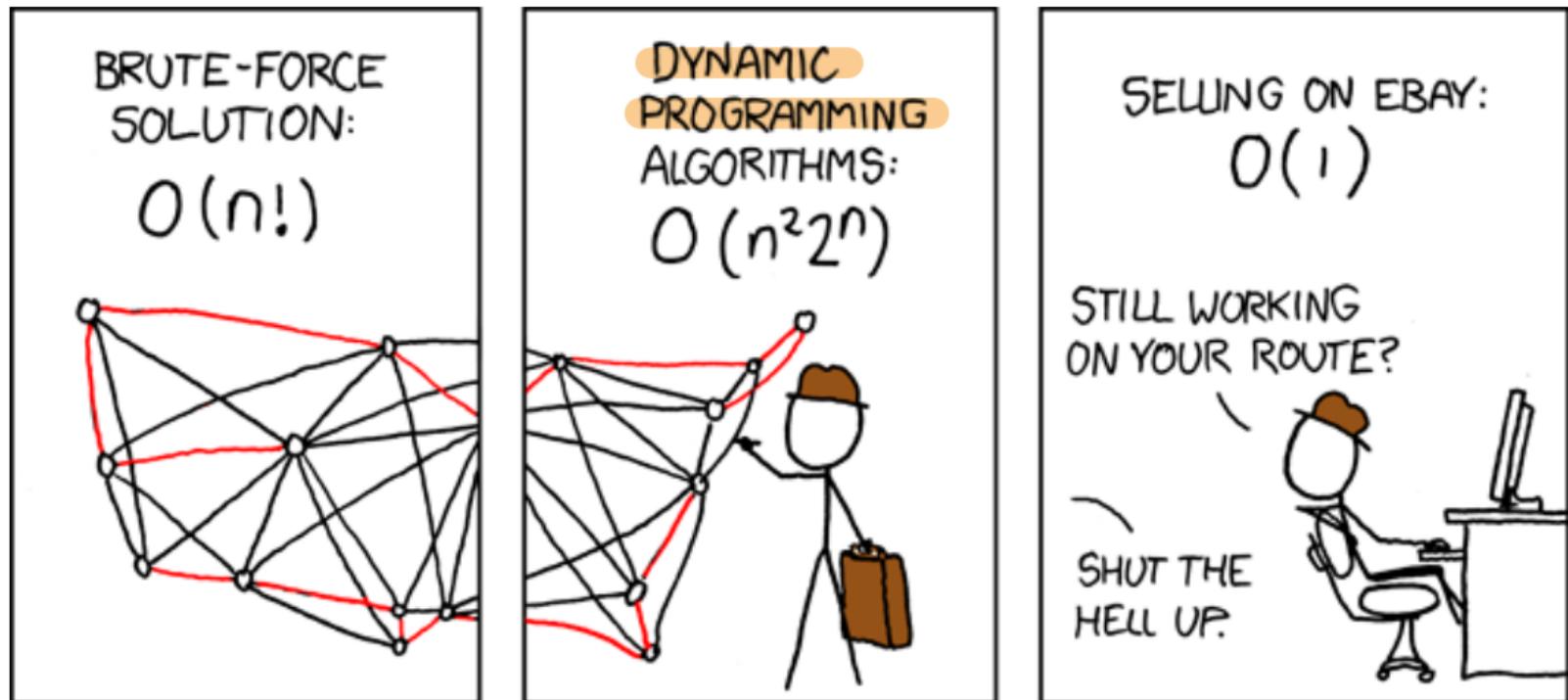




# D&A - Übungsstunde 8

*Diese Folien basieren auf denjenigen der Vorlesung, wurden aber durch den Assistenten Adel Gavranović adaptiert und erweitert*

# Comic der Woche



► xkcd

## Heutiges Programm

Intro

Follow-up

Feedback zu `[code]expert`

Quadtrees

Dynamische Programmierung

Code-Beispiele

Tipps zu `[code]expert`

Outro



[n.ethz.ch/~agavranovic](https://n.ethz.ch/~agavranovic)

▶ [Link zum Material für die Übungsstunden](#)

▶ [Webseite des Assistenten](#)

▶ [Mail an Assistenten](#)

# 1. Intro

---

Willkommen zurück!

