

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]

Quiz: Stacking Boxes

- Given: n boxes with sizes $w_i \times d_i \times h_i$
- Wanted: maximal height of a permitted stack
- Permitted stack: the base area of stacked boxes must become strictly smaller in both directions (width and depth)



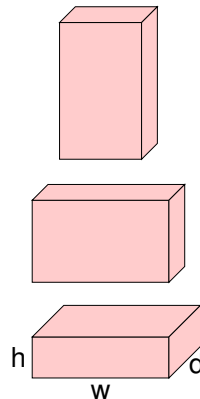
502

503

Boxen Stapeln

We assume that there are enough boxes of a kind such that each box is available in all orientations (right hand side of the figure below).

Solution: later



Box	1	2	3	4	5	6
$[w \times d \times h]$	$[1 \times 2 \times 3]$	$[1 \times 3 \times 2]$	$[2 \times 3 \times 1]$	$[3 \times 4 \times 5]$	$[3 \times 5 \times 4]$	$[4 \times 5 \times 3]$

Simpler: Fibonacci Numbers



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

Analysis: why is the recursive algorithm so slow?

504

505

Algorithm FibonacciRecursive(n)

Input : $n \geq 0$

Output : n -th Fibonacci number

if $n < 2$ **then**

 | $f \leftarrow n$

else

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

return f

Analysis

$T(n)$: Number executed operations.

■ $n = 0, 1: T(n) = \Theta(1)$

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

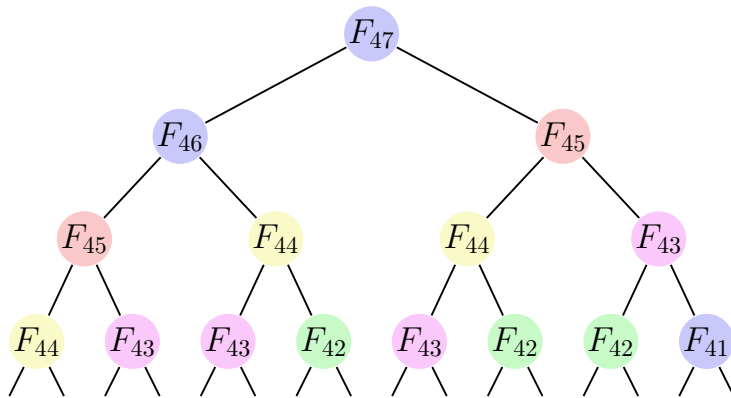
$$T(n) = T(n - 2) + T(n - 1) + c \geq 2T(n - 2) + c \geq 2^{n/2}c' = (\sqrt{2})^n c'$$

Algorithm is *exponential* in n .

506

507

Reason (visual)



Nodes with same values are evaluated (too) often.

Memoization

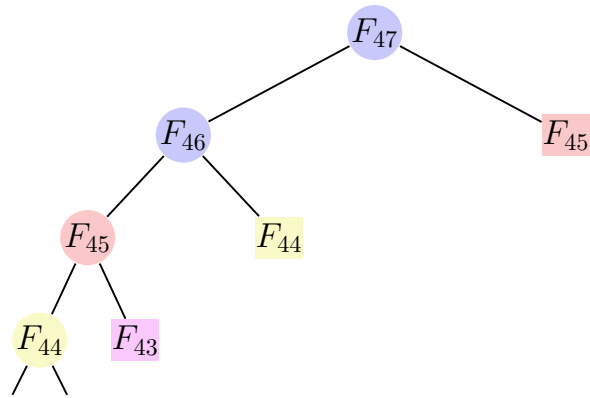
Memoization (sic) saving intermediate results.

- Before a subproblem is solved, the existence of the corresponding intermediate result is checked.
- If an intermediate result exists then it is used.
- Otherwise the algorithm is executed and the result is saved accordingly.

508

509

Memoization with Fibonacci



Rechteckige Knoten wurden bereits ausgewertet.

Algorithm FibonacciMemoization(n)

Input : $n \geq 0$

Output : n -th Fibonacci number

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

510

511

Analysis

Computational complexity:

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

Algorithm requires $\Theta(n)$ memory.²⁸

Looking closer ...

... the algorithm computes the values of F_1, F_2, F_3, \dots in the *top-down* approach of the recursion.

Can write the algorithm *bottom-up*. Then it is called *dynamic programming*.

²⁸But the naive recursive algorithm also requires $\Theta(n)$ memory implicitly.

