

# Algorithmen und Datenstrukturen

Wintersemester 2018/19

22. Vorlesung

## Dynamisches Programmieren

# Themen für den 3. Kurztest (Do, 24.01.19)

- Rot-Schwarz-Bäume (R-S-Eigenschaften, Höhe)
- Augmentieren von Datenstrukturen
- Amortisierte Analyse
- Nächstes Paar (Teile und Herrsche)
- Graphen und Breitensuche

## Anmeldung

Bis **HEUTE**, Di, 22.1., 13:00 Uhr.

# Entwurfstechniken

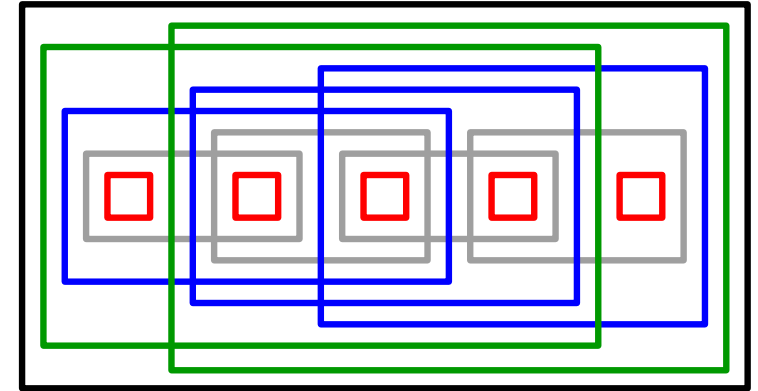
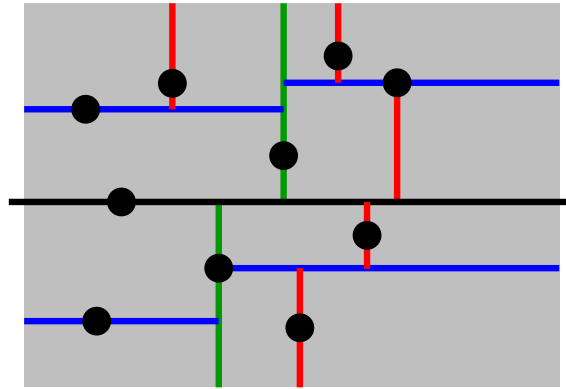
- Inkrementell
- Rekursiv
- Teile und Herrsche
- Randomisiert



- Dynamisches Programmieren

[meint hier das Arbeiten mit einer Tabelle,  
nicht das Schreiben eines Computerprogramms.]

## Vergleich



## Teile und Herrsche

- zerlegt Instanz rekursiv in *disjunkte* Teilinstanzen
- top-down
- eher für Entscheidungs- oder Berechnungsprobleme

## Dynamisches Programmieren

- zerlegt Instanz in *überlappende* Teilinstanzen, d.h. Teilinstanzen haben z.T. dieselben Teilteilinstanzen. Lösungen von Teilinstanzen werden zwischengespeichert, nicht neu berechnet.
- meist bottom-up
- meist für Optimierungsprobleme

# Fahrplan

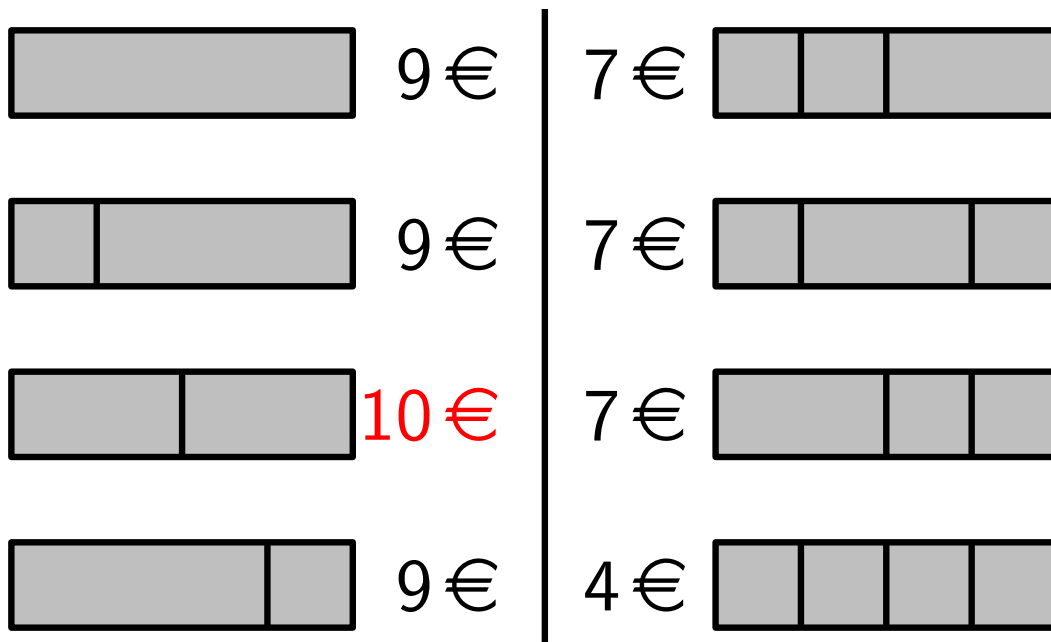
1. Struktur einer optimalen Lösung charakterisieren
2. Wert einer optimalen Lösung rekursiv definieren
3. Wert einer optimalen Lösung berechnen (meist bottom-up)
- [4. Optimale Lösung aus berechneter Information konstruieren]