## 2. Follow-up

---

# Follow-up aus vorherigen Übungsstunden

# Follow-up aus vorherigen Übungsstunden

- "Vorherige" Übungsstunde (07) ist ebenfalls auf der Website
- nicht sehr relevant

### 3. Feedback zu `[code]`expert

---

# Allgemeines zu `[code]`expert

# Allgemeines zu `[code]`expert

- Ich merke, wenn ihr die Musterlösung (leicht angepasst) kopiert

# Allgemeines zu [code]expert

- Ich merke, wenn ihr die Musterlösung (leicht angepasst) kopiert
  - Task "Task "Double Ended Queue""
  - Task "Finding a Sub-Array (Rabin-Karp)"
  - Task "Map/Filter/Reduce"

# Allgemeines zu [code]expert

- Ich merke, wenn ihr die Musterlösung (leicht angepasst) kopiert
  - Task "Task "Double Ended Queue""
  - Task "Finding a Sub-Array (Rabin-Karp)"
  - Task "Map/Filter/Reduce"
- (Hört auf damit)

# Task "Open Addressing"

# Task "Open Addressing"

- Fast niemand hatte die double hashing-Aufgabe korrekt :(
- Stellt sicher, dass ihr das an der Prüfung könnt

# Task "Trees"

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge



---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge
  - $\Theta(n^2)$

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge
  - $\Theta(n^2)$
- AVL-Trees  $\neq$  Red-Black-Trees!

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge
  - $\Theta(n^2)$
- AVL-Trees  $\neq$  Red-Black-Trees!
  - Es gab einen Fehler in der Task Description

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge
  - $\Theta(n^2)$
- AVL-Trees  $\neq$  Red-Black-Trees!
  - Es gab einen Fehler in der Task Description
  - AVL-Trees sind nicht relevant für die diesjährige Vorlesung<sup>1</sup>

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume

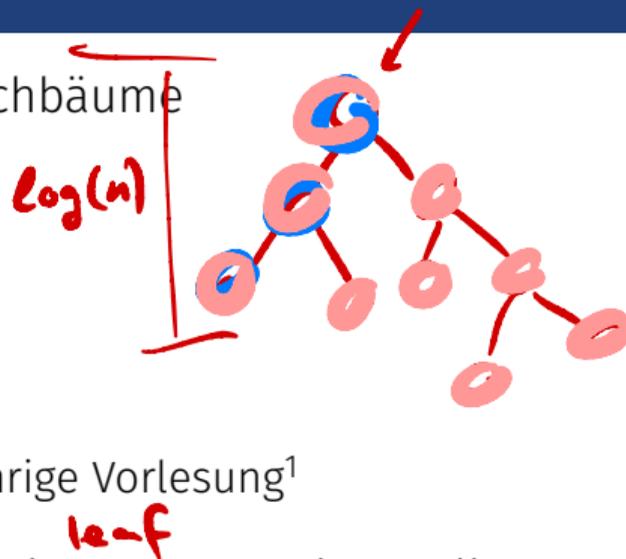
- Worst Case: Sortierte Reihenfolge
- $\Theta(n^2)$

- AVL-Trees  $\neq$  Red-Black-Trees!

- Es gab einen Fehler in der Task Description
- AVL-Trees sind nicht relevant für die diesjährige Vorlesung<sup>1</sup>

- mit Traversal meinen wir nicht "von oben nach unten" sondern "alle Nodes besuchen"

→  $\Theta(n)$



---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# Task "Trees"

- Worst Case Construction Time für Binäre Suchbäume
  - Worst Case:Sortierte Reihenfolge
  - $\Theta(n^2)$
- AVL-Trees  $\neq$  Red-Black-Trees!
  - Es gab einen Fehler in der Task Description
  - AVL-Trees sind nicht relevant für die diesjährige Vorlesung<sup>1</sup>
- mit Traversal meinen wir nicht "von oben nach unten" sondern "alle Nodes besuchen"
- `delete(k)` in Red-Black-Trees ist schwer, inbs. der "rot nach unten ziehen"-Teil aus der Vorlesung

