



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

# Dynamische Programmierung

## *Fibonacci-Zahlen*

- $F(1) = 1$
- $F(2) = 1$
- $F(n) = F(n - 1) + F(n - 2)$

## Dynamische Programmierung

Fib( $n$ )

1. **if**  $n = 1$  **return** 1
2. **if**  $n = 2$  **return** 1
3. **return** Fib( $n - 1$ ) + Fib( $n - 2$ )

## Dynamische Programmierung

### *Lemma 21*

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### *Beweis*

- Wir zeigen, dass die Laufzeit  $T(n)$  von  $\text{Fib}(n)$  größer als  $\frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n$  ist.

## Dynamische Programmierung

### *Lemma 21*

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### *Beweis*

- Wir zeigen, dass die Laufzeit  $T(n)$  von  $\text{Fib}(n)$  größer als  $\frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n$  ist.
- Damit folgt das Lemma wegen  $\left(\frac{1+\sqrt{5}}{2}\right) \geq 1,6$ .

## Dynamische Programmierung

### Lemma 21

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### Beweis

- Wir zeigen, dass die Laufzeit  $T(n)$  von  $\text{Fib}(n)$  größer als  $\frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n$  ist.
- Damit folgt das Lemma wegen  $\left(\frac{1+\sqrt{5}}{2}\right) \geq 1,6$ .
- Beweis per Induktion über  $n$ .
- (I.A.) Für  $n = 1$  ist  $T(1) \geq 1 \geq \frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)$

$$\text{Für } n = 2 \text{ ist } T(2) \geq 1 \geq \frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^2$$

## Dynamische Programmierung

### *Lemma 21*

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### *Beweis*

- (I.V.) Für  $m < n$  ist  $T(m) \geq \frac{1}{3} \left( \frac{1+\sqrt{5}}{2} \right)^m$ .

## Dynamische Programmierung

### Lemma 21

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### Beweis

- (I.V.) Für  $m < n$  ist  $T(m) \geq \frac{1}{3} \left( \frac{1+\sqrt{5}}{2} \right)^m$ .
- (I.S.) Wir haben  $T(n) \geq T(n-1) + T(n-2)$  da der Algorithmus für  $n > 2$   $\text{Fib}(n-1)$  und  $\text{Fib}(n-2)$  rekursiv aufruft. Nach (I.V.) gilt somit

$$T(n) \geq T(n-1) + T(n-2) \geq \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^{n-1} + \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^{n-2}$$



## Dynamische Programmierung

### Lemma 21

Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1,6^n)$ .

### Beweis

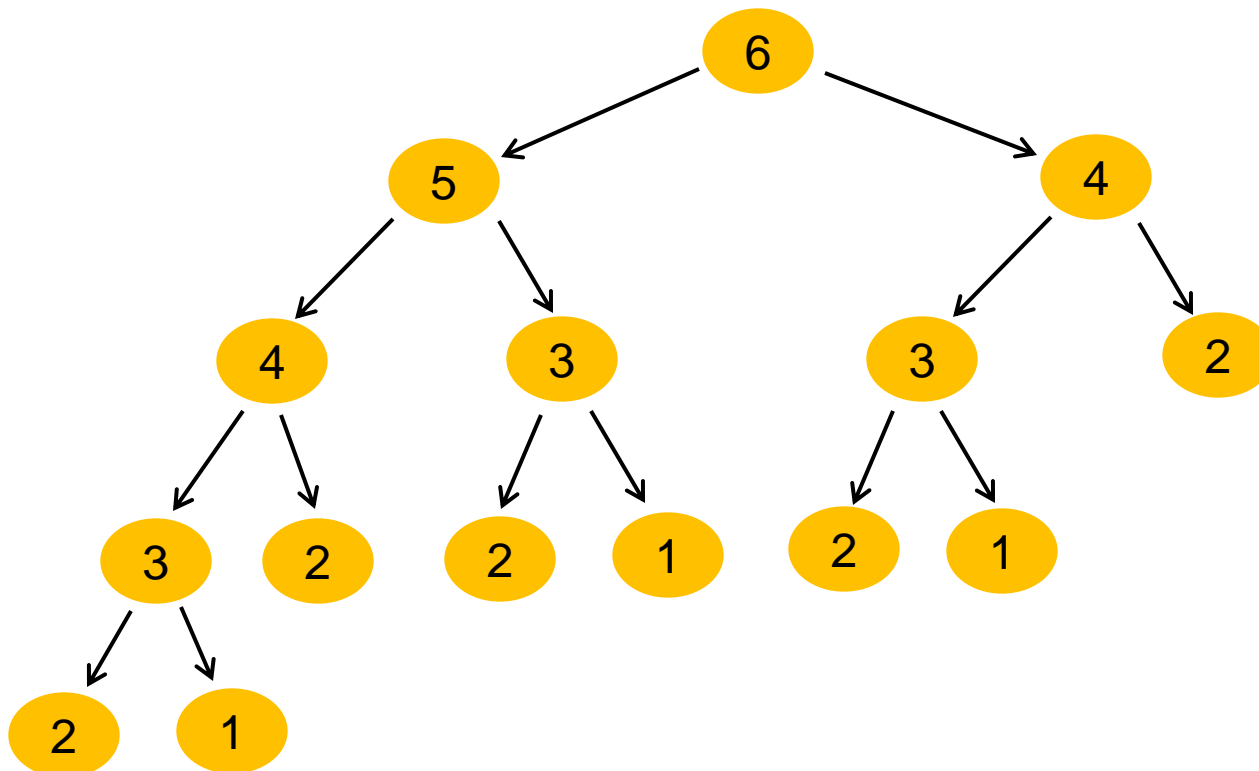
- (I.V.) Für  $m < n$  ist  $T(m) \geq \frac{1}{3} \left( \frac{1+\sqrt{5}}{2} \right)^m$ .
- (I.S.) Wir haben  $T(n) \geq T(n-1) + T(n-2)$  da der Algorithmus für  $n > 2$   $\text{Fib}(n-1)$  und  $\text{Fib}(n-2)$  rekursiv aufruft. Nach (I.V.) gilt somit

$$\begin{aligned} T(n) &\geq T(n-1) + T(n-2) \geq \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^{n-1} + \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^{n-2} \\ &= \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^{n-2} \cdot \left( 1 + \frac{1+\sqrt{5}}{2} \right) = \frac{1}{3} \cdot \left( \frac{1+\sqrt{5}}{2} \right)^n \end{aligned}$$

## Dynamische Programmierung

### *Warum ist die Laufzeit so schlecht?*

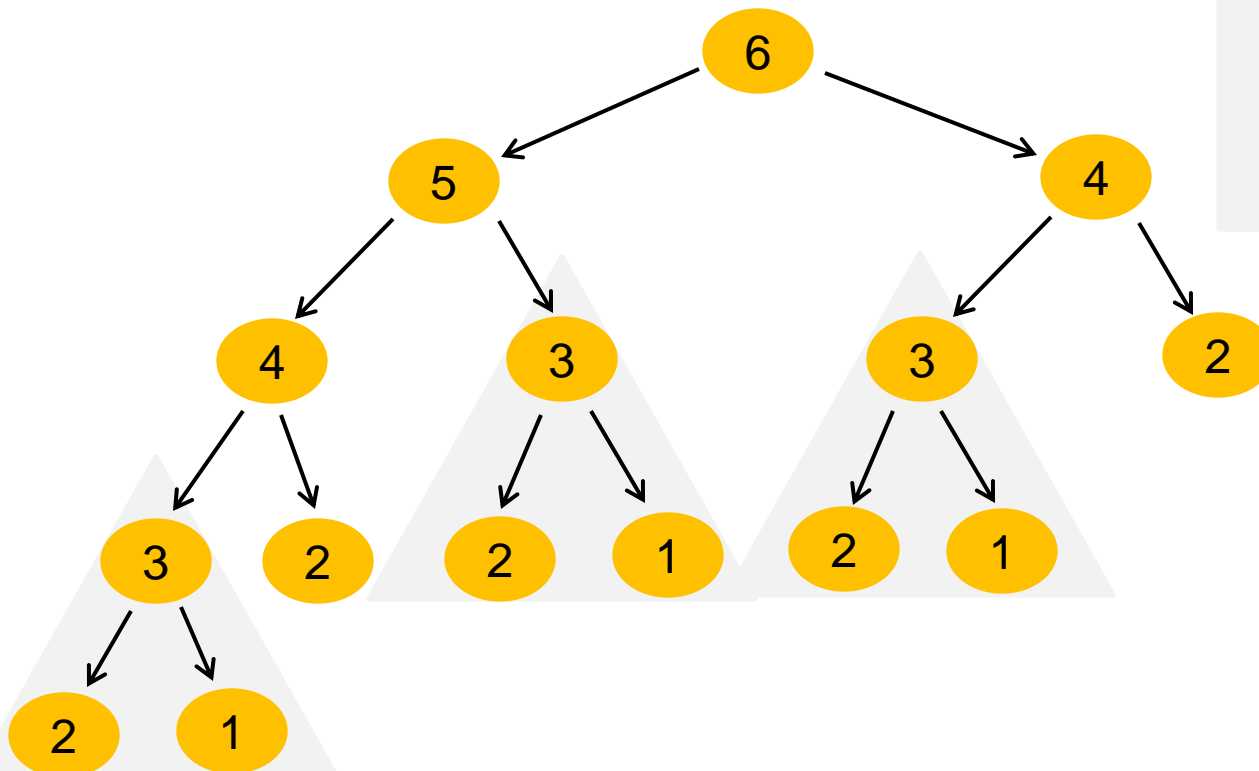
- Betrachten wir den Rekursionbaum für Aufruf `Fib(6)`



