Universität Bremen

Praktische Informatik 2
Algorithmische Grundkonzepte

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für
Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Algorithmus: Definition

- Beschreibung einer Methode zur Lösung einer gegebenen Aufgabenstellung
  - Begriff sehr allgemein fassbar: auch Kochrezepte und Aufbauanleitungen sind Algorithmen
  - In der Informatik: Algorithmen, die programmierbar sind
- Beispiele
  - Schriftliches Addieren, Subtrahieren, Multiplizieren und Dividieren
  - Aus der Position und einer Feldbeschreibung eine Bitmaske für die Erreichbarkeit von Nachbarzellen erstellen

Algorithmus: Definition

- **Eingabespezifikation**: Eingabegrößen, Anforderungen an diese
- **Ausgabespezifikation**: Ausgabegrößen, Eigenschaften dieser
- **Endliche Beschreibung**: Verfahren in endlichem Text beschrieben
- **Effektivität**: Jeder Schritt muss effektiv ausführbar sein
- **Determiniertheit**: Verfahrensablauf zu jedem Zeitpunkt vorgeschrieben
- **partielle Korrektheit**: Jedes berechnete Ergebnis genügt der Ausgabespezifikation, sofern die Eingaben der Spezifikation genügt haben
- **Terminierung**: Halt nach endlich vielen Schritten mit einem Ergebnis, sofern die Eingaben der Spezifikation genügt haben