# Ein Beispiel

## Zerlegungsproblem

Wir haben einen Stab der Länge  $n$  und kennen die Preise  $p_1, p_2, \dots, p_n$  für Stäbe der Längen  $1, 2, \dots, n$ .

Durch welche Zerlegung unseres Stabs können wir unseren **Ertrag maximieren?**



Länge $i$	1	2	3	4
Preis $p_i$ [in €]	1	5	8	9

# Ein erster Versuch

Wir haben einen Stab der Länge  $n$  und kennen die Preise  $p_1, p_2, \dots, p_n$  für Stäbe der Längen  $1, 2, \dots, n$ .

*Beispiel:  $n = 4$*

Länge $i$ [in $m$ ]	1	2	3	4
Preis $p_i$ [in €]	1	5	8	9
Quotient [€/m]	1	$2\frac{1}{2}$	$2\frac{2}{3}$	$2\frac{1}{4}$

Welche Stabzerlegung maximiert den Ertrag?

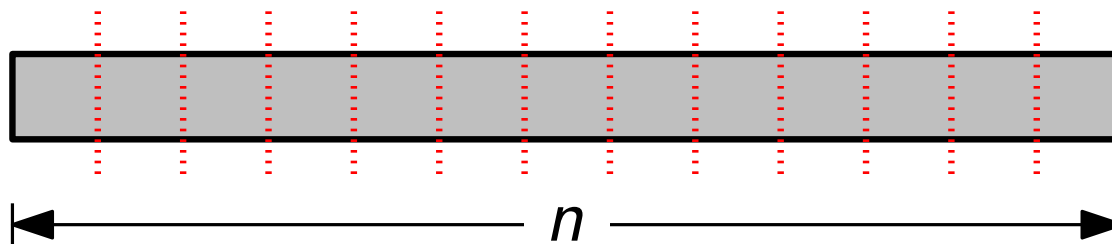
## Ein ADSler schlägt vor:

- Berechne für  $i = 1, \dots, n$  den Preis pro Meter  $q_i = p_i/i$ .
- Zerlege Stab in möglichst viele Stücke der Länge  $i$  mit  $q_i$  max.
- Streiche alle Stablängen  $\geq i$  aus der Tabelle und wiederhole den Prozess mit dem Stabrest (falls  $> 0$ ).

Funktioniert dieser Greedy-Algorithmus?  Ja? Beweisen!  
Nein? Gegenbeispiel!

# Rohe Gewalt

**Frage:** Wie viele Möglichkeiten gibt es einen Stab der Länge  $n$  zu zerlegen?



**Antw.:** Können  $n - 1$  mal entscheiden: *schneiden* oder *nicht*.  
 $\Rightarrow 2^{n-1}$  verschiedene\* Zerlegungen

