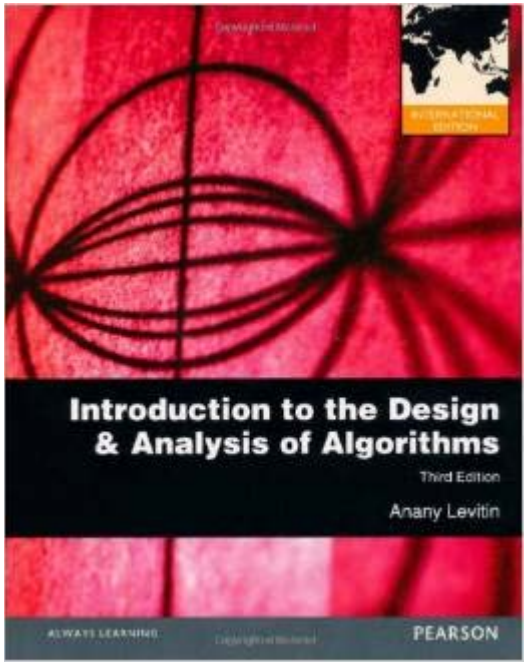
Algorithmen & Datenstrukturen

Dynamische Programmierung

Literaturangaben

Diese Lerneinheit basiert größtenteils auf dem Buch „The Design and Analysis of Algorithms“ von Anany Levitin.

In dieser Einheit behandelte Kapitel:
8 Dynamic Programming
8.1 Three Basic Examples
8.2 The Knapsack Problem (ohne Memory Functions)



Dynamische Programmierung: Übersicht

- Design-Technik für **rekursiv definierte Algorithmen** mit **überlappenden Teilproblemen**
- Entwickelt in den 50er Jahren vom amerikanischen Mathematiker **Richard Bellman**
- Ursprüngliches Anwendungsgebiet: **Optimierungsprobleme**
- „Programmierung“ wird hier in der Bedeutung „**Planung**“ verwendet

Dynamische Programmierung: Grundprinzip

- Definiere **Rekursionsgleichung** für das Problem
  - Größere Problemexemplare werden mit Hilfe der Lösungen **überlappender** kleinerer Exemplare gelöst
- Löse die kleineren Problemexemplare nur **ein einziges Mal**
- Speichere Lösungen in einer **Tabelle**
- Ermittle die **Lösung des Ausgangsproblems** mit Hilfe der Tabelle