Spezifikation
Durchführbarkeit
totale Korrektheit

Universität Bremen

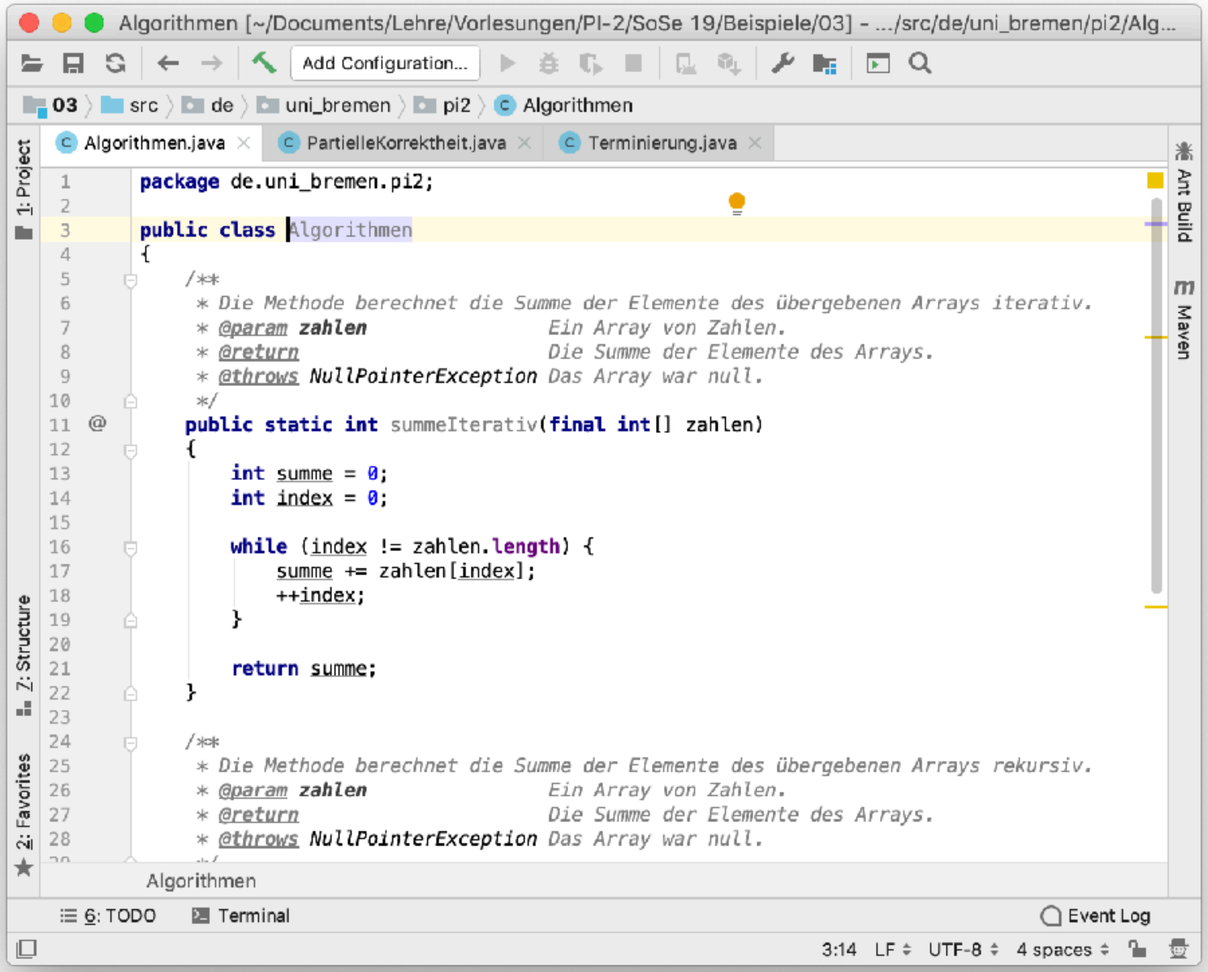
Grundschema des Algorithmenaufbaus

Name des Algorithmus und Liste der Parameter, Spezifikationen des Ein-/Ausgabeverhaltens

1. **Vorbereitung**: Einführung von Hilfsgrößen etc.
2. **Trivialfall?** Falls einfacher Fall vorliegt, Beendigung mit Ergebnis
3. **Arbeit** (Problemreduktion, Ergebnisaufbau)
   - Reduziere Problemstellung **X** auf eine einfachere Form **X‘**, mit **X > X‘**
   - Baue entsprechend der Reduktion einen Teil des Ergebnisses auf
4. **Rekursion** bzw. **Iteration**: Rufe zur Weiterverarbeitung Algorithmus mit reduziertem **X‘** erneut auf (Rekursion), bzw. fahre mit **X‘** bei 2. fort (Iteration)

Universität Bremen

Grundschema des Algorithmenaufbaus: Demo




```java
package de.uni_bremen.pi2;

public class Algorithmen
{
    /**
     * Die Methode berechnet die Summe der Elemente des übergebenen Arrays iterativ.
     * @param zahlen                Ein Array von Zahlen.
     * @return                      Die Summe der Elemente des Arrays.
     * @throws NullPointerException Das Array war null.
     */
    public static int summeIterativ(final int[] zahlen)
    {
        int summe = 0;
        int index = 0;

        while (index != zahlen.length) {
            summe += zahlen[index];
            ++index;
        }

        return summe;
    }

    /**
     * Die Methode berechnet die Summe der Elemente des übergebenen Arrays rekursiv.
     * @param zahlen                Ein Array von Zahlen.
     * @return                      Die Summe der Elemente des Arrays.
     * @throws NullPointerException Das Array war null.
```

Universität Bremen

Algorithmenmuster

- **Greedy** (gierig): Reduziere das Problem in einer Richtung
- **Divide and Conquer** (Teile und herrsche): Reduziere das Problem in zwei oder mehrere Teilprobleme
- **Backtracking**: Generiere mögliche Lösungen, wenn es nicht weiter geht, mache beim Vorgänger weiter
- **Dynamische Programmierung**: Werden Teillösungen mehrfach benötigt, merke sie dir in einer Tabelle und lies diese aus, wenn sie wieder benötigt werden

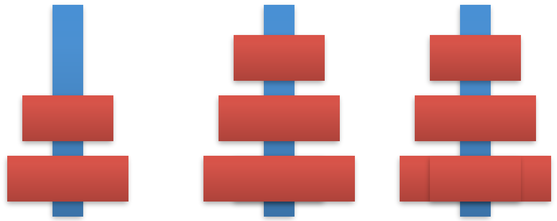
Universität Bremen

Greedy (gierig)

