

## 19. Dynamic Programming I

Fibonacci, Längste aufsteigende Teilfolge, längste gemeinsame Teilfolge, Editierdistanz, Matrixkettenmultiplikation, Matrixmultiplikation nach Strassen [Ottman/Widmayer, Kap. 1.2.3, 7.1, 7.4, Cormen et al, Kap. 15]

### Fibonacci Zahlen

☹️ (schon wieder)

$$F_n := \begin{cases} 1 & \text{wenn } n < 2 \\ F_{n-1} + F_{n-2} & \text{wenn } n \geq 3. \end{cases}$$

Analyse: warum ist der rekursive Algorithmus so langsam.

495

496

### Algorithmus FibonacciRecursive( $n$ )

**Input :**  $n \geq 0$

**Output :**  $n$ -te Fibonacci Zahl

**if**  $n \leq 2$  **then**

$f \leftarrow 1$

**else**

$f \leftarrow \text{FibonacciRecursive}(n-1) + \text{FibonacciRecursive}(n-2)$

**return**  $f$

### Analyse

$T(n)$ : Anzahl der ausgeführten Operationen.

■  $n = 1, 2$ :  $T(n) = \Theta(1)$

■  $n \geq 3$ :  $T(n) = T(n-2) + T(n-1) + c$ .

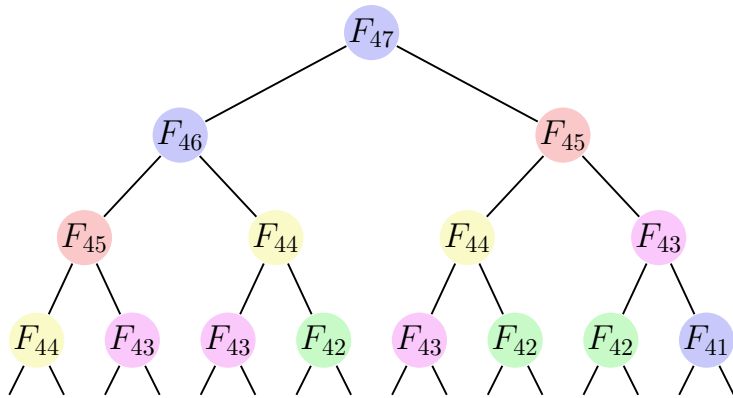
$$T(n) = T(n-2) + T(n-1) + c \geq 2T(n-2) + c \geq 2^{n/2}c' = (\sqrt{2})^n c'$$

Algorithmus ist *exponentiell (!)* in  $n$ .

497

498

## Grund, visualisiert



Knoten mit denselben Werten werden oft ausgewertet.

499

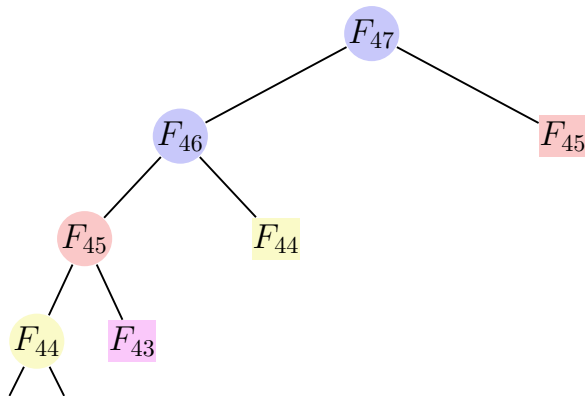
## Memoization

*Memoization* (sic) Abspeichern von Zwischenergebnissen.

- Bevor ein Teilproblem gelöst wird, wird Existenz eines entsprechenden Zwischenergebnis geprüft.
- Existiert ein gespeichertes Zwischenergebnis bereits, so wird dieses verwendet.
- Andernfalls wird der Algorithmus ausgeführt und das Ergebnis wird entsprechend gespeichert.

500

## Memoization bei Fibonacci



Rechteckige Knoten wurden bereits ausgewertet.