# Dynamische Programmierung

## Warum ist die Laufzeit so schlecht?

- Betrachten wir den Rekursionbaum für Aufruf  $\text{Fib}(6)$

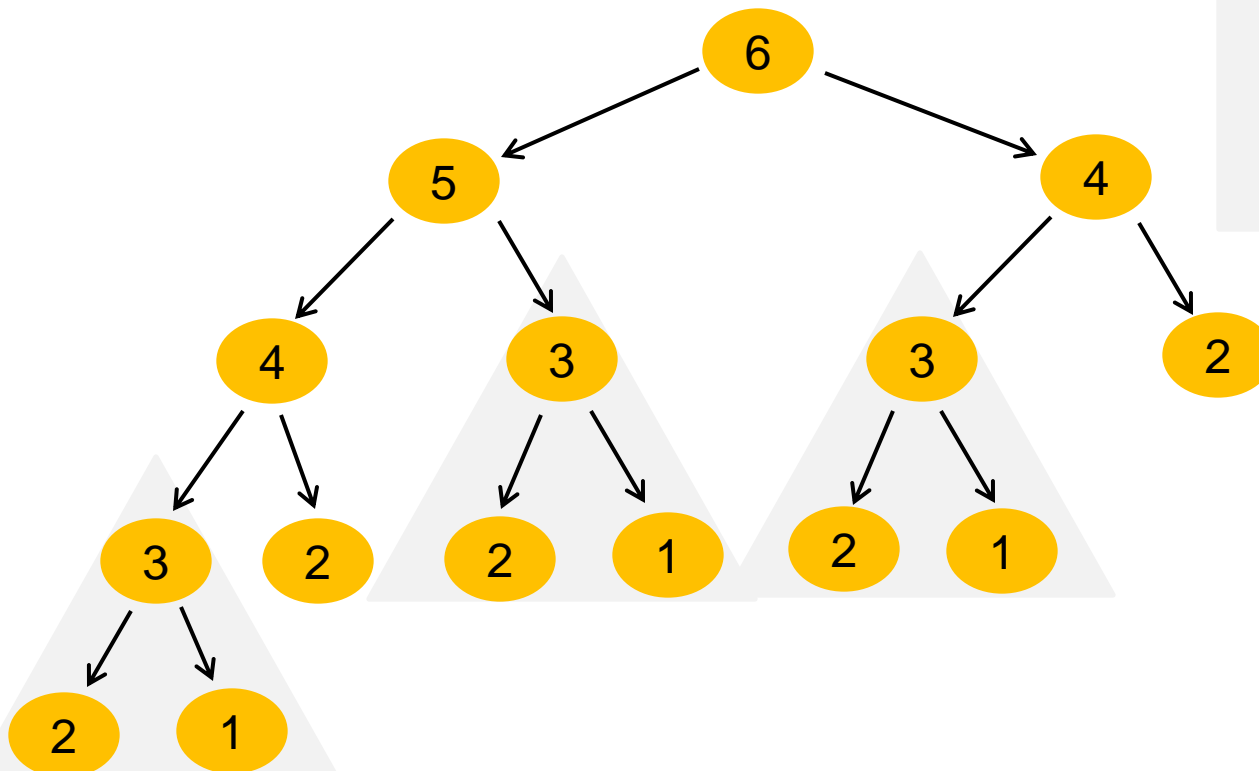


Bei der Berechnung von  $\text{Fib}(6)$  wird  $\text{Fib}(3)$  dreimal aufgerufen!

# Dynamische Programmierung

## Warum ist die Laufzeit so schlecht?

- Betrachten wir den Rekursionbaum für Aufruf  $\text{Fib}(6)$

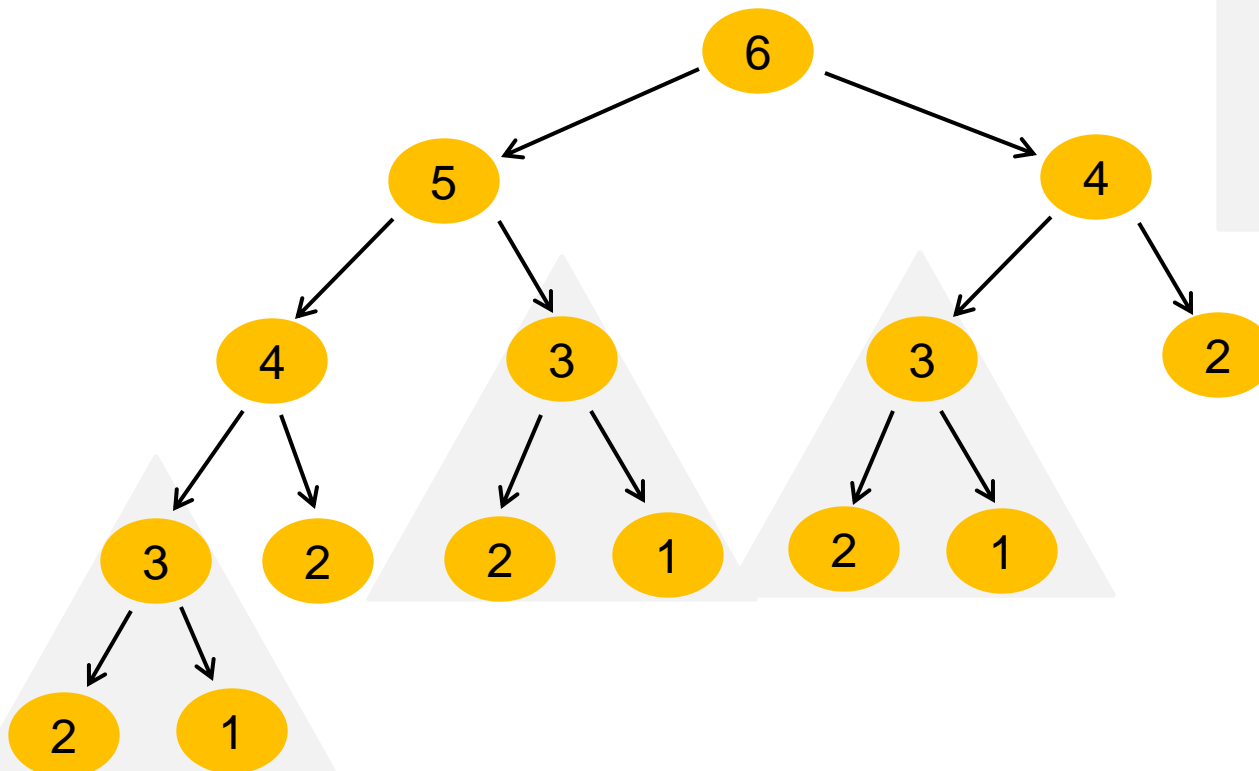


Es wird also mehrmals  
dieselbe Rechnung  
durchgeführt!

## Dynamische Programmierung

### *Warum ist die Laufzeit so schlecht?*

- Betrachten wir den Rekursionbaum für Aufruf `Fib(6)`



Bei großem  $n$  passiert dies  
sehr häufig!

# Dynamische Programmierung

## *Memoisation*

Memoisation bezeichnet die Zwischenspeicherung der Ergebnisse von Funktionsaufrufen eines rekursiven Algorithmus.

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$
2. **for**  $i \leftarrow 1$  **to**  $n$  **do**
3.      $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$
5.  $F[2] \leftarrow 1$
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do**
3.      $F[i] \leftarrow 0$
4.      $F[1] \leftarrow 1$
5.      $F[2] \leftarrow 1$
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.      $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$



## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.  $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$
5.  $F[2] \leftarrow 1$
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.  $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$  ➤ Setze  $F[1]$  auf korrekten Wert
5.  $F[2] \leftarrow 1$  ➤ Setze  $F[2]$  auf korrekten Wert
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.      $F[i] \leftarrow 0$
4.      $F[1] \leftarrow 1$  ➤ Setze  $F[1]$  auf korrekten Wert
5.      $F[2] \leftarrow 1$  ➤ Setze  $F[2]$  auf korrekten Wert
6.     **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.      $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3.     **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.      $F[i] \leftarrow 0$
4.      $F[1] \leftarrow 1$  ➤ Setze  $F[1]$  auf korrekten Wert
5.      $F[2] \leftarrow 1$  ➤ Setze  $F[2]$  auf korrekten Wert
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$  ➤ Gib Wert zurück, falls  
➤ schon berechnet
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.      $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$  ➤ Setze  $F[1]$  auf korrekten Wert
5.  $F[2] \leftarrow 1$  ➤ Setze  $F[2]$  auf korrekten Wert
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$  ➤ Gib Wert zurück, falls  
➤ schon berechnet
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$  ➤ Ansonsten berechne Wert  
➤ und speichere Wert ab
3. **return**  $F[n]$

