technische universität
dortmund

LS 2 /
Informatik

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 18

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

Beweis

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

Dynamische Programmierung

Lemma 18

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

Beweis

- Wir zeigen, dass die Laufzeit T(n) von Fib(n) größer als $\frac{1}{3}\cdot\left(\frac{1+\sqrt{5}}{2}\right)^n$.
- Damit folgt das Lemma wegen $\left(\frac{1+\sqrt{5}}{2}\right)\geq 1.6$.

Dynamische Programmierung

Lemma 18

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

Beweis

- Wir zeigen, dass die Laufzeit T(n) von 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) ≥ 1 ≥ $\frac{1}{3}\cdot\left(\frac{1+\sqrt{5}}{2}\right)$.

Dynamische Programmierung

Lemma 18

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

Beweis

- Wir zeigen, dass die Laufzeit T(n) von 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) ≥ 1 ≥ $\frac{1}{3}\cdot\left(\frac{1+\sqrt{5}}{2}\right)$.

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

Dynamische Programmierung

Lemma 18

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

Beweis

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

Dynamische Programmierung

Lemma 18

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

Beweis

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

- (I.S.) Wir haben T(n) ≥ T(n-1) + T(n-2) da der Algorithmus für n>2 Fib(n-1) und 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 18

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

Beweis

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

- (I.S.) Wir haben T(n) ≥ T(n-1) + T(n-2) da der Algorithmus für n>2 Fib(n-1) und Fib(n-2) rekursiv aufruft. Nach (I.V.) gilt somit

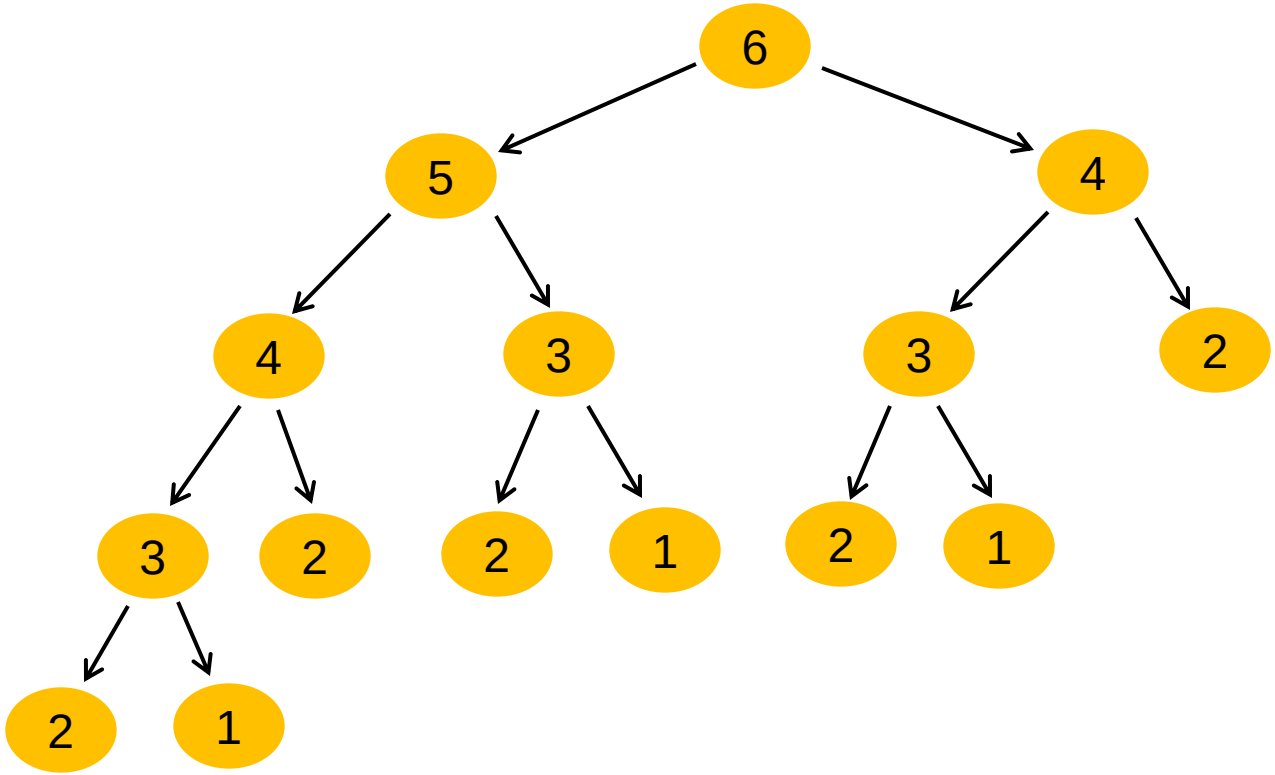
$$T(n) \geq T(n-1) + T(n-2) \geq \frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2} \right)^{n-1} + \frac{1}{3} \cdot \left(\frac{1+\sqrt{5}}{2} \right)^{n-2}$$

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

Dynamische Programmierung

Warum ist die Laufzeit so schlecht?

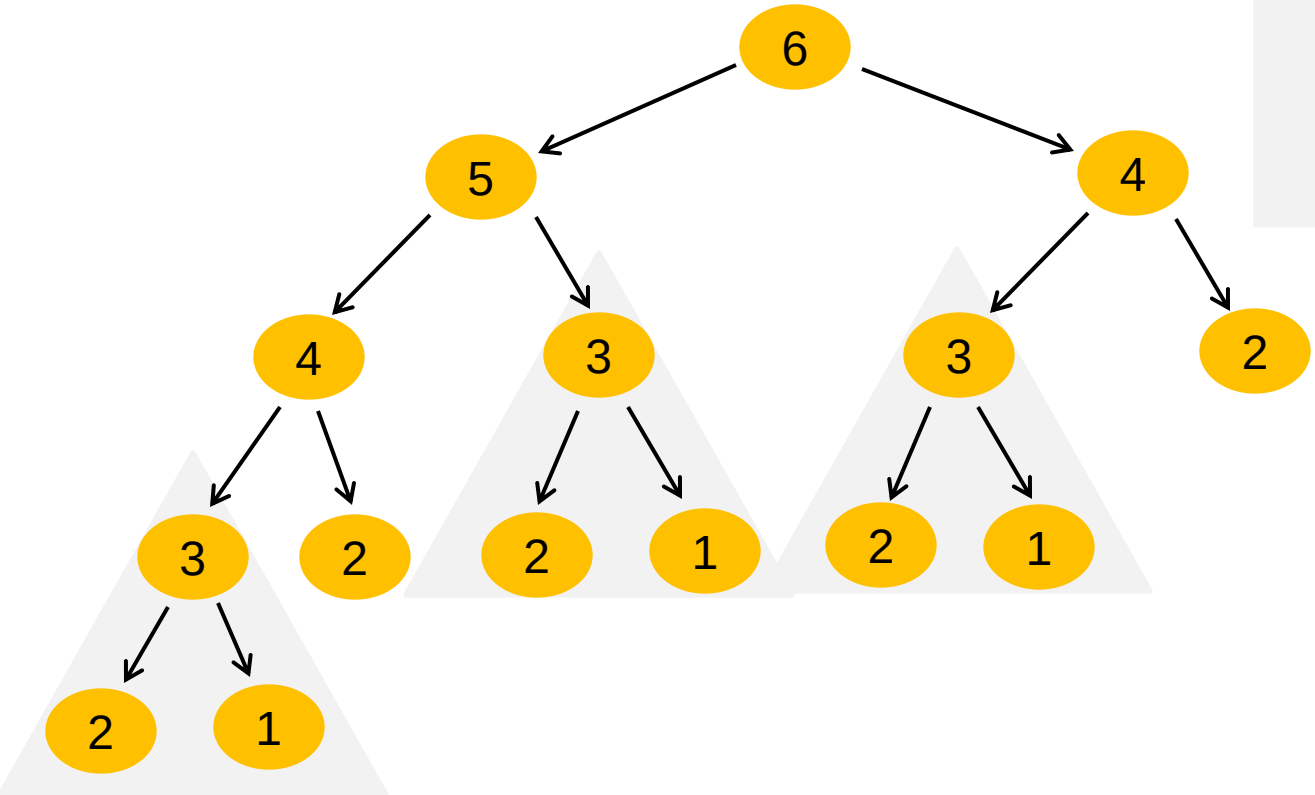
- Betrachten wir Rekursionbaum für Aufruf Fib(6)