501

## Algorithmus FibonacciMemoization( $n$ )

**Input :**  $n \geq 0$

**Output :**  $n$ -te Fibonacci Zahl

**if**  $n \leq 2$  **then**

  |  $f \leftarrow 1$

**else if**  $\exists \text{memo}[n]$  **then**

  |  $f \leftarrow \text{memo}[n]$

**else**

  |  $f \leftarrow \text{FibonacciMemoization}(n - 1) + \text{FibonacciMemoization}(n - 2)$

  |  $\text{memo}[n] \leftarrow f$

**return**  $f$

502

## Analyse

Berechnungsaufwand:

$$T(n) = T(n - 1) + c = \dots = \mathcal{O}(n).$$

Algorithmus benötigt  $\Theta(n)$  Speicher.<sup>24</sup>

<sup>24</sup>Allerdings benötigt der naive Algorithmus auch  $\Theta(n)$  Speicher für die Rekursionsverwaltung.

## Genauer hingesehen ...

... berechnet der Algorithmus der Reihe nach die Werte  $F_1, F_2, F_3, \dots$   
... verkleidet im *Top-Down* Ansatz der Rekursion.

Kann den Algorithmus auch gleich *Bottom-Up* hinschreiben. Man spricht dann auch von *dynamischer Programmierung*.

## Algorithmus FibonacciDynamicProgram(n)

**Input** :  $n \geq 0$

**Output** :  $n$ -te Fibonacci Zahl

$F[1] \leftarrow 1$

$F[2] \leftarrow 1$

**for**  $i \leftarrow 3, \dots, n$  **do**

$F[i] \leftarrow F[i - 1] + F[i - 2]$

**return**  $F[n]$

## Dynamic Programming: Vorgehen

- 1 Verwalte *DP-Tabelle* mit Information zu den Teilproblemen.  
Dimension der Tabelle? Bedeutung der Einträge?
- 2 Berechnung der *Randfälle*.  
Welche Einträge hängen nicht von anderen ab?
- 3 *Berechnungsreihenfolge* bestimmen.  
In welcher Reihenfolge können Einträge berechnet werden, so dass benötigte Einträge jeweils vorhanden sind?
- 4 Auslesen der *Lösung*.  
Wie kann sich Lösung aus der Tabelle konstruieren lassen?

Laufzeit (typisch) = Anzahl Einträge der Tabelle mal Aufwand pro Eintrag.

## Dynamic Programming: Vorgehen am Beispiel

- 1 Dimension der Tabelle? Bedeutung der Einträge?  
Tabelle der Grösse  $n \times 1$ .  $n$ -ter Eintrag enthält  $n$ -te Fibonacci Zahl.
- 2 Welche Einträge hängen nicht von anderen ab?  
Werte  $F_1$  und  $F_2$  sind unabhängig einfach "berechenbar".
- 3 In welcher Reihenfolge können Einträge berechnet werden, so dass benötigte Einträge jeweils vorhanden sind?  
 $F_i$  mit aufsteigenden  $i$ .
- 4 Wie kann sich Lösung aus der Tabelle konstruieren lassen?  
 $F_n$  ist die  $n$ -te Fibonacci-Zahl.

507

## Längste aufsteigende Teilfolge (LAT)

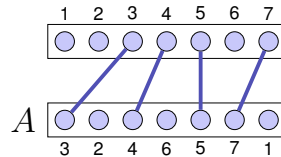


Verbinde so viele passende Anschlüsse wie möglich, ohne dass sich die Anschlüsse kreuzen.

508

## Formalisieren

- Betrachte Folge  $A = (a_1, \dots, a_n)$ .
- Suche eine längste aufsteigende Teilfolge von  $A$ .
- Beispiele aufsteigender Teilfolgen:  
(3, 4, 5), (2, 4, 5, 7), (3, 4, 5, 7), (3, 7).



