19. Dynamische Programmierung I

Memoisieren, Optimale Substruktur, Überlappende Teilprobleme, Abhängigkeiten, Allgemeines Vorgehen. Beispiele: Fibonacci, Schneiden von Eisenstangen, 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)

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

Analyse: warum ist der rekursive Algorithmus so langsam.

Algorithmus FibonacciRecursive(n)

Input: $n \geq 0$
Output: n-te Fibonacci Zahl

if $n < 2$ **then**
  $f \leftarrow n$
else
  $f \leftarrow$ FibonacciRecursive$(n-1)$ + FibonacciRecursive$(n-2)$
return f

Analyse

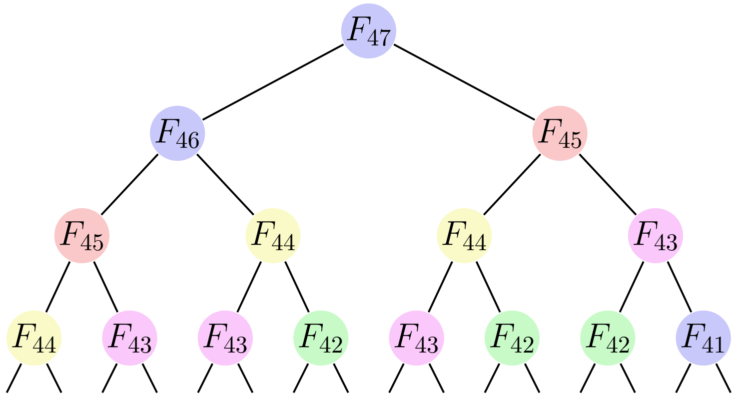
$T(n)$: Anzahl der ausgeführten Operationen.

- $n = 0, 1$: $T(n) = \Theta(1)$
- $n \geq 2$: $T(n) = T(n-2) + T(n-1) + c$.

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

Algorithmus ist *exponentiell (!)* in n.

Grund, visualisiert



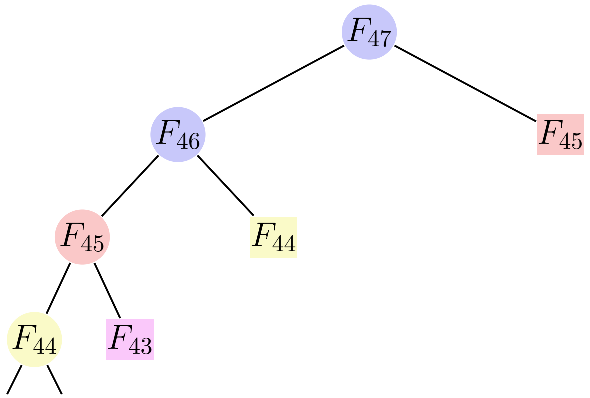
Knoten mit denselben Werten werden (zu) oft ausgewertet.

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.

Memoization bei Fibonacci



Rechteckige Knoten wurden bereits ausgewertet.

Algorithmus FibonacciMemoization(n)

Input: $n \geq 0$
Output: n-te Fibonacci Zahl

if $n \leq 2$ **then**
    $f \leftarrow 1$
else if $\exists$memo$[n]$ **then**
    $f \leftarrow$ memo$[n]$
else
    $f \leftarrow$ FibonacciMemoization$(n-1)$ + FibonacciMemoization$(n-2)$
    memo$[n] \leftarrow f$
return f

Analyse

Berechnungsaufwand:

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

denn nach dem Aufruf von $f(n-1)$ wurde $f(n-2)$ bereits berechnet.

Das lässt sich auch so sehen: Für jedes n wird $f(n)$ maximal einmal rekursiv berechnet. Laufzeitkosten: n Aufrufe mal $\Theta(1)$ Kosten pro Aufruf $n \cdot c \in \Theta(n)$. Die Rekursion verschwindet aus der Berechnung der Laufzeit.