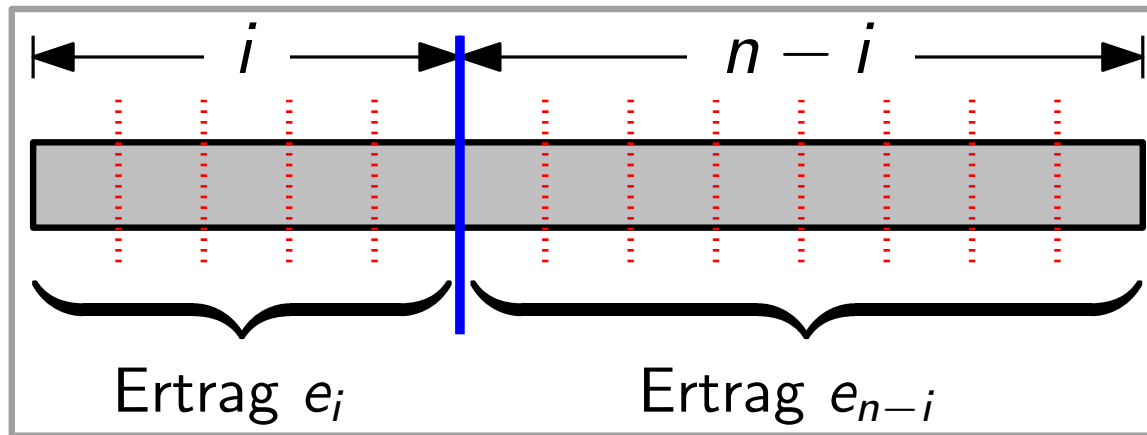
Also können wir es uns nicht leisten alle Zerlegungen durchzugehen und für jede ihren Ertrag zu berechnen.

\* ) Genauer: die gesuchte Anzahl ist die Anzahl  $p(n)$  der *Partitionen* der Zahl  $n$ , die angibt, auf wie viele Arten man  $n$  als Summe von natürlichen Zahlen schreiben kann. Es gilt  $p(n) \approx e^{\pi \sqrt{2n/3}} / (4n\sqrt{3}) \in \Theta^*((13,00195\dots)\sqrt{n})$ .



## 1. Struktur einer optimalen Lösung charakterisieren

**Def.** Für  $i = 1, \dots, n$   
 sei  $e_i$  der maximale Ertrag für einen Stab der Länge  $i$ .



Phänomen  
 der *optimalen  
 Teilstruktur!*

**Beob.** Ein Schnitt zerlegt das Problem in *unabh.* Teilprobleme.

## 2. Wert einer optimalen Lösung rekursiv definieren

Wissen nicht, welcher Schnitt in einer opt. Lösung vorkommt.

Also probieren wir einfach alle möglichen Schnitte aus:

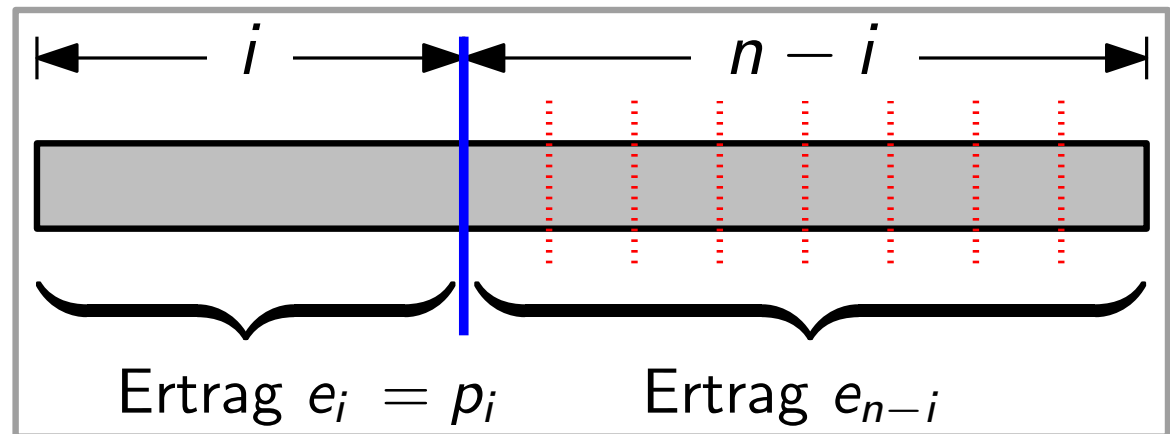
$$e_n = \max \{ p_n, e_1 + e_{n-1}, e_2 + e_{n-2}, \dots, e_{n-1} + e_1 \}$$

## 2. Wert einer optimalen Lösung rekursiv definieren

$$e_n = \max\{ p_n, e_1 + e_{n-1}, e_2 + e_{n-2}, \dots, e_{n-1} + e_1 \}$$

*Kleine Verbesserung:*

Verbiете weitere Schnitte im linken Teilstück!



