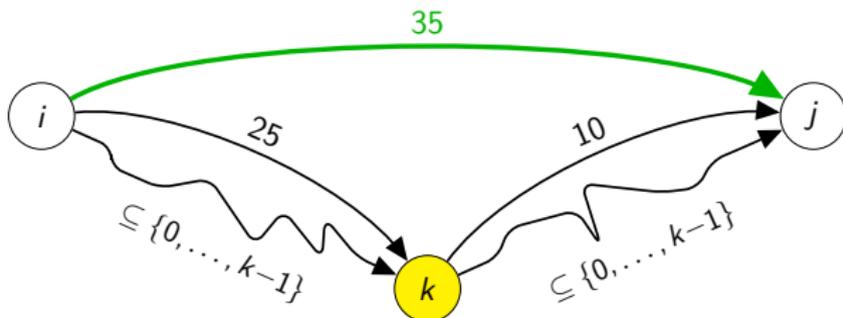


Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \dots, k\}$.

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

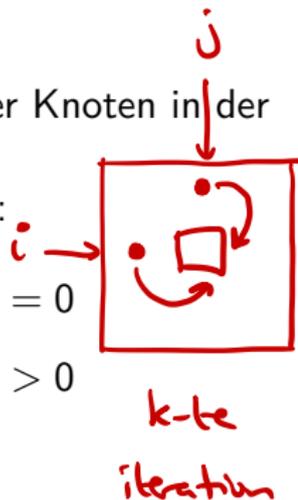


Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Pfades von i nach j über Knoten in der Menge $\{0, 1, \dots, k\}$.

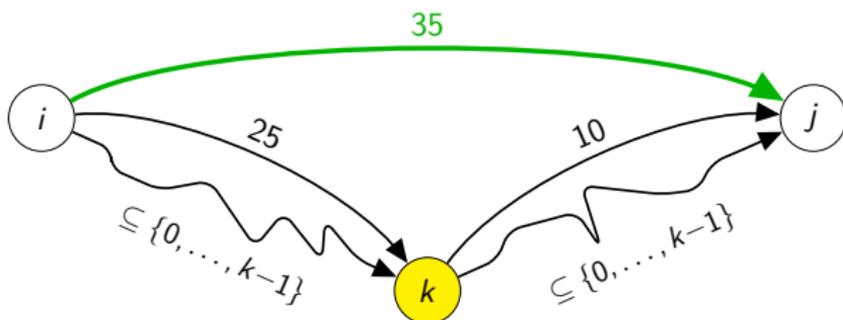
Rekursionsgleichung die dem Algorithmus zur Grunde liegt:

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

- ▶ 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 $\{0, 1, \dots, k\}$.

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: $d_{ij}^{(0)}, d_{ij}^{(1)}, d_{ij}^{(2)}, \dots$ 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).

Dynamische Programmierung

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

Dynamische Programmierung

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

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

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

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

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

Dynamische 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.

$$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$$

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

Übersicht

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

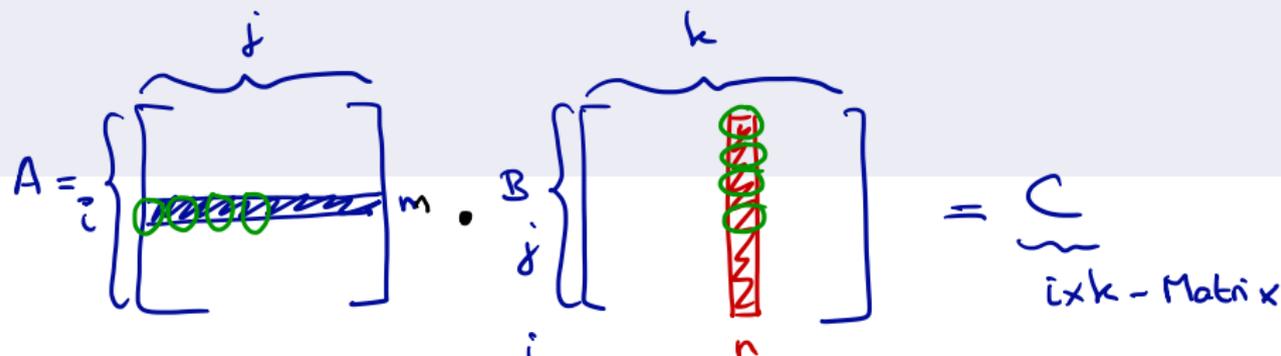
Übersicht

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

Ketten von Matrixmultiplikationen – Motivation (I)

Matrixmultiplikation

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



$$C[m, n] = \sum_{l=1}^j A[m, l] \cdot B[l, n]$$

$\underbrace{\hspace{10em}}_{j \text{ Multiplikation}}$

$i \times k \times j$

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:
$$M = A_1 \cdot A_2 \cdot \dots \cdot A_n$$

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:

$$M = A_1 \cdot A_2 \cdot \dots \cdot A_n$$

- ▶ Wir gehen davon aus, dass die Matrizen jeweils miteinander **kompatibel** sind.

$$\begin{aligned} & \text{Anzahl Spalten in } A_i \\ & = \text{Anzahl Zeilen in } A_{i+1} \end{aligned}$$

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: $n=3$

$$M = A_1 \cdot A_2 \cdot \dots \cdot A_n$$

- ▶ Wir gehen davon aus, dass die Matrizen jeweils miteinander **kompatibel** sind.
- ▶ Solche Ketten lassen sich wegen der Assoziativität der $A_1 \cdot A_2 \cdot A_3$ Matrixmultiplikation in **beliebiger Reihenfolge** berechnen / klammern.

$$(A_1 \cdot A_2) \cdot A_3 \quad \text{oder} \quad A_1 \cdot (A_2 \cdot A_3)$$

Ketten von Matrixmultiplikationen – Motivation (II)

Beispiel

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

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 $A_1 \cdot (A_2 \cdot A_3)$,

$$A_2 \cdot A_3 = 100 \times 50$$

$$\underbrace{100 \times 5 \times 50}_{100 \times 50} = 25,000$$

$$10 \times 100 \times 50 = 50,000$$

$$75,000$$

$A_2 \cdot A_3$ hat

Kardinalität

$$\underline{100 \times 50}$$

$$10 \times 100 \cdot 100 \times 50$$

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 $A_1 \cdot (A_2 \cdot A_3)$, 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 $(A_1 \cdot A_2) \cdot A_3$,

$$A_1 \cdot A_2 = 10 \times 5$$

$$10 \times 100 \times 5 = 5000$$

$$10 \times 5 \times 50 = \underline{2500}$$

$$7500$$

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 $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5\,000 + 2\,500 = 7\,500$ 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 $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5\,000 + 2\,500 = 7\,500$ Multiplikationen.