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$  ➤ Initialisiere Feld für Funktionswerte
2. **for**  $i \leftarrow 1$  **to**  $n$  **do** ➤ Setze Feldeinträge auf 0
3.      $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$  ➤ Setze  $F[1]$  auf korrekten Wert
5.  $F[2] \leftarrow 1$  ➤ Setze  $F[2]$  auf korrekten Wert
6. **return** FibMemo( $n, F$ )

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$  ➤ Gib Wert zurück, falls  
➤ schon berechnet
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$  ➤ Ansonsten berechne Wert  
➤ und speichere Wert ab
3. **return**  $F[n]$  ➤ Gib Wert zurück

## Dynamische Programmierung

Fib2( $n$ )

1. Initialisiere Feld  $F[1..n]$
2. **for**  $i \leftarrow 1$  **to**  $n$  **do**
3.      $F[i] \leftarrow 0$
4.  $F[1] \leftarrow 1$
5.  $F[2] \leftarrow 1$
6. **return** FibMemo( $n, F$ )

Laufzeit:

- $\mathbf{O}(n)$
- $\mathbf{O}(n)$
- $\mathbf{O}(n)$
- $\mathbf{O}(1)$
- $\mathbf{O}(1)$
- $1 + T(n)$

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

## Dynamische Programmierung

### *Beispiel*

FibMemo(6,  $F$ )

6

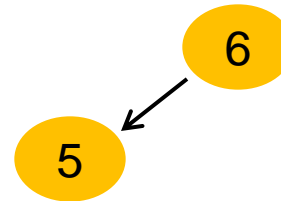
$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0



## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

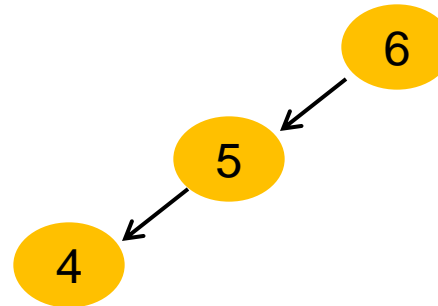


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

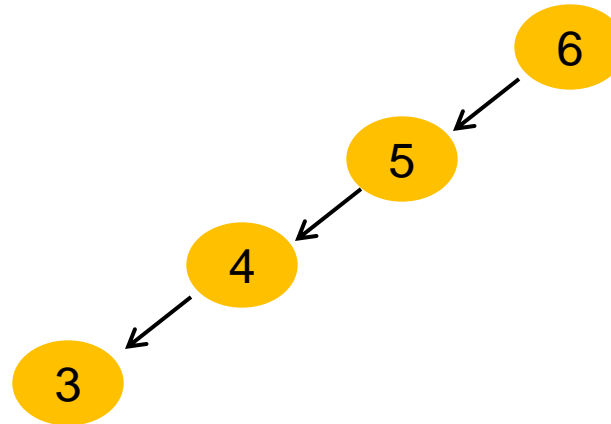


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

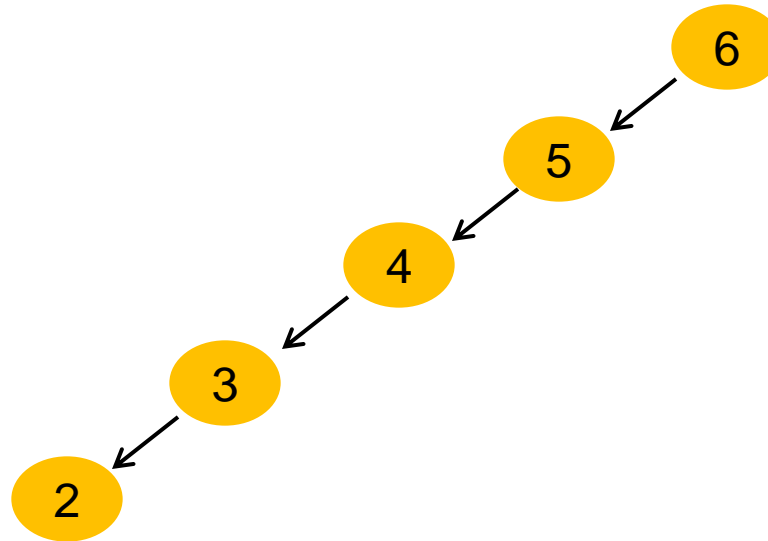


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

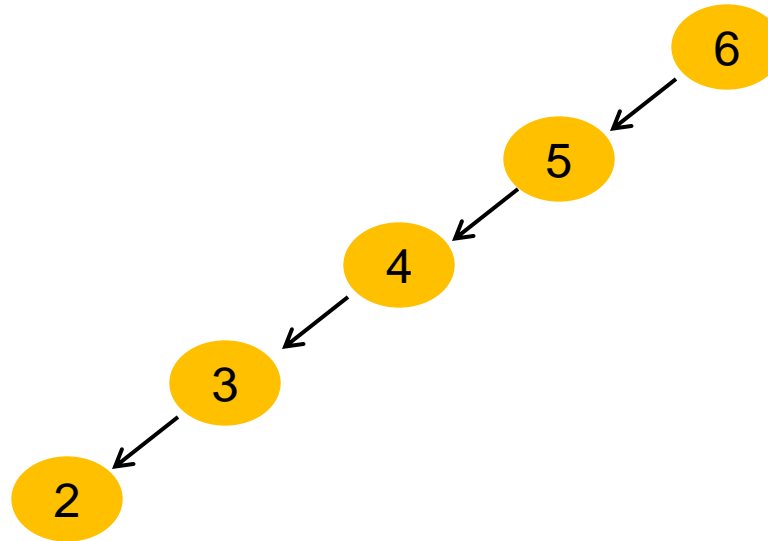


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

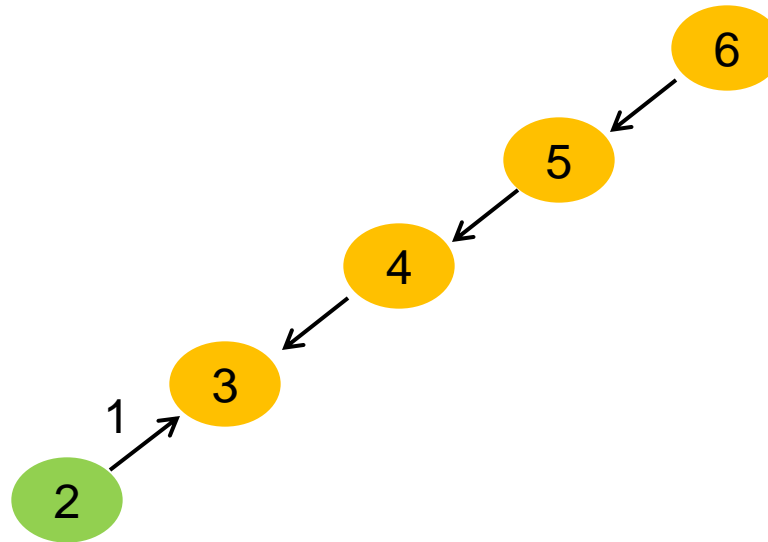


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

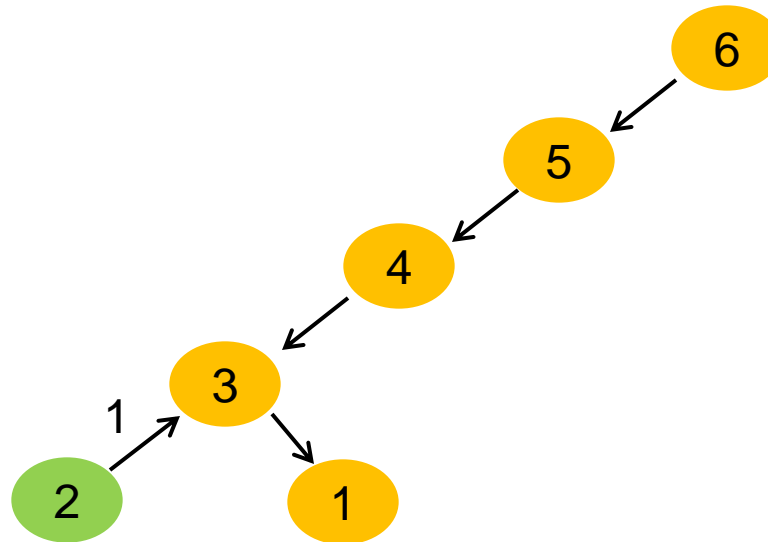


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

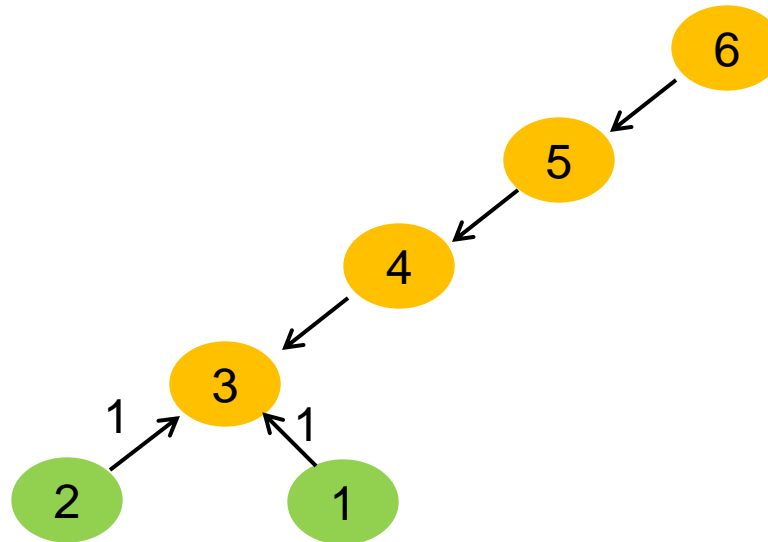


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )



