DS 26/27 – Vertiefung am Beispiel: Das Rucksackproblem

Informatik 10 – MINT-Zweig | Informatisches Problemlösen

Lernziel

Ich wende Brute-Force- und Greedy-Ansätze auf ein neues Beispielproblem (Rucksackproblem) an und vergleiche beide Strategien.

Hinweis: Dieses Notebook deckt zwei Doppelstunden (DS 26 und DS 27) ab. Die beiden Verlaufspläne unten zeigen, wie sich die Inhalte auf zwei separate 90-Minuten-Einheiten aufteilen lassen — DS 26 endet nach der Brute-Force-Lösung (Übung 1), DS 27 beginnt mit der Greedy-Heuristik (Übung 2).

🕘 Stundenverlaufsplan – DS 26 (90 Minuten)

Orientierungsrahmen für 11 SuS: Die ersten 80 Minuten bilden den Kern. Die letzten 10 Minuten sind bewusst flexibel und werden nicht mit neuem Pflichtstoff belegt.

Phase	Zeit	Inhalt/Aktion	Sozialform
Einstieg	10'	Ein konkretes Packproblem mit Gewichtsgrenze vorstellen.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Modellierung	25'	Gegenstände, Gewicht, Wert, Tragkraft, zulässige und optimale Auswahl herausarbeiten.	Unterrichtsgespräch; Tabelle am Board
Systematisches Ausprobieren	20'	Kleine Teilmengen gemeinsam bilden, bewerten und einen Brute-Force-Plan formulieren.	Plenum; gemeinsames Arbeiten an der Tabelle
Kurzer Transfer	15'	Weitere Teilmengen prüfen und die Anzahl `2^n` für kleine `n` bestimmen.	Einzelarbeit, anschließend 4 Tandems + 1 Trio
Auswertung	10'	Optimale Auswahl und exponentiell wachsenden Aufwand erklären lassen.	Plenum
Reserve / Vertiefung	10'	Python-Code nur demonstrieren oder freiwillig nachvollziehen.	Flexibel

🕘 Stundenverlaufsplan – DS 27 (90 Minuten)

Orientierungsrahmen für 11 SuS: Die ersten 80 Minuten bilden den Kern. Die letzten 10 Minuten sind bewusst flexibel und werden nicht mit neuem Pflichtstoff belegt.

Phase	Zeit	Inhalt/Aktion	Sozialform
Wiedereinstieg	10'	Brute-Force-Ergebnis und zentrale Begriffe aus DS 26 aktivieren.	Plenum
Fragend-entwickelnde Erarbeitung	25'	Wert-Gewicht-Verhältnis als mögliche schnelle Auswahlregel entdecken und ordnen.	Unterrichtsgespräch; gemeinsame Tabelle
Heuristik anwenden	20'	Greedy-Auswahl von Hand durchführen und jeden Schritt begründen.	Plenum mit kurzen Denkpausen
Kurzer Transfer	15'	Greedy-Ergebnis mit der optimalen Auswahl vergleichen und ein Gegenbeispiel untersuchen.	Einzelarbeit, anschließend 4 Tandems + 1 Trio
Bewertung	10'	Vorteile, Aufwand und fehlende Optimalitätsgarantie beurteilen.	Plenum
Reserve / Vertiefung	10'	Eigenes Gegenbeispiel oder freiwillige Python-Implementierung.	Flexibel

🗣️ Sprachhilfe

- „Es handelt sich um ein Problem/eine Probleminstanz, weil …“
- „Der Ansatz prüft/wählt … und benötigt dabei …“
- „Die Heuristik ist hier geeignet/nicht geeignet, weil …“

📌 Merksätze

- Beim **Rucksackproblem (Knapsack Problem)** stehen mehrere Gegenstände mit jeweils einem **Gewicht** und einem **Wert** zur Verfügung. Gesucht ist die Auswahl von Gegenständen mit **maximalem Gesamtwert**, deren **Gesamtgewicht** eine vorgegebene Tragkraft nicht übersteigt.
- **Brute-Force-Ansatz:** Alle möglichen **Teilmengen** der Gegenstände werden gebildet und bewertet – bei `n` Gegenständen gibt es `2^n` Teilmengen (**exponentieller Aufwand**).
- **Greedy-Heuristik:** Die Gegenstände werden nach ihrem **Wert-Gewicht-Verhältnis** (Wert ÷ Gewicht) sortiert und **so lange** eingepackt, bis kein weiterer Gegenstand mehr hineinpasst.

💡 Probleminstanz

Rucksack-Tragkraft: **10 kg**

Gegenstand	Gewicht (kg)	Wert (€)
Zelt	5	60
Schlafsack	4	40
Kocher	2	30
Proviant	3	25
Erste-Hilfe-Set	1	20

Aufgabe zum Einstieg (von Hand): Findet durch Ausprobieren mindestens zwei zulässige Kombinationen (Gesamtgewicht ≤ 10 kg) und berechnet jeweils den Gesamtwert.

(Eure Kombinationen und Werte hier)

🐍 Brute-Force-Lösung

```python
from itertools import combinations

gegenstaende = [
    ("Zelt", 5, 60),
    ("Schlafsack", 4, 40),
    ("Kocher", 2, 30),
    ("Proviant", 3, 25),
    ("Erste-Hilfe-Set", 1, 20),
]
tragkraft = 10

beste_kombination = None
bester_wert = 0

for r in range(len(gegenstaende) + 1):
    for kombination in combinations(gegenstaende, r):
        gesamtgewicht = sum(g[1] for g in kombination)
        gesamtwert = sum(g[2] for g in kombination)
        if gesamtgewicht <= tragkraft and gesamtwert > bester_wert:
            bester_wert = gesamtwert
            beste_kombination = kombination

print("Optimale Kombination:", [g[0] for g in beste_kombination])
print("Gesamtwert:", bester_wert)
print("Gesamtgewicht:", sum(g[1] for g in beste_kombination))
```

```
Optimale Kombination: ['Zelt', 'Schlafsack', 'Erste-Hilfe-Set']
Gesamtwert: 120
Gesamtgewicht: 10
```

📝 Übung 1 – Anzahl der Teilmengen

Wie viele Teilmengen müssen bei `n` Gegenständen geprüft werden? Berechnet dies für die vorliegende Instanz mit `2^n` und begründet das Ergebnis. Die folgende Code-Zelle kann optional zur Kontrolle ergänzt werden.

📝 Übung 2 – Greedy-Heuristik von Hand anwenden

Berechnet für jeden Gegenstand das **Wert-Gewicht-Verhältnis**, sortiert die Gegenstände danach absteigend und packt sie in dieser Reihenfolge, sofern sie noch passen. Vergleicht das Ergebnis mit der optimalen Brute-Force-Lösung. Haltet jeden Auswahlschritt in einer Tabelle fest.

```python
# OPTIONAL: Greedy-Heuristik für das Rucksackproblem programmieren

def greedy_rucksack(gegenstaende, tragkraft):
    # Tipp: sortiert(gegenstaende, key=lambda g: g[2]/g[1], reverse=True)
    pass
```

📝 Übung 3 – Grenzen der Heuristik

Überlegt euch (oder konstruiert) eine Probleminstanz, bei der die Greedy-Heuristik **nicht** die optimale Lösung findet. Testet eure Instanz mit beiden Ansätzen.