Problem

Finde für eine Kette von Matrizen A_1, A_2, \dots, A_n , mit Dimensionen $d_0 \times d_1, d_1 \times d_2, \dots, d_{n-1} \times d_n$, 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:

$$(A_1 \cdot A_2) \cdot A_3$$

$$A_1 \cdot (A_2 \cdot A_3)$$

$$(A_1 (A_2 \cdot A_3)) \cdot A_4$$

$$A_1 \cdot (A_2 (A_3 \cdot A_4))$$

$$(A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$$

$$((A_1 \cdot A_2) \cdot A_3) \cdot A_4$$

$$A_1 \cdot ((A_2 \cdot A_3) \cdot A_4)$$

Ketten von Matrixmultiplikationen

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

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

Ketten von Matrixmultiplikationen

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

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

Damit erhält man die Rekursionsgleichung

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

$$(A_1 \cdot \dots \cdot A_k) \cdot (A_{k+1} \cdot \dots \cdot A_n)$$

$$P(n) = \begin{cases} \sum_{k=1}^{n-1} P(k) \cdot P(n-k), & \text{falls } n > 1 \\ 1 & \text{falls } n = 1 \end{cases}$$

Betrachte $n=4$: $A_1 \cdot A_2 \cdot A_3 \cdot A_4$

$$P(4) = \sum_{k=1}^3 P(k) \cdot P(n-k)$$

$$P(1) = 1$$

$$P(2) = P(1) \cdot P(1) = 1$$

$$P(3) = \sum_{k=1}^2 P(k) \cdot P(3-k) =$$

$$P(1) \cdot P(2) + P(2) \cdot P(1) = 2$$

$$P(4) = \underbrace{P(1) \cdot P(2)}_{=2} + \underbrace{P(2) \cdot P(2)}_{=1} + \underbrace{P(3) \cdot P(1)}_{=2} = 5$$

Ketten von Matrixmultiplikationen

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

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

Damit erhält man die Rekursionsgleichung

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

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

Ketten von Matrixmultiplikationen

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

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

Damit erhält man die Rekursionsgleichung

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

- ▶ 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:

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

Damit erhält man die Rekursionsgleichung

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

- ▶ 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

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

Rekursionsgleichung

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

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

$$A_i \cdot \dots \cdot A_j$$

Rekursionsgleichung

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

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

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

Rekursionsgleichung

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

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

- ▶ Offenbar ist $m[i, i] = 0$ für alle $0 < i \leq n$.
- ▶ Die Dimension einer Teilkette $A_i \cdot \dots \cdot A_j$ ist $d_{i-1} \times d_j$.

$$\begin{array}{ccc} & \swarrow & \searrow \\ & d_{i-1} \times d_i & d_{j-1} \times d_j \\ \underline{\underline{\quad}} & & \underline{\underline{\quad}} \end{array}$$

Rekursionsgleichung

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

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

- ▶ Offenbar ist $m[i, i] = 0$ für alle $0 < i \leq n$.
- ▶ Die Dimension einer Teilkette $A_i \cdot \dots \cdot A_j$ ist $d_{i-1} \times d_j$.
- ▶ Teilen der Teilkette $A_i \cdot \dots \cdot A_j$ bei Position k ergibt:

$$m[i, j] = m[i, k] + \underbrace{d_{i-1} \cdot d_k \cdot d_j}_{\text{Multiplikation}} + m[k+1, j].$$

$$\underbrace{(A_i \cdot A_{i+1} \cdot \dots \cdot A_k)}_{d_{i-1} \times d_k} \cdot \underbrace{(A_{k+1} \cdot \dots \cdot A_j)}_{d_k \times d_j}$$

↑
k

Rekursionsgleichung

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

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

- ▶ Offenbar ist $m[i, i] = 0$ für alle $0 < i \leq n$.
- ▶ Die Dimension einer Teilkette $A_i \cdot \dots \cdot A_j$ ist $d_{i-1} \times d_j$.
- ▶ Teilen der Teilkette $A_i \cdot \dots \cdot A_j$ bei Position k ergibt:

$$m[i, j] = m[i, k] + d_{i-1} \cdot d_k \cdot d_j + m[k+1, j].$$

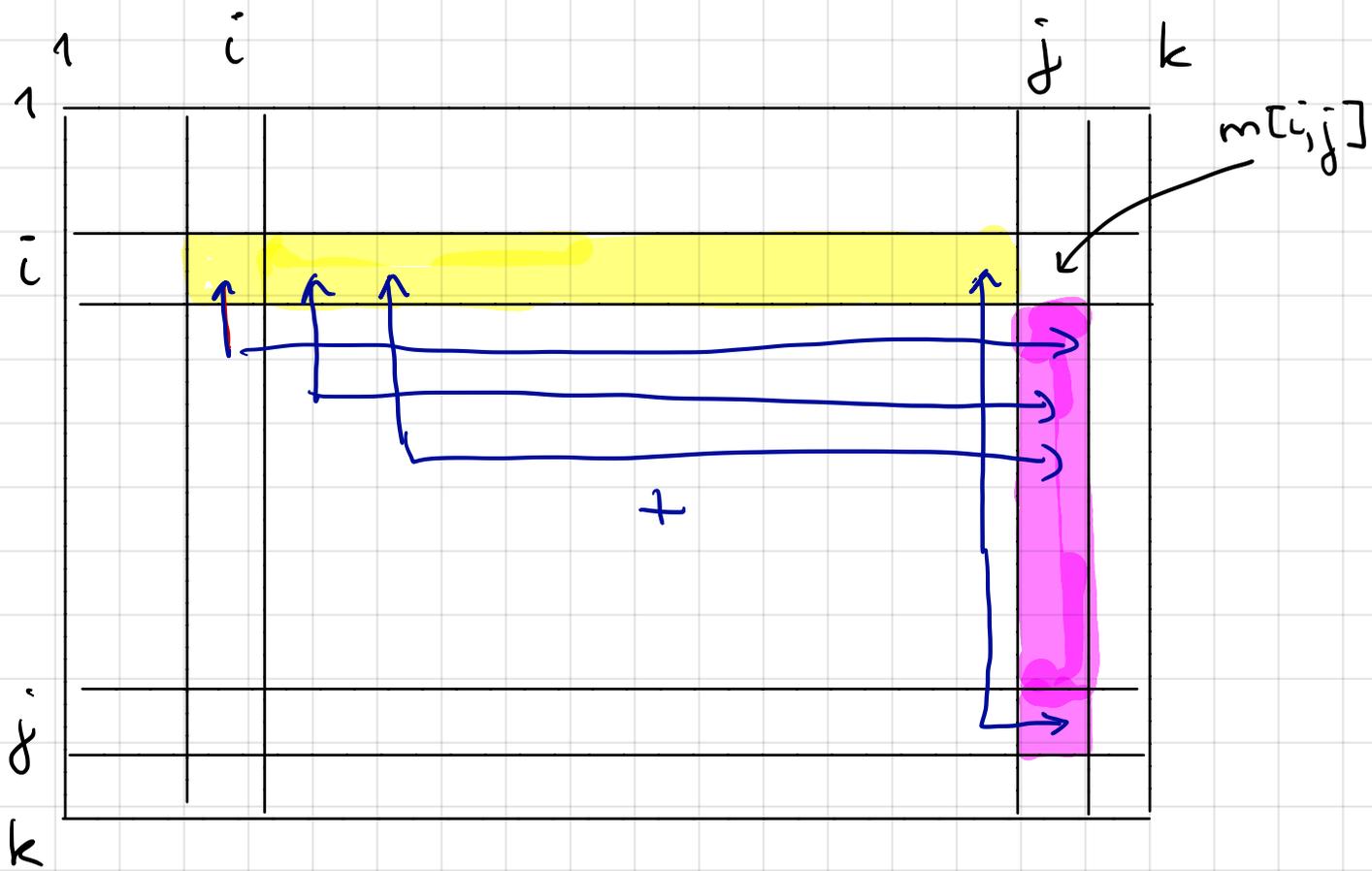
$$(A_i \dots A_k) (\dots A_j) \quad (A_i \dots A_k) (\dots A_j)$$