---

<sup>1</sup>Ausser die Dozierenden sagen etwas anderes

# C++ Tasks

- Falls ihr nicht 100% geschafft habt, studiert die Musterlösung gut

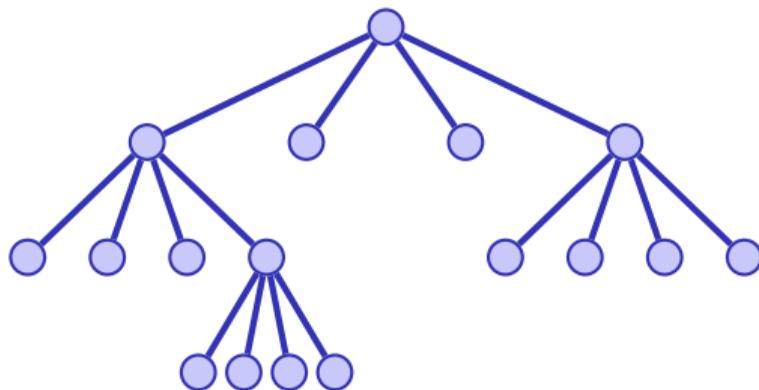
Fragen zu `[code]`expert eurerseits?

## 4. Quadtrees

---

# Quadrees

Quadrees sind Bäume, bei denen jeder Knoten höchstens **vier** Kinder hat.



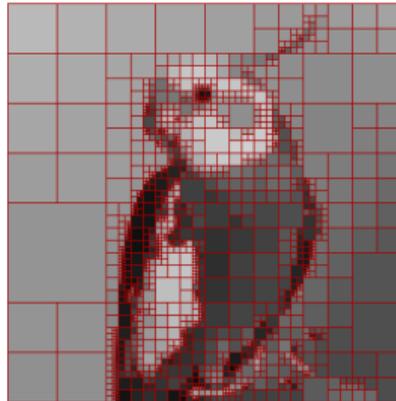
Hauptanwendung: Bildverarbeitung.

# Quadrees zur Bildkompression

**Erkenntnis:** (1) Teile das Bild rekursiv in vier Bereiche auf, (2) ordne die Bereiche Knoten in einem Quadtree zu und (3) weise jedem Blatt die Durchschnittsfarbe seines Bereichs zu.

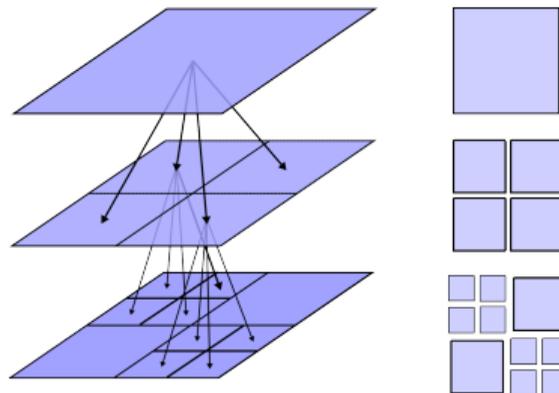
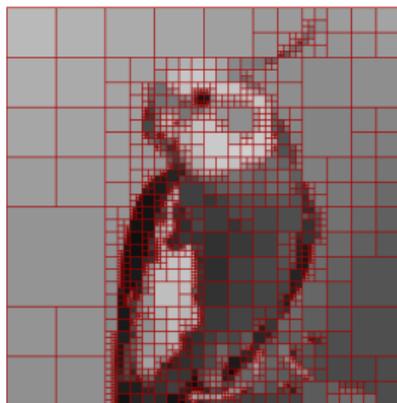
# Quadrees zur Bildkompression

**Erkenntnis:** (1) Teile das Bild rekursiv in vier Bereiche auf, (2) ordne die Bereiche Knoten in einem Quadtree zu und (3) weise jedem Blatt die Durchschnittsfarbe seines Bereichs zu.