Dynamische Programmierung

Warum ist die Laufzeit so schlecht?

- Betrachten wir Rekursionbaum für Aufruf Fib(6)

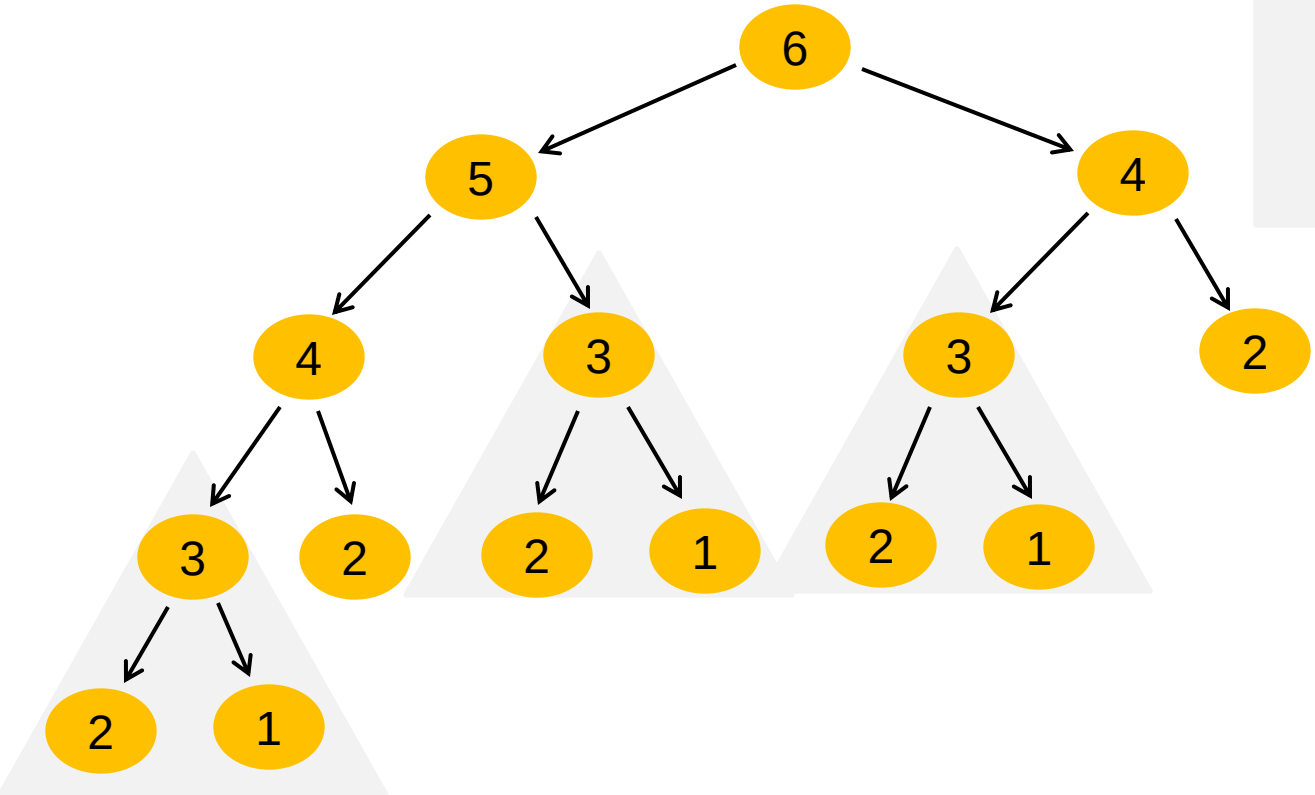


Bei der Berechnung von Fib(6) wird Fib(3) dreimal aufgerufen!

Dynamische Programmierung

Warum ist die Laufzeit so schlecht?

- Betrachten wir Rekursionbaum für Aufruf Fib(6)

