DS 24 – Optimale Lösung und Brute-Force-Ansatz

Informatik 10 – MINT-Zweig | Informatisches Problemlösen

Lernziel

Ich definiere eine optimale Lösung als beste zulässige Lösung bezüglich eines Kriteriums und beschreibe einen Brute-Force-Ansatz zu ihrer Ermittlung.

🕗 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'	Frage stellen, wie sich die garantiert kürzeste Route finden lässt.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Erarbeitung	25'	Zulässige und optimale Lösung sowie systematisches Ausprobieren aus der TSP-Instanz entwickeln.	Unterrichtsgespräch; Beiträge der SuS werden sichtbar festgehalten
Gemeinsame Umsetzung und Sicherung	20'	Rundreisen geordnet auflisten, Längen berechnen und die optimale Route bestimmen.	Plenum; Live-Coding, Tafelbild oder gemeinsame Simulation
Kurzer Transfer	15'	Anzahl der Rundreisen für kleine Städtezahlen ermitteln.	Einzelarbeit mit anschließendem Tandemabgleich (4 Tandems + 1 Trio)
Auswertung	10'	Brute Force mit Garantie und schnell wachsendem Aufwand erklären lassen.	Plenum; Lösung erklären und Fachsprache sichern
Reserve / Vertiefung	10'	Python-Auswertung oder eigene Fünf-Städte-Instanz ausschließlich optional.	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

- Eine **zulässige Lösung** erfüllt alle Bedingungen des Problems (z. B. "besucht jede Stadt genau einmal").
- Eine **optimale Lösung** ist unter allen zulässigen Lösungen diejenige, die bezüglich eines festgelegten Kriteriums (z. B. Gesamtlänge) **am besten** ist.
- Ein **Brute-Force-Ansatz** ("rohe Gewalt") ermittelt die optimale Lösung, indem er **alle** zulässigen Lösungen systematisch erzeugt, bewertet und die beste auswählt.
- Vorteil: Brute-Force liefert garantiert die **optimale** Lösung.
- Nachteil: Die Anzahl der zu prüfenden Lösungen wächst bei vielen Problemen sehr schnell (z. B. **faktoriell** beim TSP) – Brute-Force wird für größere Instanzen schnell unpraktikabel.

💡 Beispiel: TSP von Hand mit Brute-Force

Wir verwenden die Entfernungstabelle aus DS 23 (4 Städte A, B, C, D). Berechnet **von Hand** die Gesamtlänge für die Rundreise `A → B → C → D → A`:

`10 (A-B) + 35 (B-C) + 30 (C-D) + 20 (D-A) = 95`

Wiederholt das für mindestens zwei weitere Rundreisen und vergleicht die Längen.

(Eure Berechnungen hier)

🐍 Kontrolle: Brute-Force in Python

Die folgende Zelle probiert **alle** Rundreisen aus und findet die optimale Lösung – vergleicht das Ergebnis mit euren Handrechnungen.

```python
from itertools import permutations

entfernungen = {
    ("A", "B"): 10, ("A", "C"): 15, ("A", "D"): 20,
    ("B", "C"): 35, ("B", "D"): 25, ("C", "D"): 30,
}

def entfernung(ort1, ort2):
    if (ort1, ort2) in entfernungen:
        return entfernungen[(ort1, ort2)]
    return entfernungen[(ort2, ort1)]

def rundreise_laenge(reise):
    laenge = 0
    for i in range(len(reise) - 1):
        laenge += entfernung(reise[i], reise[i + 1])
    laenge += entfernung(reise[-1], reise[0])
    return laenge

staedte = ["A", "B", "C", "D"]
beste_reise = None
kuerzeste_laenge = None

for permutation in permutations(staedte[1:]):
    reise = [staedte[0]] + list(permutation)
    laenge = rundreise_laenge(reise)
    print(reise, "->", laenge)
    if kuerzeste_laenge is None or laenge < kuerzeste_laenge:
        kuerzeste_laenge = laenge
        beste_reise = reise

print("\nOptimale Rundreise:", beste_reise, "mit Länge", kuerzeste_laenge)
```

```
['A', 'B', 'C', 'D'] -> 95
['A', 'B', 'D', 'C'] -> 80
['A', 'C', 'B', 'D'] -> 95
['A', 'C', 'D', 'B'] -> 80
['A', 'D', 'B', 'C'] -> 95
['A', 'D', 'C', 'B'] -> 95

Optimale Rundreise: ['A', 'B', 'D', 'C'] mit Länge 80
```

🚀 Optionale Vertiefung – Neue Probleminstanz mit Python

Stellt eine eigene Entfernungstabelle für **5 Städte** auf (frei erfunden, plausible Werte) und passt den Python-Code oben entsprechend an, um die optimale Rundreise zu finden.

📝 Übung 2 – Wachstum beobachten

Zählt die Anzahl der geprüften Rundreisen bei 4, 5 und 6 Städten (Tipp: Es sind `(n-1)!` Reisen bei `n` Städten, wenn man die Startstadt festhält). Was fällt euch am Wachstum auf?

(Eure Beobachtung hier)

✅ Hinweis zu Übung 1

Kopiert dazu einfach den `entfernungen`-Dictionary-Block von oben, fügt die fünfte Stadt mit allen nötigen Paar-Entfernungen hinzu und erweitert die `staedte`-Liste.