- Behandle einfache und triviale Fälle
- Reduziere das Problem in **einer** Richtung
- Rekursiver Aufruf oder Iteration
- **Beispiel**: Wechselgeld mit minimaler Anzahl von Münzen, wobei genug Münzen verfügbar sind
  - Lösung: So lange die größte Münze zurückgeben, deren Wert kleiner oder gleich der noch offenen Restsumme ist, bis die Restsumme 0 ist

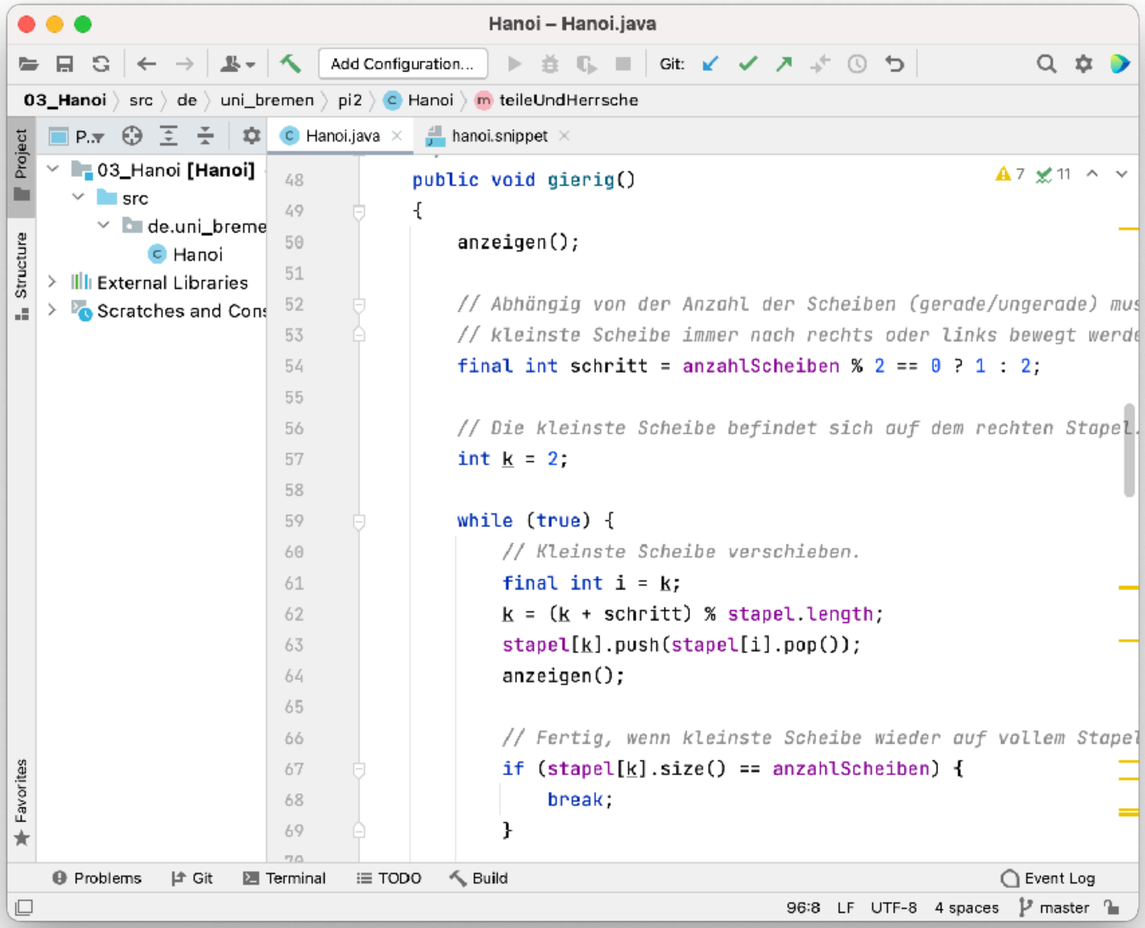
Greedy: Türme von Hanoi

- Bewege Turm von einer Stange zu einer anderen, ohne dass jemals eine größere Scheibe auf einer kleineren liegt
- Lösung
  - Bewege kleinste Scheibe nach links (zyklisch)
  - Fertig, falls Turm vollständig bewegt
  - Ansonsten bewege einzig mögliche nicht-kleinste Scheibe
  - Beginne von vorne