- ▶ Wir suchen dabei das **optimale** k , also:

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

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

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



Matrix m

 = $m[i,i], \dots, m[i, j-1]$

 = $m[i+1, j], \dots, m[j, j]$

for $i \neq j$:

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

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

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

- ▶ Wie bei Fibonacci wird z. B. das Teilproblem $m[1, 1]$ **mehrfach** verwendet: von $m[1, 2], m[1, 3], \dots, m[1, n]$ wobei n ist die Anzahl der Matrizen.

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

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

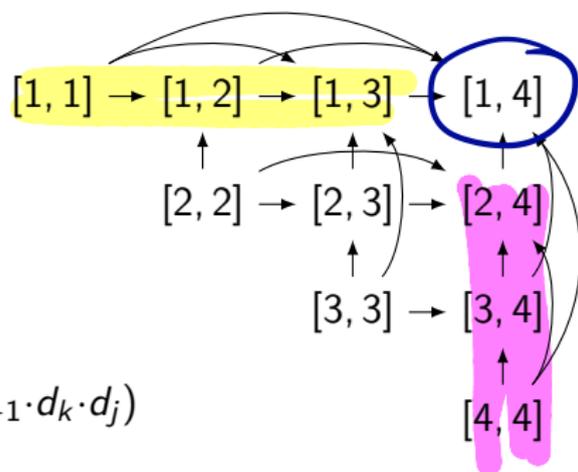
- ▶ Wie bei Fibonacci wird z. B. das Teilproblem $m[1, 1]$ **mehrfach** verwendet: von $m[1, 2], m[1, 3], \dots, m[1, n]$ wobei n ist die Anzahl der Matrizen.
- ▶ Es gibt für alle $1 \leq i < j \leq n$ ein Teilproblem, also insgesamt nur $\binom{n}{2} \in \Theta(n^2)$ Teilprobleme.

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

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

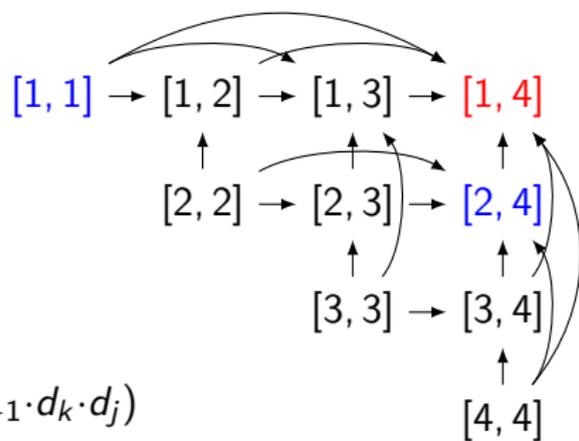
- ▶ Wie bei Fibonacci wird z. B. das Teilproblem $m[1, 1]$ **mehrfach** verwendet: von $m[1, 2], m[1, 3], \dots, m[1, n]$ wobei n ist die Anzahl der Matrizen.
- ▶ Es gibt für alle $1 \leq i < j \leq n$ ein Teilproblem, also insgesamt nur $\binom{n}{2} \in \Theta(n^2)$ Teilprobleme.
- ▶ Wählen wir eine **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)



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

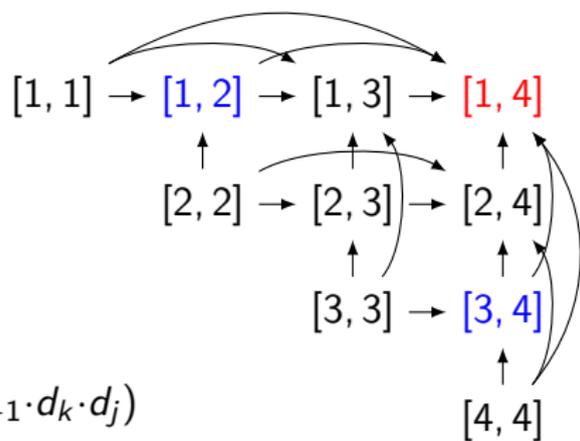
Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)



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

- Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.