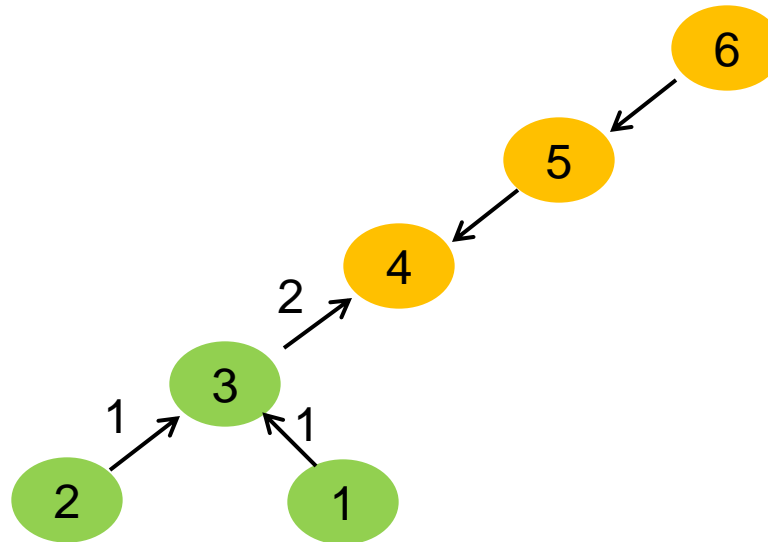
$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	0	0	0	0



## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

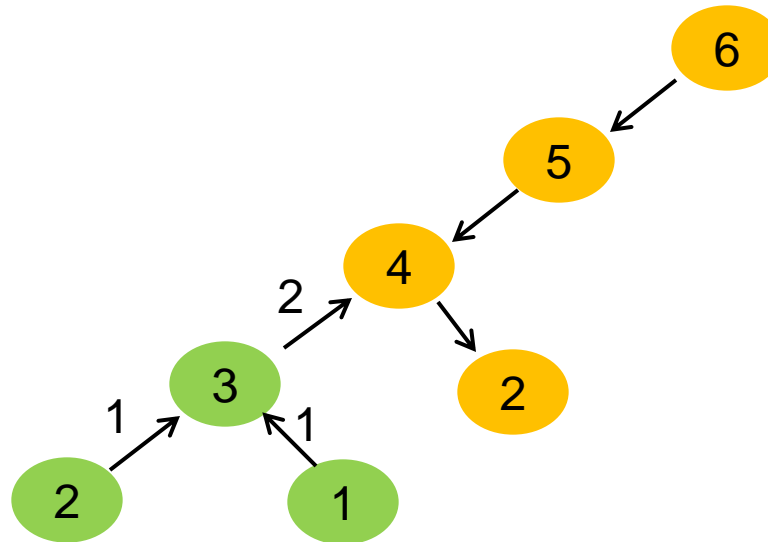


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

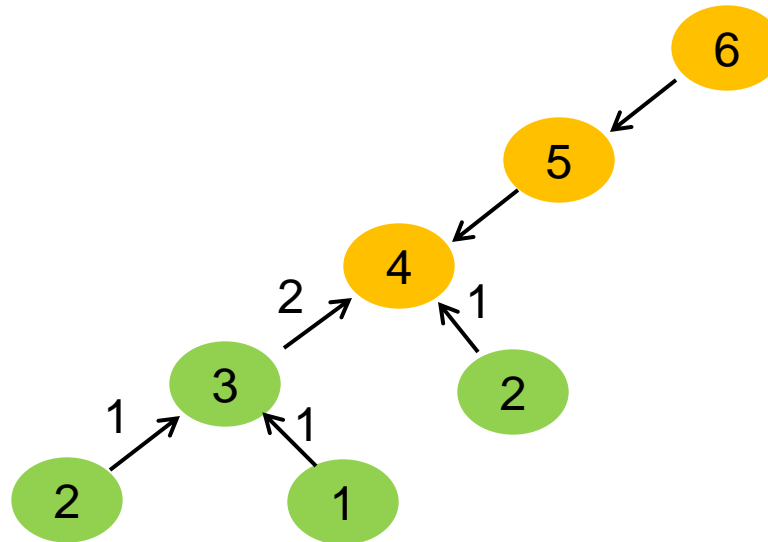


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

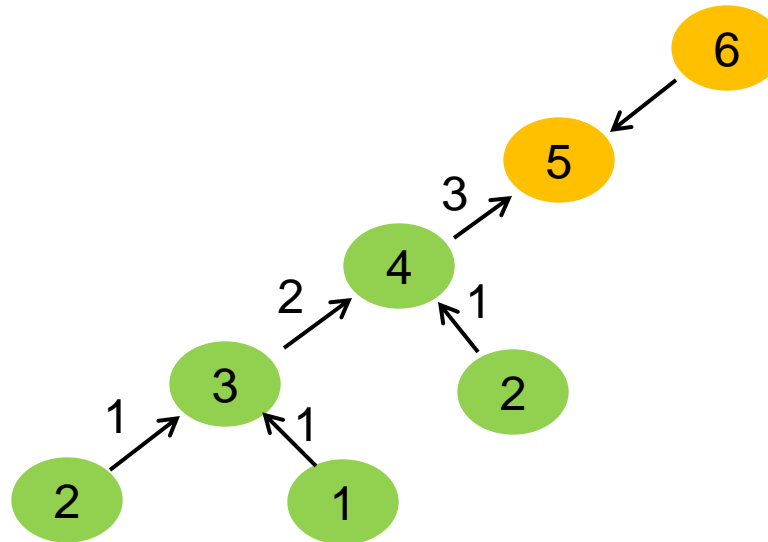


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	0	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