Universität Bremen

Greedy: Türme von Hanoi

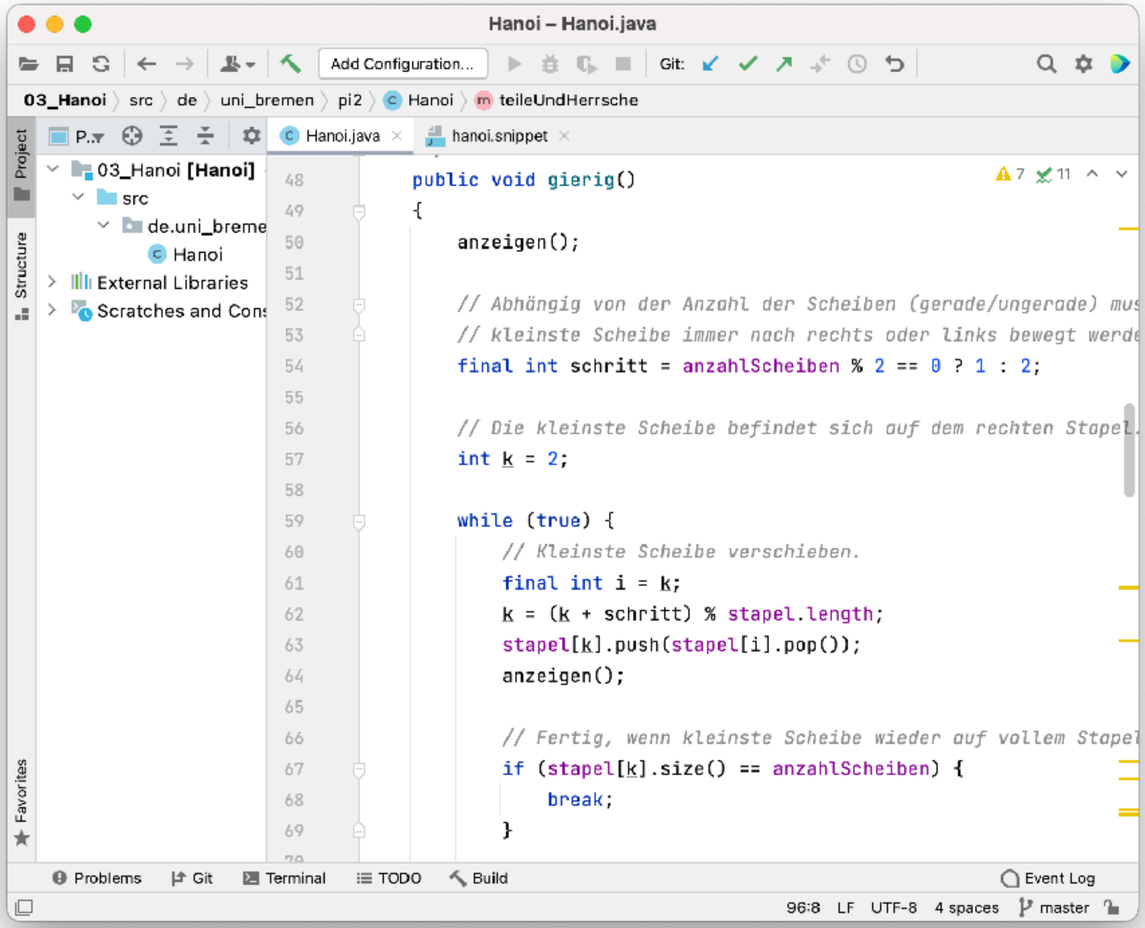


Divide and Conquer (Teile und herrsche)

- Behandle einfache und triviale Fälle
- **Teile**: Reduziere das Problem in zwei oder mehrere Teilprobleme
- **Herrsche**: Löse die Teilprobleme (typischerweise rekursiv)
- **Kombiniere**: Setze Teillösungen zur Gesamtlösung zusammen
- Teilprobleme sollten **von derselben Art** wie das Ursprungsproblem sein, damit sie mit **demselben Algorithmus** gelöst werden können