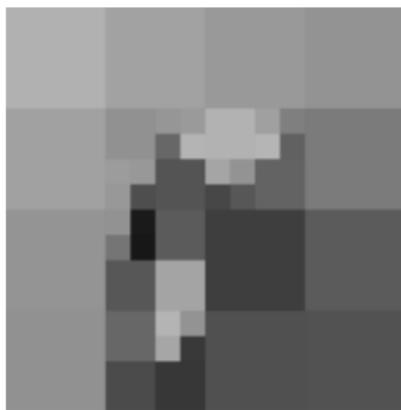
# Quadrees zur Bildkompression

**Erkenntnis:** (1) Teile das Bild rekursiv in vier Bereiche auf, (2) ordne die Bereiche Knoten in einem Quadtree zu und (3) weise jedem Blatt die Durchschnittsfarbe seines Bereichs zu.



# Quadtrees zur Bildkompression

Wann und wo soll die Rekursion gestoppt werden?



Zu früh?

# Quadtrees zur Bildkompression

Wann und wo soll die Rekursion gestoppt werden?



Ist das besser?

# Quadtrees zur Bildkompression

Wann und wo soll die Rekursion gestoppt werden?



**Frage:** Sollen wir aufhören, wenn jeder Knoten auf ein Pixel abgebildet ist?

# Quadrees zur Bildkompression

Wann und wo soll die Rekursion gestoppt werden?



**Antwort:** Wir würden das Originalbild erhalten, aber keine Speichereffizienz gewinnen.

# Quadtrees zur Bildkompression

Wir möchten

- eine möglichst genaue Annäherung, und
- so wenige Knoten wie möglich.

# Quadtrees zur Bildkompression

Wir möchten

- eine möglichst genaue Annäherung, und
- so wenige Knoten wie möglich.

Dies kann als Optimierungsproblem ausgedrückt werden:

$$H_\gamma(T, \mathbf{y}) := \gamma \cdot \underbrace{|L(T)|}_{\text{Anzahl Blätter}} + \underbrace{\sum_{r \in L(T)} \|y_r - \mu_r\|_2^2}_{\text{Summierter Approximationsfehler aller Blätter}}.$$

wo  $T$  ein Quadtree ist,  $\mathbf{y}$  die Bilddaten sind und  $\gamma \geq 0$  ein Regularisierungsparameter ist. Wir suchen die optimale Lösung  $\operatorname{argmin}_T H_\gamma(T, \mathbf{y})$ .

# Quadrees zur Bildkompression

$$H_\gamma(T, \mathbf{y}) := \gamma \cdot \underbrace{|L(T)|}_{\text{Anzahl Blätter}} + \underbrace{\sum_{r \in L(T)} \|y_r - \mu_r\|_2^2}_{\text{Summierter Approximationsfehler aller Blätter}} .$$

**Frage:** Welche Auswirkung hat ein niedriger Wert von  $\gamma$ ?

# Quadrees zur Bildkompression

$$H_\gamma(T, \mathbf{y}) := \gamma \cdot \underbrace{|L(T)|}_{\text{Anzahl Blätter}} + \underbrace{\sum_{r \in L(T)} \|y_r - \mu_r\|_2^2}_{\text{Summierter Approximationsfehler aller Blätter}} .$$

**Frage:** Welche Auswirkung hat ein niedriger Wert von  $\gamma$ ?

**Antwort:** Verbessert die Annäherung auf Kosten der Vergrößerung des Quadrees.

# Algorithmus: Minimize( $z, r, \gamma$ )

**Input:** Bilddaten  $z \in \mathbb{R}^S$ , Rechteck  $r \subset S$ , Regularisierung  $\gamma > 0$

**Output:**  $\min_T \gamma |L(T)| + \|z - \mu_{L(T)}\|_2^2$

**if**  $|r| = 0$  **then return** 0

$m \leftarrow \gamma + \sum_{s \in r} (z_s - \mu_r)^2$

**if**  $|r| > 1$  **then**

    Split  $r$  into  $r_{ll}, r_{lr}, r_{ul}, r_{ur}$

$m_1 \leftarrow \text{Minimize}(z, r_{ll}, \gamma)$ ;  $m_2 \leftarrow \text{Minimize}(z, r_{lr}, \gamma)$