Algorithmus benötigt $\Theta(n)$ Speicher.[39]

[39] 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, ... verkleidet im *Top-Down* Ansatz der Rekursion.

Man kann den Algorithmus auch gleich *Bottom-Up* hinschreiben. Das ist charakteristisch für die *dynamische Programmierung*.

Algorithmus FibonacciBottomUp(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 $F[n]$

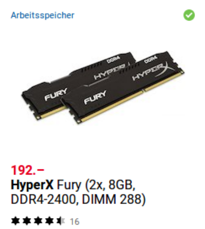
Dynamische Programmierung: Idee

- Aufteilen eines komplexen Problems in eine vernünftige Anzahl kleinerer Teilprobleme
- Die Lösung der Teilprobleme wird zur Lösung des komplexeren Problems verwendet
- Identische Teilprobleme werden nur einmal gerechnet

Dynamische Programmierung: Konsequenz

Identische Teilprobleme werden nur einmal gerechnet

$\Rightarrow$ Resultate werden zwischengespeichert



Wir tauschen Laufzeit gegen Speicherplatz

Dynamic Programming: Beschreibung

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* bestimen.
   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: Beschreibung 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.

Dynamic Programming = Divide-And-Conquer ?

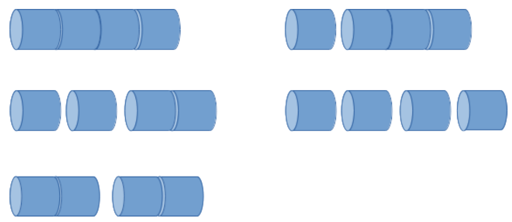
- In beiden Fällen ist das Ursprungsproblem (einfacher) lösbar, indem Lösungen von Teilproblemen herangezogen werden können. Das Problem hat *optimale Substruktur*.
- Bei Divide-And-Conquer Algorithmen (z.B. Mergesort) sind Teilprobleme unabhängig; deren Lösungen werden im Algorithmus nur einmal benötigt.
- Beim DP sind Teilprobleme nicht unabhängig. Das Problem hat *überlappende Teilprobleme*, welche im Algorithmus mehrfach gebraucht werden.
- Damit sie nur einmal gerechnet werden müssen, werden Resultate tabelliert. Dafür darf es *zwischen Teilproblemen keine zirkulären Abhängigkeiten* geben.

Schneiden von Eisenstäben

- Metallstäbe werden zerschnitten und verkauft.
- Metallstäbe der Länge $n \in \mathbb{N}$ verfügbar. Zerschneiden kostet nichts.
- Für jede Länge $l \in \mathbb{N}$, $l \leq n$ bekannt: Wert $v_l \in \mathbb{R}^+$
- Ziel: Zerschneide die Stange so (in $k \in \mathbb{N}$ Stücke), dass

$$\sum_{i=1}^{k} v_{l_i} \text{ maximal unter } \sum_{i=1}^{k} l_i = n.$$

Schneiden von Eisenstäben: Beispiel



Arten, einen Stab der Länge 4 zu zerschneiden (ohne Permutationen)

Länge	0	1	2	3	4
Preis	0	2	3	8	9

$\Rightarrow$ Bester Schnitt: 3 + 1 mit Wert 10.

Wie findet man den DP Algorithmus

0. Genaue Formulierung der gesuchten Lösung
1. Definiere Teilprobleme (und bestimme deren Anzahl)
2. Raten / Aufzählen (und bestimme die Laufzeit für das Raten)
3. Rekursion: verbinde die Teilprobleme
4. Memoisieren / Tabellieren. Bestimme die Abhängigkeiten der Teilprobleme
5. Lösung des Problems
   Laufzeit = #Teilprobleme $\times$ Zeit/Teilproblem

