



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Organisatorisches

### *Test (18.5.)*

- Bitte seien Sie um 12 Uhr im Audimax
- Hilfsmittel: ein handbeschriebenes DIN A4 Blatt

### *Übungsgruppen*

- Wenn ihre Übungsgruppe wegen eines Feiertags ausfällt, gehen Sie in eine andere
- Falls Sie an keiner Übung teilnehmen, wird dies als Fehlen gewertet

## 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

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

- Die Laufzeit von  $\text{Fib}(n)$  ist  $\Omega(1.6^n)$ .

### Beweis

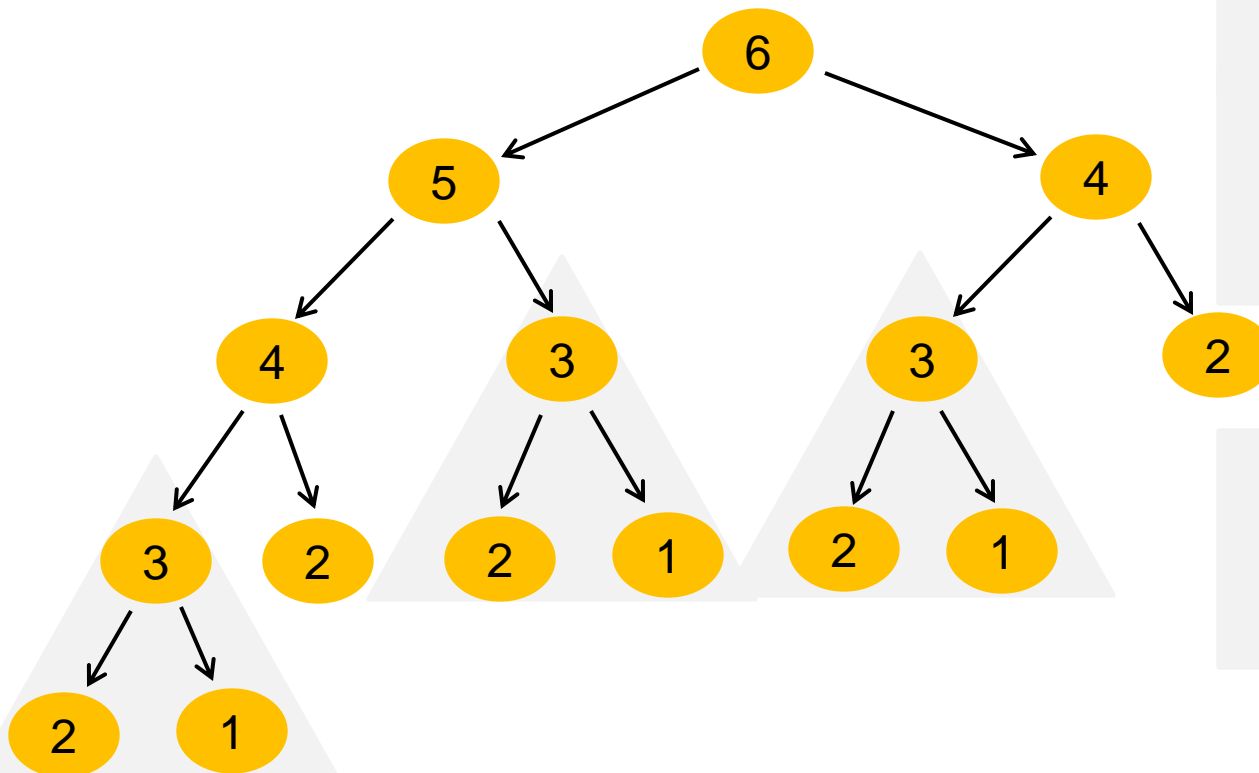
- (I.V.) Für  $m < n$  ist  $T(n) \geq \frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^m$ .
- (I.S.) Wir haben  $T(n) \geq T(n-1) + T(n-2)$  da der Algorithmus für  $n > 2$   $\text{Fib}(n-1)$  und  $\text{Fib}(n-2)$  rekursiv aufruft. Nach (I.V.) gilt somit

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

## Dynamische Programmierung

### Warum ist die Laufzeit so schlecht?

- Betrachten wir Rekursionbaum für Aufruf Fib(6)



Bei der Berechnung von Fib(6) wird Fib(3) dreimal aufgerufen!

Es wird also mehrmals dieselbe Rechnung durchgeführt!

