Algorithmentheorie

14 – Dynamische Programmierung (2)

Matrixkettenprodukt

Prof. Dr. S. Albers

Das Optimalitätsprinzip

Typische Anwendung für dynamisches Programmieren:

Optimierungsprobleme

Eine optimale Lösung für das Ausgangsproblem setzt sich aus *optimalen* Lösungen für kleinere Probleme zusammen.

Kettenprodukt von Matrizen

Gegeben: Folge (Kette) $\langle A_1, A_2, \ldots, A_n \rangle$ von Matrizen

Gesucht: Produkt $A_1 \cdot A_2 \cdot \ldots \cdot A_n$

Problem: Organisiere die Multiplikation so, dass möglichst wenig skalare Multiplikationen ausgeführt werden.

Definition: Ein Matrizenprodukt heißt *vollständig geklammert*, wenn es entweder eine einzelne Matrix oder das geklammerte Produkt zweier vollständig geklammerter Matrizenprodukte ist.

Beispiel für vollständig geklammerte Matrizenprodukte der Kette $\langle A_1, A_2, \ldots, A_n \rangle$

Alle vollständig geklammerten Matrizenprodukte der Kette $\langle A_1, A_2, A_3, A_4 \rangle$ sind:

$$(A_1 (A_2 (A_3 A_4)))$$

$$(A_1 ((A_2 A_3) A_4))$$

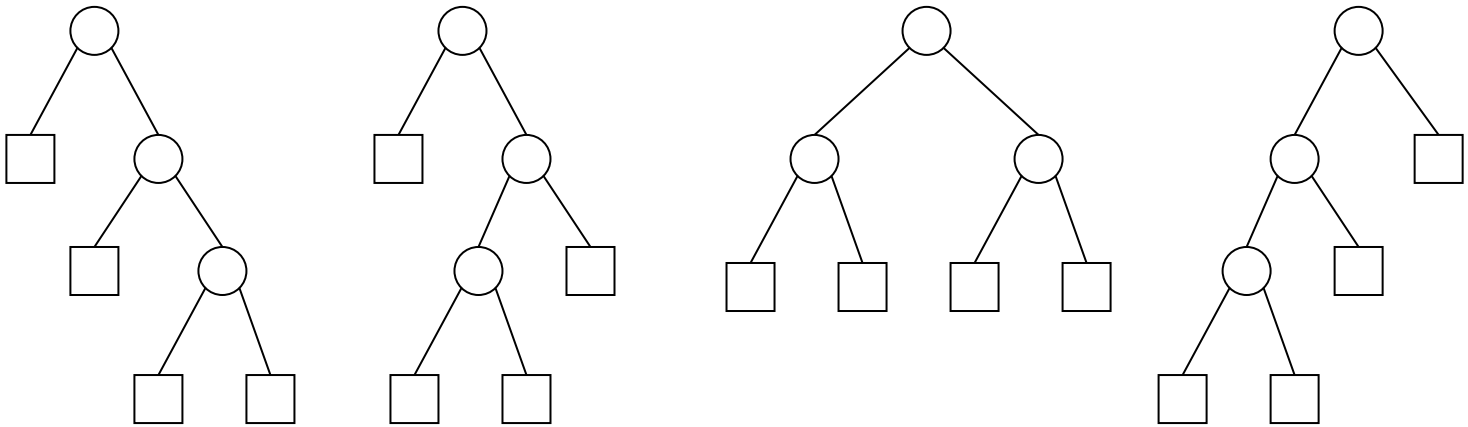
$$((A_1 A_2) (A_3 A_4))$$

$$((A_1 (A_2 A_3)) A_4)$$

$$(((A_1 A_2) A_3) A_4)$$

Anzahl der verschiedenen Klammerungen

Klammerungen entsprechen strukturell verschiedenen Bäumen.



Anzahl der verschiedenen Klammerungen

P(n) sei die Anzahl der verschiedenen Klammerungen von $A_1 \ldots A_k A_{k+1} \ldots A_n$

$$P(1) = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k) P(n-k) \quad \text{für } n \geq 2$$

$$P(n+1) = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n\sqrt{\pi n}} + \mathrm{O}\left(\frac{4^n}{\sqrt{n^5}}\right)$$

$$P(n+1) = C_n \quad n-\text{te Catalansche Zahl}$$

Bem: Finden der optimalen Klammerung durch Ausprobieren sinnlos.

Multiplikation zweier Matrizen

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

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

Algorithmus *Matrix-Mult*
Input: Eine ($p \times q$) Matrix A und eine ($q \times r$) Matrix B
Output: Die ($p \times$ r) Matrix C = A · B

```
1  for i := 1 to p do
2      for j :=1 to r do
3          C[i, j] := 0
4          for k := 1 to q do
5              C[i, j] := C[i, j] + A[i, k] ·B[k,j]
```

Anzahl Multiplikationen und Additionen: $p \cdot q \cdot r$

Bem: für zwei $n \times n$ – Matrizen werden hier n^3 Multiplikationen benötigt.
Es geht auch mit $O(n^{2.376})$ Multiplikationen.