Verallgemeinerung: Lasse Zahlen ausserhalb von  $1, \dots, n$  zu, auch mit Mehrfacheinträgen. Lasse nur strikt aufsteigende Teilfolgen zu.  
Beispiel: (2,3,3,3,5,1) mit aufsteigender Teilfolge (2,3,5).

509

## Erster Entwurf

Annahme: LAT  $L_k$  für  $k$  bekannt. Wollen nun LAT  $L_{k+1}$  für  $k+1$  berechnen.

Wenn  $a_{k+1}$  zu  $L_k$  passt, dann  $L_{k+1} = L_k \oplus a_{k+1}$

Gegenbeispiel:  $A_5 = (1, 2, 5, 3, 4)$ . Sei  $A_3 = (1, 2, 5)$  mit  $L_3 = A$ . Bestimme  $L_4$  aus  $L_3$ ?

So kommen wir nicht weiter: können nicht von  $L_k$  auf  $L_{k+1}$  schliessen.

510

## Zweiter Entwurf

Annahme: eine LAT  $L_j$  für alle  $j \leq k$  bekannt. Wollen nun LAT  $L_{k+1}$  für  $k + 1$  berechnen.

Betrachte alle passenden  $L_{k+1} = L_j \oplus a_{k+1}$  ( $j \leq k$ ) und wähle eine längste solche Folge.

Gegenbeispiel:  $A_5 = (1, 2, 5, 3, 4)$ . Sei  $A_4 = (1, 2, 5, 3)$  mit  $L_1 = (1)$ ,  $L_2 = (1, 2)$ ,  $L_3 = (1, 2, 5)$ ,  $L_4 = (1, 2, 5)$ . Bestimme  $L_5$  aus  $L_1, \dots, L_4$ ?

So kommen wir nicht weiter: können nicht von *jeweils nur einer beliebigen Lösung*  $L_j$  auf  $L_{k+1}$  schließen. Wir müssten alle möglichen LAT betrachten. Zu viel!

511

## Dritter Entwurf

Annahme: die LAT  $L_j$ , *welche mit kleinstem Element endet* sei für alle Längen  $1 \leq j \leq k$  bekannt.

Betrachte nun alle passenden  $L_j \oplus a_{k+1}$  ( $j \leq k$ ) und aktualisiere die Tabelle der längsten aufsteigenden Folgen, welche mit kleinstem Element enden.

Beispiel:  $A = (1, 1000, 1001, 2, 3, 4, \dots, 999)$

A	LAT
(1)	(1)
(1, 1000)	(1), (1, 1000)
(1, 1000, 1001)	(1), (1, 1000), (1, 1000, 1001)
(1, 1000, 1001, 2)	(1), (1, 2), (1, 1000, 1001)
(1, 1000, 1001, 2, 3)	(1), (1, 2), (1, 2, 3)

512

## DP Table

■ Idee: speichere jeweils nur das letzte Element der aufsteigenden Folge am Slot  $j$ .

■ Beispielfolge:

3 2 5 1 6 4

■ Problem: **Tabelle** enthält zum Schluss nicht die Folge, nur den letzten Wert.

■ Lösung: **Zweite Tabelle** mit den Vorgängern.

Index	1	2	3	4	5	6
Wert	3	2	5	1	6	4
Vorgänger	$-\infty$	$-\infty$	2	$-\infty$	5	1

	0	1	2	3	4	...
$-\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	
$-\infty$	3	$\infty$	$\infty$	$\infty$	$\infty$	
$-\infty$	2	$\infty$	$\infty$	$\infty$	$\infty$	
$-\infty$	2	5	$\infty$	$\infty$	$\infty$	
$-\infty$	1	5	$\infty$	$\infty$	$\infty$	
$-\infty$	1	5	6	$\infty$	$\infty$	
$-\infty$	1	4	6	$\infty$	$\infty$	

513

## Dynamic Programming Algorithmus LAT

Dimension der Tabelle? Bedeutung der Einträge?