Struktur des Problems

0. *Gesucht:* r_n = maximal erreichbarer Wert von (ganzem oder geschnittenem) Stab mit Länge n.
1. *Teilprobleme:* maximal erreichbarer Wert r_k für alle $0 \leq k < n$
2. *Rate* Länge des ersten Stückes
3. *Rekursion*

$$r_k = \max\{v_i + r_{k-i} : 0 < i \leq k\}, \quad k > 0$$
$$r_0 = 0$$

4. *Abhängigkeit:* r_k hängt (nur) ab von den Werten v_i, $1 \leq i \leq k$ und den optimalen Schnitten r_i, $i < k$
5. *Lösung* in r_n

Algorithmus RodCut(v,n)

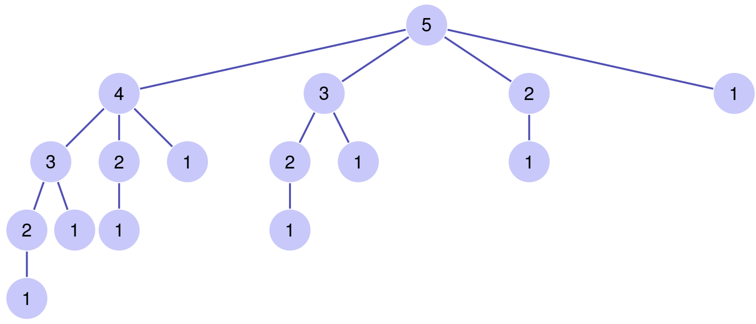
Input: $n \geq 0$, Preise v
Output: bester Wert

$q \leftarrow 0$
if $n > 0$ **then**
  for $i \leftarrow 1, \ldots, n$ **do**
    $q \leftarrow \max\{q, v_i + \text{RodCut}(v, n-i)\}$;
return q

Laufzeit $T(n) = \sum_{i=0}^{n-1} T(i) + c \quad \Rightarrow^{40} \quad T(n) \in \Theta(2^n)$

[40] $T(n) = T(n-1) + \sum_{i=0}^{n-2} T(i) + c = T(n-1) + (T(n-1) - c) + c = 2T(n-1) \quad (n > 0)$

Rekursionsbaum



Algorithmus RodCutMemoized(m, v, n)

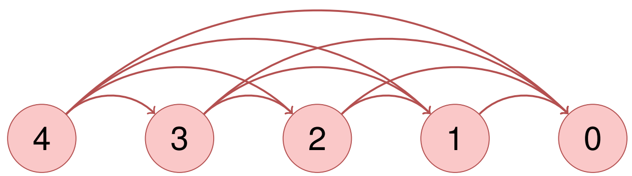
Input: $n \geq 0$, Preise v, Memoization Tabelle m
Output: bester Wert

$q \leftarrow 0$
if $n > 0$ **then**
  if $\exists\ m[n]$ **then**
    $q \leftarrow m[n]$
  else
    for $i \leftarrow 1, \ldots, n$ **do**
      $q \leftarrow \max\{q, v_i + \text{RodCutMemoized}(m, v, n-i)\}$;
    $m[n] \leftarrow q$
return q

Laufzeit $\sum_{i=1}^{n} i = \Theta(n^2)$

Teilproblem-Graph

beschreibt die Abhängigkeiten der Teilprobleme untereinander



und darf keine Zyklen enthalten

Konstruktion des optimalen Schnittes

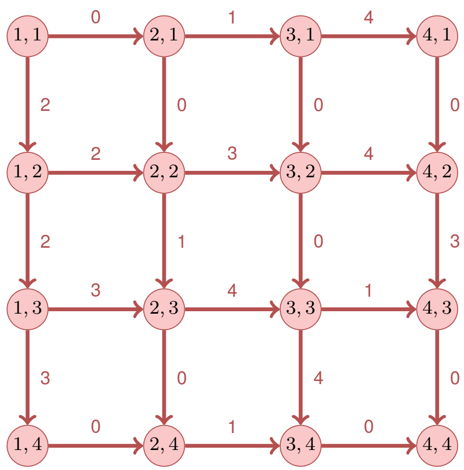
- Während der (rekursiven) Berechnung der optimalen Lösung für jedes $k \leq n$ bestimmt der rekursive Algorithmus die optimale Länge des ersten Stabes
- Speichere die Länge des ersten Stabes für jedes $k \leq n$ in einer Tabelle mit n Einträgen.