512

513

Algorithm FibonacciDynamicProgram(n)

Input : $n \geq 0$

Output : n -th Fibonacci number

$F[1] \leftarrow 1$

$F[2] \leftarrow 1$

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

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

return $F[n]$

Dynamic Programming: Idea

- Divide a complex problem into a reasonable number of sub-problems
- The solution of the sub-problems will be used to solve the more complex problem
- Identical problems will be computed only once

514

515

Dynamic Programming Consequence

Identical problems will be computed only once

⇒ Results are saved



192-
HyperX Fury (2x, 8GB,
DDR4-2400, DIMM 288)
★★★★ 16

We trade spee against
memory consumption

Dynamic Programming = Divide-And-Conquer ?

- In both cases the original problem can be solved (more easily) by utilizing the solutions of sub-problems. The problem provides *optimal substructure*.
- Divide-And-Conquer algorithms (such as Mergesort): sub-problems are independent; their solutions are required only once in the algorithm.
- DP: sub-problems are dependent. The problem is said to have *overlapping sub-problems* that are required multiple-times in the algorithm. In order to avoid redundant computations, results have to be tabulated.

516

517

Dynamic Programming: Procedure

- 1 Use a *DP-table* with information to the subproblems.
Dimension of the entries? Semantics of the entries?
- 2 Computation of the *base cases*
Which entries do not depend on others?
- 3 Determine *computation order*.
In which order can the entries be computed such that dependencies are fulfilled?
- 4 Read-out the *solution*
How can the solution be read out from the table?

Runtime (typical) = number entries of the table times required operations per entry.

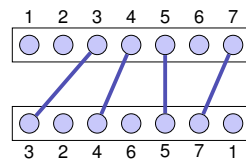
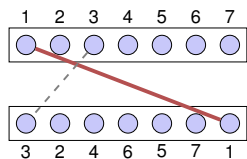
518

Dynamic Programming: Procedure with the example

- 1 Dimension of the table? Semantics of the entries?
 $n \times 1$ table. n th entry contains n th Fibonacci number.
- 2 Which entries do not depend on other entries?
Values F_1 and F_2 can be computed easily and independently.
- 3 What is the execution order such that required entries are always available?
 F_i with increasing i .
- 4 Wie kann sich Lösung aus der Tabelle konstruieren lassen?
 F_n ist die n -te Fibonacci-Zahl.

519

Longest Ascending Sequence (LAS)

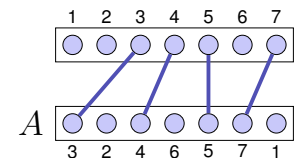


Connect as many as possible fitting ports without lines crossing.

520

Formally

- Consider Sequence $A = (a_1, \dots, a_n)$.
- Search for a longest increasing subsequence of A .
- Examples of increasing subsequences:
 $(3, 4, 5)$, $(2, 4, 5, 7)$, $(3, 4, 5, 7)$, $(3, 7)$.



Generalization: allow any numbers, even with duplicates. But only strictly increasing subsequences are permitted. Example:
 $(2, 3, 3, 3, 5, 1)$ with increasing subsequence $(2, 3, 5)$.

521

First idea

Assumption: LAS L_k known for k Now want to compute L_{k+1} for $k + 1$.

If a_{k+1} fits to L_k , then $L_{k+1} = L_k \oplus a_{k+1}$

Counterexample $A_5 = (1, 2, 5, 3, 4)$. Let $A_3 = (1, 2, 5)$ with $L_3 = A$. Determine L_4 from L_3 ?

It does not work this way, we cannot infer L_{k+1} from L_k .

Second idea.

Assumption: a LAS L_j is known for each $j \leq k$. Now compute LAS L_{k+1} for $k + 1$.

Look at all fitting $L_{k+1} = L_j \oplus a_{k+1}$ ($j \leq k$) and choose a longest sequence.

Counterexample: $A_5 = (1, 2, 5, 3, 4)$. Let $A_4 = (1, 2, 5, 3)$ with $L_1 = (1)$, $L_2 = (1, 2)$, $L_3 = (1, 2, 5)$, $L_4 = (1, 2, 5)$. Determine L_5 from L_1, \dots, L_4 ?

That does not work either: cannot infer L_{k+1} from only *an arbitrary solution* L_j . We need to consider all LAS. Too many.

522

523

Third approach

Assumption: the LAS L_j , *that ends with smallest element* is known for each of the lengths $1 \leq j \leq k$.

Consider all fitting $L_j \oplus a_{k+1}$ ($j \leq k$) and update the table of the LAS, that end with smallest possible element.

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

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)

DP Table

- Idea: save the last element of the increasing sequence L_j at slot j .
- Example: 3 2 5 1 6 4
- Problem: **Table** does not contain the subsequence, only the last value.
- Solution: **second table** with the predecessors.