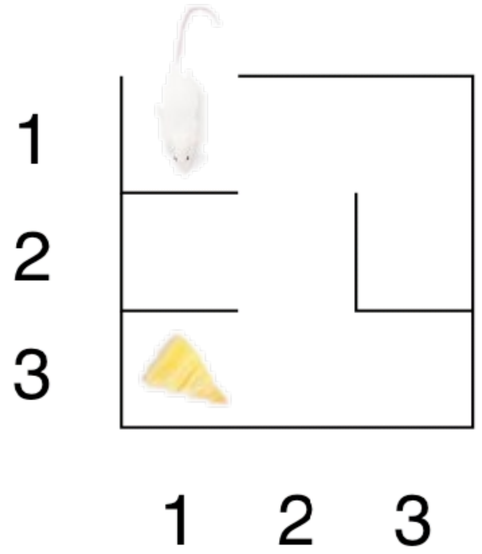
Universität Bremen

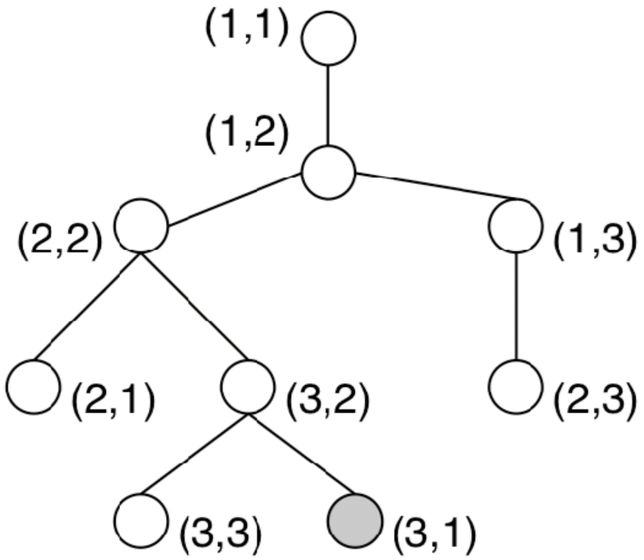
Divide and Conquer: Türme von Hanoi



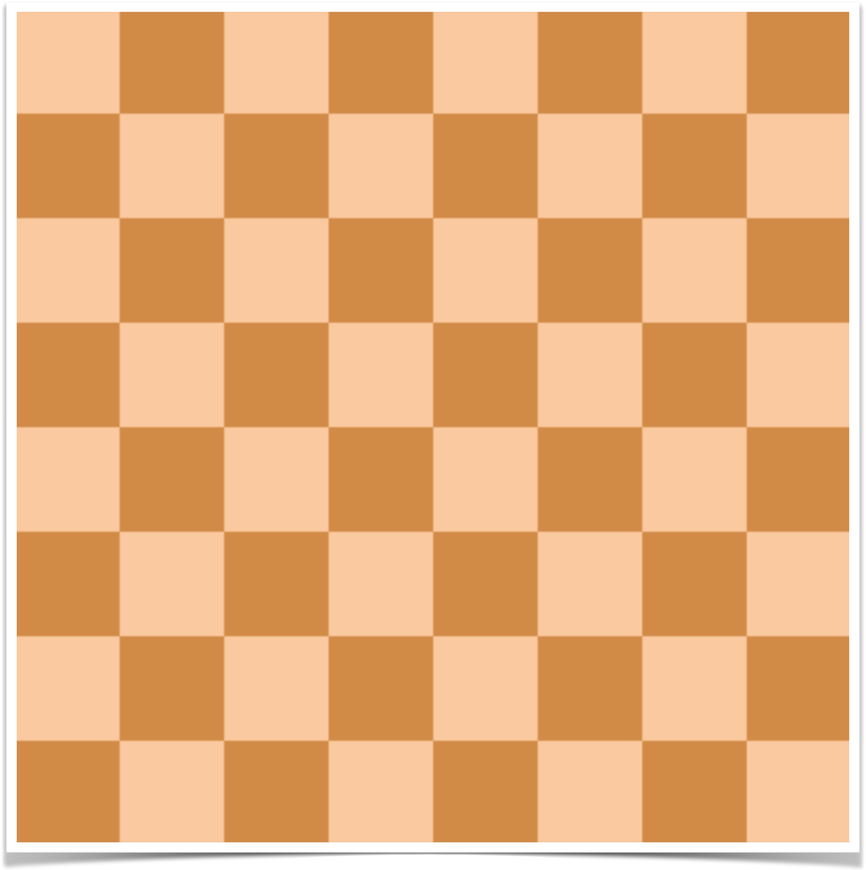
Backtracking

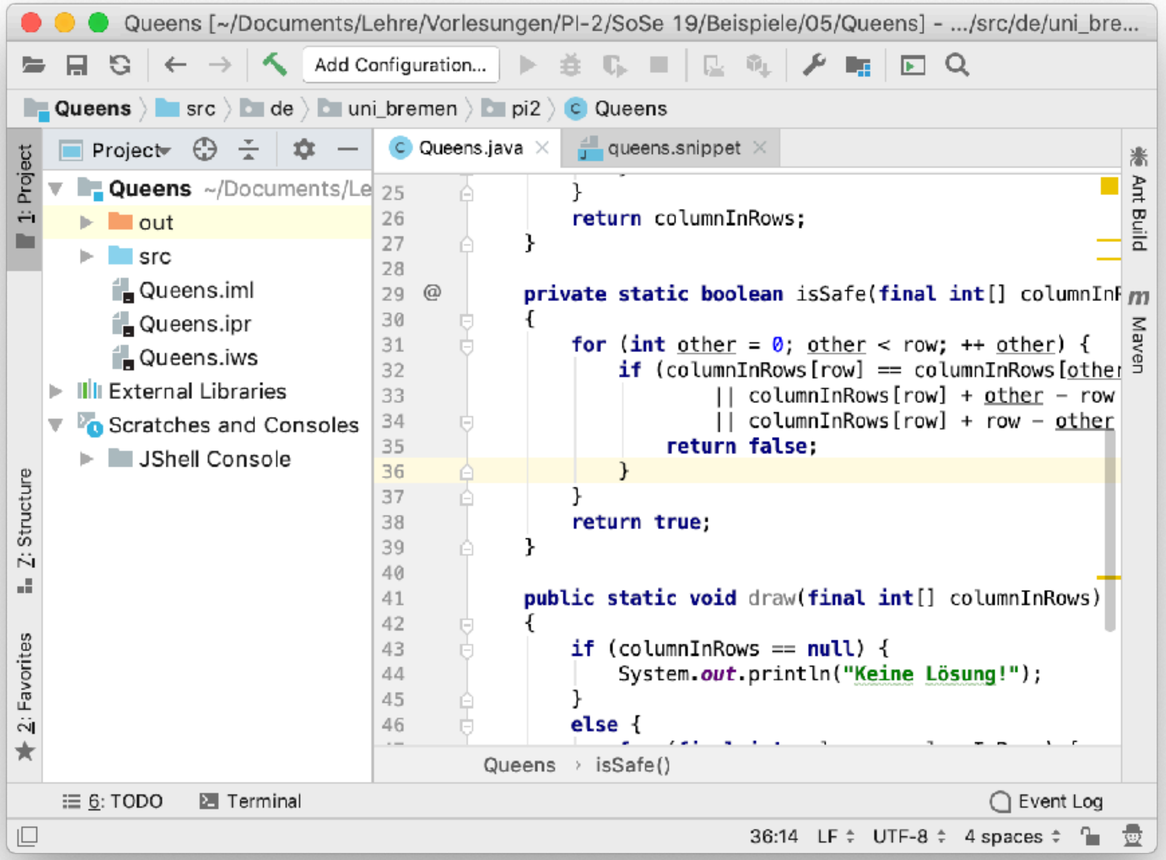
- Es gibt eine Menge von Konfigurationen K
- K_0 ist die Anfangskonfiguration
- Für jede Konfiguration K_i kann Menge der direkten Erweiterungen $K_{i,\,1\ldots n}$ bestimmt werden
- Es ist entscheidbar, ob eine Konfiguration Lösung ist
- Von jeder Konfiguration aus werden alle Erweiterungen rekursiv durchprobiert
- Hat eine Konfiguration keine Erweiterungen, wird zum Vorgänger zurückgekehrt (**backtrack**)