Bottom-Up Beschreibung am Beispiel

1. **Dimension der Tabelle? Bedeutung der Einträge?**
   Tabelle der Grösse $n \times 1$. n-ter Eintrag enthält besten Wert eines Stabes der Länge n.
2. **Welche Einträge hängen nicht von anderen ab?**
   Wert r_0 ist 0.
3. **In welcher Reihenfolge können Einträge berechnet werden, so dass benötigte Einträge jeweils vorhanden sind?**
   r_i, $i = 1, \ldots, n$.
4. **Wie kann sich Lösung aus der Tabelle konstruieren lassen?**
   r_n ist der beste Wert für eine Stange der Länge n

Kaninchen!

Ein Kaninchen sitzt auf Platz $(1, 1)$ eines $n \times n$ Gitters. Es kann nur nach Osten oder nach Süden gehen. Auf jedem Wegstück liegt eine Anzahl Rüben. Wie viele Rüben sammelt das Kaninchen maximal ein?

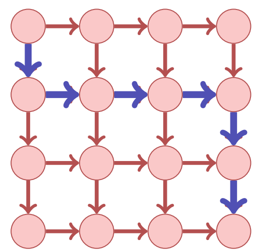


Kaninchen!

Anzahl mögliche Pfade?

- Auswahl von $n-1$ Wegen nach Süden aus $2n-2$ Wegen insgesamt.
- $$\binom{2n-2}{n-1} \in \Omega(2^n)$$

$\Rightarrow$ Naiver Algorithmus hat keine Chance



Der Weg 100011
(1:nach Süden, 0:nach Osten)

Rekursion

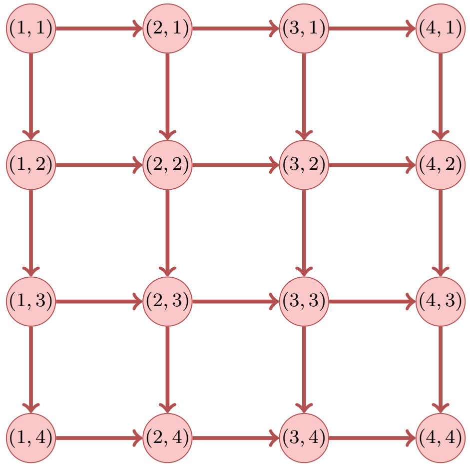
Gesucht: $T_{0,0}$ = *Maximale Anzahl Rüben von* $(0,0)$ *nach* (n,n).

Sei $w_{(i,j)-(i',j')}$ Anzahl Rüben auf Kante von (i,j) nach (i',j').

Rekursion (maximale Anzahl Rüben von (i,j) nach (n,n))

$$T_{ij} = \begin{cases} \max\{w_{(i,j)-(i,j+1)} + T_{i,j+1}, w_{(i,j)-(i+1,j)} + T_{i+1,j}\}, & i < n, j < n \\ w_{(i,j)-(i,j+1)} + T_{i,j+1}, & i = n, j < n \\ w_{(i,j)-(i+1,j)} + T_{i+1,j}, & i < n, j = n \\ 0 & i = j = n \end{cases}$$