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

Index	0	1	2	3	4	...
$(L_j)_j$	$-\infty$	1	4	6	∞	

524

525

Dynamic Programming Algorithm LAS

Table dimension? Semantics?

- Two tables $T[0, \dots, n]$ and $V[1, \dots, n]$. Start with $T[0] \leftarrow -\infty$, $T[i] \leftarrow \infty \forall i > 1$

Computation of an entry

- Entries in T sorted in ascending order. For each new entry a_{k+1} binary search for l , such that $T[l] < a_k < T[l+1]$. Set $T[l+1] \leftarrow a_{k+1}$. Set $V[k] = T[l]$.

Dynamic Programming algorithm LAS

Computation order

- 3 Traverse the list and compute $T[k]$ and $V[k]$ with ascending k

How can the solution be determined from the table?

- 4 Search the largest l with $T[l] < \infty$. l is the last index of the LAS. Starting at l search for the index $i < l$ such that $V[l] = A[i]$, i is the predecessor of l . Repeat with $l \leftarrow i$ until $T[l] = -\infty$

526

527

Analysis

Computation of the table:

- Initialization: $\Theta(n)$ Operations
- Computation of the k th entry: binary search on positions $\{1, \dots, k\}$ plus constant number of assignments.

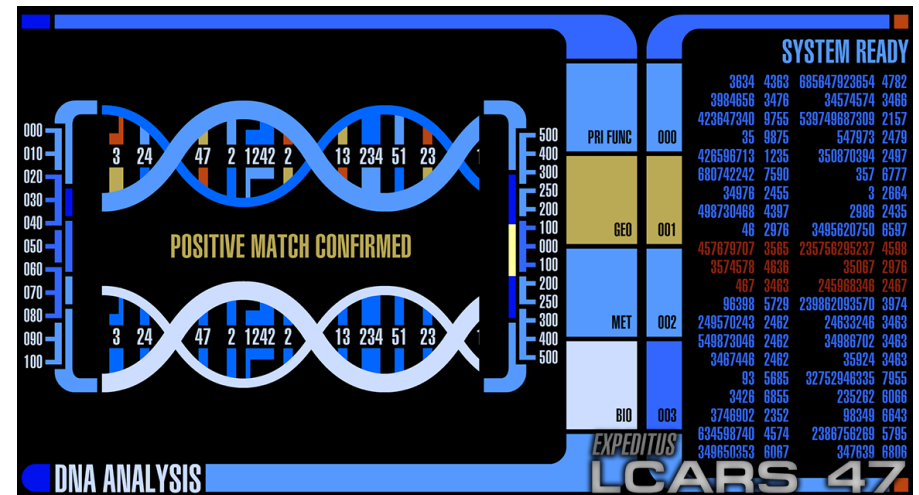
$$\sum_{k=1}^n (\log k + \mathcal{O}(1)) = \mathcal{O}(n) + \sum_{k=1}^n \log(k) = \Theta(n \log n).$$

- Reconstruction: traverse A from right to left: $\mathcal{O}(n)$.

Overall runtime:

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

DNA - Comparison (Star Trek)



528

529

DNA - Comparison

- DNA consists of sequences of four different nucleotides **A**denine **G**uanine **T**hymine **C**ytosine
- DNA sequences (genes) thus can be described with strings of A, G, T and C.
- Possible comparison of two genes: determine the **longest common subsequence**

Longest common subsequence

Subsequences of a string:

Subsequences(*KUH*): (), (*K*), (*U*), (*H*), (*KU*), (*KH*), (*UH*), (*KUH*)

Problem:

- **Input:** two strings $A = (a_1, \dots, a_m)$, $B = (b_1, \dots, b_n)$ with lengths $m > 0$ and $n > 0$.
- **Wanted:** Longest common subsequence (LCS) of A and B .

530

531

Longest Common Subsequence

Examples:

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

Ideas to solve?

T I G E R
Z I E G E

Recursive Procedure

Assumption: solutions $L(i, j)$ known for $A[1, \dots, i]$ and $B[1, \dots, j]$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$, but not for $i = m$ and $j = n$.

T I G E R
Z I E G E

Consider characters a_m, b_n . Three possibilities:

- 1 A is enlarged by one whitespace. $L(m, n) = L(m, n - 1)$
- 2 B is enlarged by one whitespace. $L(m, n) = L(m - 1, n)$
- 3 $L(m, n) = L(m - 1, n - 1) + \delta_{mn}$ with $\delta_{mn} = 1$ if $a_m = b_n$ and $\delta_{mn} = 0$ otherwise

532

533

Recursion

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

for $m, n > 0$ and base cases $L(\cdot, 0) = 0, L(0, \cdot) = 0$.