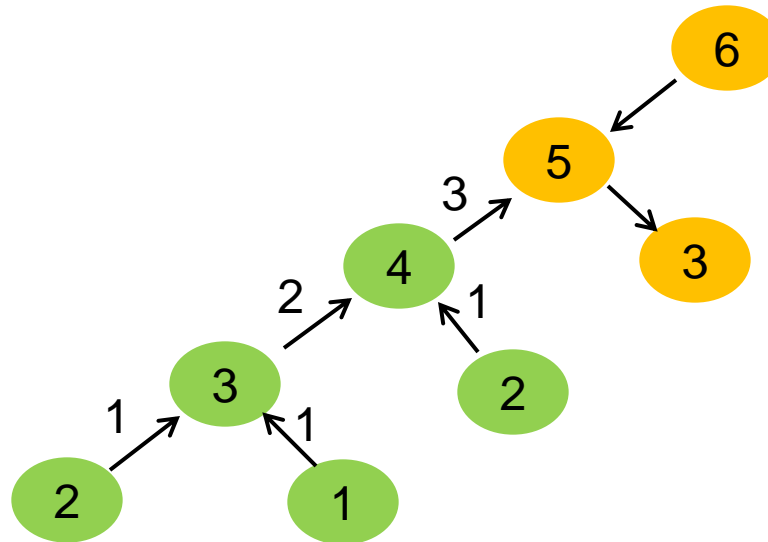


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

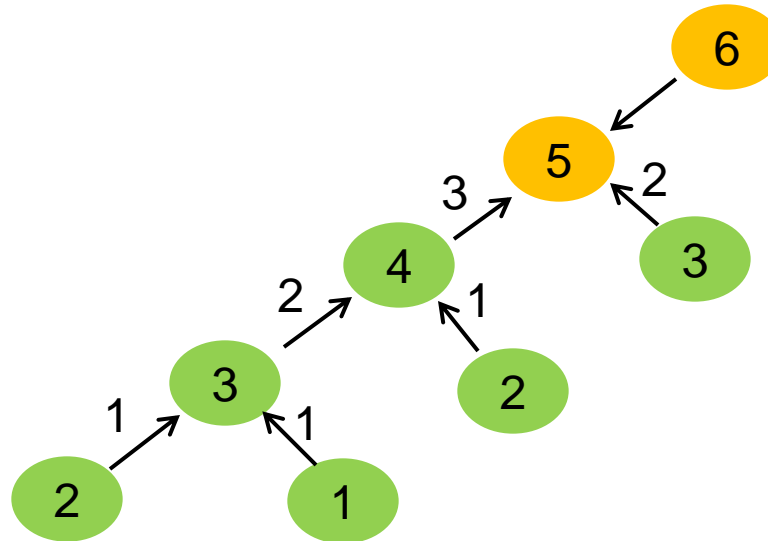


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

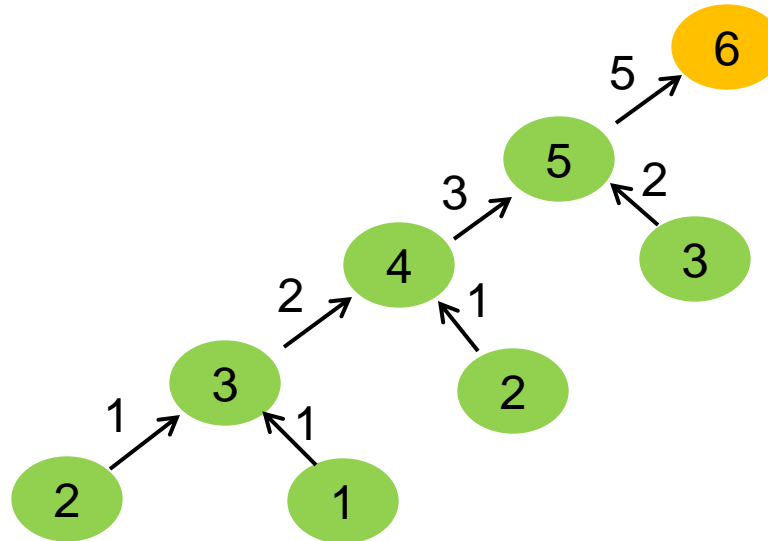


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	0	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

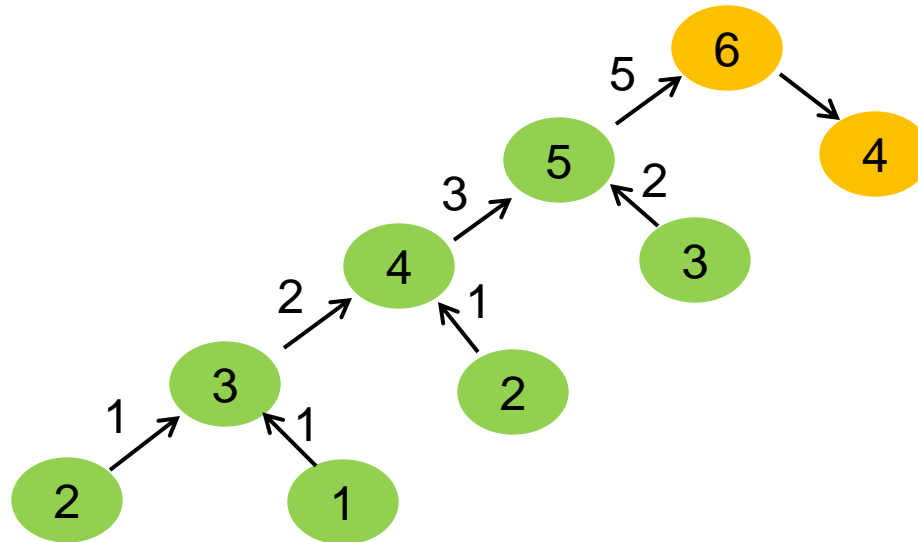


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	5	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )



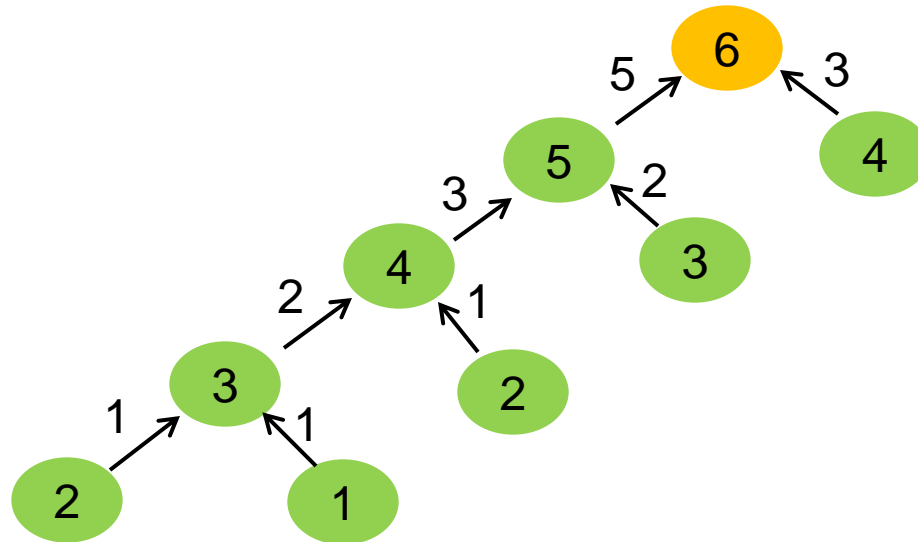
$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	5	0



## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )

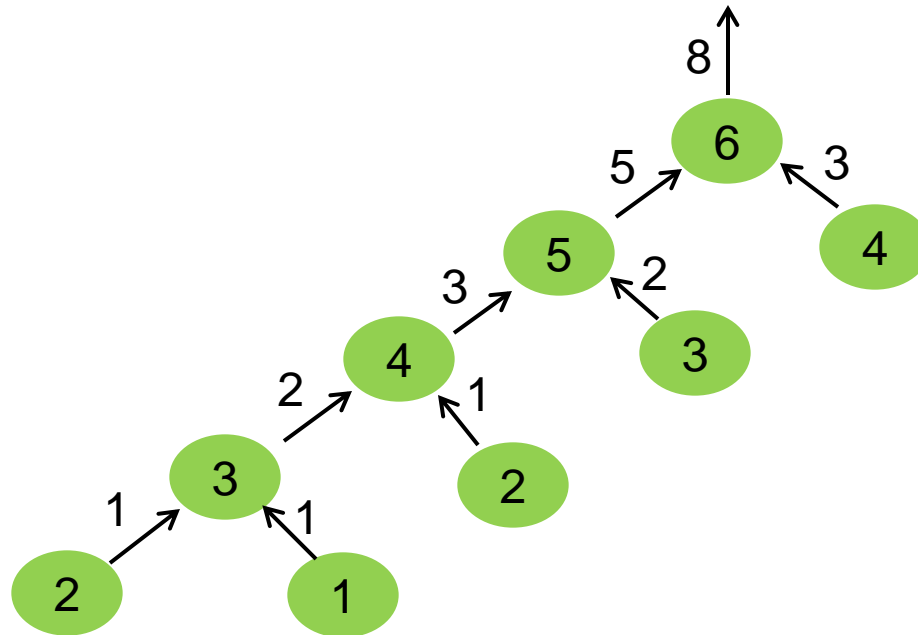


$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	5	0

## Dynamische Programmierung

### Beispiel

FibMemo(6,  $F$ )



$i$ :	1	2	3	4	5	6
$F[i]$ :	1	1	2	3	5	8

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

Welche Laufzeit hat FibMemo?

- A)  $\mathbf{O}(\log n)$
- B)  $\mathbf{O}(n)$
- C)  $\mathbf{O}(n \log n)$
- D)  $\mathbf{O}(2^n)$

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

### *Laufzeit*

- Jeder Aufruf  $\text{FibMemo}(m, F)$  generiert höchstens einmal die rekursiven Aufrufe  $\text{FibMemo}(m - 1, F)$  und  $\text{FibMemo}(m - 2, F)$

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

### Laufzeit

- Jeder Aufruf  $\text{FibMemo}(m, F)$  generiert höchstens einmal die rekursiven Aufrufe  $\text{FibMemo}(m - 1, F)$  und  $\text{FibMemo}(m - 2, F)$
- Also wird für jedes  $m < n$  die Funktion  $\text{FibMemo}(m, F)$  höchstens zweimal aufgerufen

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

### Laufzeit

- Jeder Aufruf  $\text{FibMemo}(m, F)$  generiert höchstens einmal die rekursiven Aufrufe  $\text{FibMemo}(m - 1, F)$  und  $\text{FibMemo}(m - 2, F)$
- Also wird für jedes  $m < n$  die Funktion  $\text{FibMemo}(m, F)$  höchstens zweimal aufgerufen
- Ohne die Laufzeit für die rekursiven Aufrufe benötigt FibMemo  **$O(1)$**  Zeit

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

### Laufzeit

- Jeder Aufruf  $\text{FibMemo}(m, F)$  generiert höchstens einmal die rekursiven Aufrufe  $\text{FibMemo}(m - 1, F)$  und  $\text{FibMemo}(m - 2, F)$
- Also wird für jedes  $m < n$  die Funktion  $\text{FibMemo}(m, F)$  höchstens zweimal aufgerufen
- Ohne die Laufzeit für die rekursiven Aufrufe benötigt  $\text{FibMemo}$   $\mathbf{O}(1)$  Zeit
- Wir zählen jeden Aufruf mit Parameter  $1..n$ . Daher müssen wir den Aufwand für die rekursiven Aufrufe nicht berücksichtigen. Damit ergibt sich eine Laufzeit von  $T(n) = \mathbf{O}(n)$

## Dynamische Programmierung

FibMemo( $n, F$ )