Matrizenkettenprodukt 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

a) Klammerung $((A_1 A_2) A_3)$ erfordert

$A' = (A_1 A_2)$:

$A' A_3$:

Summe:

Matrizenkettenprodukt Beispiel

A_1 :10×100 Matrix

A_2 :100×5 Matrix

A_3 : 5×50 Matrix

a) Klammerung $(A_1 (A_2 A_3))$ erfordert

$A'' = (A_2 A_3)$:

$A_1 A''$:

Summe:

Struktur der optimalen Klammerung

$(A_{i...j}) = ((A_{i...k})\ (A_{k+1....j}))$ $i \leq k < j$

Jede optimale Lösung des Matrixkettenprodukt Problems enthält optimale Lösungen von Teilproblemen.

Rekursive Bestimmung des Wertes einer optimalen Lösung:

m[i,j] sei minimale Anzahl von Operationen zur Berechung des Teilproduktes $A_{i...j}$:

m[i,j] = 0 falls $i = j$

m[i,j] = $\min_{i \leq k < j} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \}$,sonst

s[i,j] = optimaler Splitwert *k*, für den das Minimum angenommen wird

Rekursive Matrixkettenprodukt

Algorithmus *rek-mat-ket*(*p, i, j*)
Input: Eingabefolge $p = \langle p_0, p_1, ..., p_n \rangle$, $p_{i-1} \times p_i$ Dimensionen der Matrix A_i
Invariante: rek-mat-ket(*p, i, j*) liefert *m*[*i, j*]

```
1  if i = j then return 0
2  m[i, j] := ∞
3  for k := i to j – 1 do
4        m[i, j] := min(m[i,j], p_{i-1} p_k p_j +
                                  rek-mat-ket(p, i, k) +
                                  rek-mat-ket(p, k+1, j))
5  return m[i, j]
```

Aufruf: rek-mat-ket(*p*,1, *n*)

Rekursives Matrixkettenprodukt, Laufzeit

Sei $T(n)$ die Anzahl der Schritte zur Berechnung von rek-mat-ket(p,1,n).

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} \left(T(k) + T(n-k) + 1 \right)$$

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

$$\Rightarrow T(n) \geq 3^{n-1} \quad \text{(vollst. Induktion)}$$

Exponentielle Laufzeit!

Matrixkettenprodukt mit dynamischer Programmierung

Algorithmus *dyn-mat-ket*

Input: Eingabefolge $p = \langle p_0, p_1, \ldots, p_n \rangle$ $p_{i-1} \times p_i$ Dimension der Matrix A_i

Output: $m[1,n]$

```
1  n:= length(p)
2  for i:= 1 to n do m[i, i] := 0
3  for l:=2 to n do                          /* l = Länge des Teilproblems */
4    for i := 1 to n – l + 1 do              /* i ist der linke Index */
5        j := i + l – 1                      /* j ist der rechte Index*/
6        m[i, j] := ∞
7        for k := i to j - 1 do
8            m[i, j] := min(m[i, j], p_{i-1} p_k p_j + m[i, k] + m[k + 1, j])
9  return m[1, n]
```