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  $F[1..n]$  ➤ 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**  $\text{FibMemo}(n, F)$

### 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:

- $O(n)$   
 $O(n)$   
 $O(n)$   
 $O(1)$   
 $O(1)$   
 $1+T(n)$

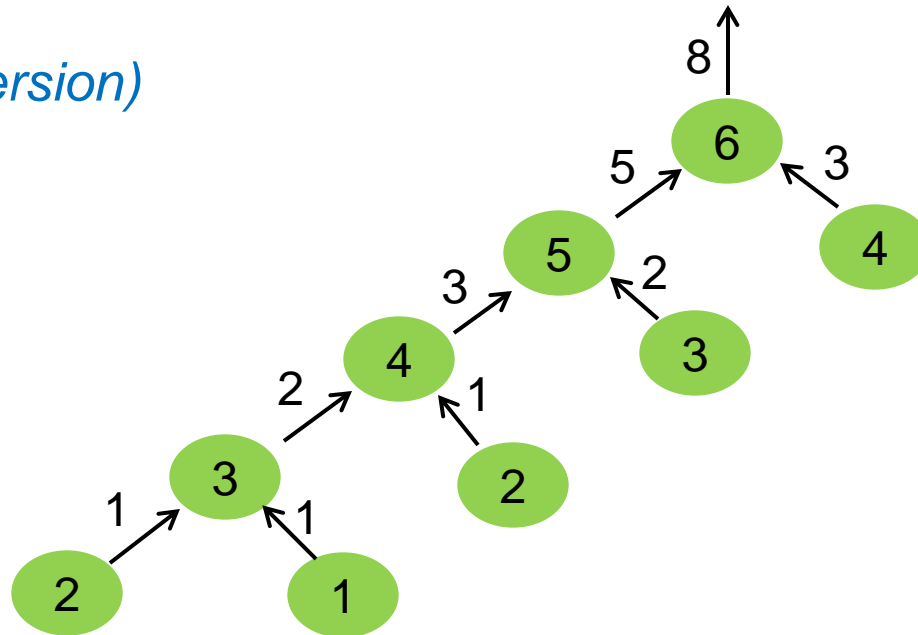
FibMemo(n,F)

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 (siehe Vollversion)*

- FibMemo(6,F)



<b>i:</b>	1	2	3	4	5	6
<b>F[i]:</b>	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**  $F[n]$

### *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  $\text{FibMemo}$   $O(1)$  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  $T(n) = O(n)$

## 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**  $F[i]$
- } Laufzeit  $O(n)$

## 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

### *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 (siehe Vollversion)*

- 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.

### Neue Frage

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

## Dynamische Programmierung

### *Allgemeinere Frage*

- Welche Zahlen lassen sich als Summe einer Teilmenge von  $M$  darstellen?

### *Noch allgemeiner*

- Sei  $M(i)$  die Menge der ersten  $i$  Zahlen von  $M$
- Für jedes  $i$ : Welche Zahlen lassen sich als Summe einer Teilmenge von  $M(i)$  darstellen?

### *Formulierung als Funktion*

- Sei  $G(i,j) = 1$  , wenn man die Zahl  $j$  als Summe einer Teilmenge der ersten  $i$  Zahlen aus  $M$  darstellen kann
- Sei  $G(i,j) = 0$  , sonst

## Dynamische Programmierung

### *Noch allgemeiner*

- Sei  $M(i)$  die Menge der ersten  $i$  Zahlen von  $M$
- Für jedes  $i$ : Welche Zahlen lassen sich als Summe einer Teilmenge von  $M(i)$  darstellen?

### *Formulierung als Funktion*

- Sei  $G(i,j) = 1$  , wenn man die Zahl  $j$  als Summe einer Teilmenge der ersten  $i$  Zahlen aus  $M$  darstellen kann
- Sei  $G(i,j) = 0$  , sonst
- Sei  $G(0,0) = 1$   
(Man kann die Null als Summe über die leere Menge darstellen)
- Sei  $G(0,j) = 0$  für  $j \neq 0$   
(Man kann keine Zahl ungleich 0 als Summe über die leere Menge darstellen)

## Dynamische Programmierung

### *Beispiel*

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

## Dynamische Programmierung

### *Formulierung als Funktion*

- Sei  $G(i,j) = 1$  , wenn man die Zahl  $j$  als Summe einer Teilmenge der ersten  $i$  Zahlen aus  $M$  darstellen kann
- Sei  $G(i,j) = 0$  , sonst
- Sei  $G(0,0) = 1$   
(Man kann die Null als Summe über die leere Menge darstellen)
- Sei  $G(0,j) = 0$  für  $j \neq 0$   
(Man kann keine Zahl ungleich 0 als Summe über die leere Menge darstellen)