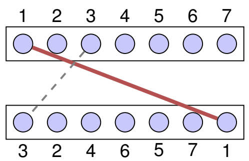
Teilproblemabhängigkeitsgraph

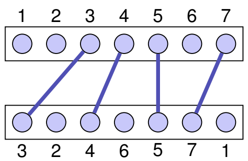


Bottom-Up Beschreibung am Beispiel

1. **Dimension der Tabelle? Bedeutung der Einträge?**
   Tabelle T der Grösse $n \times n$. Eintrag bei i,j enthält die maximale Anzahl Rüben von (i,j) nach (n,n).
2. **Welche Einträge hängen nicht von anderen ab?**
   Wert $T_{n,n}$ ist 0.
3. **In welcher Reihenfolge können Einträge berechnet werden, so dass benötigte Einträge jeweils vorhanden sind?**
   $T_{i,j}$ mit $i = n \searrow 1$ und für jedes i: $j = n \searrow 1$, (oder umgekehrt: $j = n \searrow 1$ und für jedes j: $i = n \searrow 1$).
4. **Wie kann sich Lösung aus der Tabelle konstruieren lassen?**
   $T_{1,1}$ enthält die maximale Anzahl Rüben

Längste aufsteigende Teilfolge (LAT)

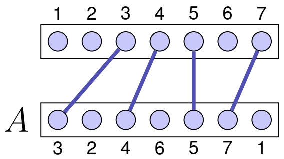




Verbinde so viele passende Anschlüsse wie möglich, ohne dass sich die Anschlüsse kreuzen.

Formalisieren

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



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

Erster Entwurf

Sei *L_i = längste Teilfolge von A_i, ($1 \leq i \leq n$).*

Annahme: LAT L_k von A_k für bekannt. Wollen nun LAT L_{k+1} für A_{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.

Zweiter Entwurf

Sei *L_i = längste Teilfolge von A_i, ($1 \leq i \leq n$).*

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 $L_1, \ldots, L_4$?

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!

Dritter Entwurf

Sei *$M_{n,i}$ = längste Teilfolge von A_n der Länge i ($1 \leq i \leq n$)*

Annahme: die LAT $M_{k,j}$ für A_k, *welche mit kleinstem Element enden* seien für alle Längen $1 \leq j \leq k$ bekannt.

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

Dritter Entwurf Beispiel

Beispiel: $A = (1, 1000, 1001, 4, 5, 2, 6, 7)$

A	LAT $M_{k,\cdot}$
1	(1)
$+1000$	$(1), (1, 1000)$
$+1001$	$(1), (1, 1000), (1, 1000, 1001)$
$+4$	$(1), (1, 4), (1, 1000, 1001)$
$+5$	$(1), (1, 4), (1, 4, 5)$
$+2$	$(1), (1, 2), (1, 4, 5)$
$+6$	$(1), (1, 2), (1, 4, 5), (1, 4, 5, 6)$
$+7$	$(1), (1, 2), (1, 4, 5), (1, 4, 5, 6), (1, 4, 5, 6, 7)$

DP Table

- Idee: speichere jeweils nur das letzte Element der aufsteigenden Folge $M_{k,j}$ 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

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

Dynamic Programming Algorithmus LAT

1 **Dimension der Tabelle? Bedeutung der Einträge?**

Zwei Tabellen $T[0,\dots,n]$ und $V[1,\dots,n]$.
$T[j]$: letztes Element der aufsteigenden Folge $M_{n,j}$
$V[j]$: Wert des Vorgängers von a_j.
Zu Beginn $T[0] \leftarrow -\infty$, $T[i] \leftarrow \infty\ \forall i > 1$

2 **Berechnung eines Eintrags**

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 $V[k] = T[l]$.

