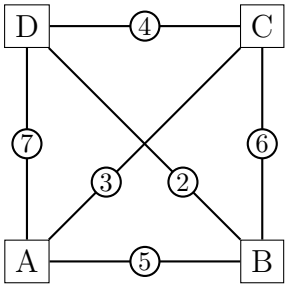
Das Travelling Salesman Problem (TSP)

Die Aufgabe

Ein Handelsvertreter soll 4 Städte besuchen und wieder in die zuerst besuchte Stadt zurückkehren. Bestimme die kürzeste Route.

Städtemodell



Distanztabelle

	A	B	C	D
A	0	5	3	7
B	5	0	6	2
C	3	6	0	4
D	7	2	4	0

Beispiele

Länge der Tour ABCDA: $5 + 6 + 4 + 7 = 22$

Länge der Tour ACDBA: $3 + 4 + 2 + 5 = 14$

Gibt es noch kürzere Wege?

Die Brute Force-Methode

Wie berechnen die Längen aller möglichen Touren und wählen die kürzeste Tour aus.

Die Menge aller Touren erhalten wir, indem wir alle möglichen Anordnungen der Städte bestimmen. Eine Anordnung von n Elementen auf n Plätzen wird *Permutation* von n Elementen genannt.

Ohne Beschränkung der Allgemeinheit können wir bei jeder Tour die Stadt A als Ausgangspunkt wählen.

Menge aller Permutationen von BCD: (Anzahl: $3 \cdot 2 \cdot 1 = 3!$)

BCD CBD DBC
BDC CDB DCB

Allgemein: Bei n Elementen gibt es $n!$ Möglichkeiten.

Laufzeitkomplexität

Wählt man wieder eine beliebige Stadt als Ausgangspunkt, so sind bei n Städten noch $(n-1)!$ Permutationen durchzurechnen.

Dabei ist für jede Permutation die Summe von n Einzeldistanzen zu addieren.

Dies ergibt eine Laufzeitkomplexität von:

$$\mathcal{O}((n-1)!) \cdot \mathcal{O}(n) = \mathcal{O}((n-1)! \cdot n) = \mathcal{O}(n!)$$

Die Distanztabelle in Python

Als Datenstruktur für die Distanzen wählen wir eine Liste von Listen (LoL), wobei die Indizes 0, 1, 2, 3 für die Städte A, B, C, D stehen. In unserem Beispiel:

```
D = [
  [0, 5, 3, 7],
  [5, 0, 6, 2],
  [3, 6, 0, 4],
  [7, 2, 4, 0],
]
```

Distanz zwischen den Städten 1 und 3: `D[1][3] = 2`

Eine Tour in Python

Wählen wir (willkürlich) die Stadt mit dem Index 0 ans Start- und Enpunkt der Tour, so genügt es, eine Rundtour durch n Städte als Liste mit $n-1$ Elementen darzustellen:

ACDBA $\Leftrightarrow$ 0, 2, 3, 1, 0 $\Leftrightarrow$ `T=[2, 3, 1]`

Die Tourlänge berechnen

```
length(ACDBA) = length(0,2,3,1,0)
              = D[0][2]+D[2][3]+D[3][1]+D[1][0]
```

Folgende Funktion berechnet die Länge einer Tour `T` (Liste):

```
def roundTripLength(D, T):
    length = D[0][T[0]] # erste Etappe
    for i in range(0, len(T)-1):
        length += D[T[i]][T[i+1]]
    length += D[T[-1]][0] # zurück an Start!
    return length
```

Erzeugung der Permutationen

Das Generieren aller Permutationen einer Liste ist nicht ganz einfach. Deshalb verwenden wir das Python-Modul `itertools`, das alle Permutationen einer Liste `L` mit der Methode `permutations(L)` erzeugt und als sogenannten Iterator zurückgibt.

```
from itertools import permutations
for p in permutations([1,2,3]):
    print(p)
```

Über eine solche Schleife lassen sich alle Tour-Längen berechnen und die aktuell kleinste herausfiltern.