$m_3 \leftarrow \text{Minimize}(z, r_{ul}, \gamma)$ ;  $m_4 \leftarrow \text{Minimize}(z, r_{ur}, \gamma)$

$m' \leftarrow m_1 + m_2 + m_3 + m_4$

**else**

$m' \leftarrow \infty$

**if**  $m' < m$  **then**  $m \leftarrow m'$

**return**  $m$

# 5. Dynamische Programmierung

---

# Dynamische Programmierung: Idee

- Aufteilen eines komplexen Problems in eine vernünftige Anzahl kleinerer Teilprobleme
- Die Lösung der Teilprobleme wird zur Lösung des komplexeren Problems verwendet
- Identische Teilprobleme werden nur einmal berechnet
  - Die Idee ist, einfach **die Ergebnisse von Teilproblemen zu speichern**, damit wir sie später bei Bedarf nicht neu berechnen müssen.

# Dynamische Programmierung: Idee

**Frage:** Welche der folgenden Fibonacci-Implementierungen würde besser abschneiden?

```
// top down
int fib(int n) {
    if (n <= 1) {
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}
```



```
// bottom up
int fib2(int n) {
    std::vector<int> f(n + 1);
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i <= n; ++i) {
        f[i] = f[i - 1] + f[i - 2];
    }
    return f[n];
}
```



# Dynamic Programming = Divide-And-Conquer ?

- In beiden Fällen ist das Ursprungsproblem (einfacher) lösbar, indem Lösungen von Teilproblemen herangezogen werden können. Das Problem hat **optimale Substruktur**.
- Bei Divide-And-Conquer Algorithmen (z.B. Mergesort) sind Teilprobleme unabhängig; deren Lösungen werden im Algorithmus nur einmal benötigt.
- Beim DP sind Teilprobleme nicht unabhängig. Das Problem hat **überlappende Teilprobleme**, welche im Algorithmus mehrfach gebraucht werden.
- Damit sie nur einmal gerechnet werden müssen, werden Resultate tabelliert. Dafür darf es **zwischen Teilproblemen keine zirkulären Abhängigkeiten** geben.



## ■ Memoisierung:

- Top-down-Ansatz
- Rekursion mit Caching von Ergebnissen
- Berechnet Werte bei Bedarf träge
- Kann effizienter sein, wenn nur wenige Werte benötigt werden

## ■ Bottom-up-Dynamische Programmierung:

- Iterativer Ansatz
- Erstellt Lösungen aus kleineren Teilproblemen
- Berechnet alle Werte in einer vordefinierten Reihenfolge
- Kann effizienter sein, wenn alle Werte benötigt werden

## Frage

In welchen der folgenden Fälle könnte die Memoisierung deutlich effizienter sein als die Bottom-up-Dynamische Programmierung?

1. Wenn alle Werte für das endgültige Ergebnis erforderlich sind
2. Wenn nur einige Werte für das endgültige Ergebnis erforderlich sind
3. Wenn das Problem überlappende Teilprobleme hat
4. Wenn das Problem iterativ gelöst werden kann

# Vergleich



## Antwort

Die Memoisierung könnte deutlich effizienter sein als die **Bottom-up-Dynamische Programmierung**, wenn **nur einige Werte für das endgültige Ergebnis erforderlich sind**

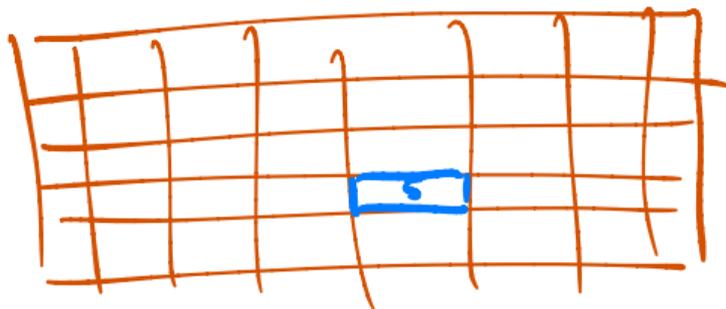
# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:**



# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?