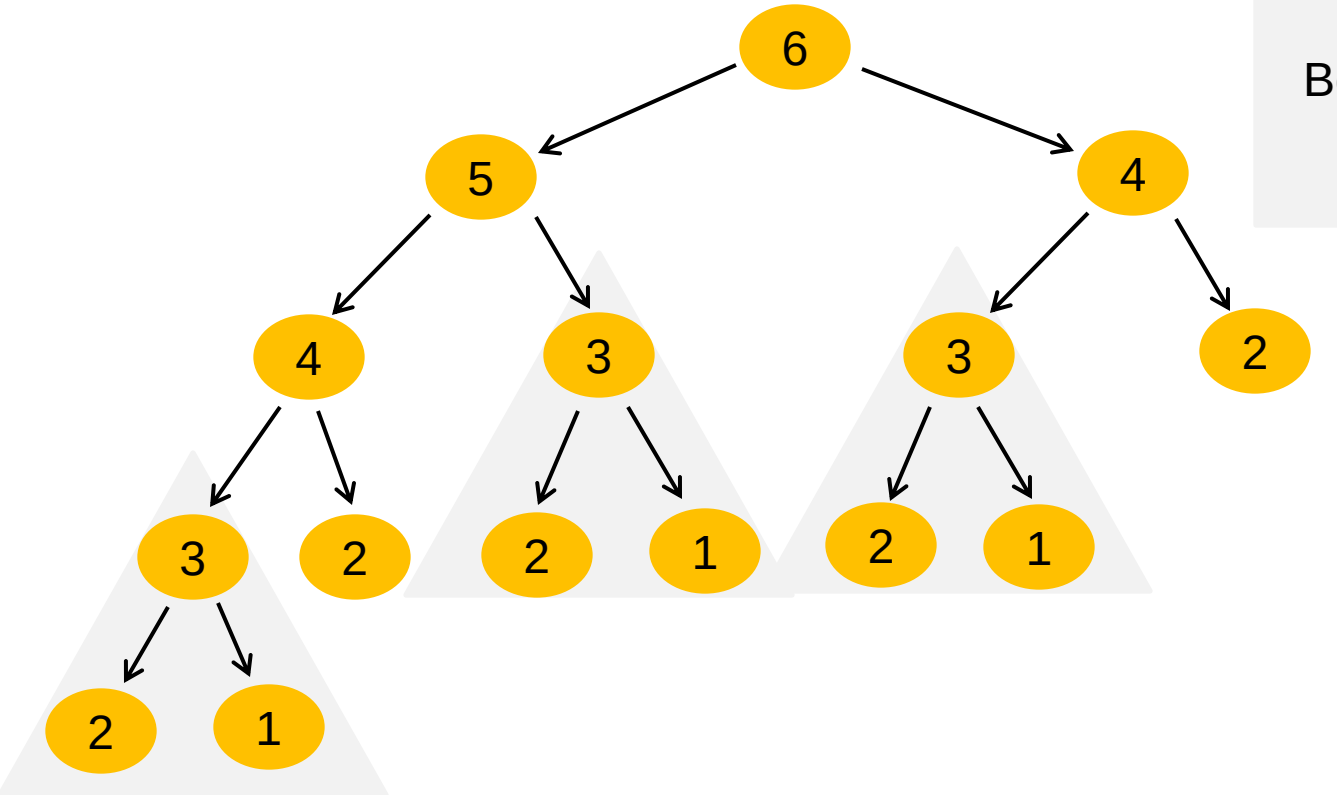


Es wird also mehrmals dieselbe Rechnung durchgeführt!

Dynamische Programmierung

Warum ist die Laufzeit so schlecht?

- Betrachten wir 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 ←1 **to** n **do**
3. F[i] ← 0
4. F[1] ← 1
5. F[2] ← 1
6. **return** FibMemo(n,F)

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]

2. F[n] ← FibMemo(n-1, F) + 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 ←1 **to** n **do**
3. F[i] ← 0
4. F[1] ← 1
5. F[2] ← 1
6. **return** FibMemo(n,F)

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Dynamische Programmierung

Fib2(n)

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

- Initialisiere Feld für Funktionswerte
- Setze Feldeinträge auf 0

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + 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 ←1 **to** n **do** ➢ Setze Feldeinträge auf 0
3. F[i] ← 0
4. F[1] ← 1 ➢ Setze F[1] auf korrekten Wert
5. F[2] ← 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] ← FibMemo(n-1, F) + 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 ←1 **to** n **do** ➢ Setze Feldeinträge auf 0
3. F[i] ← 0
4. F[1] ← 1 ➢ Setze F[1] auf korrekten Wert
5. F[2] ← 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] ← FibMemo(n-1, F) + 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 ←1 **to** n **do** ➢ Setze Feldeinträge auf 0
3. F[i] ← 0
4. F[1] ← 1 ➢ Setze F[1] auf korrekten Wert
5. F[2] ← 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] ← FibMemo(n-1, F) + 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 ←1 **to** n **do** ➢ Setze Feldeinträge auf 0
3. F[i] ← 0
4. F[1] ← 1 ➢ Setze F[1] auf korrekten Wert
5. F[2] ← 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] ← FibMemo(n-1, F) + 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 ←1 **to** n **do** ➢ Setze Feldeinträge auf 0
3. F[i] ← 0
4. F[1] ← 1 ➢ Setze F[1] auf korrekten Wert
5. F[2] ← 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] ← FibMemo(n-1, F) + FibMemo(n-2, F) ➢ Ansonsten berechne Wert ➢ und speichere Wert ab
3. **return** F[n] ➢ Gib Wert zurück

Dynamische Programmierung

Fib2(n)	Laufzeit:
1. Initialisiere Feld F[1..n]	O(n)
2. **for** i ←1 **to** n **do**	O(n)
3. F[i] ← 0	O(n)
4. F[1] ← 1	O(1)
5. F[2] ← 1	O(1)
6. **return** FibMemo(n,F)	1+T(n)