Backtracking: n-Damen-Problem





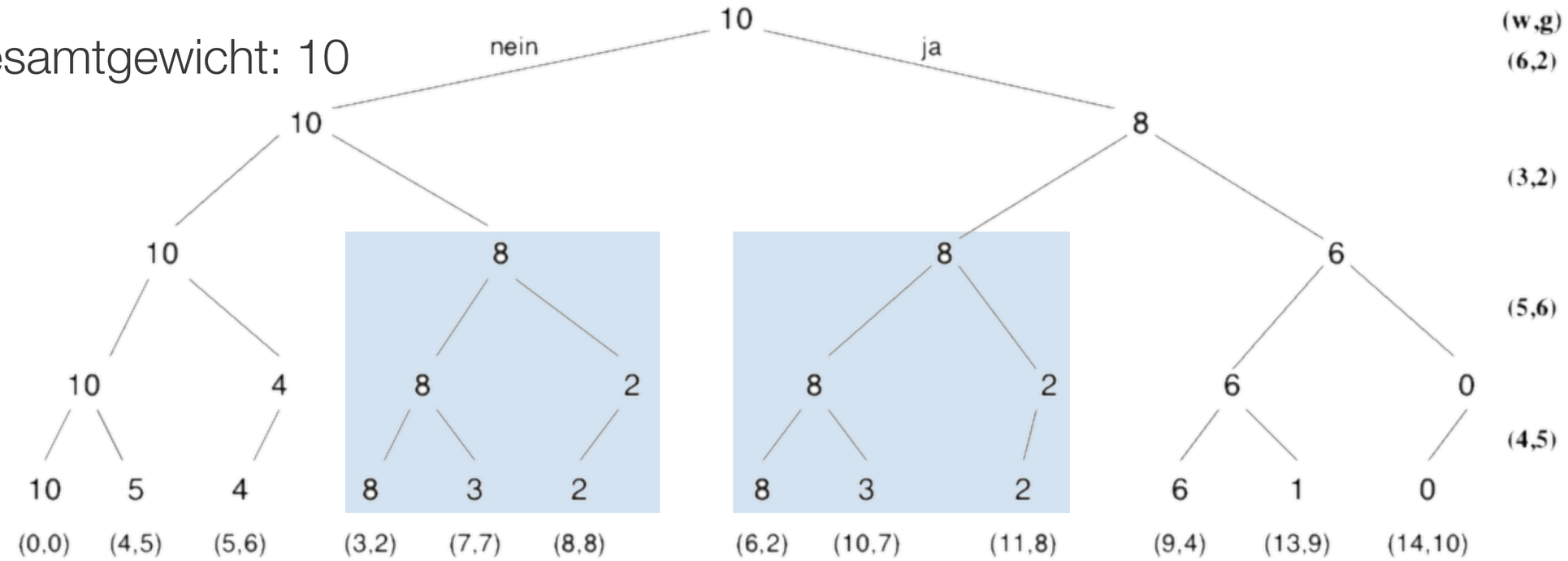
Universität Bremen

Dynamische Programmierung

- Werden Teillösungen mehrfach benötigt, speichere sie in einer **Tabelle**
- Lies die Tabelle später aus, wenn die Teillösungen wieder benötigt werden
- **Beispiel**: Rucksackproblem
  - Fülle Rucksack so mit Gegenständen, dass das zulässige Gesamtgewicht nicht überschritten wird und ihr Wert maximal ist