Optimale Gesamtlösung aus optimalen Teillösungen ermittelbar

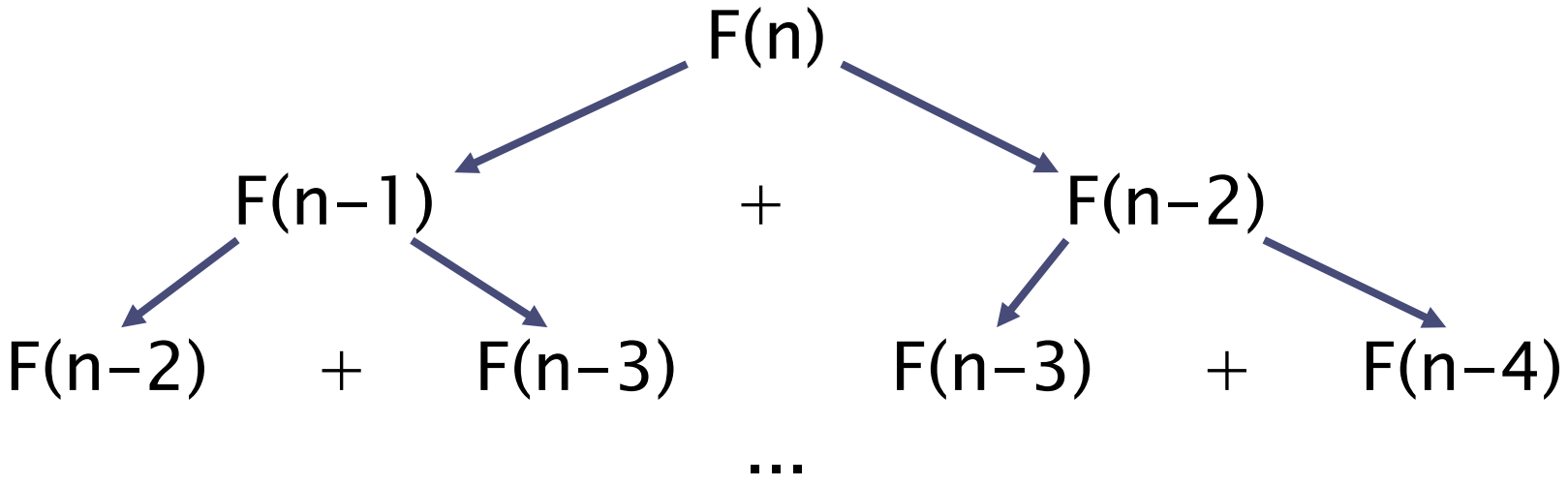
DP nur bei Überlappung sinnvoll

Vermeide redundante Arbeit

Space/Time-Tradeoff

Beispiel 1: Fibonacci-Zahlen (I)

- Bekannte Definition:
  - F(n) = F(n-1) + F(n-2)
  - F(0) = 0
  - F(1) = 1
- **Rekursive Top-Down-Berechnung**



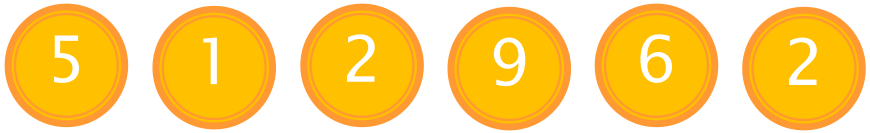
Beispiel 1: Fibonacci-Zahlen (II)

- **Iterative Bottom-Up-Berechnung** mit **Speicherung** der Ergebnisse
  - $F(0) = 0$
  - $F(1) = 1$
  - $F(2) = 0 + 1 = 1$
  - …
  - $F(n-2) = \ldots$
  - $F(n-1) = \ldots$
  - $F(n) = F(n-1) + F(n-2)$

0	1	1	…	F(n-2)	F(n-1)	F(n)

- Zeiteffizienz?
- Speicherplatzeffizienz?

Beispiel 2: Coin-row problem

- **Gegeben:**
  Eine Reihe von n Münzen mit positiven, ganzzahligen, aber nicht notwendigerweise unterschiedlichen Werten $c_1, c_2, \ldots, c_n$
- **Ziel:**
  Wähle **möglichst wertvolle Teilmenge** von Münzen aus
- **Randbedingung:**
  Es dürfen **nicht zwei (ursprünglich) benachbarte** Münzen gewählt werden
- **Beispiel:** 

Was ist die beste Auswahl?

DP-Lösung des Coin-row problems (I)

- Definiere: F(n) ist das Maximum aus der Reihe der ersten n Münzen
- Betrachte Auswahlen ohne und mit der letzten Münze:
  - Wertvollste Auswahl, die Münze n nicht enthält
  - Wertvollste Auswahl, die Münze n enthält
- Rekursionsgleichung:
  $F(n) = \max\{F(n-1), c_n + F(n-2)\}$ für $n > 1$
  $F(0) = 0$, $F(1) = c_1$

DP-Lösung des Coin-row Problems (II)

$F(n) = \max\{F(n-1), c_n + F(n-2)\}$ für $n > 1$
$F(0) = 0, F(1) = c_1$

index	0	1	2	3	4	5	6
coins	–	5	1	2	9	6	2
F( )							

- Welche Münzen werden gewählt?
- Zeit-Effizienz?
- Speicherplatz-Effizienz?

Ergebnisse für alle kürzeren Münzreihen wurden ebenfalls ermittelt

Beispiel 3: Münzwechselproblem

- **Gegeben:**
  Münzen (in unbegrenzter Anzahl) mit den ganzzahligen Nennwerten $d_1 < d_2 < \ldots < d_m$ und $d_1 = 1$
- **Ziel:**
  Wechsele einen nicht-negativen, ganzzahligen Betrag n in eine äquivalente Menge Münzen
- **Randbedingung:**
  Es sollen möglichst wenige Münzen verwendet werden
- **Beispiel:**
  - Nennwerte der Münzen: 1, 3, 4
  - Zu wechselnder Betrag: 6

Minimale Anzahl von Münzen?