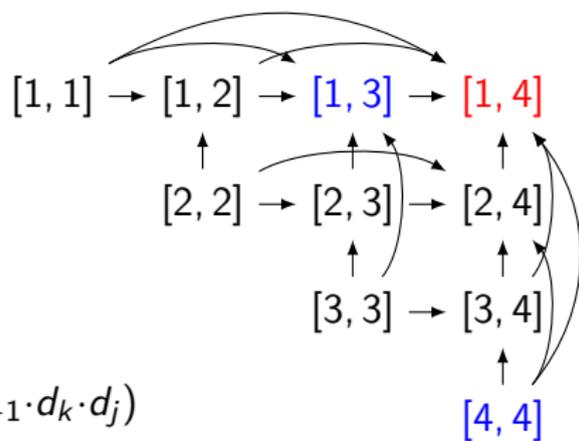
Ketten von Matrixmultiplikationen – Bottom-Up-Lösung (II)



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

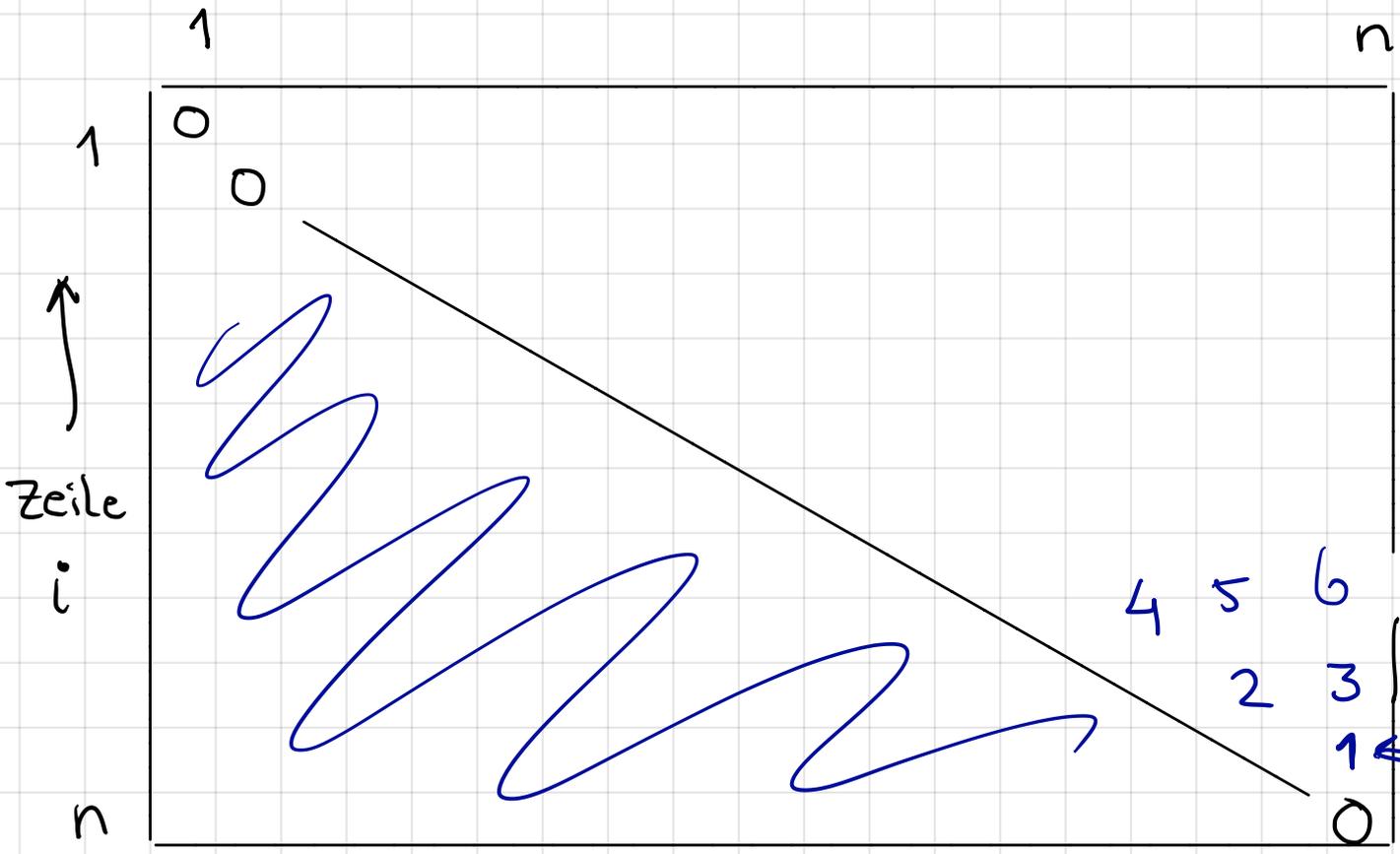
- ▶ Damit ist eine Zeitkomplexität von $\Theta(n^3)$ bei einem Platzbedarf von $\Theta(n^2)$ möglich.
- ▶ Erinnerung: Die naive Variante hätte $\Omega(2^n)$ Zeit, allerdings bei nur $\Theta(1)$ Platzbedarf, benötigt (*Time-Memory-Tradeoff*).

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



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

- ▶ 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.



→ Spalte j

$$m[i,j] = \left\{ \begin{array}{l} 0 \end{array} \right.$$

falls $i=j$

Anfang
 $i=n-1$
 $j=n$

Ketten von Matrixmultiplikationen – Algorithmus

```

1 // Eingabe: Die Dimensionen der Matrizen: dim[i]=di für i=0...n
2 int matMultOrder(int dim[n+1], int n) {
3     int m[n,n]; // hier 0-basiert!
4     for (int i = 0; i < n; i++)
5         m[i,i] = 0; // Diagonale
6     for (int i = n-1; i >= 0; i--) // Zeilen
7         for (int j = i+1; j < n; j++) { // Spalten
8             int curMin = +inf;
9             for (int k = i; k < j; k++) {
10                curMin = min(curMin,
11                    m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
12            }
13            m[i,j] = curMin;
14        }
15    return m[0,n-1];
16 }

```

min $\left. \begin{matrix} \dots \\ \dots \end{matrix} \right\} \dots$
 $i \leq k < j$

rekursionsgleichung

Ketten von Matrixmultiplikationen – Algorithmus

```
1 // Eingabe: Die Dimensionen der Matrizen: dim[i]=di für i=0...n
2 int matMultOrder(int dim[n+1], int n) {
3     int m[n,n]; // hier 0-basiert!
4     for (int i = 0; i < n; i++)
5         m[i,i] = 0; // Diagonale
6     for (int i = n-1; i >= 0; i--) // Zeilen
7         for (int j = i+1; j < n; j++) { // Spalten
8             int curMin = +inf;
9             for (int k = i; k < j-1; k++) {
10                curMin = min(curMin,
11                    m[i,k] + m[k+1,j] + dim[i]*dim[k+1]*dim[j+1]);
12            }
13            m[i,j] = curMin;
14        }
15     return m[0,n-1];
16 }
```

