DYNAMISCHE PROGRAMMIERUNG (DP)

Sortieren: Invarianten sind hilfreich beim Bearbeiten von Daten

DP: selbe Idee für Berechnen von Werten: Rekursion

beides verwandt zu Induktion

Vorbereitung: Memoization / Bottom-Up-Berechnung

Beispiel: Fibonacci-Zahlen $F_1 = F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$ für $n \geq 3$

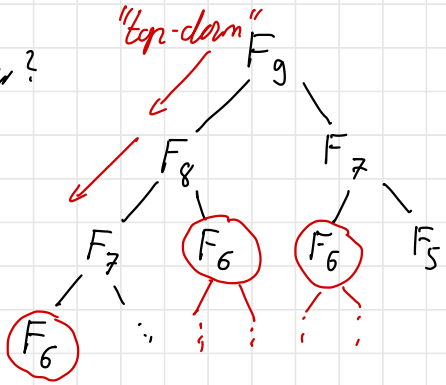
```
Fib(n)
    if n ≤ 2 : f ← 1
    else f ← Fib(n-1) + Fib(n-2)
    return f
```

Laufzeit: $T(n) = T(n-1) + T(n-2) + c$

$\geq 2 \cdot T(n-2) \geq 4 \cdot T(n-4) \geq \ldots \geq \Omega(2^{n/2}) = \Omega(\sqrt{2}^n)$

exponentiell: extrem teuer

Warum so teuer?

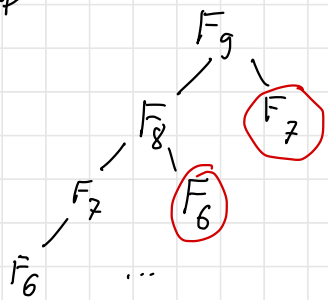


mehrfach berechnet

Lösung 1: merken! (Memoization)

```
FibM(n)
    if schon gespeichert : return memo[n]
    if n≤2 : f←1
    else f←FibM(n-1) + FibM(n-2)
    memo[n]←f
    return f
```

Laufzeit: $O(n)$



top-down, rekursiv

schon berechnet
→ kein weiteres Abstiegen

Lösung 2: bottom-up (iterativ)

```
Fib_bottom_up(n)
    F[1]←1
    F[2]←1
    for i = 3...n : F[i]←F[i-1]+F[i-2]
    return F[n]
```

Laufzeit: $O(n)$

Vor-/Nachteile von bottom-up-Programmen

+ oft einfacher Code
+ vermeidet Verwaltungsaufwand für den Computer:
  „call stack" für top-down-Programme
  Extremfall: sehr große Rekursionstiefe → „stack overflow"
+ Berechnungsreihenfolge ist explizit (kann auch negativ sein, too much info)
− gedanklich manchmal komplizierter als top-down-Rekursion

Kernidee von DP:

- Definiere Teilproblem ← kein allgemeingültiges Rezept, ähnlich wie Invariante → Üben!
- Löse Teilproblem rekursiv oder iterativ

Problem 1 (Erinnerung): Maximum Subarray Sum

a_1 a_2 ... a_n

Teilproblem: Randmax $R_j = \max_{i \le j} S_{ij}$ $\quad (S_{ij} = a_i + \dots + a_j)$

Rekursion: $R_j := \max\{a_j, a_j + R_{j-1}\}$

Auslesen der Lösung: $S^* = \max\{R_1, \dots, R_n, 0\}$

Problem 2: Jump Game

Input: Array $A[1 \dots n]$ von positiven ganzen Zahlen

Spiel: - Starte in 1.

- Von Position i dürfen wir höchstens $A[i]$ nach vorne springen (auf eine beliebige Position zwischen $i+1, \dots, i + A[i]$)

Gesucht: minimale Zahl an Sprüngen, um n zu erreichen

Beispiel:

1	3	5	3	2	1	2	4	4	2	9

5 Sprünge

4 Sprünge

Antwort: 4

Versuch 1:

Teilproblem: $S[i]$:= Mindestzahl an Sprüngen, um i zu erreichen

Rekursion: $S[i] = \min \{ 1 + S[j] \mid 1 \le j < i \text{ und } j + A[j] \ge i \}$

Kann von diesen letzten Ursprüngen erreicht werden

können i von j aus erreichen

z.B.:

	1	3	5	3	2	1	2	4	4	2	9
$S[i]$	0	1	2	2	2	3	3	3	4	4	4

Bottom-up code:

```
S[1] ← 0
for i = 2 ... n
    S[i] ← ∞        ← Zahl, die garantiert größer ist als das Ergebnis, hier gehen auch S[i] := n oder S[i] := i
    for j = 1 ... i-1
        if j + A[j] ≥ i : S[i] ← min { S[i], 1 + S[j] }
return S[n]
```

Laufzeit: $\Theta\left(\sum_{i=1}^{n} (i-1)\right) = \Theta(n^2)$

GEHT ES BESSER?