Berechnungsbeispiel

A_1 30×35

A_2 35×15

A_3 15×5

A_4 5×10

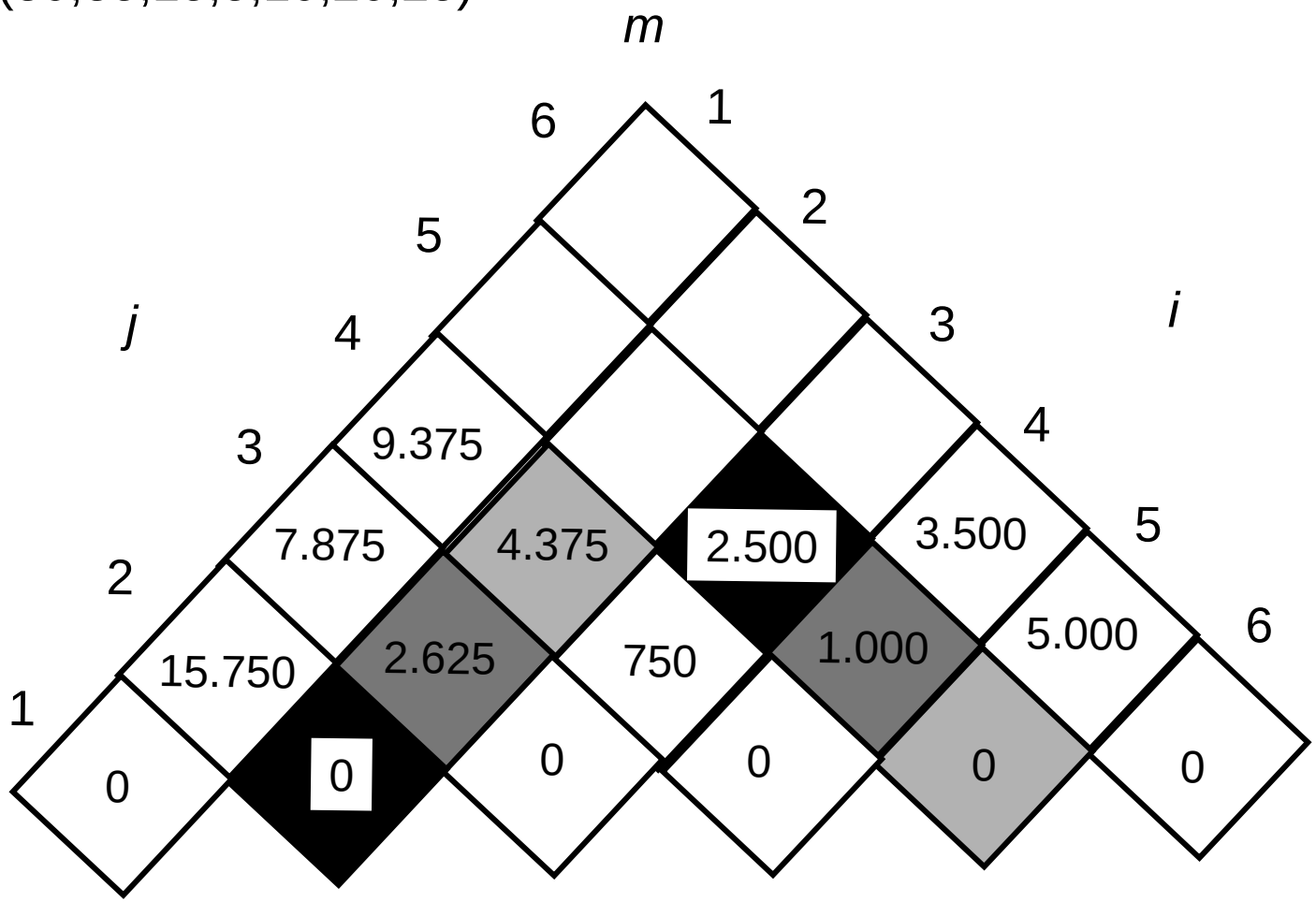
A_5 10×20

A_6 20×25

P = (30,35,15,5,10,20,25)

Berechnungsbeispiel

P = (30,35,15,5,10,20,25)