# Dynamische Programmierung

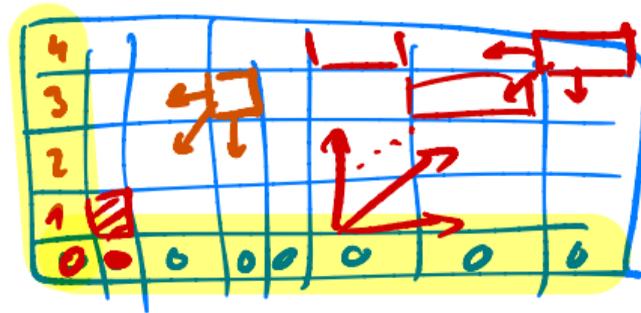
Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:**

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?



# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge:**

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge:** In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge:** In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit:**

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge:** In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit:** Wie lässt sich die Lösung am Ende aus der Tabelle auslesen? Laufzeit des Algorithmus?

# Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle:** Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags:** Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge:** In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit:** Wie lässt sich die Lösung am Ende aus der Tabelle auslesen? Laufzeit des Algorithmus?

# Überprüfung

Wählen Sie aus, welche Eigenschaften ein Problem haben muss, damit ein dynamischer Programmieransatz geeignet ist:

- Optimale Teilstruktur ✍
- Echtzeit-Problembehandlung
- Unabhängige Teilprobleme
- Speichereffiziente Lösung
- Rekursive Struktur
- Überlappende Teilprobleme
- Zirkuläre Abhängigkeiten
- Tabellierung oder Memoisierungspotenzial
- Kleiner Zustandsraum



Wählen Sie die Problemmerkmale, die benötigt werden, damit die dynamische Programmierung angemessen ist:

- **Optimale Teilstruktur**

- Echtzeit-Problembehandlung

- Unabhängige Teilprobleme

- Speichereffiziente Lösung

- **Rekursive Struktur**

- **Überlappende Teilprobleme**

- Zirkuläre Abhängigkeiten

- **Tabellierung oder Memoisierungspotenzial**

- Kleiner Zustandsraum

# Beispiel: Münzwechsel-Problem

## Definition

Gegeben sei ein Satz von Münzwerten und ein Zielbetrag. Finde die **minimale Anzahl von Münzen**, die benötigt werden, um den Zielbetrag zu erreichen. **Beachte, dass dieselbe Münzwert mehrmals verwendet werden kann.**

$[1, 2, 5]$

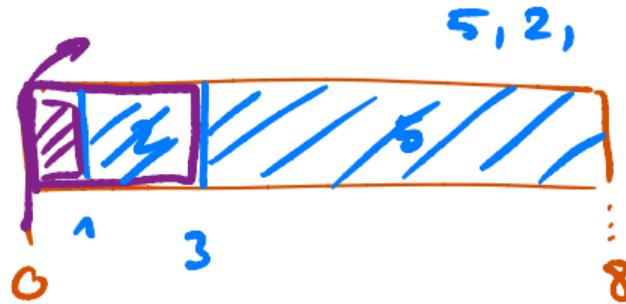
## Beispiel

Gegeben seien Münzen =  $[2, 4, 7]$  und Zielbetrag =  $8$ , die Lösung ist  $2 (4 + 4)$ .

## Bemerkung

Wenn das Problem keine Lösung hat, sollte der Algorithmus  $-1$  zurückgeben.

# Münzwechsel-Problem

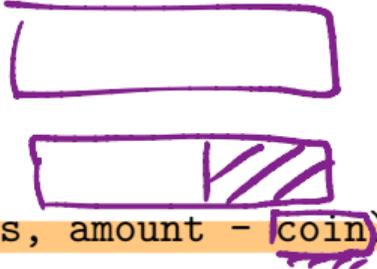


## Aufgabe

Entwerfen Sie einen rekursiven Algorithmus, um die Aufgabe zu lösen.