	\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

534

Dynamic Programming algorithm LCS

Dimension of the table? Semantics?

- 1 Table $L[0, \dots, m][0, \dots, n]$. $L[i, j]$: length of a LCS of the strings (a_1, \dots, a_i) and (b_1, \dots, b_j)

Computation of an entry

- 2 $L[0, i] \leftarrow 0 \forall 0 \leq i \leq m, L[j, 0] \leftarrow 0 \forall 0 \leq j \leq n$. Computation of $L[i, j]$ otherwise via $L[i, j] = \max(L[i-1, j-1] + \delta_{ij}, L[i, j-1], L[i-1, j])$.

535

Dynamic Programming algorithm LCS

3 Computation order

Rows increasing and within columns increasing (or the other way round).

4 Reconstruct solution?

- 4 Start with $j = m, i = n$. If $a_i = b_j$ then output a_i and continue with $(j, i) \leftarrow (j-1, i-1)$; otherwise, if $L[i, j] = L[i, j-1]$ continue with $j \leftarrow j-1$ otherwise, if $L[i, j] = L[i-1, j]$ continue with $i \leftarrow i-1$.
Terminate for $i = 0$ or $j = 0$.

536

Analysis LCS

- Number table entries: $(m+1) \cdot (n+1)$.
- Constant number of assignments and comparisons each. Number steps: $\mathcal{O}(mn)$
- Determination of solution: decrease i or j . Maximally $\mathcal{O}(n+m)$ steps.

Runtime overall:

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

537

Editing Distance

Editing distance of two sequences $A = (a_1, \dots, a_m)$,
 $B = (b_1, \dots, b_m)$.

Editing operations:

- Insertion of a character
- Deletion of a character
- Replacement of a character

Question: how many editing operations at least required in order to transform string A into string B .

TIGER ZIGER ZIEGER ZIEGE

Editing Distance = Levenshtein Distance

Procedure?

- Two dimensional table $E[0, \dots, m][0, \dots, n]$ with editing distances $E[i, j]$ of strings $A_i = (a_1, \dots, a_i)$ and $B_j = (b_1, \dots, b_j)$.
- Consider the last characters of A_i and B_j . Three possible cases:

- 1 Delete last character of A_i : $E[i-1, j] + 1$.
- 2 Append character to A_i : $E[i, j-1] + 1$.
- 3 Replace A_i by B_j : $E[i-1, j-1] + 1 - \delta_{ij}$.

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

²⁹or append character to B_j

³⁰or delete last character of B_j

DP Table

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

	∅	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

Algorithm: exercise

Matrix-Chain-Multiplication

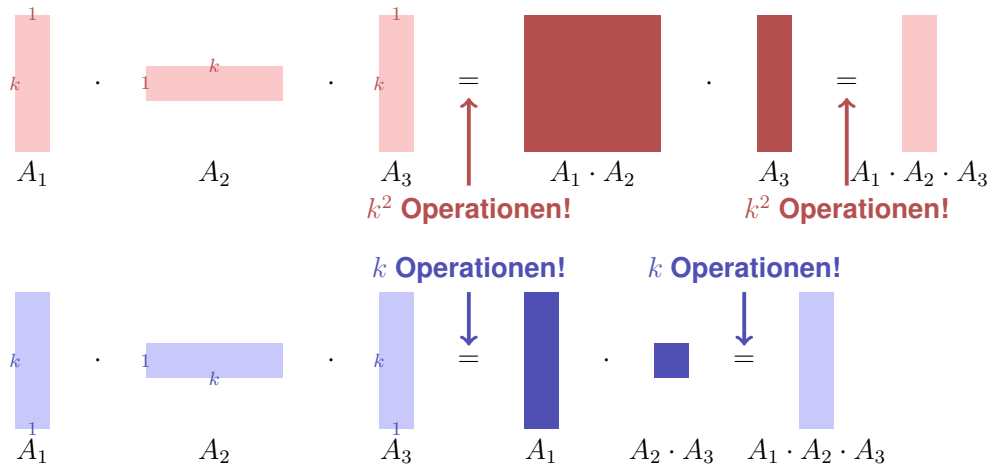
Task: Computation of the product $A_1 \cdot A_2 \cdot \dots \cdot A_n$ of matrices A_1, \dots, A_n .

Matrix multiplication is associative, i.e. the order of evaluation can be chosen arbitrarily

Goal: efficient computation of the product.

Assumption: multiplication of an $(r \times s)$ -matrix with an $(s \times u)$ -matrix provides costs $r \cdot s \cdot u$.