1. **if**  $F[n] > 0$  **then return**  $F[n]$
2.  $F[n] \leftarrow \text{FibMemo}(n - 1, F) + \text{FibMemo}(n - 2, F)$
3. **return**  $F[n]$

### Laufzeit

- Jeder Aufruf  $\text{FibMemo}(m, F)$  generiert höchstens einmal die rekursiven Aufrufe  $\text{FibMemo}(m - 1, F)$  und  $\text{FibMemo}(m - 2, F)$
- Also wird für jedes  $m < n$  die Funktion  $\text{FibMemo}(m, F)$  höchstens zweimal aufgerufen
- Ohne die Laufzeit für die rekursiven Aufrufe benötigt  $\text{FibMemo}$   $\mathbf{O}(1)$  Zeit
- Wir zählen jeden Aufruf mit Parameter  $1..n$ . Daher müssen wir den Aufwand für die rekursiven Aufrufe nicht berücksichtigen. Damit ergibt sich eine Laufzeit von  $T(n) = \mathbf{O}(n)$



## Dynamische Programmierung

### *Beobachtung*

- Die Funktionswerte werden bottom-up berechnet

### *Grundidee der dynamischen Programmierung*

- Berechne die Funktionswerte iterativ und bottom-up

FibDynamischeProgrammierung( $n$ )

1. Initialisiere Feld  $F[1..n]$
2.  $F[1] \leftarrow 1$
3.  $F[2] \leftarrow 1$
4. **for**  $i \leftarrow 3$  **to**  $n$  **do**
5.      $F[i] \leftarrow F[i - 1] + F[i - 2]$
6. **return**  $F[n]$

## Dynamische Programmierung

### Beobachtung

- Die Funktionswerte werden bottom-up berechnet

### Grundidee der *dynamischen Programmierung*

- Berechne die Funktionswerte iterativ und bottom-up

FibDynamischeProgrammierung( $n$ )

1. Initialisiere Feld  $F[1..n]$
  2.  $F[1] \leftarrow 1$
  3.  $F[2] \leftarrow 1$
  4. **for**  $i \leftarrow 3$  **to**  $n$  **do**
  5.      $F[i] \leftarrow F[i - 1] + F[i - 2]$
  6. **return**  $F[n]$
- } Laufzeit  $\mathbf{O}(n)$

## Dynamische Programmierung

### *Dynamische Programmierung*

- Formuliere Problem rekursiv
- Löse die Rekursion „bottom-up“ durch schrittweises Ausfüllen einer Tabelle der möglichen Lösungen

### *Wann ist dynamische Programmierung effizient?*

- Die Anzahl unterschiedlicher Funktionsaufrufe (Größe der Tabelle) ist klein
- Bei einer „normalen Ausführung“ des rekursiven Algorithmus ist mit vielen Mehrfachausführungen zu rechnen

## Dynamische Programmierung

### *Analyse der dynamischen Programmierung*

- Korrektheit: Für uns wird es genügen, eine Korrektheit der rekursiven Problemformulierung zu zeigen  
(für einen vollständigen formalen Korrektheitsbeweis wäre auch noch die korrekte Umsetzung des Auffüllens der Tabelle mittels Invarianten zu zeigen. Dies ist aber normalerweise offensichtlich)
- Laufzeit: Die Laufzeitanalyse ist meist recht einfach, da der Algorithmus typischerweise aus geschachtelten **for**-Schleifen besteht

### *Hauptschwierigkeit bei der Algorithmenentwicklung*

- Finden der Rekursion

## Dynamische Programmierung

### *Aufgabe*

Entwickeln Sie jeweils einen Algorithmus für die Maximumssuche in einem Feld  $A$

- (a) nach dem Teile-und-Herrsche-Prinzip
- (b) als gierigen Algorithmus
- (c) mit Hilfe von dynamischer Programmierung

## Dynamische Programmierung

### *Ein Spielzeugbeispiel*

- Maximum von  $n$  Zahlen
- Eingabe: Array  $A$  der Größe  $n$
- Ausgabe: Wert des Maximums der Zahlen in  $A$

### *Ziel*

- Dynamische Programmierung anhand dieses Beispiels durchspielen
- Natürlich keine Laufzeitverbesserung

## Dynamische Programmierung

### *Entwicklung der Rekursionsgleichung*

- Eingabe besteht aus  $n$  Elementen
- Idee: Ordne die Elemente von 1 bis  $n$  (das Eingabefeld  $A$  gibt z.B. eine solche Ordnung)
- Drücke optimale Lösung für die ersten  $i$  Elemente als Funktion der optimalen Lösung ersten  $i - 1$  Elemente aus

## Dynamische Programmierung

### *Entwicklung der Rekursionsgleichung*

- Eingabe besteht aus  $n$  Elementen
- Idee: Ordne die Elemente von 1 bis  $n$  (das Eingabefeld  $A$  gibt z.B. eine solche Ordnung)
- Drücke optimale Lösung für die ersten  $i$  Elemente als Funktion der optimalen Lösung ersten  $i - 1$  Elemente aus

### *Beispiel*

- Sei  $\text{Max}(i) = \max_{1 \leq j \leq i} \{A[j]\}$
- Dann gilt  $\text{Max}(1) = A[1]$  (Rekursionsabbruch)
- $\text{Max}(i) = \max\{\text{Max}(i - 1), A[i]\}$



## Dynamische Programmierung

MaxDynamic( $A$ )

1.  $n \leftarrow \text{length}[A]$
2. Initialisiere Feld  $\text{Max}[1..n]$
3.  $\text{Max}[1] \leftarrow A[1]$
4. **for**  $i \leftarrow 2$  **to**  $n$  **do**
5.      $\text{Max}[i] \leftarrow \max\{\text{Max}[i - 1], A[i]\}$
6. **return**  $\text{Max}[n]$

## Dynamische Programmierung

MaxDynamic( $A$ )

1.  $n \leftarrow \text{length}[A]$
2. Initialisiere Feld  $\text{Max}[1..n]$
3.  $\text{Max}[1] \leftarrow A[1]$
4. **for**  $i \leftarrow 2$  **to**  $n$  **do**
5.      $\text{Max}[i] \leftarrow \max\{\text{Max}[i - 1], A[i]\}$
6. **return**  $\text{Max}[n]$

*Wie kann man nur aus  $\text{Max}[1..n]$  den Index eines größten Elements von  $A$  herausfinden?*

- Sei  $j$  der größte Wert für den gilt  $\text{Max}[j] > \text{Max}[j - 1]$   
(mit der Konvention  $\text{Max}[0] = -\infty$ )
- Dann ist  $j$  der gesuchte Index
- Beweis  $\rightarrow$  Übung

## Dynamische Programmierung

### *Was lernen wir aus der Maximumsberechnung?*

- Wenn wir es mit Mengen zu tun haben, können wir eine Ordnung der Elemente einführen und die Rekursion durch Zurückführen der optimalen Lösung für  $i$  Elemente auf die Lösung für  $i - 1$  Elemente erhalten
- Benötigt wird dabei eine Beschreibung der optimalen Lösung für  $i - 1$  Elemente (hier nur der Wert der Lösung)
- Die Lösung selbst (der Index des Maximums) kann nachher aus der Tabelle rekonstruiert werden

## Dynamische Programmierung

### *Problem: Partition*

- Eingabe: Menge  $M$  mit  $n$  natürlichen Zahlen
- Ausgabe: Ja, gdw. man  $M$  in zwei Mengen  $L$  und  $R$  partitionieren kann, so dass  $\sum_{x \in L} x = \sum_{x \in R} x$  gilt; nein sonst

### *Beispiel*

- 4, 7, 9, 10, 13, 23

## Dynamische Programmierung

### *Problem: Partition*

- Eingabe: Menge  $M$  mit  $n$  natürlichen Zahlen
- Ausgabe: Ja, gdw. man  $M$  in zwei Mengen  $L$  und  $R$  partitionieren kann, so dass  $\sum_{x \in L} x = \sum_{x \in R} x$  gilt; nein sonst

### *Beispiel*

- 4, 7, 9, 10, 13, 23

## Dynamische Programmierung

### *Problem: Partition*

- Eingabe: Menge  $M$  mit  $n$  natürlichen Zahlen
- Ausgabe: Ja, gdw. man  $M$  in zwei Mengen  $L$  und  $R$  partitionieren kann, so dass  $\sum_{x \in L} x = \sum_{x \in R} x$  gilt; nein sonst

### *Beispiel*

- 4, 7, 9, 10, 13, 23
- $4 + 7 + 9 + 13 = 33$

## Dynamische Programmierung

### *Problem: Partition*

- Eingabe: Menge  $M$  mit  $n$  natürlichen Zahlen
- Ausgabe: Ja, gdw. man  $M$  in zwei Mengen  $L$  und  $R$  partitionieren kann, so dass  $\sum_{x \in L} x = \sum_{x \in R} x$  gilt; nein sonst

### *Beispiel*

- 4, 7, 9, 10, 13, 23
- $4 + 7 + 9 + 13 = 33$
- $10 + 23 = 33$

## Dynamische Programmierung

### *Problem: Partition*

- Eingabe: Menge  $M$  mit  $n$  natürlichen Zahlen
- Ausgabe: Ja, gdw. man  $M$  in zwei Mengen  $L$  und  $R$  partitionieren kann, so dass  $\sum_{x \in L} x = \sum_{x \in R} x$  gilt; nein sonst

### *Beispiel*

- 4, 7, 9, 10, 13, 23
- $4 + 7 + 9 + 13 = 33$
- $10 + 23 = 33$
- Ausgabe: Ja



## Dynamische Programmierung

### *Beobachtung*

- Sei  $M$  eine Menge mit  $n$  natürlichen Zahlen.  
 $M$  kann genau dann in zwei Mengen  $L, R$  mit  $\sum_{x \in L} x = \sum_{x \in R} x$  partitioniert werden, wenn es eine Teilmenge  $L$  von  $M$  gibt mit  $\sum_{x \in L} x = W/2$ , wobei  $W = \sum_{x \in M} x$  die Summe aller Zahlen aus  $M$  ist.

