Aufgabe 5: „Stadtführung“

Team-ID: 00128

Team-Name: E29C8CF09F8E89

Bearbeiter/-innen dieser Aufgabe:
Leonhard Masche

7. November 2023

Inhaltsverzeichnis

1 Lösungsidee

Da der Startpunkt der Tour egal ist, und nur essentielle Orte („Knoten“) bestehen bleiben müssen, werden alle nonessentiellen Knoten vom Anfang der Tour entfernt. Nun werden alle Start- und End-Indices von Subtouren generiert. Dazu wird von jedem Index in der Tour (linker Pointer) aus ein zweiter Pointer inkrementiert, bis entweder: ein

Knoten mit demselben Namen gefunden wurde (die Subtour wird hinzugefügt), oder ein essentieller Knoten gefunden wurde. Aus diesen Subtouren wird (entsprechend dem Weighted Interval Scheduling Problem) die größtmögliche Reihenfolge von Subtouren ermittelt. Diese werden nun entfernt, und das Ergebnis ausgegeben.

2 Umsetzung

Das Programm (`program.py`) ist in Python umgesetzt und mit einer Umgebung ab der Version 3.8 ausführbar.

Beim Ausführen der Datei wird zuerst nach der Zahl des Beispiels gefragt. Dieses wird nun aus der Datei `input/tour{n}.txt` geladen und bearbeitet. Die Tour wird als chronologisch sortierte Liste gespeichert. Die Subtouren werden (wie beschrieben im vorherigen Abschnitt) durch zwei verschachtelte `for`-Schleifen ermittelt. Durch dynamic programming wird die längste Kombination von Subtouren ermittelt, um diese dann aus der Liste zu entfernen.[1] Zum Schluss wird die gekürzte Tour in die Konsole ausgegeben. Das Programm läuft mit einer Zeitkomplexität von $\mathcal{O}(n^2)$ und alle Beispiele werden in unter 0.05ms bearbeitet.

3 Beispiele

Hier wird das Programm auf die 5 Beispiele von der Website angewendet:

- **tour1.txt**

```
Berechnet in 0.02ms


Tour:   (Gesamtlänge 1020)
1. Brauerei (1613)
2. Karzer (1665)
3. Rathaus (1678,1739)
4. Euler-Brücke (1768)
5. Fibonacci-Gaststätte (1820)
6. Schiefes Haus (1823)
```

[1] Vgl. Lalla Mouatadid. *Dynamic Programming: Weighted Interval Scheduling*. URL: `https://www.cs.toronto.edu/~lalla/373s16/notes/WIS.pdf` (besucht am 07. 11. 2023).

```
7. Theater (1880)
8. Emmy-Noether-Campus (1912,1998)
9. Euler-Brücke (1999)
10. Brauerei (2012)
```

- **tour2.txt**

```
Berechnet in 0.03ms

Tour:   (Gesamtlänge 940)
1. Karzer (1665)
2. Rathaus (1678,1739)
3. Euler-Brücke (1768)
4. Fibonacci-Gaststätte (1820)
5. Schiefes Haus (1823)
6. Theater (1880)
7. Emmy-Noether-Campus (1912,1998)
8. Euler-Brücke (1999)
9. Brauerei (2012)
```

- **tour3.txt**

```
Berechnet in 0.02ms

Tour:   (Gesamtlänge 1220)
1. Observatorium (1874)
2. Piz Spitz (1898)
3. Panoramasteg (1912,1952)
4. Ziegenbrücke (1979)
5. Talstation (2005)
```

- **tour4.txt**

```
Berechnet in 0.03ms

Tour:   (Gesamtlänge 1640)
1. Dom (1596)
2. Bogenschütze (1610,1683)
3. Schnecke (1698)
```

4. Fischweiher (1710)
5. Reiterhof (1728)
6. Schnecke (1742)
7. Schmiede (1765)
8. Große Gabel (1794,1874)
9. Fingerhut (1917)
10. Stadion (1934)
11. Marktplatz (1962)
12. Baumschule (1974)
13. Polizeipräsidium (1991)
14. Blaues Pferd (2004)