Dynamic Programming Algorithmus LAT

3 **Berechnungsreihenfolge**

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

4 **Wie kann sich Lösung aus der Tabelle konstruieren lassen?**

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 $T[l] = -\infty$

Analyse

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

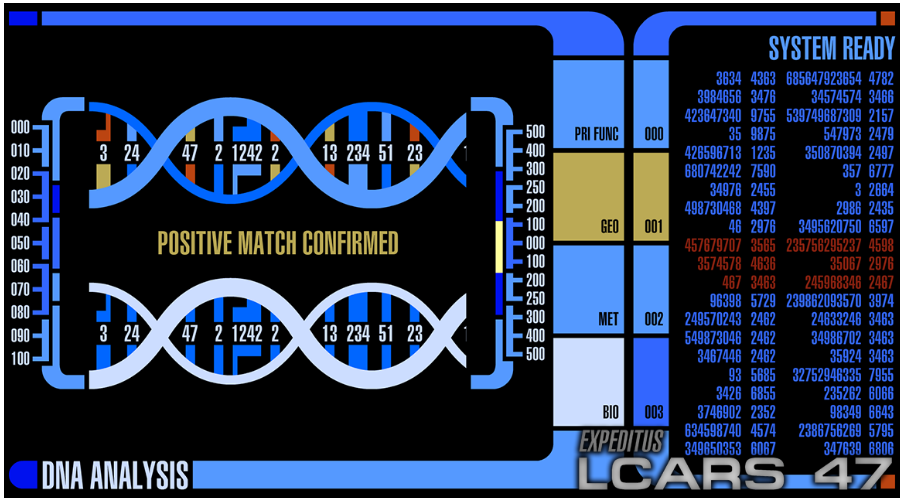
$$\sum_{k=1}^{n} (\log k + \mathcal{O}(1)) = \mathcal{O}(n) + \sum_{k=1}^{n} \log(k) = \Theta(n \log n).$$

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

Somit Gesamtlaufzeit

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

DNA - Vergleich (Star Trek)



DNA - Vergleich

- DNA besteht aus Sequenzen von vier verschiedenen Nukleotiden **A**denin **G**uanin **T**hymin **C**ytosin
- DNA-Sequenzen (Gene) werden mit Zeichenketten aus A, G, T und C beschrieben.
- Ein möglicher Vergleich zweier Gene: Bestimme **Längste gemeinsame Teilfolge**

Das Problem, die längste gemeinsame Teilfolge zu finden ist ein Spezialfall der minimalen Editierdistanz. Die folgenden Folien werden daher in der Vorlesung nicht behandelt.

[Längste Gemeiname Teilfolge]

Teilfolgen einer Zeichenkette:

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

Problem:

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

[Längste Gemeiname Teilfolge]

Beispiele:

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

Ideen zur Lösung?

T	I		G	E	R
Z	I	E	G	E	

[Rekursives Vorgehen]

Annahme: Lösungen $L(i,j)$ bekannt für $A[1,\dots,i]$ und $B[1,\dots,j]$ für alle $1 \le i \le m$ und $1 \le j \le n$, 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

[Rekursion]

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

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

	∅	Z	I	E	G	E
∅	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

[Dynamic Programming Algorithmus LGT]

1. **Dimension der Tabelle? Bedeutung der Einträge?**

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

2. **Berechnung eines Eintrags**

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

[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 und fahre fort mit $(j, i) \leftarrow (j-1, i-1)$; sonst, falls $L[i, j] = L[i, j-1]$ fahre fort mit $j \leftarrow j-1$; sonst, falls $L[i, j] = L[i-1, j]$ fahre fort mit $i \leftarrow i-1$. Terminiere für $i = 0$ oder $j = 0$.

[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:

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

Minimale Editierdistanz

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

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

Minimale Editierdistanz

Gesucht: Günstigste zeichenweise Transformation $A_n \rightarrow B_m$ mit Kosten

Operation	Levenshtein	LGT[41]	allgemein
c einfügen	1	1	ins(c)
c löschen	1	1	del(c)
Ersetzen $c \rightarrow c'$	$\mathbb{1}(c \neq c')$	$\infty \cdot \mathbb{1}(c \neq c')$	repl(c, c')