*Also gilt:*

$$e_n = \max\{ p_n, p_1 + e_{n-1}, p_2 + e_{n-2}, \dots, p_{n-1} + e_1 \}$$

$$= \max_{1 \leq i \leq n} \{ p_i + e_{n-i} \}, \quad \text{wobei } e_0 := 0.$$

**Vorteil:** Wert einer opt. Lösung ist Summe aus einer Zahl der Eingabe und *einem* Wert einer opt. Teillösung.

### 3. Wert einer optimalen Lösung berechnen: *top-down*

Wir wissen:  $e_n = \max_{1 \leq i \leq n} \{p_i + e_{n-i}\}$ , wobei  $e_0 := 0$ .

```

StangenZerlegung(int[] p, int n = p.length)
  if n == 0 then return 0
  q = -∞
  for i = 1 to n do
    q = max{q, p[i] + StangenZerlegung(p, n - i)}
  return q

```

**Laufzeit:** Sei  $A(n)$  die Gesamtzahl von Aufrufen von  $\text{StangenZerlegung}(p, \cdot)$  beim Ausführen von  $\text{StangenZerlegung}(p, n)$

$$\Rightarrow A(0) = 1$$

$$\text{und } A(n) = 1 + \sum_{i=1}^n A(n-i) = 1 + \sum_{j=0}^{n-1} A(j) \stackrel{?}{=} 2^n$$

*Beweis?!*



### 3. Wert einer optimalen Lösung berechnen: *mit Tabelle*

**Zeit-Speicher-Tausch** (engl. *time-memory trade-off*)

```
MemoStangenZerlegung(int[] p, int n = p.length)
```

```
  e = new int[0..n]
```

```
  e[0] = 0
```

```
  for i = 1 to n do
```

```
    e[i] =  $-\infty$ 
```

```
  return HauptStangenZerlegung(p, n, e)
```

```
HauptStangenZerlegung(int[] p, int n, int[] e)
```

```
  if  $e[n] \geq 0$  or  $n == 0$  then return e[n]
```

```
  q =  $-\infty$ 
```

```
  for i = 1 to n do
```

```
    q = max{q, p[i] + HauptStangenZerlegung(p, n - i, e)}
```

```
  e[n] = q; return q
```

**Laufzeit?**

– Wie letzte Folie?

– Asymptotisch schneller?

### 3. Wert einer optimalen Lösung berechnen: *bottom-up*

BottomUpStangenZerlegung(int[]  $p$ , int  $n$ )

$e = \text{new int}[0..n]$

$e[0] = 0$

**for**  $j = 1$  **to**  $n$  **do**

$q = -\infty$

**for**  $i = 1$  **to**  $j$  **do**

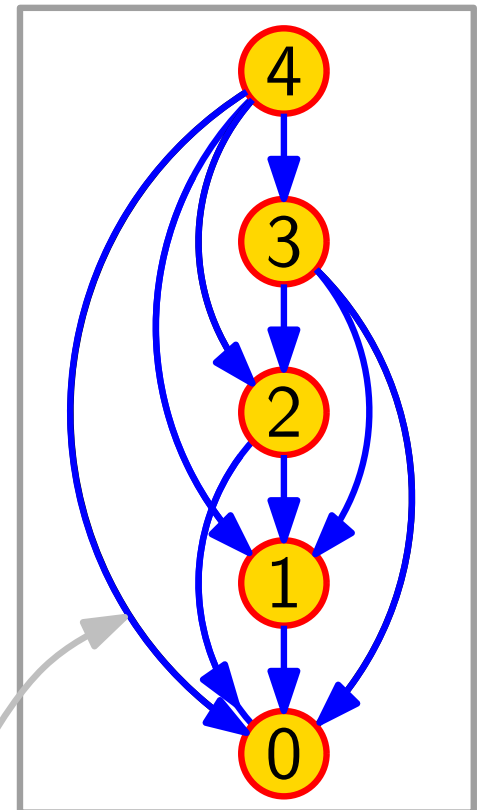
$q = \max\{q, p[i] + e[j - i]\}$

$e[j] = q$

**return**  $q$

Neu: kein  
rekursiver  
Aufruf!

Kante  $(j, i)$  bedeutet:  
Teilinstanz  $j$  benützt Wert einer  
opt. Lösung von Teilinstanz  $i$ .



Graph der  
Teilinstanzen

**Beob.** Größe des Graphen (d.h. in erster Linie Anz. Kanten) ist proportional zur Laufzeit des DP (Anz. Additionen).

**Satz.** BottUpSZerl() und MemoSZerl() laufen in  $O(n^2)$  Zeit.

## 4. Optimale Lösung aus berechneter Info. konstruieren

```
GibZerlegungAus(int[] p, int n)
  ℓ = new int[0..n]; e = new int[0..n]
  ErweiterteBottomUpZerlegung(p, e, ℓ, n)
  while n > 0 do           // gib wiederholt Länge des 1. Teilstücks aus
    _ print ℓ[n]; n = n - ℓ[n]
```