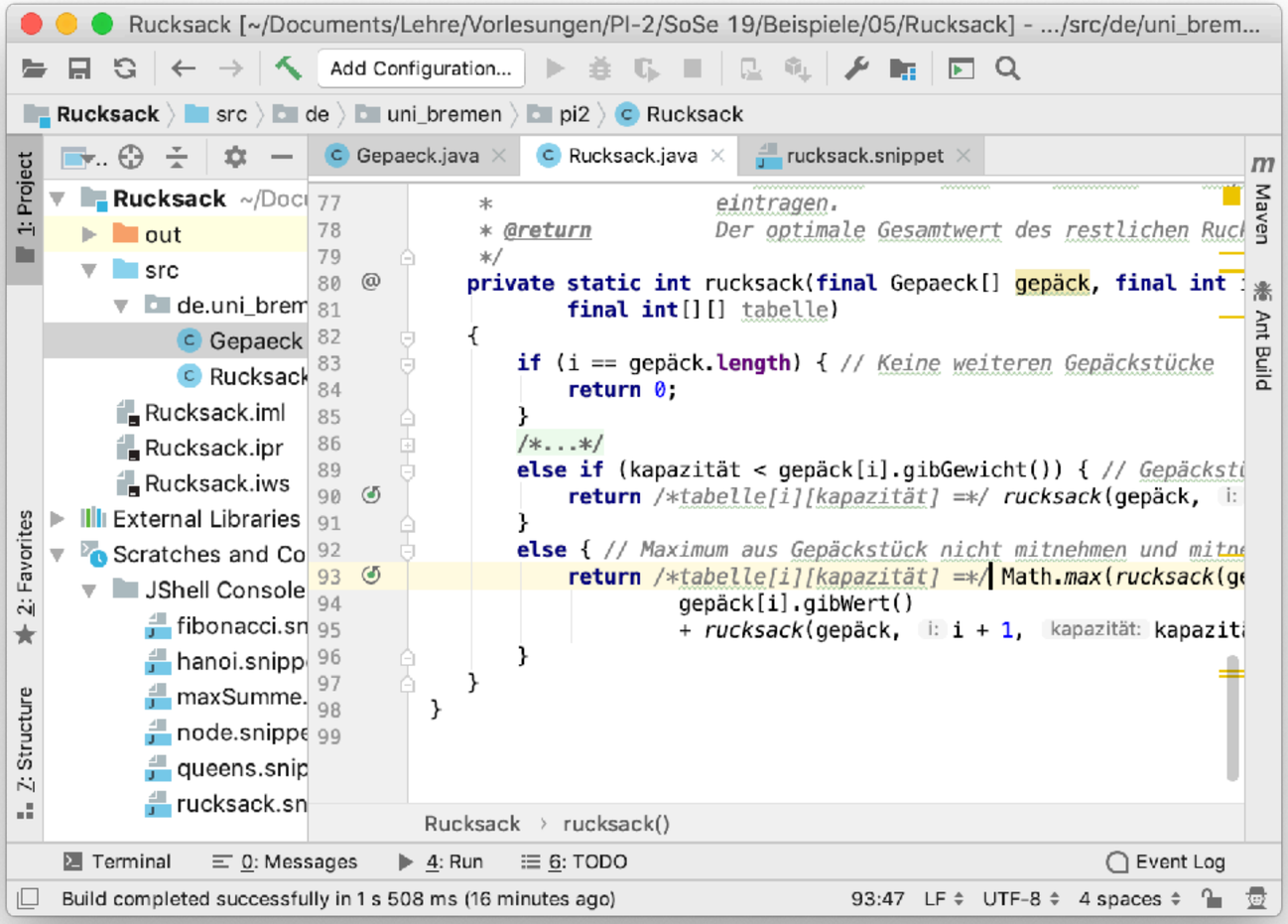
Dynamische Programmierung: Rucksackproblem – Beispiel

- Gewichte $g_1 = 2$, $g_2 = 2$, $g_3 = 6$, $g_4 = 5$
- Werte $w_1 = 6$, $w_2 = 3$, $w_3 = 5$, $w_4 = 4$
- Maximales Gesamtgewicht: 10



Universität Bremen

Dynamische Programmierung: Rucksackproblem



Universität Bremen

Zusammenfassung der Konzepte

- **Algorithmus**
- **Greedy**
- **Divide and Conquer**
- **Backtracking**
- **Dynamische Programmierung**