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

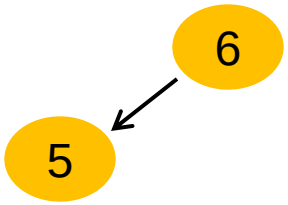
6

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

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

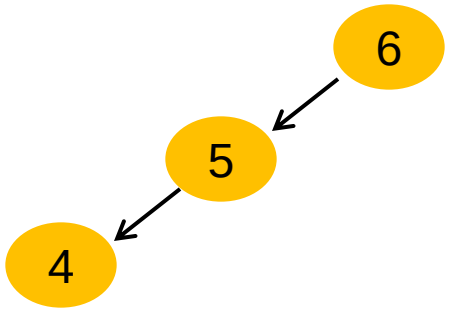


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

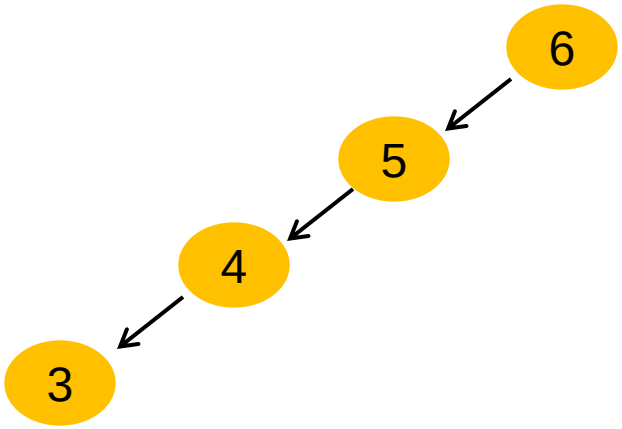


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

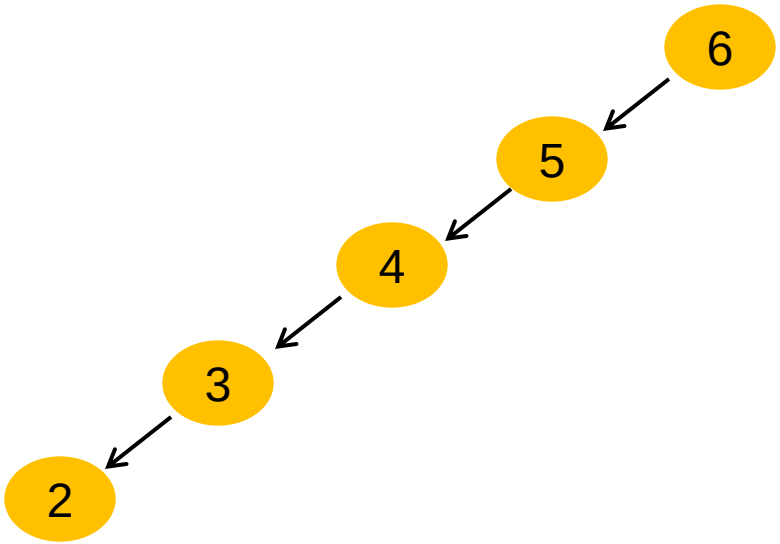


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

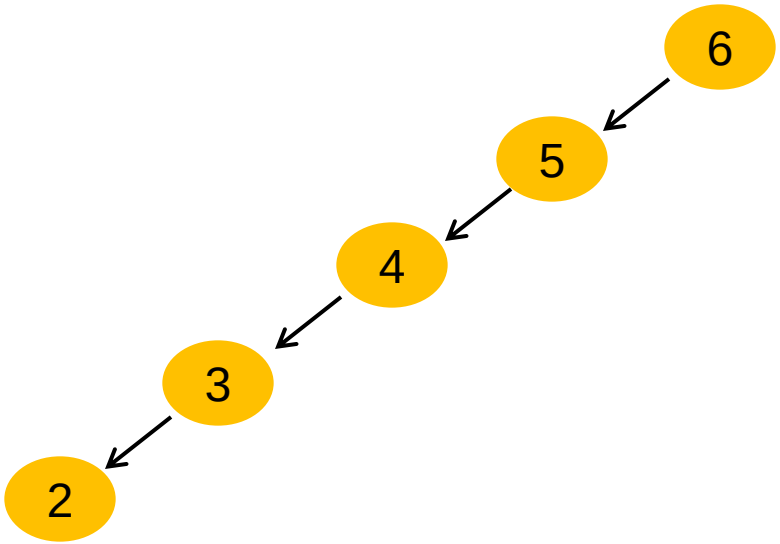


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

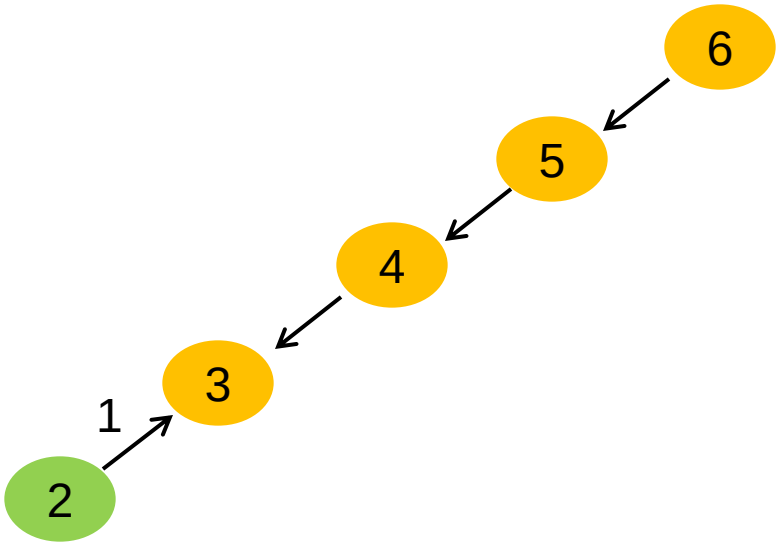


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