DP-Lösung des Münzwechselproblems (I)

- Definiere: F(n) ist die minimale Anzahl von Münzen, die in der Summe n ergeben
- Betrachte Münzwechsel mit einer Münze weniger, also für die Werte $n - d_j$ für $j = 1, 2, \ldots, m$ so dass $n \geq d_j$
- Rekursionsgleichung:

$$F(n) = \min_{j\,:\,n \geq d_j} \{F(n-d_j)\} + 1 \text{ für } n > 0$$

$$F(0) = 0$$

DP-Lösung des Münzwechselproblems (II)

$$F(n) = \min_{j : n \geq d_j} \{F(n-d_j)\} + 1 \text{ für } n > 0$$

$$F(0) = 0$$

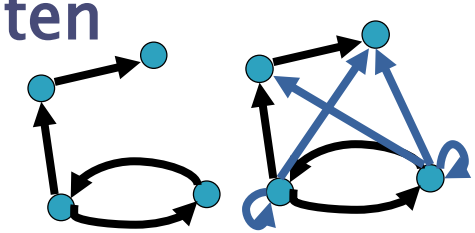
n	0	1	2	3	4	5	6
F( )							
Münze							

- Welche Münzen werden gewählt?
- Zeit-Effizienz?
- Speicherplatz-Effizienz?

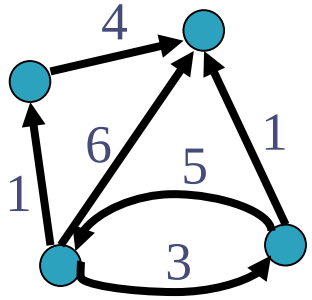
Ergebnisse für alle kleineren Beträge wurden ebenfalls ermittelt

Dynamische Programmierung: Weitere Beispiele

- Berechnung von **Binomialkoeffizienten**
- **Warshall-Algorithmus** (Berechnung der **transitiven Hülle**)



- Konstruktion des **optimalen binären Suchbaums**
- **Floyd-Algorithmus** (Berechnung der **kürzesten Pfade zwischen allen Knoten**)



- Einige **Varianten** schwieriger Optimierungsprobleme
  - **Problem des Handelsreisenden**
  - **Rucksackproblem**

Rucksackproblem mit Dynamischer Programmierung (I)

Rekapitulation des Problems:

Gegeben:

- n Gegenstände mit
  - **ganzzahligen** Gewichten: w_1 w_2 … w_n
  - Wertangaben: v_1 v_2 … v_n
  - Rucksack mit Kapazität W
- **Aufgabe**: Finde wertvollste Teilmenge, die noch in den Rucksack passt

Achtung: Beschränkung auf ganzzahlige Gewichte vereinfacht Problem erheblich!

Rucksackproblem mit Dynamischer Programmierung (II)

- Betrachte **kleinere Problemexemplare** definiert für die ersten **i Gegenstände** und die **Kapazität j** ($j \leq W$)
- Sei **V[i,j]** der **optimale Wert** für solch ein Exemplar
- Dann gilt

$$V[i,j] = \begin{cases} V[i-1,j] & \text{falls } j - w_i < 0 \\ \max\{V[i-1,j],\ v_i + V[i-1,j-w_i]\} & \text{falls } j - w_i \geq 0 \end{cases}$$

- Startbedingungen:
  - $V[0,j] = 0$
  - $V[i,0] = 0$

Rucksackproblem mit DP: Beispiel

- Rucksack mit Kapazität $W = 5$ kg

Gegenstand	Gewicht	Wert
1	2 kg	12 €
2	1 kg	10 €
3	3 kg	20 €
4	2 kg	15 €

Welche Gegenstände wurden eingepackt?

		Kapazität j					
		0	1	2	3	4	5
	0						
Gegenstand 1	1						
Gegenstand 1–2	2						
Gegenstand 1–3	3						
Gegenstand 1–4	4						?