




# Informatik II

## Dynamische Programmierung

G. Zachmann  
 Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)




- Zweite Technik für Algorithmenentwurf
- Zum Namen:
  - "*Dynamische ...*" hat nichts mit "Dynamik" zu tun, sondern mit Tabulierung
  - "... Programmierung" hat nichts mit "Programmieren" zu tun, sondern mit "Verfahren"
  - vergleiche "lineares Programmieren", "Integer-Programmieren" (alles Begriffe aus der Optimierungstheorie)
- Typische Anwendung für dynamisches Programmieren: Optimierung
- Vergleich mit Divide-and-Conquer:
  - Ähnlich: DP berechnet Lsg eines Problems aus Lösungen von Teilproblemen
  - Anders: Teilprobleme werden oft **nicht** rekursiv gelöst sondern **iterativ**, beginnend mit den Lösungen der kleinsten Teilprobleme (*bottom-up*)
- Wird häufig angewandt, wenn rekursiver Ansatz immer wieder dieselben Teilprobleme lösen würde

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 2

## Matrix Chain Multiplication Problem (MCMP)

- Gegeben: Folge (Kette)  $A_1, A_2, \dots, A_n$  von Matrizen mit **verschiedenen** Dimensionen
- Gesucht: Produkt  $A_1 \cdot A_2 \cdot \dots \cdot A_n$
- Aufgabe:  
Organisiere die Multiplikationen so, daß möglichst wenig skalare Multiplikationen ausgeführt werden. (Man nutzt Assoziativität aus.)
- Definition:  
Ein Matrizenprodukt heißt **vollständig geklammert**, wenn es entweder eine einzelne Matrix oder das geklammerte Produkt zweier vollständig geklammerter Matrizenprodukte ist.

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 3

## Multiplikation zweier Matrizen

$$A = (a_{ij})_{p \times q}, B = (b_{ij})_{q \times r}, A \cdot B = C = (c_{ij})_{p \times r}$$

$$c_{ij} = \sum_{k=1}^q a_{ik} b_{kj}$$

$$p \begin{matrix} \boxed{A} \\ q \end{matrix} \cdot q \begin{matrix} \boxed{B} \\ r \end{matrix} = \begin{matrix} \boxed{C} \\ r \end{matrix} p$$

```
# Eingabe: p×q Matrix A q×r Matrix B
# Ausgabe: p×r Matrix C = A·B
for i in range( 0,p ):
    for j in range( 0,r ):
        C[i,j] = 0
        for k in range( 0,q ):
            C[i,j] += A[i,k] * B[k,j]
```

- Anzahl der Multiplikationen und Additionen =  $p \cdot q \cdot r$
- Bem: für 2  $n \times n$ -Matrizen werden hier  $n^3$  Multiplikationen benötigt, es geht auch mit  $O(n^{2.378})$

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 4

## Beispiel

- Berechnung des Produkts von  $A_1, A_2, A_3$  mit
  - $A_1$  :  $10 \times 100$  Matrix
  - $A_2$  :  $100 \times 5$  Matrix
  - $A_3$  :  $5 \times 50$  Matrix

<p>1. Klammerung <math>((A_1 A_2) A_3)</math> erfordert</p> <table border="0"> <tr> <td><math>A' = (A_1 A_2)</math></td> <td>5000 Mult.</td> <td></td> <td></td> </tr> <tr> <td><math>A' A_3</math></td> <td>2500 Mult.</td> <td></td> <td></td> </tr> <tr> <td>Summe:</td> <td>7500 Mult.</td> <td></td> <td></td> </tr> </table>	$A' = (A_1 A_2)$	5000 Mult.			$A' A_3$	2500 Mult.			Summe:	7500 Mult.			<p>2. Klammerung <math>(A_1 (A_2 A_3))</math> erfordert</p> <table border="0"> <tr> <td><math>A'' = (A_2 A_3)</math></td> <td>25000 Mult.</td> <td></td> <td></td> </tr> <tr> <td><math>A_1 A''</math></td> <td>50000 Mult.</td> <td></td> <td></td> </tr> <tr> <td>Summe:</td> <td>75000 Mult.</td> <td></td> <td></td> </tr> </table>	$A'' = (A_2 A_3)$	25000 Mult.			$A_1 A''$	50000 Mult.			Summe:	75000 Mult.		
$A' = (A_1 A_2)$	5000 Mult.																								
$A' A_3$	2500 Mult.																								
Summe:	7500 Mult.																								
$A'' = (A_2 A_3)$	25000 Mult.																								
$A_1 A''$	50000 Mult.																								
Summe:	75000 Mult.																								