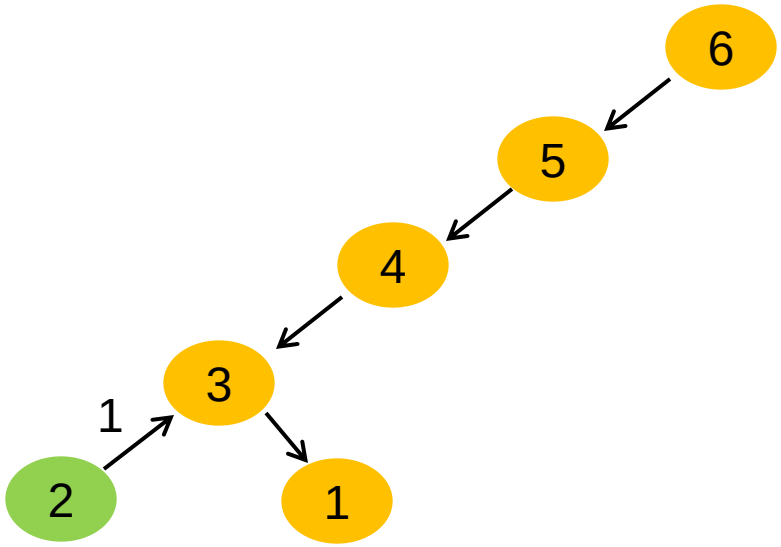


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

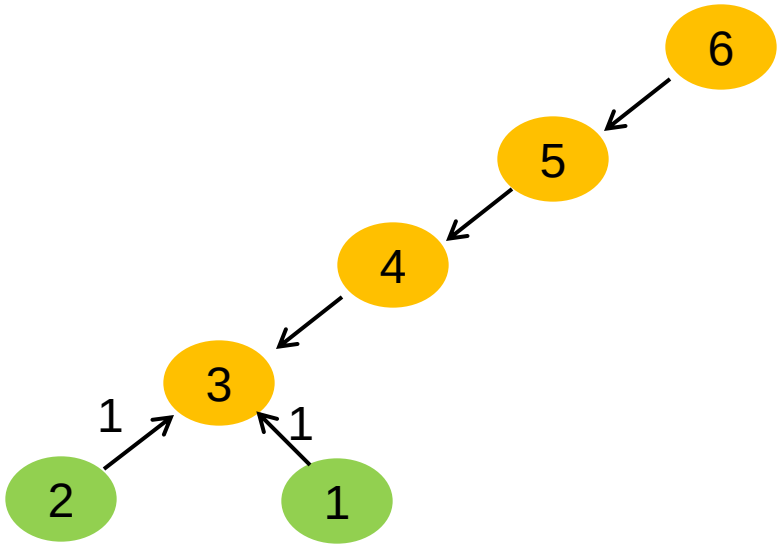


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

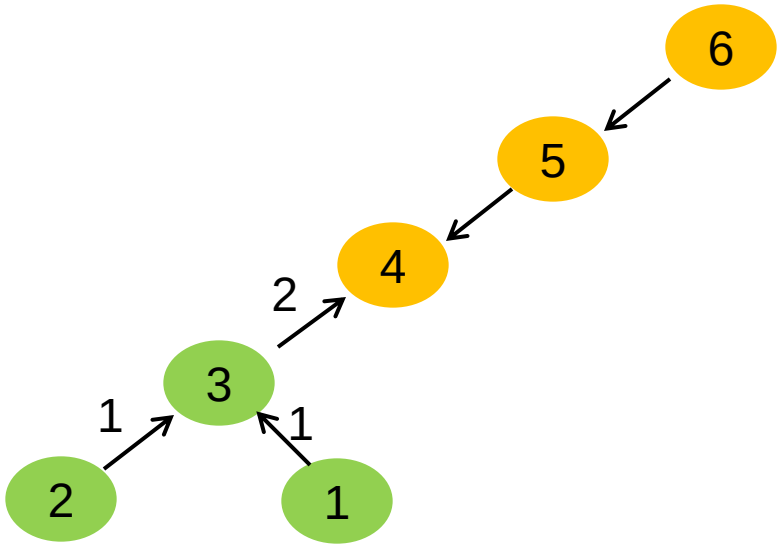


i:	1	2	3	4	5	6
F[i]:	1	1	0	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

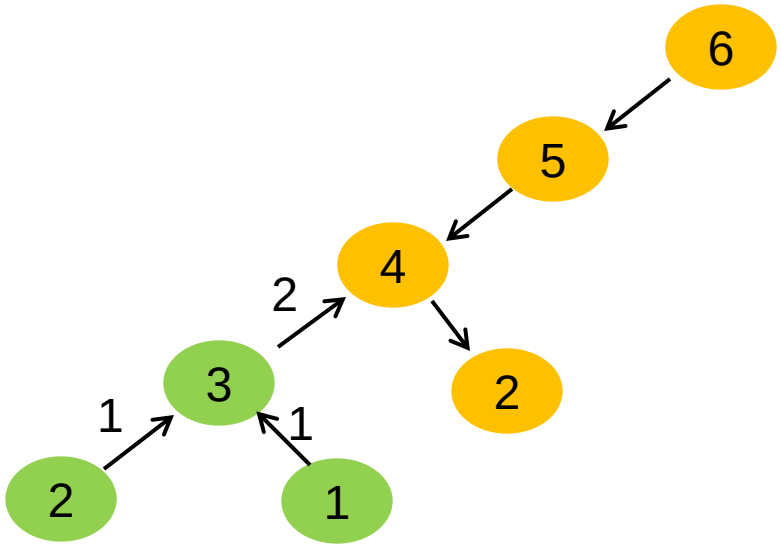


i:	1	2	3	4	5	6
F[i]:	1	1	2	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

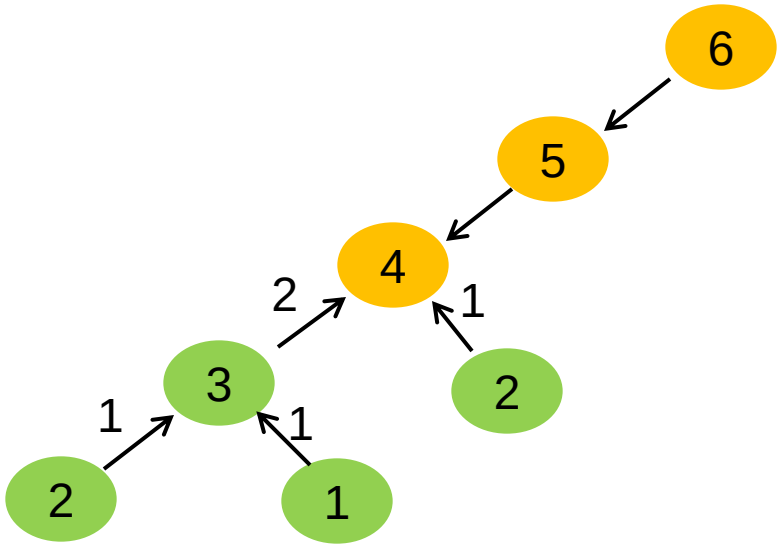


i:	1	2	3	4	5	6
F[i]:	1	1	2	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

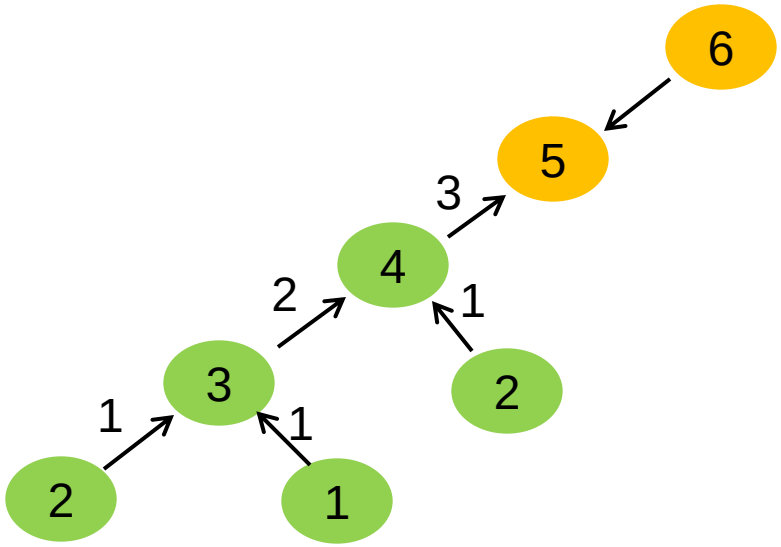


i:	1	2	3	4	5	6
F[i]:	1	1	2	0	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