Berechnungsbeispiel

$$m[2,5] =$$

$$\min_{2 \le k < 5} \left(m[2,k] + m[k+1,5] + p_1 p_k p_5 \right)$$

$$\min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 \end{cases}$$

$$\min \begin{cases} 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

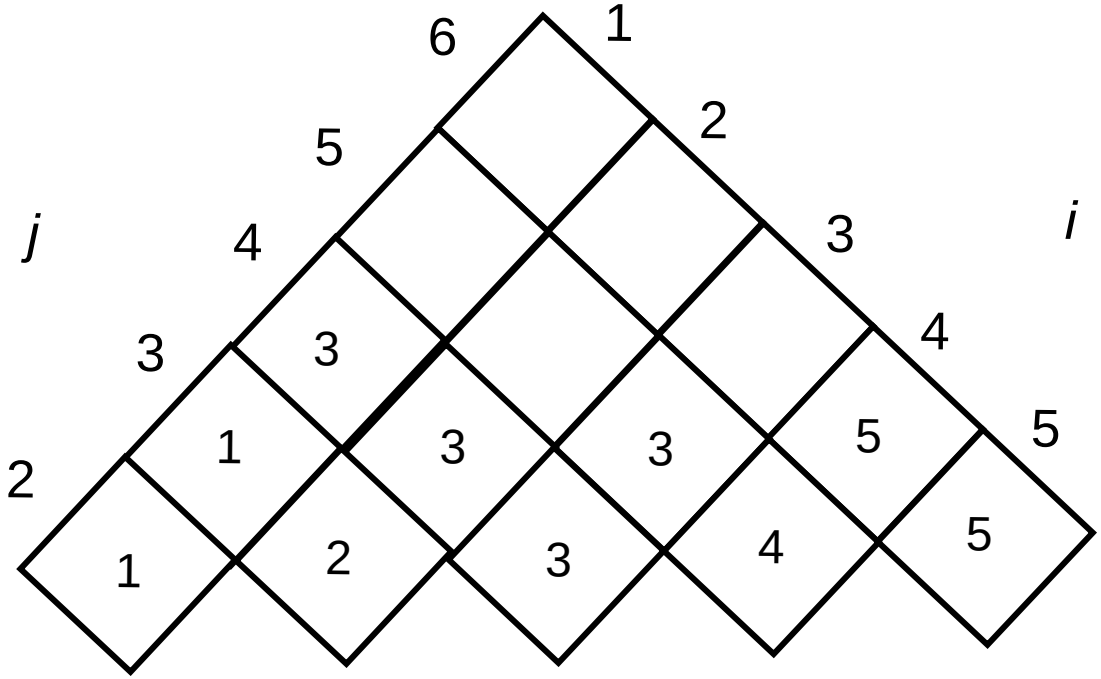
$$= 7125$$

Matrixkettenprodukt und optimale Splitwerte mit dynamischer Programmierung

Algorithmus *dyn-mat-ket(p)*
Input: Eingabefolge $p = \langle p_0, p_1, \ldots, p_n \rangle$ $p_{i-1} \times p_i$ Dimension der Matrix A_i
Output: $m[1,n]$ und eine Matrix $s[i,j]$ von opt. Splitwerten

```
1  n := length(p)
2  for i := 1 to n do m[i, i] := 0
3  for l := 2 to n do
4      for i := 1 to n – l +1 do
5          j := i + l – 1
6          m[i, j] := ∞
7          for k := i to j – 1 do
8              q := m[i, j]
9              m[i, j] := min(m[i, j], p_{i-1} p_k p_j + m[i, k] + m[k + 1, j])
10             if m[i, j] < q then s[i, j] := k
11 return (m[1, n], s)
```

Berechnungsbeispiel für Splitwerte



Berechnung der optimalen Klammerung

Algorithmus *Opt-Klam*

Input: Die Sequenz *A* der Matrizen, die Matrix *s* der optimalen Splitwerte, zwei Indizes *i* und *j*

Output: Eine optimale Klammerung von $A_{i...j}$

```
1   if i < j
2      then X := Opt-Klam(A, s, i, s[i, j])
3           Y := Opt-Klam(A, s, s[i, j] + 1, j)
4           return (X·Y)
5      else return A_i
```

Aufruf: *Opt–Klam*(*A*, *s*, 1, *n*)

Matrixkettenprodukt mit dynamischer Programmierung (Top-down Ansatz)

Notizblock-Methode zur Beschleunigung einer rekursiven Problemlösung:

Ein Teilproblem wird nur beim *ersten Auftreten* gelöst, die Lösung wird in einer Tabelle gespeichert und bei jedem späteren Auftreten desselben Teilproblems wird die Lösung (ohne erneute Rechnung!) in der Lösungstabelle nachgesehen.

Memoisiertes Matrixkettenprodukt (Notizblock-Methode)

Algorithmus *mem-mat-ket*(*p, i, j*)

Invariante: mem-mat-ket(*p, i, j*) liefert $m[i, j]$ und $m[i, j]$ hat den korrekten Wert, falls $m[i, j] < \infty$

1 **if** $i = j$ **then return** 0

2 **if** $m[i, j] < \infty$ **then return** $m[i, j]$

3 **for** $k := i$ **to** $j - 1$ **do**

4 $m[i, j] := \min(m[i, j], p_{i-1}\, p_k\, p_j +$
mem-mat-ket(*p, i, k*) +
mem-mat-ket(*p*, $k + 1$, *j*))

5 **return** $m[i, j]$

Memoisiertes Matrixkettenprodukt (Notizblock-Methode)

Aufruf:

```
1  n:= length(p) – 1
2  for i := 1 to n do
3      for j := 1 to n do
4          m[i, j] := ∞
5  mem-mat-ket(p,1,n)
```

Zur Berechnung aller Einträge $m[i, j]$ mit Hilfe von mem-mat-ket genügen insgesamt $O(n^3)$ Schritte.

$O(n^2)$ Einträge
jeder Eintrag $m[i, j]$ wird einmal eingetragen
jeder Eintrag $m[i, j]$ wird zur Berechnung eines Eintrages $m[i', j']$ betrachtet, falls $i' = i$ und $j' > j$ oder $j' = j$ und $i' < i$
→ $\leq 2n$ Einträge benötigen $m[i, j]$

Bemerkung zum Matrixkettenprodukt

1. Es gibt einen Algorithmus mit Laufzeit $O(n \log n)$, der optimale Klammerung findet.

3. Es gibt einen Algorithmus mit linearer Laufzeit $O(n)$, der eine Klammerung findet mit Multiplikationsaufwand $\leq 1.155\ M_{opt}$