Versuch 2:

Teilproblem: $M[k]$:= maximaler Index, den wir in k Sprüngen erreichen

Rekursion: $M[k] = \max \{ i + A[i] \mid 1 \le i \le M[k-1] \}$

```
k ← 0, M[0] ← 1
while M[k] < n :
        k ← k + 1
        M[k] ← max { i + A[i] | 1 ≤ i ≤ M[k-1] }
return k
```

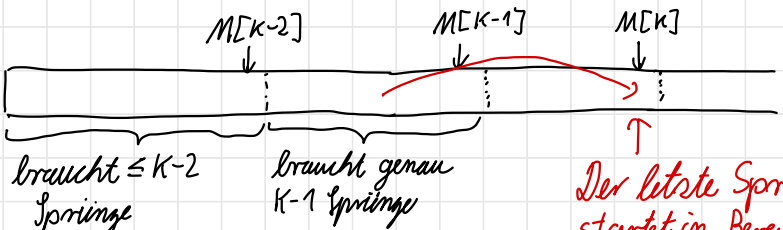
Wie immer: worst-case-Laufzeit gemeint

Laufzeit: Für $A = \boxed{1\,1\,1 \ldots 1}$ steigt $M[k]$ immer nur um 1

→ worst-case Laufzeit ist mindestens $\sum_{i=1}^{n-1} (i-1) = \Omega(n^2)$

Geht es besser?]A: selbes Teilproblem, verbesserte Rekursion:

$$M[K] = \max\{ i + A[i] \mid M[K-2] < i \leq M[K-1] \}$$



Der letzte Sprung startet im Bereich, wo man genau K-1 Sprünge braucht. Sonst käme man mit ≤ K-1 Sprüngen hierhin.

```
K ← 0, M[0] ← 1, M[-1] ← 0
while M[K] < n:
        K ← K+1
        M[K] ← max{ i + A[i] | M[K-2] < i ≤ M[K-1] }
return K
```

hat $M[K-1] - M[K-2]$ Elemente

Gesamtlaufzeit für Berechnung aller max: $O(n)$

↝ Laufzeit $O(n)$ (da restliche Berechnungen auch in $O(n)$)

Geht es besser?

NEIN: Für Input $A =$ [1 1 ... 1] müssen wir uns jeden Eintrag anschauen:

Angenommen, wir haben alle Felder außer dem i-ten angeschaut ($i \neq n-1, n$)

1 1 1 ... 1	?	1 1 1 ... 1
	i	

↝ Information reicht nicht, um sicher die richtige Antwort zu geben.

↝ worst-case-Laufzeit $\Omega(n)$

Problem 3: Längste gemeinsame Teilfolge LGT

T I G E R
Z I E G E

S E M E S T E R
V E R B E S S E R N

Teilfolge: - Reihenfolge muss stimmen
- Lücken ok

Alignment:

T	I	_	G	E	R
Z	I	E	G	E	_

Allgemein: Wie lang ist die LGT von $A[1...n]$ und $B[1..m]$?

1. Versuch: Teilproblem: LGT von $A[1..i]$ und $B[1...i]$

⇝ müssen LGT für $i+1$ aus LGT von i berechnen.

Beispiel: $LGT\left(\begin{matrix} T & I & G & E \\ Z & I & E & G \end{matrix}\right) = 2$ $\quad i=4$

Was ist $LGT\left(\begin{matrix} T & I & G & E & R \\ Z & I & E & G & E \end{matrix}\right)$? $\quad i=5$

Unklar: Zur Antwort müssten wir schon die „richtige" LGT für $i=4$ kennen.

→ Information reicht nicht

→ Müssen uns alle Endungen von LGT merken

2. Versuch: Teilproblem $L(i,j)$:= Länge LGT von $A[1..i]$, $B[1..j]$

Rekursion:

$i=0$ oder $j=0 \Rightarrow L(i,j)=0$

$i,j>0$: 1. Möglichkeit: Benutze sowohl $A[i]$ als auch $B[j]$
Geht nur, falls $A[i]=B[j]$

2. Möglichkeit: Benutze $A[i]$ nicht } überlappen sich, ist aber nicht schlimm
3. Möglichkeit: Benutze $B[j]$ nicht }

Also: Falls $A[i]=B[j]$, dann

genauere Überlegung: überflüssig

$$L(i,j) = \max\{1+L(i-1,j-1), L(i-1,j), L(i,j-1)\}$$

Sonst $L(i,j) = \max\{L(i-1,j), L(i,j-1)\}$

Berechnung bottom-up durch Füllen der DP-Tabelle:

$L(i,j)$	$i=$	0	1	2	3	4	5
		–	T	I	G	E	R
$j=$ 0	–	0	0	0	0	0	0
1	Z	0	0	0	0	0	0
2	I	0	0	1	1	1	1
3	E	0	0	1	1	2	2
4	G	0	0	1	2	2	2
5	E	0	0	1	2	3	3

Lösung

Reihenfolge: z.B. zeilenweise von oben nach unten

LGT finden wir durch Rückverfolgen:
„Wie kam dieser Wert zustande?"
Jeder Diagonalschritt gibt einen Buchstaben der LGT

Laufzeit $O(mn)$, Speicher $O(mn)$

kann verbessert werden, indem wir uns immer nur die letzte Zeile merken $\rightsquigarrow O(n)$
Alternativ: $O(m)$
$\rightsquigarrow$ Speicher: $O(\min\{n,m\})$

Bemerkungen (optional):

Geht es (viel) besser?

- Falls einer der Strings sehr kurz, $n \leq m$: JA, in Zeit $O(n+m^2)$.
  (Annahme: fixes Alphabet)

- Für $n = m$ (oder $n = \Theta(m)$): jahrzehntelang offen
- Wurde 2015 von Bringmann (damals ETH) und Künnemann gelöst.
- Antwort: NEIN, es gibt für kein $\varepsilon > 0$ einen $O(n^{2-\varepsilon})$-Algorithmus
  (unter gewissen Annahmen)
- Kleine Verbesserung sind möglich: Es gibt einen $O(n^2/\log n)$-Algorithmus

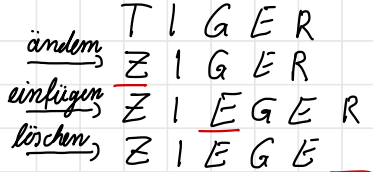
Problem 4: Editierdistanz

Gegeben zwei Zeichenfolgen $A[1..n]$, $B[1..m]$

Wollen A in B umwandeln mit Editieroperationen:

- Zeichen einfügen
- Zeichen löschen
- Zeichen ändern

z.B.:



Gesucht: minimale Anzahl Operationen

Das ist nicht dasselbe Problem wie LGT, z.B.
LGT: LCTIGER, ZIEGE) = 3 = LCTIGER, BIBER)
Editierdistanz: EDCTIGER, ZIEGE) = 3 ≠ 2 = EDCTIGER, BIBER)

Teilproblem: $ED(i,j)$:= Editierdistanz von $A[1..i]$ zu $B[1..j]$

Rekursion: Was passiert am Ende mit $A[i]$ in optimaler Lösung?

Fall 1: Wird irgendwann gelöscht. → kann man auch gleich machen.

Dann ist $ED(i,j) = \underbrace{1+}_{\text{lösche } A[i]} ED(i-1,j)$

Fall 2: Wird am Ende auf etwas in $B[1..j-1]$ gematcht.

⇒ $B[j]$ wird irgendwann eingefügt → gleich machen

Dann $ED(i,j) = \underbrace{1+}_{\text{füge } B[j] \text{ ein}} ED(i,j-1)$

Fall 3: Wird am Ende auf $B[j]$ gematcht.

Dann $ED(i,j) = \begin{cases} ED(i-1,j-1) & , \text{falls } A[i] = B[j] \\ \underbrace{1+}_{\text{ändere } A[i] \text{ in } B[j]} ED(i-1,j-1) & , \text{falls } A[i] \neq B[j] \end{cases}$

Also:

$$i,j>0: ED(i,j) = \min\{ ED(i-1,j)+1,\ ED(i,j-1)+1,\ ED(i-1,j-1)+\begin{cases}0 & \text{falls } A[i]=B[j]\\ 1 & \text{falls } A[i]\neq B[j]\end{cases} \}$$

$A[i]$ löschen

$B[j]$ einfügen

$A[i]$ durch $B[j]$ ersetzen

$$ED(i,0) = i$$
$$ED(0,j) = j$$

Implementierung: Fülle wieder DP-Tabelle aus

ED(i,j)	i=0	1	2	3	4	5
	–	T	I	G	E	R
j=0 –	0	1	2	3	4	5
1 Z	1	1	2	3	4	5
2 I	2	2	1	2	3	4
3 E	3	3	2	2	2	3
4 G	4	4	3	2	3	3
5 E	5	5	4	3	2	3

Lösung

Reihenfolge: z.B. zeilenweise (von oben nach unten)

Operationen: wieder durch Rückverfolgen

T I G E R → Z I E G E: ① lösche R

T I G E → Z I E G E: ok, ok

T I G → Z I E G: ok, ok

T I → Z I E

② füge E ein

T I E → Z I E: ok, ok

T I → Z I: ok

T → Z: ③ ändere T→Z

Z

Laufzeit: $O(m \cdot n)$

Speicher: $O(m \cdot n)$, geht auch in $O(\min\{m,n\})$

Geht es besser? Bemerkungen zu LGT übertragen sich auf ED.