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

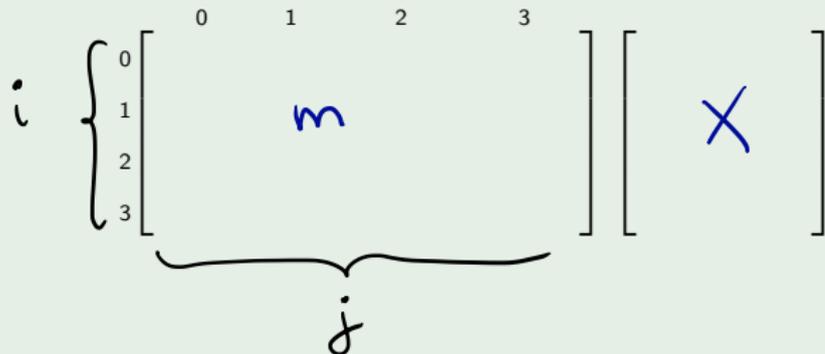
Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \text{dim}[i] * \text{dim}[k+1] * \text{dim}[j+1]);$$

$$\text{dim}[] = \{30, \quad 1, \quad 40, \quad 10, \quad 25\};$$



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

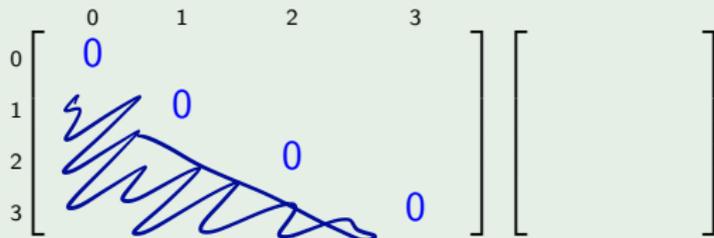
Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \text{dim}[i] * \text{dim}[k+1] * \text{dim}[j+1]);$

$\text{dim}[] = \{30, 1, 40, 10, 25\};$



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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i,j] = \min(m[i,k] + m[k+1,j] + \text{dim}[i] \cdot \text{dim}[k+1] \cdot \text{dim}[j+1]);$

$\text{dim}[] = \{30, 1, 40, 10, 25\};$

$$\begin{array}{c}
 0 \\
 1 \\
 2 \\
 3
 \end{array}
 \begin{bmatrix}
 0 & & & & \\
 & 0 & & & \\
 & & \times & & \\
 & & & 0 & \\
 & & & & 0
 \end{bmatrix}
 \begin{bmatrix}
 & & & & \\
 & & & & \\
 & & & & \\
 & & & & \\
 & & & &
 \end{bmatrix}
 \begin{bmatrix}
 & & & & \\
 & & & & \\
 & & & & \\
 & & & &
 \end{bmatrix}
 \begin{array}{c}
 i = 2, j = 3 \\
 A_2 A_3
 \end{array}$$

$0 + 0 + 40 \cdot 10 \cdot 25$
 $(k=2): A_2 \cdot A_3$

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$$

$$\dim[] = \{30, 1, 40, 10, 25\};$$

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

$$0 + 0 + 1 \cdot 40 \cdot 10$$

$$(k=1): A_1 \cdot A_2$$

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$$

$$\dim[] = \{30, 1, 40, 10, 25\};$$

$$\min \left\{ \begin{array}{l} \cancel{0 + 10000 + 1 \cdot 40 \cdot 25,} \\ \quad (k=1): A_1 \cdot (A_2 A_3) \end{array} \right. \quad \left. \begin{array}{l} \overbrace{400 + 0 + 1 \cdot 10 \cdot 25}^{650} \\ \quad (k=2): \underbrace{(A_1 A_2)} \cdot \underbrace{A_3} \end{array} \right\}$$

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$$

$$\dim[] = \{30, 1, 40, 10, 25\};$$

$$\begin{array}{c}
 0 \\
 1 \\
 2 \\
 3
 \end{array}
 \begin{array}{c}
 0 \\
 1 \\
 2 \\
 3
 \end{array}
 \begin{array}{c}
 0 \\
 1 \\
 2 \\
 3
 \end{array}
 \begin{array}{c}
 0 \\
 1 \\
 2 \\
 3
 \end{array}
 \left[\begin{array}{cccc}
 0 & 1200 & & \\
 & 0 & 400 & 650 \\
 & & 0 & 10\,000 \\
 & & & 0
 \end{array} \right]
 \left[\begin{array}{cc}
 0 & \\
 & 1 & 2 \\
 & & 2
 \end{array} \right]
 \begin{array}{l}
 i = 0, j = 1 \\
 A_0 A_1
 \end{array}$$

$$0 + 0 + 30 \cdot 1 \cdot 40$$

$$(k=0): A_0 \cdot A_1$$

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

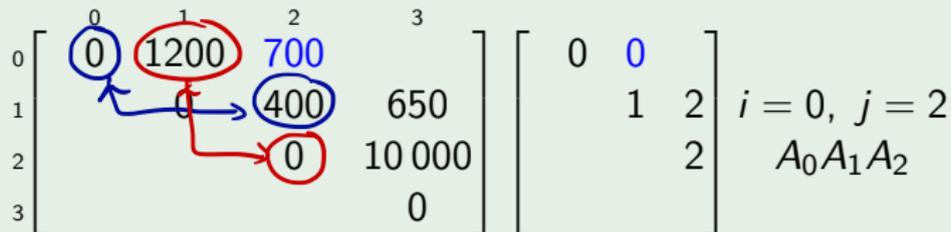
Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$$

$$\dim[] = \{30, 1, 40, 10, 25\};$$



$$0 + 400 + 30 \cdot 1 \cdot 10, \quad 1200 + 0 + 30 \cdot 40 \cdot 10$$

$$(k=0): A_0 \cdot (A_1 A_2) \quad (k=1): (A_0 A_1) \cdot A_2$$

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

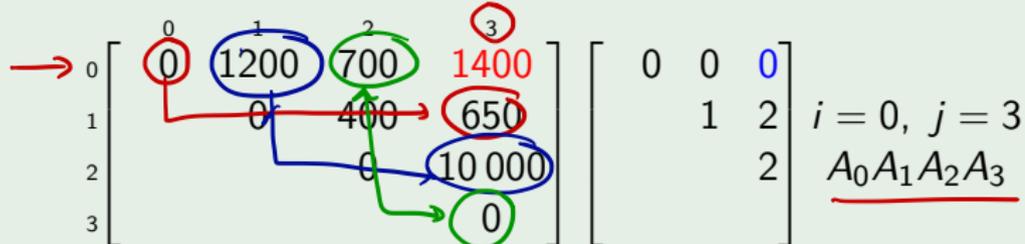
Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$