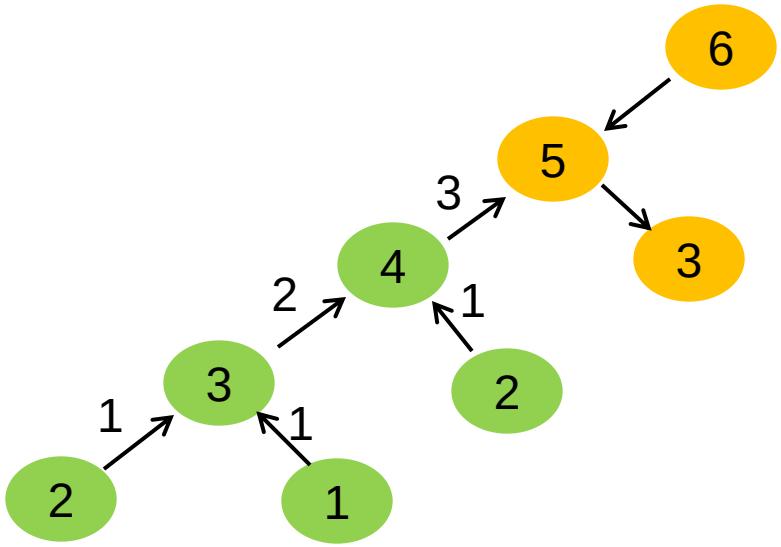


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

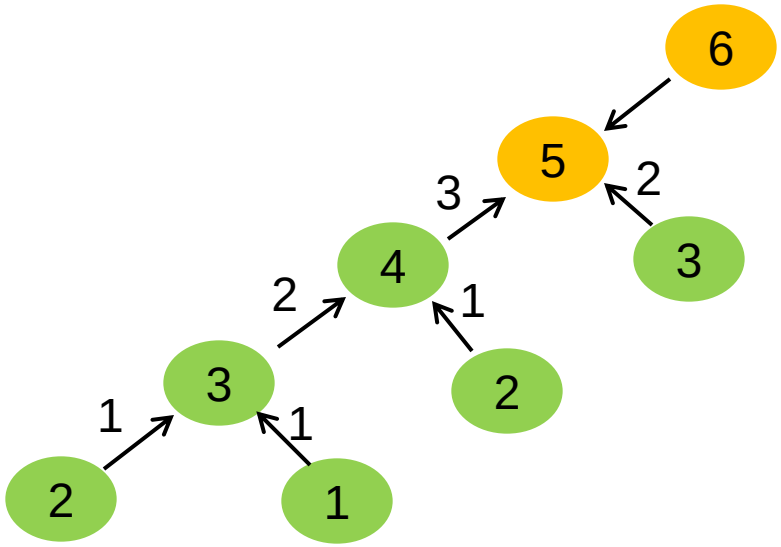


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

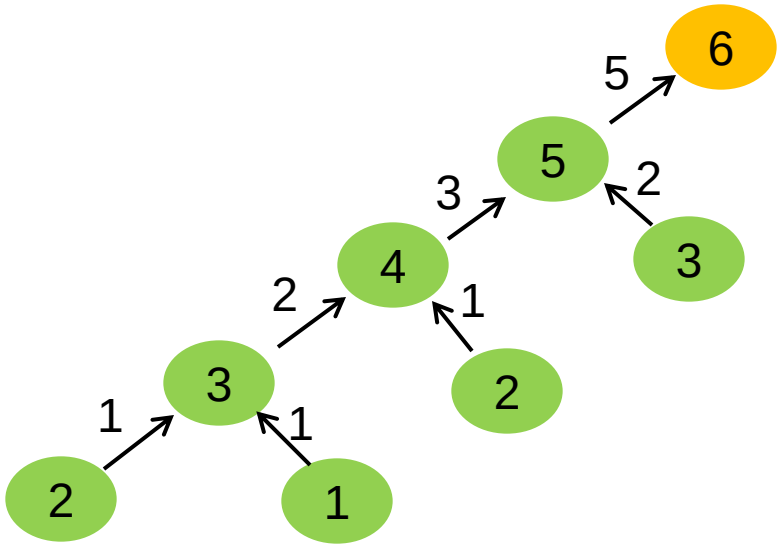


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	0	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

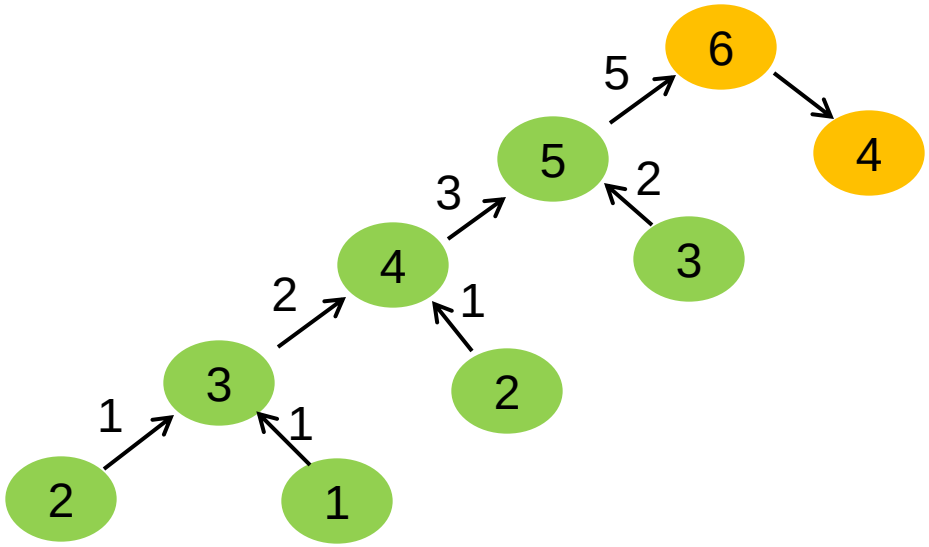


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	5	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

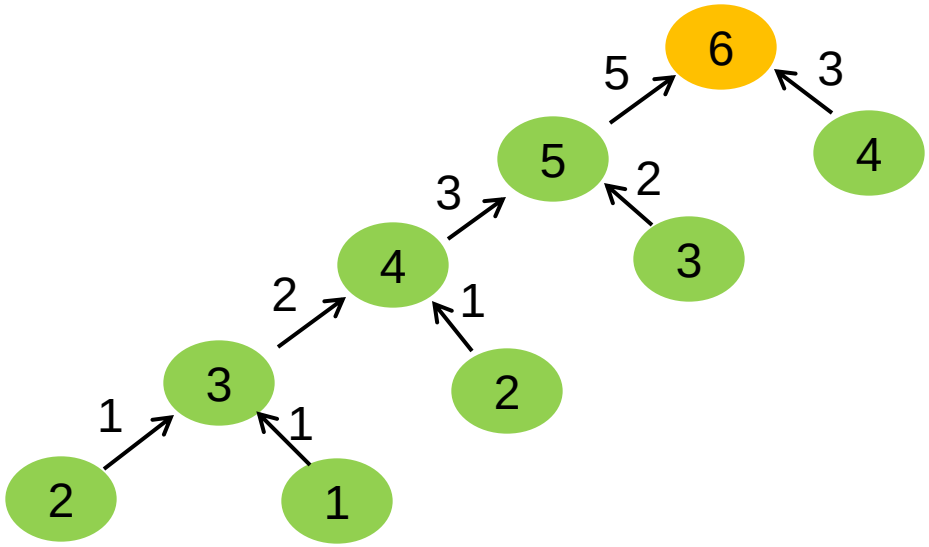


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	5	0

Dynamische Programmierung

Beispiel

- FibMemo(6,F)