### *Rekursion*

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

## Dynamische Programmierung

### PartitionMemoInit(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  $G[0..\text{length}[M]][0..W]$
  6. **for**  $i \leftarrow 0$  **to**  $\text{length}[M]$  **do**
  7.      $G[i,0] \leftarrow 1$
  8.     **for**  $j \leftarrow 1$  **to**  $W$  **do**
  9.          $G[i,j] \leftarrow -1$
  10. **for**  $j \leftarrow 1$  **to**  $W$  **do**
  11.      $G[0,j] \leftarrow 0$
  12. **if**  $\text{PartitionMemo}(M, G, n, W/2) = 1$  **then return** 1
  13. **else return** 0
- Berechne  $W$
  - Keine 2 gleich großen Teilmengen
  - Initialisiere Feld  $G$
  - Setze dabei  $G[i,0]$  auf 1 und
  - Setze dabei  $G[0,j]$ ,  $j > 0$ , auf 0
  - $G[i,j] = -1$  heißt, Funktionswert noch
  - nicht bekannt

## Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if**  $j < 0$  **return** 0

2. **if**  $G[i,j] \neq -1$  **then return**  $G[i,j]$

3. **if** PartitionMemo(M,G,i-1,j)=1

or PartitionMemo(M,G,i-1,j-M[i]) **then**  $G[i,j]=1$

4. **else**  $G[i,j]=0$

5. **return**  $G[i,j]$

➤ Gibt es Teilmenge S von  $M[1..i]$  mit  $\sum_{x \in S} x = j$  ?

➤ Wenn j ungültig gib false zurück

➤  $G[i,j]$  bereits berechnet?

➤ Sonst, berechne  $G[i,j]$  nach

➤ Rekursion

## 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  $G[0..\text{length}[M]][0..W]$
6. **for**  $i \leftarrow 0$  **to**  $\text{length}[M]$  **do** ➤ Berechnung
7.     **for**  $j \leftarrow 0$  **to**  $W/2$  **do** ➤ von G
8.         **if**  $j=0$  **then**  $G[i,j] \leftarrow 1$
9.         **else if**  $i=0$  **then**  $G[i,j] \leftarrow 0$
10.             **else if**  $G[i-1,j] = 1$  **or**  $(M[i] \leq j \text{ und } G[i-1,j-M[i]])$  **then**  $G[i,j] \leftarrow 1$
11.             **else**  $G[i,j] \leftarrow 0$
12. **return**  $G[\text{length}[M], W/2]$

## Dynamische Programmierung

*Beispiel:  $M = 1, 4, 3, 5, 7$  (siehe Vollversion)*

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

## Dynamische Programmierung

*Beispiel:  $M = 1, 4, 3, 5, 7$  (siehe Vollversion)*

M[5]=7

i \ 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.
- Ansonsten berechnet der Algorithmus die Funktion  $G$ , mit
- $G[i,0] = 1$  für alle  $i$
- $G[0,j] = 0$  für alle  $j > 0$
- $G[i,j] = 1$ , gdw.  $G[i-1,j] = 1$  oder  $(j \geq M[i] \text{ und } G[i-1, j-M[i]] = 1)$
- Der Algorithmus gibt 1 zurück, gdw.  $G[\text{length}[M], W/2] = 1$ .

## Dynamische Programmierung

### *Lemma 22*

- Algorithmus PartitionDynamicProg ist korrekt.

### *Beweis*

- Wir zeigen per Induktion über  $i$ :
- $G[i,j] = 1$ , 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  $G[k,j]$  korrekt berechnet
- (I.S.) Z.z.  $G[i,j] = 1$ , 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.  $G[i,j] = 1$ , gdw. man  $j$  als Summe einer Teilmenge von  $M[1..i]$  darstellen kann.
- „ $\leq$ “ Kann man  $j$  als Summe einer Teilmenge von  $M[1..i]$ , 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  $G[i-1,j]=1$  ist und somit auch  $G[i,j]=1$ . Im zweiten Fall muss die Teilmenge von  $M[1..i-1]$  Summe  $j-M[i]$  haben. Nach (I.V.) ist dann aber  $G[i-1,j-M[i]]=1$  und somit  $G[i,j] = 1$ .
- „ $\Rightarrow$ “ Ist  $G[i,j]=1$ , so war entweder  $G[i-1,j]=1$  oder  $G[i-1,j-M[i]]=1$ . 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.
- Somit gilt  $G[\text{length}[M],W/2] = 1$  gdw. man  $j$  als Summe einer Teilmenge von  $M[1..\text{length}[M]]$  darstellen kann.

## Dynamische Programmierung

### Satz 23

- Sei  $M$  eine Menge von  $n$  natürlichen Zahlen und  $R$  die Summe der Zahlen aus  $M$ . Algorithmus PartitionDynamicProg löst Partition in Zeit  $O(nW)$ .

### Beweis

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

## Dynamische Programmierung

### *Bemerkung*

- Partition ist ein NP-vollständiges Problem
- Damit gibt es wahrscheinlich keinen polynomieller 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