## Dynamische Programmierung

### Beobachtung

- Sei  $M$  eine Menge mit  $n$  natürlichen Zahlen.  
 $M$  kann genau dann in zwei Mengen  $L, R$  mit  $\sum_{x \in L} x = \sum_{x \in R} x$  partitioniert werden, wenn es eine Teilmenge  $L$  von  $M$  gibt mit  $\sum_{x \in L} x = W/2$ , wobei  $W = \sum_{x \in M} x$  die Summe aller Zahlen aus  $M$  ist.

### Neue Frage

- Gibt es  $L \subseteq M$  mit  $\sum_{x \in L} x = W/2$ ?
- Gibt es  $L \subseteq M$  mit  $\sum_{x \in L} x = U$  für ein vorgegebenes  $U$ ?

## Dynamische Programmierung

### *Fragestellung*

Gibt es  $L \subseteq M$  mit  $\sum_{x \in L} x = U$  für ein vorgegebenes  $U$ ?

### *Beobachtungen*

- Entscheidungsproblem (Antwort ist ja oder nein)
- Einfache Rekursion nicht möglich  
(die Lösung für  $i - 1$  Elemente sagt i.a. nichts über die Lösung für  $i$  Elemente aus)
- Benötige daher alle Lösungen für  $i - 1$  Elemente!

## Dynamische Programmierung

### *Allgemeinere Frage*

- Welche Zahlen lassen sich als Summe einer Teilmenge von  $M$  darstellen?
- Ausgabe: Feld  $E[0 \dots W]$  mit  $E[j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M$  darstellen kann
- $W$  ist die Summe aller Zahlen aus  $M$

## Dynamische Programmierung

### *Idee*

- Vorgehen wie bei Max: Berechne Lösung für die ersten  $i$  Elemente
- Neu: Lösung ist nicht eine Zahl, sondern ein Feld  $\rightarrow$  2D Lösung

## Dynamische Programmierung

### Gesuchte Lösungen

- Sei  $E(i, j) = 1$  , wenn man die Zahl  $j$  als Summe einer Teilmenge der ersten  $i$  Zahlen aus  $M$  darstellen kann
- Sei  $E(i, j) = 0$ , sonst
- Zeile  $i$  von  $E$  entspricht einer Lösung für die Zahlen aus  $M[1..i]$

### Hinweis

- $E(..)$  bezeichnet eine Lösung eines Entscheidungsproblems
- $E[..]$  bezeichnet die vom Algorithmus berechnete Lösung (beide sollten natürlich gleich sein)

## Dynamische Programmierung

### *Beispiel*

- $M = \{13, 7, 10, 15\}$
- $E(1,20) = 0$ , da man 20 nicht als Summe einer Teilmenge der ersten Zahl aus  $M$  darstellen kann
- $E(2,20) = 1$ , da man  $20 = 13 + 7$  als Summe einer Teilmenge der ersten beiden Zahlen aus  $M$  darstellen kann

## Dynamische Programmierung

### *Rekursion (Struktur der Lösungen des Entscheidungsproblems)*

- $E(i, j) = 1$ , wenn  $E(i - 1, j) = 1$  oder  $(j \geq M[i] \text{ und } E(i - 1, j - M[i]) = 1)$
- $E(i, j) = 0$ , sonst

### *Rekursionsabbruch*

- $E(0, 0) = 1$   
(Man kann die Null als Summe über die leere Menge darstellen)
- $E(0, j) = 0$  für  $j > 0$   
(Man kann keine Zahl ungleich 0 als Summe über die leere Menge darstellen)



## Dynamische Programmierung

PartitionDynamicProg( $M$ )

1.  $W \leftarrow 0$
2. **for**  $i \leftarrow 1$  **to**  $\text{length}[M]$  **do**
3.      $W \leftarrow W + M[i]$
4. **if**  $W$  ist ungerade **then return** 0
5. Initialisiere Feld  $E[0..\text{length}[M]][0..W]$
6. **for**  $i \leftarrow 0$  **to**  $\text{length}[M]$  **do**
7.     **for**  $j \leftarrow 0$  **to**  $W/2$  **do**
8.         **if**  $j = 0$  **then**  $E[i, j] \leftarrow 1$
9.         **else if**  $i = 0$  **then**  $E[i, j] \leftarrow 0$
10.         **else if**  $E[i - 1, j] = 1$  **or**  $(M[i] \leq j \text{ und } E[i - 1, j - M[i]] = 1)$  **then**  $E[i, j] \leftarrow 1$
11.         **else**  $E[i, j] \leftarrow 0$
12. **return**  $E[\text{length}[M], W/2]$

## Dynamische Programmierung

PartitionDynamicProg( $M$ )

1.  $W \leftarrow 0$
  2. **for**  $i \leftarrow 1$  **to**  $\text{length}[M]$  **do**
  3.      $W \leftarrow W + M[i]$
  4. **if**  $W$  ist ungerade **then return** 0
  5. Initialisiere Feld  $E[0..\text{length}[M]][0..W]$
  6. **for**  $i \leftarrow 0$  **to**  $\text{length}[M]$  **do**
  7.     **for**  $j \leftarrow 0$  **to**  $W/2$  **do**
  8.         **if**  $j = 0$  **then**  $E[i, j] \leftarrow 1$
  9.         **else if**  $i = 0$  **then**  $E[i, j] \leftarrow 0$
  10.         **else if**  $E[i - 1, j] = 1$  **or** ( $M[i] \leq j$  und  $E[i - 1, j - M[i]] = 1$ ) **then**  $E[i, j] \leftarrow 1$
  11.         **else**  $E[i, j] \leftarrow 0$
  12. **return**  $E[\text{length}[M], W/2]$
- Berechnen von  $W$  und
  - Initialisierung von  $E$

## Dynamische Programmierung

PartitionDynamicProg( $M$ )

1.  $W \leftarrow 0$
  2. **for**  $i \leftarrow 1$  **to**  $\text{length}[M]$  **do**
  3.      $W \leftarrow W + M[i]$
  4. **if**  $W$  ist ungerade **then return** 0
  5. Initialisiere Feld  $E[0..\text{length}[M]][0..W]$
  6. **for**  $i \leftarrow 0$  **to**  $\text{length}[M]$  **do**
  7.     **for**  $j \leftarrow 0$  **to**  $W/2$  **do**
  8.         **if**  $j = 0$  **then**  $E[i, j] \leftarrow 1$
  9.         **else if**  $i = 0$  **then**  $E[i, j] \leftarrow 0$
  10.         **else if**  $E[i - 1, j] = 1$  **or** ( $M[i] \leq j$  und  $E[i - 1, j - M[i]] = 1$ ) **then**  $E[i, j] \leftarrow 1$
  11.         **else**  $E[i, j] \leftarrow 0$
  12. **return**  $E[\text{length}[M], W/2]$
- Berechnen von  $W$  und
  - Initialisierung von  $E$
  - Berechnung von  $E$

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

<b>j \ i</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>0</b>											
<b>1</b>											
<b>2</b>											
<b>3</b>											
<b>4</b>											
<b>5</b>											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

<b>i \ j</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>0</b>	1										
<b>1</b>											
<b>2</b>											
<b>3</b>											
<b>4</b>											
<b>5</b>											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

<b>i \ j</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>0</b>	1	0	0	0	0	0	0	0	0	0	0
<b>1</b>											
<b>2</b>											
<b>3</b>											
<b>4</b>											
<b>5</b>											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[1] = 1$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1										
2											
3											
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[1] = 1$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1									
2											
3											
4											
5											



# Dynamische Programmierung

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$$M[1] = 1$$

[illegible]

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[2] = 4$$

<b>i \ j</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>0</b>	1	0	0	0	0	0	0	0	0	0	0
<b>1</b>	1	1	0	0	0	0	0	0	0	0	0
<b>2</b>	1										
<b>3</b>											
<b>4</b>											
<b>5</b>											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[2] = 4$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1									
3											
4											
5											

## Dynamische Programmierung

$$M[2] = 4$$

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0							
3											
4											
5											

## Dynamische Programmierung

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$$M[2] = 4$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1						
3											
4											
5											

