Aufgabe 5: Stadtführung

Teilnahme-ID: 69408

Bearbeiter dieser Aufgabe:  
Tim Krome

2. November 2023

[Lösungsidee 1](#_Toc151246612)

[Vorüberlegung 1](#_Toc151246613)

[Finden der kürzesten verkürzten Route 2](#_Toc151246614)

[Verändern des Anfangs- bzw. Endpunkts 2](#_Toc151246615)

[Umsetzung 3](#_Toc151246616)

[Einlesen der Eingabedatei 3](#_Toc151246617)

[Definieren einer Klasse zum Speichern von Routen 3](#_Toc151246618)

[Finden der kürzesten verkürzten Route 4](#_Toc151246619)

[Ausgabe des Wegs 6](#_Toc151246620)

[Beispiele 6](#_Toc151246621)

[Beispiel 1 6](#_Toc151246622)

[Beispiel 2 7](#_Toc151246623)

[Beispiel 3 8](#_Toc151246624)

[Beispiel 4 8](#_Toc151246625)

[Beispiel 5 9](#_Toc151246626)

[Quellcode 10](#_Toc151246627)

# Lösungsidee

## Vorüberlegung

Es soll laut Aufgabenstellung die kürzeste verkürzte Route bestimmt werden. Es kann mehrere, zueinander im Konflikt stehende Möglichkeiten geben, geschlossene Teilrouten wegzulassen. Beispielsweise kann in der Route A1\*-B2-C4-D5-B6-D7-A8\* entweder die Teilroute (D5)-D6-(D7) oder die Teilroute (B2)-C4-D5-(B6) weggelassen werden[[1]](#footnote-1). Daher müssen alle möglichen verkürzten Routen ermittelt werden und die kürzeste Route aus allen möglichen verkürzten Routen muss ausgewählt werden.

Zum Verknüpfen der Tourpunkte zu einer Route verwende ich ein iteratives Verfahren. Zunächst definiere ich zwei Begriffe:

* *Geschlossene Route:* Eine Teilroute der ursprünglichen Route, die in der nächsten Iteration fortgesetzt werden kann.
* *Geöffnete Route:* Eine Teilroute der ursprünglichen Route, die zu einem Ort X führt und erst wieder fortgesetzt bzw. zu einer geschlossenen Route umgewandelt werden kann, wenn Ort X erneut erreicht wird.

## Finden der kürzesten verkürzten Route

Angenommen, die ursprüngliche Route beginnt und endet mit einem essentiellen Punkt, dann kann folgendes Verfahren angewendet werden, um die kürzeste verkürzte Route zu finden:

1. Die Routenpunkte der ursprünglichen Route werden in chronologischer Reihenfolge sortiert.
2. Eine zunächst leere Menge für mögliche Teilrouten wird initialisiert, die Menge wird im Folgenden als *possible\_routes* bezeichnet.
3. Zur Menge *possible\_routes* wird eine geschlossene Teilroute hinzugefügt, die nur aus dem ersten Routenpunkt der ursprünglichen Route besteht.
4. Anschließend wird über alle Routenpunkte in chronologischer Reihenfolge iteriert. In jeder Iteration wird folgendes durchgeführt:  
     
   Jede **geschlossene** Teilroute aus *possible\_routes* wird mit dem derzeit betrachteten Routenpunkt fortgesetzt. Auch für jede **geöffnete** Teilroute aus *possible\_routes* wird versucht, sie mit dem derzeit betrachteten Routenpunkt fortzusetzen und somit in eine geschlossene Route umzuwandeln (dies ist nur möglich, wenn der letzte besuchte Ort der Route dem Ort des aktuell betrachteten Routenpunkts entspricht). Durch das Schließen und Fortsetzen einer geöffneten Route entsteht eine verkürzte Route, in der ein geschlossenes Teilstück weggelassen wurde.   
     
   Anschließend wird für jede geschlossene Route aus *possible\_routes* eine **Kopie** der jeweiligen Route erstellt und als **geöffnete** Route zu *possible\_routes* hinzugefügt. So wird sozusagen ein Vermerk gesetzt, dass ab dem aktuell betrachteten Tourpunkt ein geschlossenes Teilstück weggelassen werden kann, wenn in einer späteren Iteration erneut auf den Ort des aktuell betrachteten Tourpunkts getroffen wird.