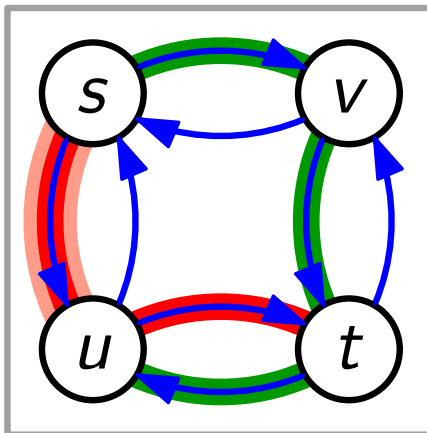
```
ErweiterteBottomUpZerlegung(int[] p, int[] e, int[] ℓ, int n)
  e[0] = 0
  for j = 1 to n do
    q = -∞
    for i = 1 to j do
      if q < p[i] + e[j - i] then
        q = p[i] + e[j - i]
        ℓ[j] = i           // merke Länge des 1. Teilstücks
    e[j] = q
```

# Längste Wege

Gegeben: ungewichteter gerichteter Graph  $G = (V, E)$  mit  $s, t \in V$ ,  $s \neq t$  und  $t$  von  $s$  erreichbar.

Gesucht: ein längster einfacher  $s$ - $t$ -Weg,

d.h. eine Folge  $\langle s = v_0, v_1, \dots, v_k = t \rangle$  mit  $v_0 v_1, \dots, v_{k-1} v_k \in E$ ,  $v_i \neq v_j$  (für  $i \neq j$ ) und  $k$  maximal.




$\langle s, u, t \rangle$  ist ein längster einfacher  $s$ - $t$ -Weg.

Aber:

$\langle s, u \rangle$  ist *kein* längster einfacher  $s$ - $u$ -Weg;

$\langle s, v, t, u \rangle$  ist ein längster einfacher  $s$ - $u$ -Weg!

## Fahrplan

1. Struktur einer optimalen Lösung charakterisieren 
2. Wert einer optimalen Lösung rekursiv definieren
3. Wert einer optimalen Lösung berechnen (meist bottom-up)

<sup>\*</sup>) Es ist NP-schwer für  $(G, s, t, k)$  zu entscheiden, ob  $G$  einen einfachen  $s$ - $t$ -Weg der Länge  $k$  enthält. (Vgl. Hamilton-Weg!)

# Längste Wege in azyklischen Graphen

Gegeben: gewichteter gerichteter *kreisfreier* Graph  $G = (V, E; w)$  mit  $s, t \in V$ ,  $s \neq t$  und  $t$  von  $s$  erreichbar.

Gesucht: ein längster  $s$ - $t$ -Weg.

**Beob<sub>1</sub>** In kreisfreien Graphen sind alle Wege einfach.

**Beob<sub>2</sub>** Dieses Problem hat optimale Teilstruktur, denn:  
Ein längster  $s$ - $t$ -Weg  $\pi$  gehe durch  $u$ , d.h.

$$\pi = s \xrightarrow{\pi_{su}} u \xrightarrow{\pi_{ut}} t.$$

Dann gilt:

$\pi_{su}$  ist längster  $s$ - $u$ -Weg;  $\pi_{ut}$  ist längster  $u$ - $t$ -Weg –  
sonst wäre  $\pi$  kein längster  $s$ - $t$ -Weg.

Außerdem gilt  $V(\pi_{su}) \cap V(\pi_{ut}) = \{u\}$ ;  
sonst gäbe es einen Kreis!



# Algorithmus nach Fahrplan

1. Struktur einer optimalen Lösung charakterisieren ✓

2. Wert einer optimalen Lösung rekursiv definieren

$$d_v = \max_{uv \in E} d_u + w(u, v) \quad // \text{ Länge eines längsten } s\text{-}v\text{-Wegs}$$

3. Wert einer optimalen Lösung berechnen (hier bottom-up)

–  $G$  topologisch sortieren

–  $d$ -Werte initialisieren:  $d_s = 0$  und  $d_v = -\infty$  für alle  $v \neq s$

– for-Schleife durch Knoten v.l.n.r.  $d$ -Werte berechnen

*so!*

**Übrigens:** *kürzeste* Wege in kreisfreien Graphen gehen analog (mit min statt max und  $+\infty$  statt  $-\infty$ )

# Und jetzt?

Im Buch [CLRS] werden weitere, praxisrelevante Probleme mit dynamischem Programmieren gelöst:

- Ketten von Matrixmultiplikationen
- Längste gemeinsame Teilfolge (in Zeichenketten)
- Optimale binäre Suchbäume

*Lesen Sie Kapitel 15.2–5 !!!*