Beispiel

T	I	G	E	R		T	I	_	G	E	R		T→Z	+E	-R		
Z	I	E	G	E		Z	I	E	G	E	_		Z→T	-E	+R		

[41] Längste gemeinsame Teilfolge – Spezialfall des Editierproblems

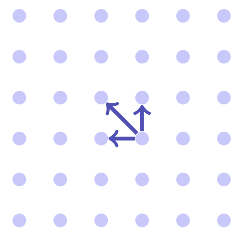
DP

- 0 $E(n,m)$ = minimale Anzahl Editieroperationen (ED Kosten) für $a_{1\dots n} \to b_{1\dots m}$
- 1 Teilprobleme $E(i,j)$ = ED von $a_{1\dots i}$. $b_{1\dots j}$. #TP $= n \cdot m$
- 2 Raten/Probieren Kosten$\Theta(1)$
  - $a_{1..i} \to a_{1\dots i-1}$ (löschen)
  - $a_{1..i} \to a_{1\dots i}b_j$ (einfügen)
  - $a_{1..i} \to a_{1\dots i_1}b_j$ (ersetzen)
- 3 Rekursion

$$E(i,j) = \min \begin{cases} \mathsf{del}(a_i) + E(i-1,j), \\ \mathsf{ins}(b_j) + E(i,j-1), \\ \mathsf{repl}(a_i,b_j) + E(i-1,j-1) \end{cases}$$

DP

- 4 Abhängigkeiten



$\Rightarrow$ Berechnung von links oben nach rechts unten. Zeilen- oder Spaltenweise.

- 5 Lösung steht in $E(n,m)$

Beispiel (Levenshteinabstand)

$$E[i,j] \leftarrow \min\left\{E[i-1,j]+1, E[i,j-1]+1, E[i-1,j-1]+\mathbb{1}(a_i \neq b_j)\right\}$$

	∅	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

Editierschritte: von rechts unten nach links oben, der Rekursion folgend. Bottom-Up Beschreibung des Algorithmus: Übung

Bottom-Up DP Algorithmus ED]

Dimension der Tabelle? Bedeutung der Einträge?

1 Tabelle $E[0,\dots,m][0,\dots,n]$. $E[i,j]$: Minimaler Editierabstand der Zeichenketten $(a_1,\dots,a_i)$ und $(b_1,\dots,b_j)$

Berechnung eines Eintrags

2 $E[0,i] \leftarrow i \;\forall 0 \le i \le m$, $E[j,0] \leftarrow i \;\forall 0 \le j \le n$. Berechnung von $E[i,j]$ sonst mit $E[i,j] =$
$\min\{\mathsf{del}(a_i) + E(i-1,j), \mathsf{ins}(b_j) + E(i,j-1), \mathsf{repl}(a_i,b_j) + E(i-1,j-1)\}$

Bottom-Up DP Algorithmus ED

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 $E[i, j] = \text{repl}(a_i, b_j) + E(i-1, j-1)$ gilt, gib $a_i \to b_j$ aus und fahre fort mit $(j, i) \leftarrow (j-1, i-1)$; sonst, falls $E[i, j] = \text{del}(a_i) + E(i-1, j)$ gib $\text{del}(a_i)$ aus fahre fort mit $j \leftarrow j-1$; sonst, falls $E[i, j] = \text{ins}(b_j) + E(i, j-1)$, gib $\text{ins}(b_j)$ aus und fahre fort mit $i \leftarrow i-1$. Terminiere für $i = 0$ und $j = 0$.