$\dim[] = \{30, 1, 40, 10, 25\};$



$0 + 650 + 30 \cdot 1 \cdot 25,$ $1200 + 10000 + 30 \cdot 40 \cdot 25,$ $700 + 0 + 30 \cdot 10 \cdot 25$
 $(k=0): A_0 \cdot (A_1 A_2 A_3)$ $(k=1): (A_0 A_1) \cdot (A_2 A_3)$ $(k=2): (A_0 A_1 A_2) \cdot A_3$

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] \cdot \dim[k+1] \cdot \dim[j+1]);$

$\dim[] = \{30, 1, 40, 10, 25\};$

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

$0 + 650 + 30 \cdot 1 \cdot 25,$ $1200 + 10\,000 + 30 \cdot 40 \cdot 25,$ $700 + 0 + 30 \cdot 10 \cdot 25$
 $(k=0): A_0 \cdot (A_1 A_2 A_3)$ $(k=1): (A_0 A_1) \cdot (A_2 A_3)$ $(k=2): (A_0 A_1 A_2) \cdot A_3$

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] * \dim[k+1] * \dim[j+1]);$$

$$\dim[] = \{30, 1, 40, 10, 25\};$$

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

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \text{dim}[i] * \text{dim}[k+1] * \text{dim}[j+1]);$

$\text{dim}[] = \{30, 1, 40, 10, 25\};$

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

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \dim[i] * \dim[k+1] * \dim[j+1]);$

$\dim[] = \{30, 1, 40, 10, 25\};$

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

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

Ketten von Matrixmultiplikationen – Beispiel

Beispiel

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

$m[i, j] = \min(m[i, k] + m[k+1, j] + \text{dim}[i] * \text{dim}[k+1] * \text{dim}[j+1]);$

$\text{dim}[] = \{30, 1, 40, 10, 25\};$

$$\begin{array}{c}
 \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \left[\begin{array}{cccc}
 0 & 1200 & 700 & \mathbf{1400} \\
 & 0 & 400 & 650 \\
 & & 0 & 10\,000 \\
 & & & 0
 \end{array} \right] \left[\begin{array}{ccc}
 0 & 0 & 0 \\
 & 1 & 2 \\
 & & 2
 \end{array} \right]
 \end{array}$$

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

Übersicht

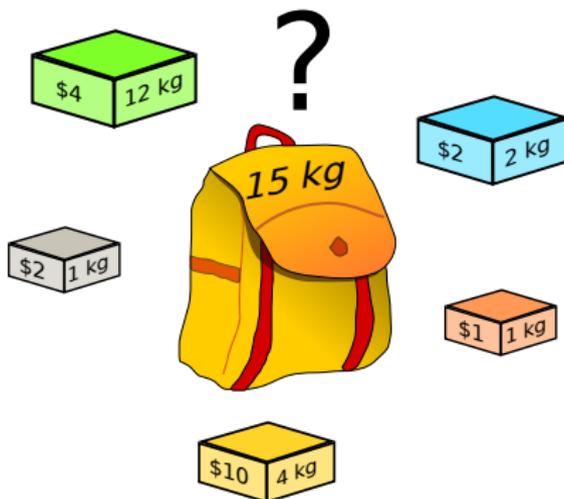
- 1 Motivation
- 2 Dynamische Programmierung
 - Rekursionsgleichungen
- 3 Anwendungen
 - Ketten von Matrixmultiplikationen
 - Das Rucksackproblem
 - Longest Common Subsequence (LCS)

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 , 15 kg

Das Rucksackproblem (II)

Gegeben:

- ▶ Maximale Tragkraft M ,
- ▶ n Gegenstände: $G = \{1, \dots, n\}$,

$n=5$

Das Rucksackproblem (II)

Gegeben:

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

$\{1, 2, 3, \dots\}$

Das Rucksackproblem (II)

Gegeben:

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

$\{1, 2, 3, \dots\}$

Das Rucksackproblem (II)

Gegeben:

- ▶ Maximale Tragkraft M ,
- ▶ n Gegenstände: $G = \{1, \dots, n\}$,
- ▶ 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: $G = \{1, \dots, n\}$,
- ▶ 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 $\sum_{i \in S} w_i \leq M$.

Das Rucksackproblem (II)

Gegeben:

- ▶ Maximale Tragkraft M ,
- ▶ n Gegenstände: $G = \{1, \dots, n\}$,
- ▶ 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 $\sum_{i \in S} w_i \leq M$.

Beispiel: die nächste Rakete zum ISS

Objekt	1	2	3	4	5	6	7	8
Gewicht	153	54	191	66	239	107	148	249
Profit	232	73	201	50	141	79	48	38
Profitaichte	1.52	1.35	1.05	0.76	0.59	0.58	0.32	0.15