# Münzwechsel Rekursive Lösung

```
int coinChange(const std::vector<int>& coins, int amount) {  
    if (amount == 0) {  
        return 0;  
    }  
    int minCoins = INT_MAX;  
    for (int coin : coins) {  
        if (amount - coin >= 0) {  
            int temp = coinChange(coins, amount - coin);  
            if (temp != -1) {  
                minCoins = std::min(minCoins, temp + 1);  
            }  
        }  
    }  
    return minCoins == INT_MAX ? -1 : minCoins;  
}
```

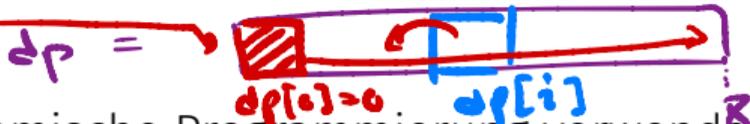


# Münzwechsel-Problem

## Aufgabe

Entwerfen Sie einen DP-Algorithmus, um die Aufgabe zu lösen.

# Münzwechsel Dynamische Programmierung



Wir können die dynamische Programmierung verwenden, um dieses Problem zu lösen, indem wir ein eindimensionales Array erstellen, bei dem  $dp[i]$  die minimale Anzahl von Münzen darstellt, die zur Herstellung des Betrags  $i$  benötigt werden:

- Setze jedes Element in  $dp$  auf einen Wert, der größer ist als die maximal mögliche Anzahl von Münzen.
- Setze  $dp[0] = 0$ .
- Für jede Münze  $c$ , iteriere durch das Array und aktualisiere  $dp[i]$ , wenn  $dp[i-c]+1$  einen niedrigeren Wert hat.

$$c \in \{2, 4, 7\}$$

Eine vollständige Beschreibung eines dynamischen Problems umfasst immer die folgenden Aspekte:

- ✓ **Definition der Teilprobleme / der DP-Tabelle:** Welche ist die Tabelle? Was ist die Bedeutung jedes Eintrags?
- ✓ **Rekursion: Berechnung eines Eintrags:** Wie berechnet man den Wert aus den Werten von anderen Einträgen? Welche Einträge sind von anderen Einträgen ab?
- ✓ **Topologische Ordnung: Berechnungsreihenfolge:** In welcher Reihenfolge muss man die Einträge berechnen?

# Coin Change DP Solution

```
int coinChange(const std::vector<int>& coins, int amount) {  
    std::vector<int> dp(amount + 1, (amount + 1)); ←   
    dp[0] = 0;  
    for (int coin : coins) { C ∈ {1, 2, 4}  
        for (int i = coin; i <= amount; ++i) { über dp[]  
            dp[i] = std::min(dp[i], dp[i - coin] + 1);  
        }  
    }  
    return dp[amount] <= amount ? dp[amount] : -1;  
}
```

# DP Visualization

$$dp[i] \leftarrow \min(dp[i], dp[i-c] + 1)$$

Münzen: [1, 2, 4] | Ziel: 8

i	0	1	2	3	4	5	6	7	8
dp[i]	0	1	2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

Anfangszustand des dp-Arrays.

$ni = (dp[1], dp[1-1])$   
 $\infty, dp[0] + 1$   
MAX\_INT

- Für jede Münze  $c$ , iteriere durch das Array und aktualisiere  $dp[i]$ , wenn  $dp[i-c] + 1$  einen niedrigeren Wert hat.

Eine vollständige Beschreibung e  
**immer** die folgenden Aspekte:

– Definition des Teilproblems  $i$ .

# DP Visualization

Münzen: [~~1~~, 2, 4] Ziel: 8

i	0	1	2	3	4	5	6	7	8
dp[i]	0	1	2	3	4	5	6	7	8

Nach der Verarbeitung der ersten Münze.