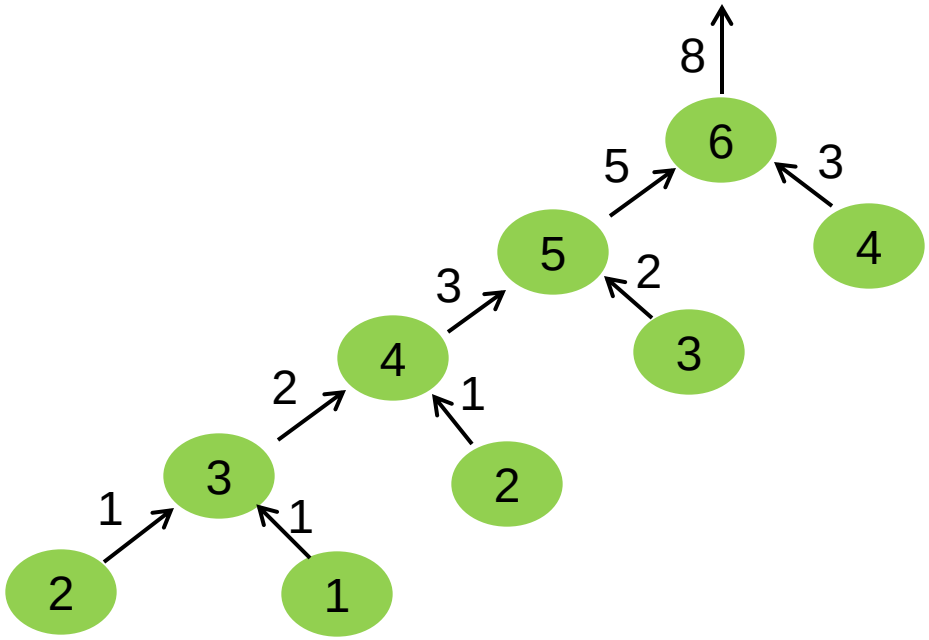


i:	1	2	3	4	5	6
F[i]:	1	1	2	3	5	0

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] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Laufzeit

- Jeder Aufruf FibMemo(m,F) generiert höchstens einmal die rekursiven Aufrufe FibMemo(m-1,F) und FibMemo(m-2,F)

Dynamische Programmierung

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Laufzeit

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

Dynamische Programmierung

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Laufzeit

- Jeder Aufruf FibMemo(m,F) generiert höchstens einmal die rekursiven Aufrufe FibMemo(m-1,F) und FibMemo(m-2,F)
- Also wird für jedes m<n die Funktion 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] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Laufzeit

- Jeder Aufruf FibMemo(m,F) generiert höchstens einmal die rekursiven Aufrufe FibMemo(m-1,F) und FibMemo(m-2,F)
- Also wird für jedes m<n die Funktion FibMemo(m,F) höchstens zweimal aufgerufen
- Ohne die Laufzeit für die rekursiven Aufrufe benötigt 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

FibMemo(n,F)

1. **if** F[n]>0 **then** return F[n]
2. F[n] ← FibMemo(n-1, F) + FibMemo(n-2, F)
3. **return** F[n]

Laufzeit

- Jeder Aufruf FibMemo(m,F) generiert höchstens einmal die rekursiven Aufrufe FibMemo(m-1,F) und FibMemo(m-2,F)
- Also wird für jedes m<n die Funktion FibMemo(m,F) höchstens zweimal aufgerufen
- Ohne die Laufzeit für die rekursiven Aufrufe benötigt 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] ← 1
3. F[2] ← 1
4. **for** i ← 3 **to** n **do**
5. F[i] ← 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] ← 1
3. F[2] ← 1
4. **for** i ← 3 **to** n **do**
5. F[i] ← F[i-1] + F[i-2]
6. **return** F[n]

} 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

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

Dynamische Programmierung

MaxMemo(i,Max,A)

1. **if** Max[i]≠-∞ **then** **return** Max[i]
2. Max[i] ← max{Max(i-1,Max,A) ,A[i]}
3. **return** Max[i]

MaxMemoInit(A)

1. Initialisiere Feld Max[1..n] ➢ Initialisiere Feld für Funktionswerte
2. **for** i ←1 **to** n **do**
3. Max[i] ← - ∞
4. Max[1] ← A[1]
5. **return** MaxMemo(n,Max,A)

Dynamische Programmierung

MaxDynamic(Max,A)

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

Dynamische Programmierung

MaxDynamic(Max,A)

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

Wie kann man nur aus Max[1..n] den Index eines größten Elements von A herausfinden?

- Sei j der größte Wert für den gilt Max[j] > Max[j-1]
  (mit der Konvention Max[0] = -∞)
- Dann ist j der gesuchte Index
- Beweis -> 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?

Dynamische Programmierung

Formulierung einer Rekursion/Induktion

- Wir nehmen an, dass die Zahlen aus M in einem Feld M[1..n] gegeben sind
- Wir bezeichnen mit M(i) die ersten i Zahlen dieses Feldes, also die Zahlen aus M[1..i]
- Wir bauen die Lösung Schritt für Schritt auf
- Wenn wir wissen, welche Zahlen sich als Summe einer Teilmenge von M(i-1) darstellen lassen, wie kommen wir auf die Zahlen, die sich als Summe einer Teilmenge von M(i) darstellen lassen?
- Sei G(i-1) die Menge der Zahlen, die sich als Summe einer Teilmenge von M(i-1) darstellen lassen
- Dann gilt $G(i) = G(i-1) \cup G(i-1) + M[i]$, wobei $G(i-1) + M[i]$ die Menge $\{x+M[i] : x \in G(i-1)\}$ bezeichnet
- Rekursionsanfang: $G(0) = \{0\}$

Dynamische Programmierung

Problem:

- Wie stelle ich G(i) im Rechner dar?
- Wir werden einfach für jedes G(i) ein Feld G[i,0…W] anlegen, wobei G[i,j]=1 ist, wenn sich j als Summe einer Teilmenge von G(i) darstellen läßt und 0 sonst
- W ist dabei die Summe aller Elemente aus M

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

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)

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≠0
  (Man kann keine Zahl ungleich 0 als Summe über die leere Menge darstellen)

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≠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≠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≥M[i] und G[i-1,j-M[i]] = 1)
- G[i,j] = 0, sonst

Dynamische Programmierung

PartitionMemoInit(M)

1. W←0
2. **for** i ← 1 **to** length[M] **do**
3. W ← W+ M[i]
4. **if** W ist ungerade **then return** 0
5. Initialisiere Feld G[0..length[M]][0..W]
6. **for** i ← 0 **to** length[M] **do**
7. G[i,0] ← 1
8. **for** j ← 1 **to** W **do**
9. G[i,j] ← -1
10. **for** j ← 1 to W **do**
11. G[0,j] ← 0
12. **if** PartitionMemo(M,G, n, W/2)=1 **then return** 1
13. **else return** 0

Dynamische Programmierung

PartitionMemoInit(M)

1. W←0
2. **for** i ← 1 **to** length[M] **do**
3. W ← W+ M[i]
4. **if** W ist ungerade **then return** 0
5. Initialisiere Feld G[0..length[M]][0..W]
6. **for** i ← 0 **to** length[M] **do**
7. G[i,0] ← 1
8. **for** j ← 1 **to** W **do**
9. G[i,j] ← -1
10. **for** j ← 1 to W **do**
11. G[0,j] ← 0
12. **if** PartitionMemo(M,G, n, W/2)=1 **then return** 1
13. **else return** 0