- **tour5.txt**

Berechnet in 0.04ms

Tour: (Gesamtlänge 2460)
1. Hexentanzplatz (1703)
2. Eselsbrücke (1711)
3. Dreibannstein (1724,1752)
4. Schmetterling (1760)
5. Dreibannstein (1781)
6. Märchenwald (1793,1840)
7. Eselsbrücke (1855,1877)
8. Reiterdenkmal (1880)
9. Riesenrad (1881,1902)
10. Dreibannstein (1911)
11. Olympisches Dorf (1924)
12. Haus der Zukunft (1927)
13. Stellwerk (1931,1942)
14. Labyrinth (1955)
15. Gauklerstadl (1961)
16. Planetarium (1971)
17. Känguruhfarm (1976)
18. Balzplatz (1978)
19. Dreibannstein (1998)

20. Labyrinth (2013)
21. CO2-Speicher (2022)
22. Gabelhaus (2023)

4 Quellcode

program.py

```python
import os
from typing import Dict, List, Tuple
from time import time


def load_tour(
    path: str,
) -> List[Tuple[str, int, bool, int]]:
    """
    Öffne ein Beispiel und gebe die Tour zurück.

    Parameters
    ----------
    path : str
        Der Dateipfad der Beispieldatei relativ zur `program.py`-Datei.

    Returns
    -------
    List[Tuple[str, int, bool, int]]
        Chronologisch sortierte Liste mit den Punkten der Tour.
        Ein Punkt ist ein Tuple mit:
        - dem Name des Ortes
        - dem Jahr der Besichtigung
        - ob der Ort essentiell ist
        - dem Abstand zum vorherigen Punkt
    """
    with open(os.path.join(os.path.dirname(__file__), path), "r", encoding="utf8") as
    ↪  f:
        # Dimensionen einlesen
        n = int(f.readline())

        tour = []
        for i in range(n):
```

```
            # Lesen einer Zeile
            name, year, ess, dist = f.readline().split(",")
            # Konvertieren zu Datentypen
            year = int(year)
            ess = ess == "X"
            dist = int(dist)
            # Hinzufügen zu Tour
            tour.append([name, [year], ess, dist])

        if not tour == (tour_ := list(sorted(tour, key=lambda x: x[1][0]))):
            print(
                "Die Eingabe-Tour war nicht chronologisch sortiert. "
                "Das könnte zu Problemen bei der kumulativen Distanz führen. "
                "Anyways, ..."
            )
            tour = tour_
    return tour


def main(tour: List[Tuple[str, List[int], bool, int]]):
    timed = time()  # Zeitmessung

    offset_dist = 0  # Variable, um entfernte Strecke zu speichern

    # Unwichtige Knoten vom Anfang entfernen
    x = 0
    while not tour[x][2]:
        x += 1
    offset_dist -= tour[x][3]
    tour = tour[x:]

    # Subtouren finden
    subtours: Dict[int, Tuple[int, int]] = {}  # {j: (i, v)}
    for i in range(len(tour) - 1):
        for j in range(i + 1, len(tour)):
            if tour[i][0] == tour[j][0]:
                subtours[j] = (i, tour[j][3] - tour[i][3])
                break
            elif tour[j][2]:
                break

    # Beste Subtouren-Kombination finden (Weighted Interval Scheduling)
```

```
    best = {-1: (0, [])}  # 1: (value, ((i, j), (i, j)))
    for j in range(len(tour)):
        if j in subtours:
            i, v = subtours[j]
            best[j] = max(
                best[j - 1], (best[i][0] + v, best[i][1] + [(i, j)]), key=lambda x:
                ↪ x[0]
            )
        else:
            best[j] = best[j - 1]  # nur leerraum

    # Subtouren entfernen
    for i, j in reversed(best[len(tour) - 1][1]):
        tour[j][1] = tour[i][1] + tour[j][1]
        offset_dist += tour[i][3] - tour[j][3]
        tour = tour[:i] + tour[j:]

    print(f"Berechnet in {(time() - timed)*1000:.2f}ms\n")
    # Tour ausgeben
    print(f"Tour:   (Gesamtlänge {tour[-1][3] + offset_dist})")
    for i, (name, years, *_) in enumerate(tour):
        print(f"{i+1}. {name} ({','.join(map(str, years))})")


# Haupt-Loop
while True:
    try:
        n_bsp = int(input("Bitte Nummer des Beispiels eingeben:\n> "))
        tour = load_tour(f"input/tour{n_bsp}.txt")
        print()
        main(tour)
    except TimeoutError as e:  # Error-Handling
        print(f"{e.__class__.__name__}: {e}")
    print()
```