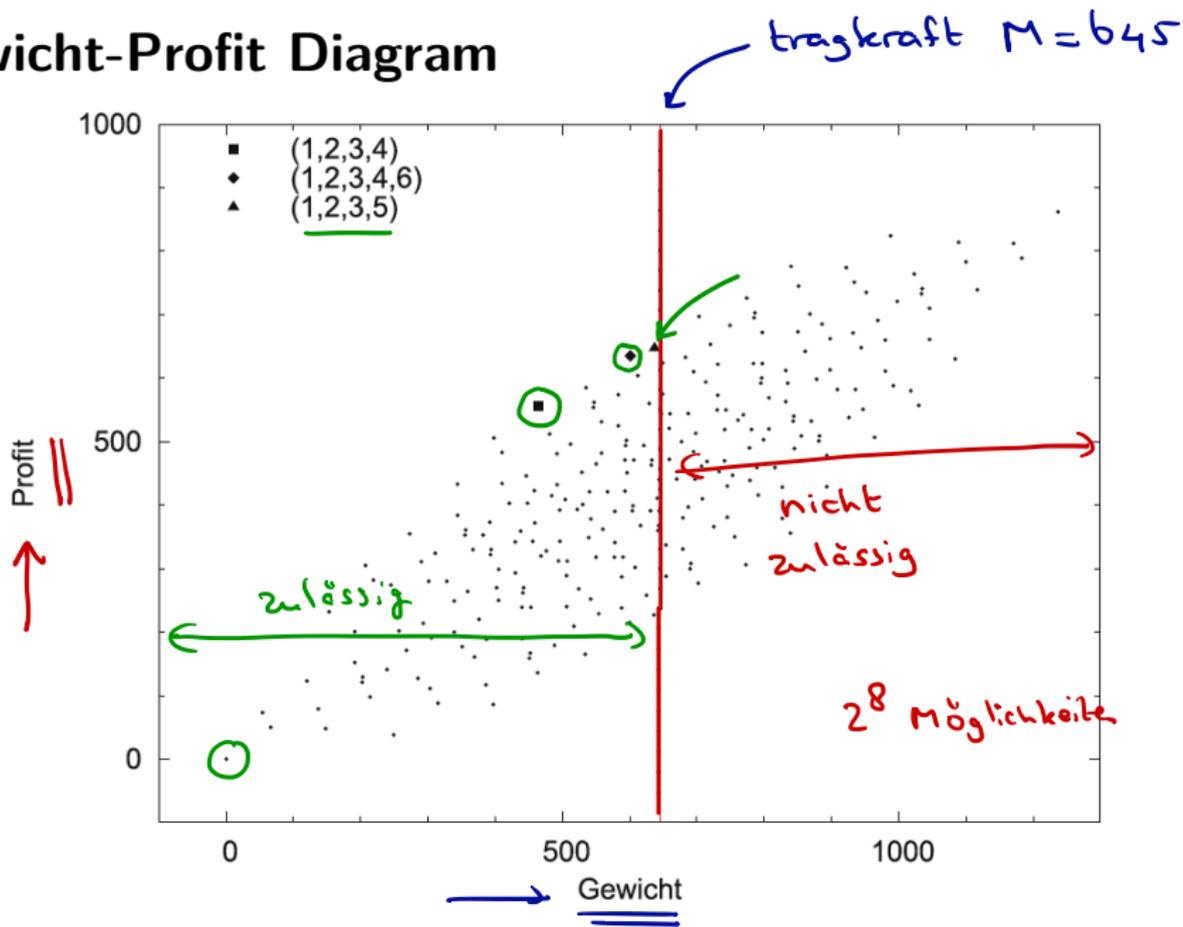
absteigend sortiert nach
Profitaichte

Tragkraft $M = 645$ kg

1, 2, 3, 4

153 + 54 + 191
+ 66

Gewicht-Profit Diagramm



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)

n Gegenstände

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 , Wert: c_n) 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 , Wert: c_n) 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 , Wert: c_n) mitgenommen wird.

Betrachten wir beide Fälle:

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

Mit: c_{\max} wäre dann gleich $c'_{\max} + c_n$ für $M' = M - w_n$.
 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 , Wert: c_n) mitgenommen wird.

Betrachten wir beide Fälle:

- ① **Ohne:** c_{\max} wäre dann gleich c'_{\max} für $M' = M$.
- ② **Mit:** c_{\max} wäre dann gleich $c'_{\max} + c_n$ für $M' = M - w_n$.
Falls $M' < 0$, dann setzen wir $c'_{\max} = -\infty$ („geht nicht“).

⇒ 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 $\{1, \dots, i\}$ berücksichtigt.

$$\subseteq \{1, \dots, n\}$$

Das Rucksackproblem – Rekursionsgleichung (II)

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

Es ergibt sich folgende Rekursionsgleichung:

$$C[i, j] = \begin{cases} 0 & \text{für } i = 0, j \geq 0 \\ -\infty & \text{für } j < 0 \\ \max(\underbrace{C[i-1, j]}_{\text{Gegenstand } i \text{ wird nicht mitgenommen}}, \underbrace{c_i + C[i-1, j-w_i]}_{\text{\(i\) wurde mitgenommen}}) \end{cases}$$

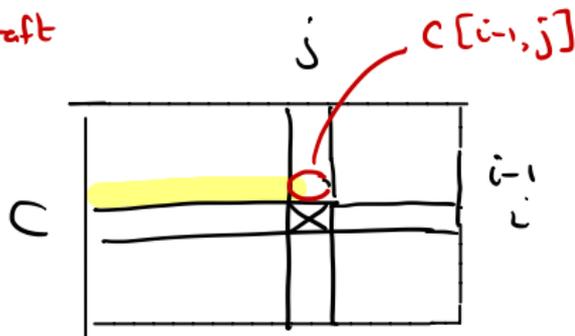
Das Rucksackproblem – Rekursionsgleichung (II)

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

Es ergibt sich folgende Rekursionsgleichung:

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

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



Das Rucksackproblem – Rekursionsgleichung (II)

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

Es ergibt sich folgende Rekursionsgleichung:

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

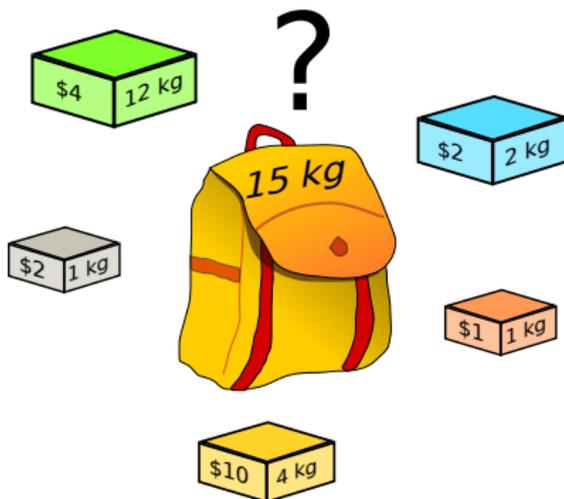
- ▶ Dann ist $c_{\max} = C[n, M]$.
- ▶ Diese Rekursionsgleichung lösen wir nun bottom-up, indem wir die Rucksäcke mit allen möglichen Gewichten $\{0, \dots, M\}$ berechnen, wenn wir jeweils einen weiteren Gegenstand hinzunehmen.

Das Rucksackproblem – Algorithmus

```
1 // Eingabe: Gewichte w[i], Werte c[i], Tragkraft M
2 // Index 0 bei Gewichten und Werten ungenutzt
3 int knapDP(int w[n+1], int c[n+1], int n, int M) {
4     int C[n+1,M+1];
5     for (int j = 0; j <= M; j++)
6         C[0,j] = 0;
7     for (int i = 1; i <= n; i++)
8         for (int j = 0; j <= M; j++)
9             if (w[i] <= j) {
10                C[i,j] = max(C[i-1,j], c[i] + C[i-1,j-w[i]]);
11            } else {
12                C[i,j] = C[i-1,j]; // passt nicht
13            }
14     return C[n,M];
15 }
```

- ▶ 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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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	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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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	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

$w[] = \{ X, 2, 12, 1, 1, \underline{4} \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

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

$w[] = \{ X, 2, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

5

←
4

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

$w[] = \{ X, 2, 12, 1, \textcircled{1}, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

$w[] = \{ X, 2, 12, \underline{1}, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

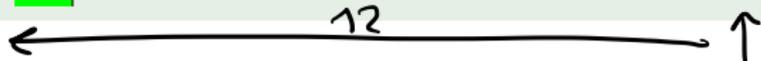
Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um c_{\max} zu erreichen.