Does it matter?



542

Recursion

- Assume that the best possible computation of $(A_1 \cdot A_2 \cdots A_i)$ and $(A_{i+1} \cdot A_{i+2} \cdots A_n)$ is known for each i .
- Compute best i , done.

$n \times n$ -table M . entry $M[p, q]$ provides costs of the best possible bracketing $(A_p \cdot A_{p+1} \cdots A_q)$.

$$M[p, q] \leftarrow \min_{p \leq i < q} (M[p, i] + M[i + 1, q] + \text{costs of the last multiplication})$$

543

Computation of the DP-table

- Base cases $M[p, p] \leftarrow 0$ for all $1 \leq p \leq n$.
- Computation of $M[p, q]$ depends on $M[i, j]$ with $p \leq i \leq j \leq q$, $(i, j) \neq (p, q)$.
In particular $M[p, q]$ depends at most from entries $M[i, j]$ with $i - j < q - p$.
Consequence: fill the table from the diagonal.

544

Analysis

DP-table has n^2 entries. Computation of an entry requires considering up to $n - 1$ other entries.

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

Readout the order from M : exercise!

545

Digression: matrix multiplication

Consider the multiplication of two $n \times n$ matrices.

Let

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

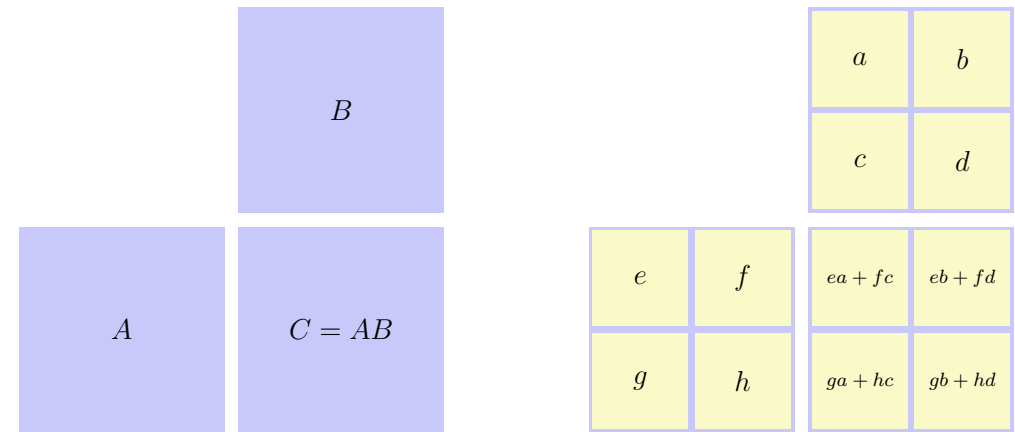
$$C = A \cdot B$$

then

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

Naive algorithm requires $\Theta(n^3)$ elementary multiplications.

Divide and Conquer

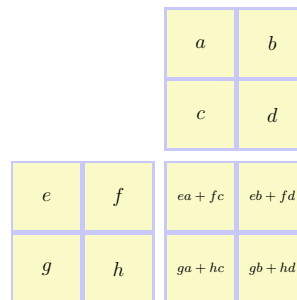


546

547

Divide and Conquer

- Assumption $n = 2^k$.
- Number of elementary multiplications:
 $M(n) = 8M(n/2), M(1) = 1$.
- yields $M(n) = 8^{\log_2 n} = n^{\log_2 8} = n^3$. No advantage 😞



548

Strassen's Matrix Multiplication

- **Nontrivial observation by Strassen (1969):**

It suffices to compute the seven products

$$A = (e + h) \cdot (a + d), B = (g + h) \cdot a,$$

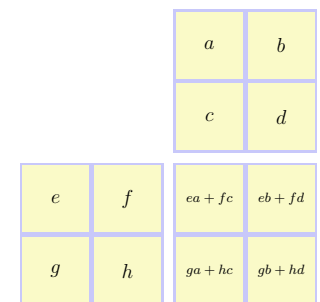
$$C = e \cdot (b - d), D = h \cdot (c - a), E = (e + f) \cdot d,$$

$$F = (g - e) \cdot (a + b), G = (f - h) \cdot (c + d). \text{ Denn:}$$

$$ea + fc = A + D - E + G, eb + fd = C + E,$$

$$ga + hc = B + D, gb + hd = A - B + C + F.$$

- This yields $M'(n) = 7M(n/2), M'(1) = 1$.
Thus $M'(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$.
- Fastest currently known algorithm:
 $\mathcal{O}(n^{2.37})$



549