1 Zwei Tabellen  $T[0, \dots, n]$  und  $V[1, \dots, n]$ . Zu Beginn  $T[0] \leftarrow -\infty$ ,  $T[i] \leftarrow \infty \forall i > 1$

Berechnung eines Eintrags

2 Einträge in  $T$  aufsteigend sortiert. Für jeden Neueintrag  $a_{k+1}$  binäre Suche nach  $l$ , so dass  $T[l] < a_k < T[l + 1]$ . Setze  $T[l + 1] \leftarrow a_{k+1}$ . Setze  $V[k] = T[l]$ .

514

## Dynamic Programming Algorithmus LAT

### Berechnungsreihenfolge

- 3 Beim Traversieren der Liste werden die Einträge  $T[k]$  und  $V[k]$  mit aufsteigendem  $k$  berechnet.

### Wie kann sich Lösung aus der Tabelle konstruieren lassen?

- 4 Suche das grösste  $l$  mit  $T[l] < \infty$ .  $l$  ist der letzte Index der LAT. Suche von  $l$  ausgehend den Index  $i < l$ , so dass  $V[l] = A[i]$ ,  $i$  ist der Vorgänger von  $l$ . Repetiere mit  $l \leftarrow i$  bis  $T[l] = -\infty$

515

## Analyse

### Berechnung Tabelle:

- Initialisierung:  $\Theta(n)$  Operationen
- Berechnung  $k$ -ter Eintrag: Binäre Suche auf Positionen  $\{1, \dots, k\}$  plus konstante Anzahl Zuweisungen.

$$\sum_{k=1}^n (\log k + \mathcal{O}(1)) = \mathcal{O}(n) + \sum_{k=1}^n \log(k) = \Theta(n \log n).$$

- Rekonstruktion: Traversiere  $A$  von rechts nach links:  $\mathcal{O}(n)$ .

Somit Gesamtlaufzeit

$$\Theta(n \log n).$$

516

## Längste Gemeinsame Teilfolge

Teilfolgen einer Zeichenkette:

Teilfolgen(KUH): ( $\epsilon$ ), (K), (U), (H), (KU), (KH), (UH), (KUH)

Problem:

- **Eingabe:** Zwei Zeichenketten  $A = (a_1, \dots, a_m)$ ,  $B = (b_1, \dots, b_n)$  der Längen  $m > 0$  und  $n > 0$ .
- **Gesucht:** Eine längste gemeinsame Teilfolge (LGT) von  $A$  und  $B$ .

Sinnvolle Anwendung: Ähnlichkeit von DNA-Sequenzen in der Biologie.

517

## Längste Gemeinsame Teilfolge

Beispiele:

$LGT(IGEL, KATZE) = E$ ,  $LGT(TIGER, ZIEGE) = IGE$

Ideen zur Lösung?

T I G E R  
Z I E G E

518

## Rekursives Vorgehen

Annahme: Lösungen  $L(i, j)$  bekannt für  $A[1, \dots, i]$  und  $B[1, \dots, j]$  für alle  $1 \leq i \leq m$  und  $1 \leq j \leq n$ , jedoch nicht für  $i = m$  und  $j = n$ .

T I G E R  
Z I E G E

Betrachten Zeichen  $a_m, b_n$ . Drei Möglichkeiten:

- 1  $A$  wird um ein Leerzeichen erweitert.  $L(m, n) = L(m, n - 1)$
- 2  $B$  wird um ein Leerzeichen erweitert.  $L(m, n) = L(m - 1, n)$
- 3  $L(m, n) = L(m - 1, n - 1) + \delta_{mn}$  mit  $\delta_{mn} = 1$  wenn  $a_m = b_n$  und  $\delta_{mn} = 0$  sonst

519

## Rekursion

$$L(m, n) \leftarrow \max \{L(m - 1, n - 1) + \delta_{mn}, L(m, n - 1), L(m - 1, n)\}$$

für  $m, n > 0$  und Randfälle  $L(\cdot, 0) = 0, L(0, \cdot) = 0$ .