- ▶ 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

$w[] = \{ X, 2, \underline{12}, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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

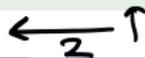
Offen ist noch die Frage, *welche* Gegenstände ($S \subseteq G$) nun eigentlich mitgenommen werden müssen, um c_{\max} zu erreichen.

- ▶ 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

$w[] = \{ X, \underline{2}, 12, 1, 1, 4 \}$, $c[] = \{ X, 2, 4, 2, 1, 10 \}$, $M = 15$

	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



Übersicht

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

Longest Common Subsequence – einige Begriffe

Teilsequenz

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

j L

seehofer

Longest Common Subsequence – einige Begriffe

Teilsequenz

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

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

Longest Common Subsequence – einige Begriffe

Teilsequenz

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

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

Beispiel

bcdb und *aa* sind Teilsequenzen von $A = *abcdbab*$.

Longest Common Subsequence – einige Begriffe

Teilsequenz

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

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

Beispiel

bcdb und *aa* sind Teilsequenzen von $A = *abcdbab*$.

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

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

Beispiel

*bcd***b** und *aa* sind Teilsequenzen von $A = \text{abc**b**dab}$.

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 $A = \text{abc**b**dab}$ und $B = \text{bd**c**aba}$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

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

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abc**bd**ab$ und $B = **bd**caba$, aber keine LCS.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

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

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abc**bd**ab$ und $B = **bd**caba$, aber keine LCS.

bcba ist eine LCS von $A = abc**bd**ab$ und $B = **bd**caba$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

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

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abc**bd**ab$ und $B = **bd**caba$, aber keine LCS.

bcba ist eine LCS von $A = abc**bd**ab$ und $B = **bd**caba$.

bdab ist auch eine LCS von $A = abc**bd**ab$ und $B = **bd**caba$.

Longest Common Subsequence – einige Begriffe

Problem der längsten gemeinsamen Teilsequenz

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

$a_1 a_2 \dots a_m$ – pro Buchst.
2 Möglichk.

Beispiel

bca ist eine gemeinsame Teilsequenz von $A = abc**bd**ab$ und $B = **bd**caba$, aber keine LCS.

bcba ist eine LCS von $A = abc**bd**ab$ und $B = **bd**caba$.

bdab ist auch eine LCS von $A = abc**bd**ab$ und $B = **bd**caba$.

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



abc b d a

$i=3$

abc

$i=4$

abc b

Longest Common Subsequence – Eigenschaften (I)

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

Lemma: Optimale Teilstruktur

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

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

low
—

uw
—

Longest Common Subsequence – Eigenschaften (I)

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

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

$$LCS(A_m, B_n) = LCS(A_m, B_{n-1}) \quad \text{oder}$$

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

Longest Common Subsequence – Eigenschaften (I)

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

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

$$LCS(A_m, B_n) = LCS(A_m, B_{n-1}) \quad \text{oder}$$

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

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: $\underline{L[i, j]} = \underline{|LCS(A_i, B_j)|}$

$$A_i = a_1 a_2 \dots a_i$$

$$B_j = b_1 b_2 \dots b_j$$

Longest Common Subsequence – Rekursiongleichung

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

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

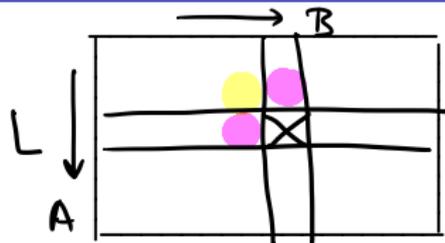
- Das lässt sich direkt als Algorithmus umsetzen (Hausaufgabe).

$$A = a_1 \dots a_{j-1} \boxed{a_j} \quad \textcircled{1}$$

$$B = b_1 b_2 \dots b_{j-1} \boxed{b_j} \quad \textcircled{2}$$

$a_i \neq b_j$

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).
- ▶ Dessen Laufzeit ist $O(\underbrace{|A|} \cdot \underbrace{|B|})$, 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:

- ↖ bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- ↑ bei $a_i \neq b_j$, für $L[i, j] = L[i - 1, j]$
- ← bei $a_i \neq b_j$, für $L[i, j] = L[i, j - 1]$ und nicht ↑

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:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 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:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\leftarrow 1$	$\swarrow 1$	$\leftarrow 1$				
E	0	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\uparrow 1$	$\swarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\swarrow 2$
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:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\leftarrow 1$	$\swarrow 1$	$\leftarrow 1$				
E	0	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\uparrow 1$	$\swarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\swarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
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:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 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:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0											
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0											
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\leftarrow 1$	$\swarrow 1$	$\leftarrow 1$				
E	0	$\uparrow 0$	$\uparrow 0$	$\swarrow 1$	$\uparrow 1$	$\swarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\swarrow 2$
U	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
T	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
S	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
C	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
H	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$						
L	0											
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 3	\leftarrow 3	\leftarrow 3	\leftarrow 3	\leftarrow 3
A	0											
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0											
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0											

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
\rightarrow D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Longest Common Subsequence – Beispiel

Zur Verdeutlichung verwenden wir folgende Notation:

- \swarrow bei $a_i = b_j$, und $L[i, j] = L[i - 1, j - 1] + 1$
- \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	\uparrow 0	\uparrow 0	\uparrow 0	\swarrow 1	\leftarrow 1	\swarrow 1	\leftarrow 1				
E	0	\uparrow 0	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\leftarrow 2	\swarrow 2
U	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
T	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
S	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
C	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
H	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2						
L	0	\uparrow 0	\uparrow 0	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow 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	\swarrow 4	\leftarrow 4	\leftarrow 4	\leftarrow 4
N	0	\swarrow 1	\leftarrow 1	\uparrow 1	\uparrow 1	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\swarrow 5	\leftarrow 5	\leftarrow 5
D	0	\uparrow 1	\uparrow 1	\uparrow 1	\swarrow 2	\uparrow 2	\uparrow 2	\uparrow 3	\uparrow 4	\uparrow 5	\swarrow 6	\leftarrow 6

Letzte Vorlesung

Letzte Vorlesung

Montag 9. Juli, 08:30 (Hörsaal H01). Bis dann!