Matrix-Kettenmultiplikation

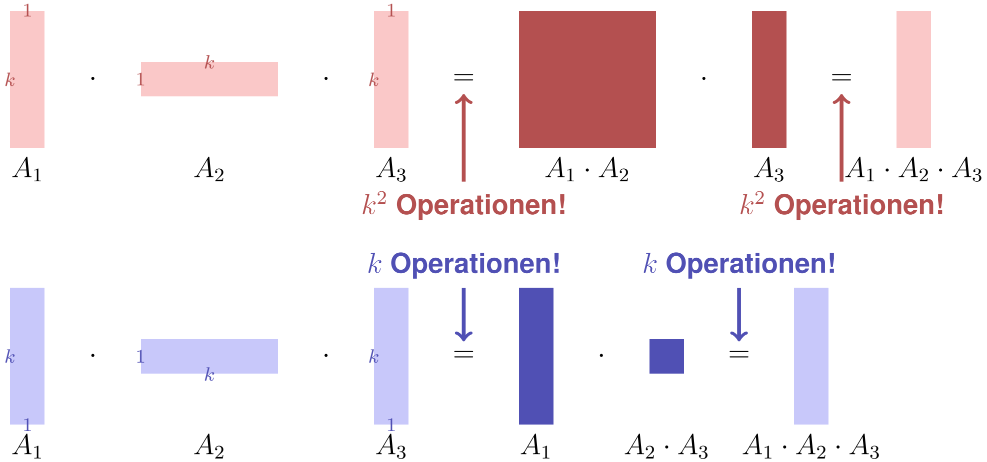
Aufgabe: Berechnung des Produktes $A_1 \cdot A_2 \cdot ... \cdot A_n$ von Matrizen $A_1, \dots, A_n$.

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

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?



Rekursion

- Annahme, dass die bestmögliche Berechnung von $(A_1 \cdot A_2 \cdots A_i)$ und $(A_{i+1} \cdot A_{i+2} \cdots A_n)$ 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 $(A_p \cdot A_{p+1} \cdots A_q)$.

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

Berechnung der DP-Tabelle

- Randfälle: $M[p,p] \leftarrow 0$ für alle $1 \le p \le n$.
- Berechnung von $M[p,q]$ hängt ab von $M[i,j]$ mit $p \le i \le j \le q$, $(i,j) \ne (p,q)$.
  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.

Analyse

DP-Tabelle hat n^2 Einträge. Berechung 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

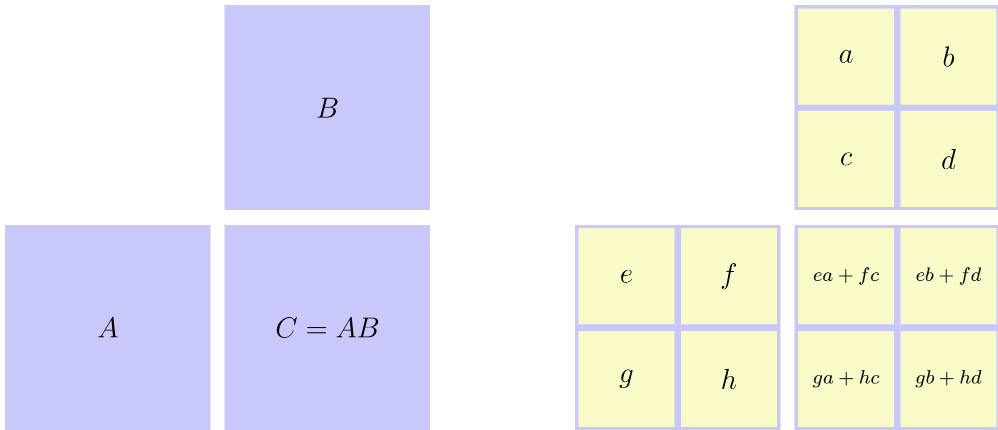
$$A = (a_{ij})_{1\le i,j\le n}, B = (b_{ij})_{1\le i,j\le n}, C = (c_{ij})_{1\le i,j\le n},$$
$$C = A \cdot B$$

dann

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

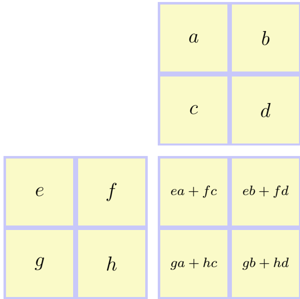
Naiver Algorithmus benötigt $\Theta(n^3)$ elementare Multiplikationen.

Divide and Conquer



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 ☹



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 $M'(n) = 7M(n/2)$, $M'(1) = 1$. Also $M'(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$.
- Schnellster bekannter Algorithmus: $\mathcal{O}(n^{2.37})$