# DP Visualization

Münzen: [1, 2, 4]    Ziel: 8

i	0	1	2	3	4	5	6	7	8
dp[i]	0	1	1	2	2	3	3	4	4

Nach der Verarbeitung der zweiten Münze.

# DP Visualization

Münzen: [1, 2, 4] Ziel: 8

i	0	1	2	3	4	5	6	7	8
dp[i]	0	1	1	2	1	2	2	3	2

Nach der Verarbeitung der dritten und letzten Münze. Antwort:  $dp[8] = 2$ .

# Münzwechsel-Zeitkomplexität

## Frage

Wie vergleicht sich die Zeitkomplexität des DP-Algorithmus mit dem naiven rekursiven Algorithmus?

# Münzwechsel-Zeitkomplexität

## Frage

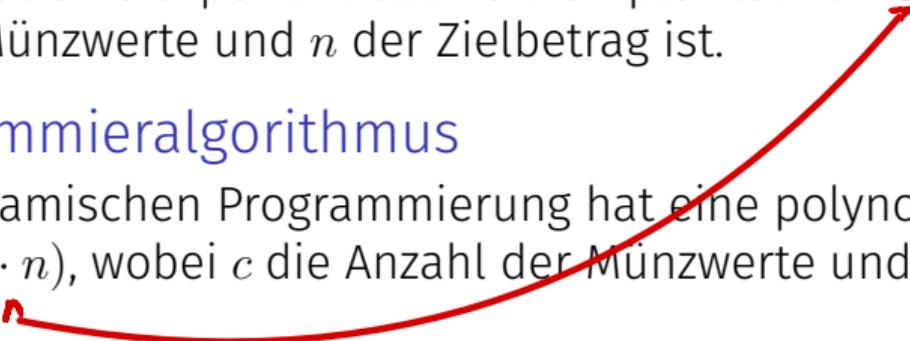
Wie vergleicht sich die Zeitkomplexität des DP-Algorithmus mit dem naiven rekursiven Algorithmus?

## Naiver Algorithmus

Der naive Algorithmus hat eine exponentielle Zeitkomplexität von  $O(c^n)$ , wobei  $c$  die Anzahl der Münzwerte und  $n$  der Zielbetrag ist.

## Dynamischer Programmieralgorithmus

Der Algorithmus der dynamischen Programmierung hat eine polynomielle Zeitkomplexität von  $O(c \cdot n)$ , wobei  $c$  die Anzahl der Münzwerte und  $n$  der Zielbetrag ist.



Exercise class 08: Quadtrees and Dynamic Programming auf Code-Expert

- Region Point Quadtree
- Maximum sum of an increasing subsequence ←

## 7. Tipps zu `expert`

---

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## **Aufgabe "Maximum Sum Triangle"**

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## **Aufgabe "Maximum Sum Triangle"**

- Wichtig!

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## **Aufgabe "Maximum Sum Triangle"**

- Wichtig!
- Verwendet das Rezept

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## **Aufgabe "Maximum Sum Triangle"**

- Wichtig!
- Verwendet das Rezept
- Lasst euch Zeit

# Tipps für nächste [code]expert -Aufgaben

## Aufgabe "Image Segmentation"

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## Aufgabe "Enumerating Palindromes" DP

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## Aufgabe "Maximum Sum Triangle" DP

- Wichtig!
- Verwendet das Rezept
- Lasst euch Zeit

## Aufgabe "Maximum average sum partition of an array"

# Tipps für nächste [code]expert -Aufgaben

## **Aufgabe "Image Segmentation"**

- Viel Mathematik und Nachdenken
- Viel Algorithmus implementieren
- Wenig Code (etwa 20 Zeilen)

## **Aufgabe "Enumerating Palindromes"**

- Verwendet das Rezept
- Verwendet nicht ChatGPT

## **Aufgabe "Maximum Sum Triangle"**

- Wichtig!
- Verwendet das Rezept
- Lasst euch Zeit

## **Aufgabe "Maximum average sum partition of an array"**

- Verwendet das Rezept

## 8. Outro

---

# Allgemeine Fragen?

Bis zum nächsten Mal

Schönes Wochenende!