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 5

## Problemstellung

- Gegeben: Folge von Matrizen  $A_1, A_2, \dots, A_n$  und die Dimensionen  $p_0, p_1, \dots, p_n$ , wobei Matrix  $A_i$  Dimension  $p_{i-1} \times p_i$  hat
- Gesucht: Multiplikationsreihenfolge, die die Anzahl der Multiplikationen minimiert
- Beachte: der Algorithmus führt die Multiplikationen nicht aus, er bestimmt nur die optimale Reihenfolge
- Verallgemeinerung: ermittle die optimale Ausführungsreihenfolge für eine Menge von Operationen
  - im Compilerbau: Code-Optimierung
  - bei Datenbanken: Anfrageoptimierung

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 6

Beispiel für  $\langle A_1 \cdot A_2 \cdot \dots \cdot A_n \rangle$

- Alle vollständig geklammerten Matrizenprodukte der Kette  $\langle A_1, A_2, A_3, A_4 \rangle$  sind:
 
$$\left( A_1 (A_2 (A_3 A_4)) \right), \left( A_1 ((A_2 A_3) A_4) \right), \left( (A_1 A_2) (A_3 A_4) \right),$$

$$\left( (A_1 (A_2 A_3)) A_4 \right), \left( ((A_1 A_2) A_3) A_4 \right)$$
- Klammerungen entsprechen strukturell verschiedenen Auswertungsbäumen (siehe 1. Semester)

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 7

Anzahl der verschiedenen Klammerungen

- $P(n)$  sei die Anzahl der verschiedenen Klammerungen von  $A_1 \cdot \dots \cdot A_k \cdot A_{k+1} \cdot \dots \cdot A_n$ :
 
$$P(1) = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k) \quad \text{für } n \geq 2$$
- Definition:
 
$$P(n+1) =: C_n = n\text{-te Catalan'sche Zahl}$$
- Es gilt:
 
$$P(n+1) = \frac{1}{n+1} \binom{2n}{n} \approx \frac{4^n}{n\sqrt{\pi n}} + O\left(\frac{4^n}{\sqrt{n^5}}\right) \quad (\text{o. Bew.})$$
- Folge: Finden der optimalen Klammerung durch Ausprobieren aller Möglichkeiten sinnlos

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 8

## Struktur der optimalen Klammerung

- Sei  $A_{i...j}$  das Produkt der Matrizen  $i$  bis  $j$ ;  $A_{i...j}$  ist eine  $p_{i-1} \times p_j$ -Matrix
- Behauptung:  
Jede optimale Lösung des MCMP enthält optimale Lösungen von Teilproblemen.
- Anders gesagt:  
Jede optimale Lösung des MCMP setzt sich zusammen aus optimalen Lösungen von bestimmten Teilproblemen.

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 9

## Beweis (durch Widerspruch)

- Sei eine optimale Lösung so geklammert  

$$A_{i...j} = (A_{i...k}) (A_{k+1...j}) \quad , \quad i \leq k < j$$
- Behauptung → Klammerung innerhalb  $A_{i...k}$  muß ihrerseits optimal sein
- Ann.: die Klammerung von  $A_{i...k}$  innerhalb der optimalen Lsg zu  $A_{i...j}$  sei nicht optimal
- Es gibt bessere Klammerung von  $A_{i...k}$  mit geringeren Kosten
- Setze diese Teillösung in Lösung zu  $A_{i...j} = (A_{i...k}) (A_{k+1...j})$  ein
- Ergibt eine bessere Lösung → Widerspruch zur Ann. der Optimalität der ursprünglichen Lsg zu  $A_{i...j}$

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 10

## Rekursive Lösung

- Auf der höchsten Stufen werden 2 Matrizen multipliziert, d.h., für jedes  $k$ ,  $1 \leq k \leq n-1$ , ist  $(A_{1\dots n}) = ((A_{1\dots k}) (A_{k+1\dots n}))$
- Die optimalen Kosten können beschrieben werden als
  - $i = j \rightarrow$  Folge enthält nur eine Matrix, keine Kosten
  - $i < j \rightarrow$  kann geteilt werden, indem jedes  $k$ ,  $i \leq k < j$  betrachtet wird, Kosten für bestimmtes  $k$  = "Kosten links" + "Kosten rechts" + Kosten für  $(A_{i\dots k}) \cdot (A_{k+1\dots j})$
- Daraus lässt sich die folgende rekursive Regel ableiten:
  - $m[i,j]$  sei minimale Anzahl von Operationen zur Berechnung des Teilprodukts  $A_{i\dots j}$
$$m[i,j] = \begin{cases} 0, & \text{falls } i = j \\ \min_{i \leq k < j} \{ m[i,k] + m[k+1,j] + p_{i-1}p_kp_j \} \end{cases}$$

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 11

## Naiver rekursiver Algo

- Invariante:  $mcm\_rek(p, i, j)$  liefert  $m[i,j]$

```
def mcm_rek( p,i,j ):
    if i = j:
        return 0
    m = ∞
    for k in range( i,j ):
        q = p[i-1]*p[k]*p[j] + \
            mcm_rek(p,i,k) + \
            mcm_rek(p,k+1,j)
        if q < m:
            m = q
    return m
```

- Aufruf:  $mcm\_rek(p, 1, n)$

G. Zachmann Informatik 2 - SS 06 Dynamische Programmierung 12



## Laufzeit der naïven rekursiven Lösung



- Sei  $T(n)$  die Anzahl der Schritte zur Berechnung von rek-mat-kat für Eingabefolge der Länge  $n$

$$T(1) = 1$$

$$T(n) = 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 2)$$

$$\approx 2n + 2 \sum_{k=1}^{n-1} T(k) \Rightarrow$$

$$T(n) \geq 2^{n-1} \quad (\text{vollständige Induktion})$$

- exponentielle Laufzeit!