## Dynamische Programmierung

Beispiel:  $M = \{1, 4, 3, 5, 7\}$

$$M[2] = 4$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1					
3											
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[2] = 4$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3											
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1										
4											
5											



## Dynamische Programmierung

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1 ↓	0	0	1	1	0	0	0	0	0
3	1	1									
4											
5											

## Dynamische Programmierung

$$M[3] = 3$$

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0 ↓	0	1	1	0	0	0	0	0
3	1	1	0								
4											
5											

## Dynamische Programmierung

$$M[3] = 3$$

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1							
4											
5											

## Dynamische Programmierung

Beispiel:  $M = \{1, 4, 3, 5, 7\}$

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1					
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0				
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	1	1		
4											
5											

## Dynamische Programmierung

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$$M[3] = 3$$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	1	1	0	0
4											
5											

## Dynamische Programmierung

$$M[4] = 5$$

*Beispiel:  $M = \{1, 4, 3, 5, 7\}$*

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	1	1	0	0
4	1										
5											



## Dynamische Programmierung

$$M[4] = 5$$

Beispiel:  $M = \{1, 4, 3, 5, 7\}$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	1	1	0	0
4	1	1	0	1	1	1	1	1	1	1	1
5											

## Dynamische Programmierung

$$M[5] = 7$$

*Beispiel:*  $M = \{1, 4, 3, 5, 7\}$

$i \backslash j$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	1	1	0	0
4	1	1	0	1	1	1	1	1	1	1	1
5	1	1	0	1	1	1	1	1	1	1	1

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- PartitionDynamicProg berechnet zunächst die Summe  $W$  der Elemente aus  $M$ . Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- PartitionDynamicProg berechnet zunächst die Summe  $W$  der Elemente aus  $M$ . Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion  $E$ , mit
- $E[0,0] = 1$
- $E[0,j] = 0$  für alle  $j > 0$
- $E[i,j] = 1$ , gdw.  $E[i-1,j] = 1$  oder  $(j \geq M[i] \text{ und } E[i-1,j-M[i]] = 1)$

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- PartitionDynamicProg berechnet zunächst die Summe  $W$  der Elemente aus  $M$ . Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion  $E$ , mit
- $E[0,0] = 1$
- $E[0,j] = 0$  für alle  $j > 0$
- $E[i,j] = 1$ , gdw.  $E[i-1,j] = 1$  oder ( $j \geq M[i]$  und  $E[i-1,j-M[i]] = 1$ )
- **Der Algorithmus gibt 1 zurück, gdw.  $E[\text{length}[M], W/2] = 1$ .**

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- PartitionDynamicProg berechnet zunächst die Summe  $W$  der Elemente aus  $M$ . Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion  $E$ , mit
- $E[0,0] = 1$
- $E[0,j] = 0$  für alle  $j > 0$
- $E[i,j] = 1$ , gdw.  $E[i-1,j] = 1$  oder  $(j \geq M[i] \text{ und } E[i-1,j-M[i]] = 1)$
- Der Algorithmus gibt 1 zurück, gdw.  $E[\text{length}[M], W/2] = 1$ .

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- Wir zeigen per Induktion über  $i$ :
- $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- Wir zeigen per Induktion über  $i$ :
- $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann
- (I.A.) Für  $i = 0, j = 0$  kann man  $j$  als Summe der leeren Teilmenge darstellen  
Für  $i = 0, j > 0$ , kann man  $j$  nicht als Summe einer Teilmenge der leeren Menge darstellen



## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- Wir zeigen per Induktion über  $i$ :
- $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann
- (I.A.) Für  $i = 0, j = 0$  kann man  $j$  als Summe der leeren Teilmenge darstellen  
Für  $i = 0, j > 0$ , kann man  $j$  nicht als Summe einer Teilmenge der leeren Menge darstellen
- (I.V.) Für alle  $k < i$  und alle  $j$  wird  $E[k, j]$  korrekt berechnet

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- Wir zeigen per Induktion über  $i$ :
- $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann
- (I.A.) Für  $i = 0, j = 0$  kann man  $j$  als Summe der leeren Teilmenge darstellen  
Für  $i = 0, j > 0$ , kann man  $j$  nicht als Summe einer Teilmenge der leeren Menge darstellen
- (I.V.) Für alle  $k < i$  und alle  $j$  wird  $E[k, j]$  korrekt berechnet
- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i - 1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i - 1]$ .

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i - 1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i - 1]$ . Im ersten Fall folgt aus (I.V.), dass  $E[i - 1, j] = 1$  ist und somit auch  $E[i, j] = 1$ .

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i - 1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i - 1]$ . Im ersten Fall folgt aus (I.V.), dass  $E[i - 1, j] = 1$  ist und somit auch  $E[i, j] = 1$ . **Im zweiten Fall muss die Teilmenge von  $M[1..i - 1]$  Summe  $j - M[i]$  haben.**

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i - 1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i - 1]$ . Im ersten Fall folgt aus (I.V.), dass  $E[i - 1, j] = 1$  ist und somit auch  $E[i, j] = 1$ . Im zweiten Fall muss die Teilmenge von  $M[1..i - 1]$  Summe  $j - M[i]$  haben. **Nach (I.V.) ist dann aber  $E[i - 1, j - M[i]] = 1$  und somit  $E[i, j] = 1$ .**

## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i-1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i-1]$ . Im ersten Fall folgt aus (I.V.), dass  $E[i-1, j] = 1$  ist und somit auch  $E[i, j] = 1$ . Im zweiten Fall muss die Teilmenge von  $M[1..i-1]$  Summe  $j - M[i]$  haben. Nach (I.V.) ist dann aber  $E[i-1, j - M[i]] = 1$  und somit  $E[i, j] = 1$ .
- „ $\Rightarrow$ “ Ist  $E[i, j] = 1$ , so war entweder  $E[i-1, j] = 1$  oder  $E[i-1, j - M[i]] = 1$ . Nach (I.V.) kann man entweder  $j$  oder  $j - M[i]$  als Teilmenge von  $M[1..i-1]$  darstellen. Somit kann man  $j$  als Teilmenge von  $M[1..i]$  darstellen.



## Dynamische Programmierung

### Lemma 22

Algorithmus PartitionDynamicProg ist korrekt.

### Beweis

- (I.S.) Z.z.  $E[i, j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\Leftarrow$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen, so kann man  $j$  entweder als Teilmenge von  $M[1..i - 1]$  darstellen oder als  $M[i]$  vereinigt mit einer Teilmenge von  $M[1..i - 1]$ . Im ersten Fall folgt aus (I.V.), dass  $E[i - 1, j] = 1$  ist und somit auch  $E[i, j] = 1$ . Im zweiten Fall muss die Teilmenge von  $M[1..i - 1]$  Summe  $j - M[i]$  haben. Nach (I.V.) ist dann aber  $E[i - 1, j - M[i]] = 1$  und somit  $E[i, j] = 1$ .
- „ $\Rightarrow$ “ Ist  $E[i, j] = 1$ , so war entweder  $E[i - 1, j] = 1$  oder  $E[i - 1, j - M[i]] = 1$ . Nach (I.V.) kann man entweder  $j$  oder  $j - M[i]$  als Teilmenge von  $M[1..i - 1]$  darstellen. Somit kann man  $j$  als Teilmenge von  $M[1..i]$  darstellen.

## Dynamische Programmierung

### *Lemma 22*

Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

Somit gilt  $E[\text{length}[M], W/2] = 1$  gdw. man  $W/2$  als Summe einer Teilmenge von  $M[1..\text{length}[M]]$  darstellen kann.

## Dynamische Programmierung

### *Laufzeitanalyse*

Die Laufzeit wird durch die zwei geschachtelten **for**-Schleifen dominiert und ist höchstens  $\mathbf{O}(1) \cdot n \cdot W$ .

### *Erweiterung der $\mathbf{O}$ -Notation auf zwei Variablen*

$\mathbf{O}(f(n, m)) = \{g(n, m) \mid \exists n_0, m_0, c > 0, \text{ so dass für alle } n \geq n_0, m \geq m_0 \text{ gilt,}$   
dass  $g(n, m) \leq c \cdot f(n, m)\}$

### *Bemerkung*

Diese Definition kann in konstruierten Fällen zu nicht gewünschten Aussagen führen! (z.B. wenn  $g(1, m) = m^2$  ist und  $g(n, m) = m$  für  $n > 1$ )

## Dynamische Programmierung

### Satz 23

Sei  $M$  eine Menge von  $n$  natürlichen Zahlen und  $W$  die Summe der Zahlen aus  $M$ . Algorithmus PartitionDynamicProg löst Partition in Zeit  $\mathbf{O}(nW)$ .

### Beweis

- Die Korrektheit des Algorithmus folgt aus Lemma 22.
- Die Laufzeit ist offensichtlich  $\mathbf{O}(nW)$ .

## Dynamische Programmierung

### *Bemerkung*

- Partition ist ein NP-vollständiges Problem
- Damit gibt es wahrscheinlich keinen polynomiellen Algorithmus für Partition

### *Warum ist unser Algorithmus nicht polynomiell?*

- Die Laufzeit hängt von  $W$  ab
- Sind die Zahlen aus  $M$  exponentiell groß, so ist die Laufzeit ebenfalls exponentiell