	$\emptyset$	Z	I	E	G	E
$\emptyset$	0	0	0	0	0	0
T	0	0	0	0	0	0
I	0	0	1	1	1	1
G	0	0	1	1	2	2
E	0	0	1	2	2	3
R	0	0	1	2	2	3

520

## Dynamic Programming Algorithmus LGT

Dimension der Tabelle? Bedeutung der Einträge?

- 1 Tabelle  $L[0, \dots, m][0, \dots, n]$ .  $L[i, j]$ : Länge einer LGT der Zeichenketten  $(a_1, \dots, a_i)$  und  $(b_1, \dots, b_j)$

Berechnung eines Eintrags

- 2  $L[0, i] \leftarrow 0 \forall 0 \leq i \leq m, L[j, 0] \leftarrow 0 \forall 0 \leq j \leq n$ . Berechnung von  $L[i, j]$  sonst mit  $L[i, j] = \max(L[i - 1, j - 1] + \delta_{ij}, L[i, j - 1], L[i - 1, j])$ .

521

## Dynamic Programming Algorithmus LGT

Berechnungsreihenfolge

- 3 Abhängigkeiten berücksichtigen: z.B. Zeilen aufsteigend und innerhalb von Zeilen Spalten aufsteigend.

Wie kann sich Lösung aus der Tabelle konstruieren lassen?

- 4 Beginne bei  $j = m, i = n$ . Falls  $a_i = b_j$  gilt, gib  $a_i$  aus, sonst falls  $L[i, j] = L[i, j - 1]$  fahre mit  $j \leftarrow j - 1$  fort, sonst falls  $L[i, j] = L[i - 1, j]$  fahre mit  $i \leftarrow i - 1$  fort. Terminiere für  $i = 0$  oder  $j = 0$ .

522

## Analyse LGT

- Anzahl Tabelleneinträge:  $(m + 1) \cdot (n + 1)$ .
- Berechnung jeweils mit konstanter Anzahl Zuweisungen und Vergleichen. Anzahl Schritte  $\mathcal{O}(mn)$
- Bestimmen der Lösung: jeweils Verringerung von  $i$  oder  $j$ . Maximal  $\mathcal{O}(n + m)$  Schritte.

Laufzeit insgesamt:

$$\mathcal{O}(mn).$$

## Editierdistanz

Editierdistanz von zwei Zeichenketten  $A = (a_1, \dots, a_m)$ ,  $B = (b_1, \dots, b_m)$ .

**Editieroperationen:**

- Einfügen eines Zeichens
- Löschen eines Zeichens
- Änderung eines Zeichens

Frage: Wie viele Editieroperationen sind mindestens nötig, um eine gegebene Zeichenkette  $A$  in eine Zeichenkette  $B$  zu überführen.

TIGER ZIGER ZIEGER ZIEGE

Editierdistanz= Levenshtein Distanz

523

524

## Vorgehen?

- Zweidimensionale Tabelle  $E[0, \dots, m][0, \dots, n]$  mit Editierdistanzen  $E[i, j]$  zwischen Worten  $A_i = (a_1, \dots, a_i)$  und  $B_j = (b_1, \dots, b_j)$ .
- Betrachte die jeweils letzten Zeichen von  $A_i$  und  $B_j$ . Drei mögliche Fälle:
  - 1 Lösche letztes Zeichen von  $A_i$ :  $E[i - 1, j] + 1$ .
  - 2 Füge Zeichen zu  $A_i$  hinzu:<sup>26</sup>  $E[i, j - 1] + 1$ .
  - 3 Ersetze  $A_i$  durch  $B_j$ :  $E[i - 1, j - 1] + 1 - \delta_{ij}$ .

$$E[i, j] \leftarrow \min \{ E[i - 1, j] + 1, E[i, j - 1] + 1, E[i - 1, j - 1] + 1 - \delta_{ij} \}$$

<sup>25</sup>oder füge Zeichen zu  $B_j$  hinzu

<sup>26</sup>oder lösche letztes Zeichen von  $B_j$

## DP Tabelle

$$E[i, j] \leftarrow \min \{ E[i - 1, j] + 1, E[i, j - 1] + 1, E[i - 1, j - 1] + 1 - \delta_{ij} \}$$

	∅	Z	I	E	G	E
∅	0	1	2	3	4	5
T	1	1	2	3	4	5
I	2	2	1	2	3	4
G	3	3	2	2	2	3
E	4	4	3	2	3	2
R	5	5	4	3	3	3

Algorithmus: Übung!

