DS 28 – Aufwand verschiedener Lösungsansätze vergleichen

Informatik 10 – MINT-Zweig | Informatisches Problemlösen

Lernziel

Ich vergleiche den Aufwand verschiedener Lösungsansätze miteinander.

> Der Begriff **Aufwand** wird hier in einem **didaktisch reduzierten, anschaulichen** Sinn verwendet (Anzahl der geprüften Möglichkeiten bzw. Laufzeit) – eine formale Komplexitätstheorie folgt erst in der Oberstufe.

🕘 Stundenverlaufsplan (Doppelstunde, 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 und Aktivierung	10'	An bekannte Anzahlen aus TSP und Rucksackproblem anknüpfen.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Erarbeitung	25'	Faktorielles und exponentielles Wachstum anschaulich über Wertetabellen entwickeln.	Unterrichtsgespräch; Beiträge der SuS werden sichtbar festgehalten
Gemeinsame Umsetzung und Sicherung	20'	Eine kleine Messreihe gemeinsam interpretieren; Code dient nur der Demonstration.	Plenum; Live-Coding, Tafelbild oder gemeinsame Simulation
Kurzer Transfer	15'	Für ausgewählte Werte von `n` die Zahl `2^n` berechnen und vergleichen.	Einzelarbeit mit anschließendem Tandemabgleich (4 Tandems + 1 Trio)
Auswertung	10'	Unterschied zwischen geprüften Möglichkeiten und gemessener Zeit erklären lassen.	Plenum; Lösung erklären und Fachsprache sichern
Reserve / Vertiefung	10'	Diagramm oder Hochrechnung als Vertiefung.	Flexibel; bei Bedarf für Fragen, Fehlersuche oder Abschluss nutzen

🗣️ 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

- Der **Aufwand** eines Lösungsansatzes lässt sich anschaulich als **Anzahl der zu prüfenden Möglichkeiten** oder als **tatsächlich gemessene Rechenzeit** beschreiben.
- Beim TSP (Brute-Force) wächst der Aufwand **faktoriell** mit der Anzahl der Städte: `(n-1)!`
- Beim Rucksackproblem (Brute-Force) wächst der Aufwand **exponentiell** mit der Anzahl der Gegenstände: `2^n`
- Greedy-Heuristiken prüfen dagegen **wesentlich weniger Möglichkeiten**; der Aufwand entsteht hier vor allem durch das Sortieren und das einmalige Durchlaufen der Gegenstände.

- Schon bei mäßig größeren Probleminstanzen wird der Unterschied zwischen Brute-Force und Heuristik **drastisch** spürbar.

🐍 Messung: Wachstum des Brute-Force-Aufwands (TSP)

Die folgende Zelle misst, wie stark die Anzahl der zu prüfenden Rundreisen und die tatsächliche Rechenzeit mit der Anzahl der Städte wächst.

```python
import time
from itertools import permutations

def anzahl_und_zeit_bruteforce(anzahl_staedte):
    staedte = list(range(anzahl_staedte))
    start = time.time()
    anzahl = 0
    for _ in permutations(staedte[1:]):
        anzahl += 1
    dauer = time.time() - start
    return anzahl, dauer

print(f"{'Städte':>7} | {'Anzahl Routen':>15} | {'Zeit (s)':>10}")
for n in range(2, 9):
    anzahl, dauer = anzahl_und_zeit_bruteforce(n)
    print(f"{n:>7} | {anzahl:>15} | {dauer:>10.4f}")
```

```
 Städte |   Anzahl Routen |   Zeit (s)
      2 |               1 |     0.0000
      3 |               2 |     0.0000
      4 |               6 |     0.0000
      5 |              24 |     0.0000
      6 |             120 |     0.0000
      7 |             720 |     0.0000
      8 |            5040 |     0.0002
```

📝 Übung 1 – Tabelle interpretieren

Betrachtet die Ausgabe oben:

1. Um welchen Faktor wächst die Anzahl der Routen jeweils von `n` zu `n+1` Städten?
2. Schätzt anhand des Musters (ohne es auszuführen!), wie lange die Berechnung bei **12 Städten** ungefähr dauern würde.

(Eure Antworten hier)

📝 Übung 2 – Rucksackproblem: Brute-Force vs. Greedy im Vergleich

Erstellt für eine wachsende Anzahl an Gegenständen (z. B. 5, 10, 15, 18) eine Tabelle mit der Anzahl der von Brute Force geprüften Teilmengen (`2^n`). Vergleicht dieses Wachstum mit der Greedy-Heuristik, die die Gegenstände sortiert und anschließend einmal durchläuft. Die Code-Zelle kann optional zur Kontrolle ergänzt werden.

```python
# OPTIONAL: Tabelle mit n und 2**n zur Kontrolle ausgeben
for n in [5, 10, 15, 18]:
    pass  # TODO: 2**n berechnen und ausgeben
```

🚀 Zusatzaufgabe – Diagramm

Falls ihr mit `matplotlib` vertraut seid: Stellt das faktorielle Wachstum (TSP) und das exponentielle Wachstum (Rucksack) grafisch gegenüber.

```python
import matplotlib.pyplot as plt
import math

n_werte = list(range(1, 11))
faktoriell = [math.factorial(n) for n in n_werte]
exponentiell = [2**n for n in n_werte]

plt.plot(n_werte, faktoriell, marker="o", label="Fakultät (TSP)")
plt.plot(n_werte, exponentiell, marker="o", label="2^n (Rucksack)")
plt.yscale("log")
plt.xlabel("Anzahl n")
plt.ylabel("Anzahl Möglichkeiten (log-Skala)")
plt.legend()
plt.title("Wachstum des Brute-Force-Aufwands")
plt.show()
```