- Berechne W

Dynamische Programmierung

PartitionMemoInit(M)

1. W←0
2. **for** i ← 1 **to** length[M] **do**
3. W ← W+ M[i]
4. **if** W ist ungerade **then return** 0
5. Initialisiere Feld G[0..length[M]][0..W]
6. **for** i ← 0 **to** length[M] **do**
7. G[i,0] ← 1
8. **for** j ← 1 **to** W **do**
9. G[i,j] ← -1
10. **for** j ← 1 to W **do**
11. G[0,j] ← 0
12. **if** PartitionMemo(M,G, n, W/2)=1 **then return** 1
13. **else return** 0

- Berechne W
- Keine 2 gleich großen Teilmengen

Dynamische Programmierung

```
PartitionMemoInit(M)
1.  W←0
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0
5.  Initialisiere Feld G[0..length[M]][0..W]
6.  for i ← 0 to length[M] do
7.      G[i,0] ← 1
8.      for j ← 1 to W do
9.          G[i,j] ← -1
10. for j ← 1 to W do
11.     G[0,j] ← 0
12. if 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

PartitionMemoInit(M)

```
1.  W←0                                          ➢ Berechne W
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0              ➢ Keine 2 gleich großen Teilmengen
5.  Initialisiere Feld G[0..length[M]][0..W]     ➢ Initialisiere Feld G
6.  for i ← 0 to length[M] do                    ➢ Setze dabei G[i,0] auf 1 und
7.      G[i,0] ← 1                               ➢ Setze dabei G[0,j], j>0, auf 0
8.      for j ← 1 to W do                        ➢ G[i,j] = -1 heißt, Funktionswert noch
9.          G[i,j] ← -1                          ➢ nicht bekannt
10. for j ← 1 to W do
11.     G[0,j] ← 0
12. if PartitionMemo(M,G, n, W/2)=1 then return 1
13. else return 0
```

Dynamische Programmierung

```
PartitionMemoInit(M)
1.  W←0                                          ➢ Berechne W
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0              ➢ Keine 2 gleich großen Teilmengen
5.  Initialisiere Feld G[0..length[M]][0..W]     ➢ Initialisiere Feld G
6.  for i ← 0 to length[M] do                    ➢ Setze dabei G[i,0] auf 1 und
7.      G[i,0] ← 1                               ➢ Setze dabei G[0,j], j>0, auf 0
8.      for j ← 1 to W do                        ➢ G[i,j] = -1 heißt, Funktionswert noch
9.          G[i,j] ← -1                          ➢ nicht bekannt
10. for j ← 1 to W do
11.     G[0,j] ← 0
12. if PartitionMemo(M,G, n, W/2)=1 then return 1
13. else return 0
```

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

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

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

- Gibt es Teilmenge L von M[1..i] mit $\sum_{x \in L} x = j$?
- Wenn j ungültig gib false zurück

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

- Gibt es Teilmenge L von M[1..i] mit $\sum_{x \in L} x = j$?
- Wenn j ungültig gib false zurück
- G[i,j] bereits berechnet?

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

- Gibt es Teilmenge L von M[1..i] mit $\sum_{x \in L} x = j$?
- Wenn j ungültig gib false zurück
- G[i,j] bereits berechnet?
- Sonst, berechne G[i,j]
- nach Rekursion

Dynamische Programmierung

PartitionMemo(M,G, i, j)

1. **if** j<0 **return 0**
2. **if** G[i,j]≠-1 **then return** G[i,j]
3. **if** PartitionMemo(M,G,i-1,j)=1
   or PartitionMemo(M,G,i-1,j-M[i])=1 **then** G[i,j]=1
4. **else** G[i,j]=0
5. **return** G[i,j]

- Gibt es Teilmenge L von M[1..i] mit $\sum_{x \in L} x = j$?
- Wenn j ungültig gib false zurück
- G[i,j] bereits berechnet?
- Sonst, berechne G[i,j]
- nach Rekursion

Dynamische Programmierung

PartitionDynamicProg(M)

```
1.  W←0
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0
5.  Initialisiere Feld G[0..length[M]][0..W]
6.  for i ← 0 to length[M] do
7.      for j ← 0 to W/2 do
8.          if j=0 then G[i,j] ← 1
9.          else if i=0 then G[i,j] ← 0
10.                 else if G[i-1,j]=1 or (M[i]≤j und G[i-1,j-M[i]]=1) then G[i,j] ← 1
11.                         else G[i,j] ← 0
12. return G[length[M],W/2]
```

Dynamische Programmierung

```
PartitionDynamicProg(M)
1.  W←0
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0
5.  Initialisiere Feld G[0..length[M]][0..W]
6.  for i ← 0 to length[M] do
7.      for j ← 0 to W/2 do
8.          if j=0 then G[i,j] ← 1
9.          else if i=0 then G[i,j] ← 0
10.                 else if G[i-1,j]=1 or (M[i]≤j und G[i-1,j-M[i]]=1) then G[i,j] ← 1
11.                         else G[i,j] ← 0
12. return G[length[M],W/2]
```

- Berechnen von W und
- Initialisierung von G

Dynamische Programmierung

PartitionDynamicProg(M)

1. W←0
2. **for** i ← 1 **to** length[M] **do**
3. W ← W+ M[i]
4. **if** W ist ungerade **then return** 0
5. Initialisiere Feld G[0..length[M]][0..W]
6. **for** i ← 0 **to** length[M] **do**
7. **for** j ← 0 **to** W/2 **do**
8. **if** j=0 **then** G[i,j] ← 1
9. **else if** i=0 **then** G[i,j] ← 0
10. **else if** G[i-1,j]=1 **or** (M[i]≤j und G[i-1,j-M[i]]=1) **then** G[i,j] ← 1
11. **else** G[i,j] ← 0
12. **return** G[length[M],W/2]

- Berechnen von W und
- Initialisierung von G
- Berechnung
- von G

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

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

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

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 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											
2											
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[1]=1

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										
2											
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[1]=1

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									
2											
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[1]=1

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											
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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										
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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									
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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							
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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						
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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					
3											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[2]=4

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											
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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										
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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									
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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								
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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							
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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					
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

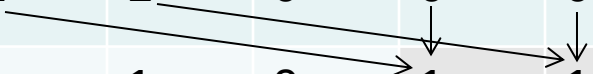
i \ j	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0				
4											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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		
4											
5											



Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[3]=3

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											
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[4]=5

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										
5											

Dynamische Programmierung

Beispiel: M = 1, 4, 3, 5, 7

M[4]=5

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											

Dynamische Programmierung

M[5]=7

Beispiel: M = 1, 4, 3, 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