525

526



## Matrix-Kettenmultiplikation

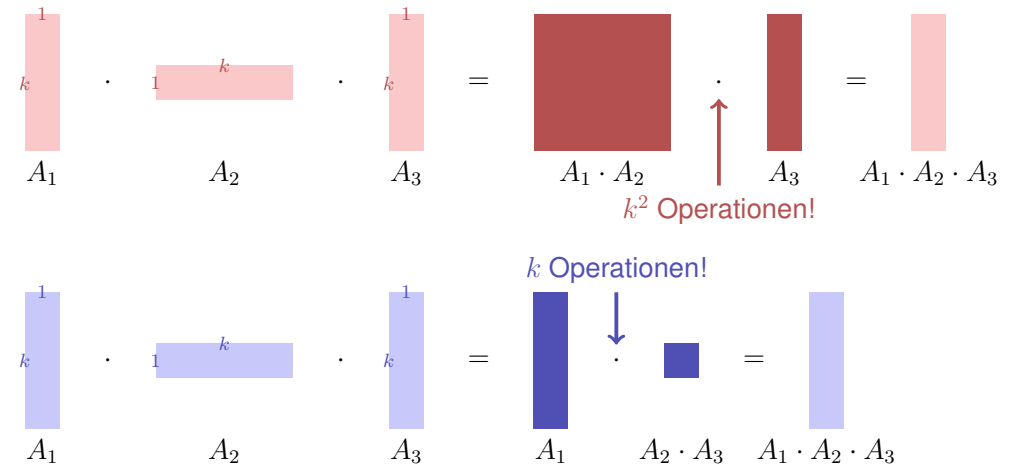
Aufgabe: Berechnung des Produktes  $A_1 \cdot A_2 \cdot \dots \cdot A_n$  von Matrizen  $A_1, \dots, A_n$ .

Matrizenmultiplikation ist assoziativ, d.h. Klammerung kann beliebig gewählt.

Ziel: möglichst effiziente Berechnung des Produktes.

Annahme: Multiplikation einer  $(r \times s)$ -Matrix mit einer  $(s \times u)$ -Matrix hat Kosten  $r \cdot s \cdot u$ .

## Macht das einen Unterschied?



527

528

## Rekursion

- Annahme, dass die bestmögliche Berechnung von  $(A_1 \cdot A_2 \cdot \dots \cdot A_i)$  und  $(A_{i+1} \cdot A_{i+2} \cdot \dots \cdot A_n)$  für jedes  $i$  bereits bekannt ist.
- Bestimme bestes  $i$ , fertig.

$n \times n$ -Tabelle  $M$ . Eintrag  $M[p, q]$  enthält Kosten der besten Klammerung von  $(A_p \cdot A_{p+1} \cdot \dots \cdot A_q)$ .

$$M[p, q] \leftarrow \min_{p \leq i < q} (M[p, i] + M[i + 1, q] + \text{Kosten letzte Multiplikation})$$

## Berechnung der DP-Tabelle

- Randfälle:  $M[p, p] \leftarrow 0$  für alle  $1 \leq p \leq n$ .
- Berechnung von  $M[p, q]$  hängt ab von  $M[i, j]$  mit  $p \leq i \leq j \leq q$ ,  $(i, j) \neq (p, q)$ .  
 Insbesondere hängt  $M[p, q]$  höchstens ab von Einträgen  $M[i, j]$  mit  $i - j < q - p$ .  
 Folgerung: Fülle die Tabelle von der Diagonale ausgehend.

529

530

## Analyse

DP-Tabelle hat  $n^2$  Einträge. Berechnung eines Eintrages bedingt Betrachten von bis zu  $n - 1$  anderen Einträgen.

Gesamtlaufzeit  $\mathcal{O}(n^3)$ .

Auslesen der Reihenfolge aus  $M$ : Übung!

## Exkurs: Matrixmultiplikation

Betrachten Multiplikation zweier  $n \times n$ -Matrizen.

Seien

$$A = (a_{ij})_{1 \leq i, j \leq n}, B = (b_{ij})_{1 \leq i, j \leq n}, C = (c_{ij})_{1 \leq i, j \leq n}, \\ C = A \cdot B$$

dann

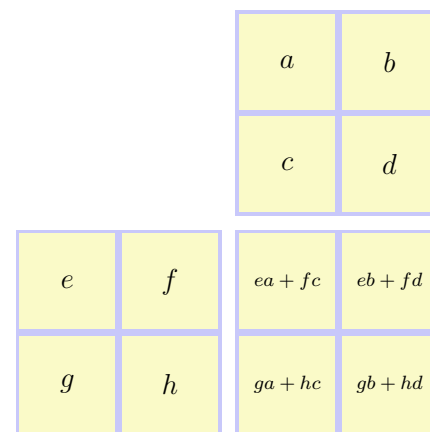
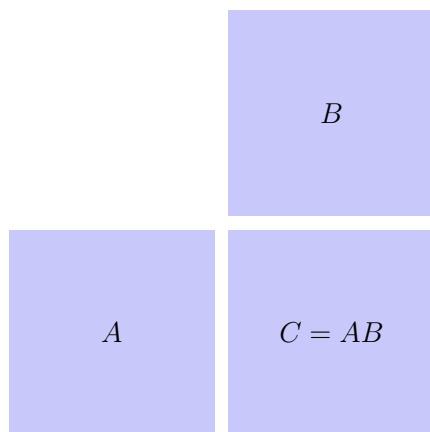
$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

Naiver Algorithmus benötigt  $\Theta(n^3)$  elementare Multiplikationen.

531

532

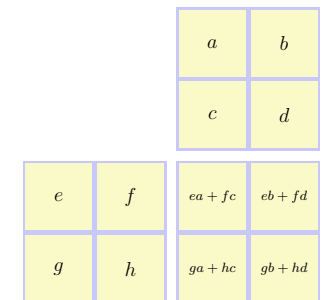
## Divide and Conquer



533

## Divide and Conquer

- Annahme  $n = 2^k$ .
- Anzahl elementare Multiplikationen:  
 $M(n) = 8M(n/2), M(1) = 1$ .
- Ergibt  $M(n) = 8^{\log_2 n} = n^{\log_2 8} = n^3$ . Kein Gewinn 😞



534

## Strassens Matrixmultiplikation

### ■ Nichttriviale Beobachtung von Strassen

(1969): Es genügt die Berechnung der sieben Produkte  $A = (e + h) \cdot (a + d)$ ,  $B = (g + h) \cdot a$ ,  $C = e \cdot (b - d)$ ,  $D = h \cdot (c - a)$ ,  $E = (e + f) \cdot d$ ,  $F = (g - e) \cdot (a + b)$ ,  $G = (f - h) \cdot (c + d)$ . Denn:  
 $ea + fc = A + D - E + G$ ,  $eb + fd = C + E$ ,  
 $ga + hc = B + D$ ,  $gb + hd = A - B + C + F$ .

### ■ Damit ergibt sich

$$M'(n) = 7M(n/2), M'(1) = 1.$$

$$\text{Also } M'(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}.$$

### ■ Schnellster bekannter Algorithmus:

$$\mathcal{O}(n^{2.37})$$

		a	b
		c	d
e	f	ea + fc	eb + fd
g	h	ga + hc	gb + hd