Sollte es sich beim derzeit betrachteten Routenpunkt um einen **essentiellen Routenpunkt** handeln, dann werden alle Teilrouten bis auf die kürzeste geschlossene Route aus *possible\_routes* entfernt, da der derzeit betrachtete Tourpunkt ein essentieller Tourpunkt ist, somit auf jeden Fall besucht werden muss und daher nur der kürzeste Weg, der zu diesem Tourpunkt führt, der Anfang der kürzesten Gesamtroute sein kann.

1. Da vorausgesetzt wurde, dass die ursprüngliche Route auf einem essentiellen Tourpunkt endet, enthält die Menge *possible\_routes* am Ende genau eine verkürzte Teilroute, bei der es sich um die kürzeste verkürzte Route handelt, die trotzdem noch alle essentiellen Punkte abdeckt.

## Verändern des Anfangs- bzw. Endpunkts

Es ist es allerdings auch möglich, dass die Route mit nicht essentiellen Tourpunkten beginnt und endet. Die verkürzte Route muss nach wie vor denselben Anfangs- wie Endort haben, dieser darf allerdings durch das Weglassen von nicht essentiellen Tourpunkten am Anfang und Ende der Route verändert werden. Beispielsweise darf die Route A1-B2-C3-D4\*-C5-A6 zu C3-D4\*-C5 verkürzt werden (alter Anfang- und Endort war A, neuer Anfangs- und Endort ist C). Da die ursprüngliche Route ein Rundgang ist, handelt es sich auch hierbei letztendlich um das Weglassen einer geschlossenen Teilroute (im obigen Beispiel wurde die geschlossene Teilroute (C5)-A6-A1-B2-(C3) weggelassen). Wenn die ursprüngliche Route mit einem nicht essentiellen Tourpunkt beginnt und / oder endet, dann kann das Finden der kürzesten verkürzten Route so umgesetzt werden:

1. Zuerst wird die ursprüngliche Route in zwei Teilrouten aufgeteilt. Die erste Teilroute geht vom ersten Routenpunkt bis zum ersten essentiellen Routenpunkt, die zweite Teilroute geht vom ersten essentiellen Routenpunkt bis zum letzten Routenpunkt.
2. Anschließend werden unter Verwendung des im vorherigen Kapitel beschriebenen Verfahrens alle Möglichkeiten ermittelt, die erste Teilroute zu verkürzen, es werden dabei alle Punkte der ersten Teilroute als möglicher Anfangspunkt der Teilroute durchprobiert.
3. Danach werden unter Verwendung des im vorherigen Kapitel beschriebenen Verfahrens alle Möglichkeiten ermittelt, die zweite Teilroute zu verkürzen, es werden dabei alle möglichen Endpunkte als Endpunkt der Teilroute durchprobiert (der letzte essentielle Routenpunkt und alle Punkte nach dem letzten essentiellen Punkt sind mögliche Endpunkte).
4. Zuletzt werden alle Möglichkeiten ermittelt, je zwei verkürzte Teilrouten wieder zu einer Gesamtroute zusammenzufügen, bei der Anfangs- und Endort gleich sind. Die kleinste mögliche Gesamtroute ist die kürzeste verkürzte Route bzw. die Route, die ermittelt werden soll.

# Umsetzung

Die Lösungsidee wird in Python 3.10 implementiert. Zum Kopieren von Datenstrukturen verwende ich die *deepcopy* Methode des Python-Standard-Libraries *copy*.

## Einlesen der Eingabedatei

Die Eingabedatei wird vom Nutzer als erstes Kommandozeilenargument angegeben und vom Programm eingelesen.

from copy import deepcopy

import sys

# Textdatei einlesen

# sys.argv[1] ist das erste Kommandozeilenargument

with open(sys.argv[1]) as f:

input\_lines = f.read().split("\n")  
  
Anschließend werden die Tourpunkte als Quadrupel der Form *(Ort, Jahr, ist\_essentiell[[2]](#footnote-2), kumulierte\_Distanz)* eingelesen und in einer Liste namens *tourpunkte* gespeichert. Die Liste wird anschließend so sortiert, dass die Tourpunkte in ihr in chronologischer Reihenfolge vorliegen (Der erste Tourpunkt der Liste ist demnach der mit der niedrigsten Jahreszahl).

num\_tourpunkte = int(input\_lines.pop(0).strip())

tourpunkte = []

for i in range(num\_tourpunkte):

line = input\_lines.pop(0).split(",")

tourpunkte.append([line[0], line[1], line[2], int(line[3].strip())])

tourpunkte = sorted(tourpunkte, key = lambda k : int(k[1]))

## Definieren einer Klasse zum Speichern von Routen

Damit eine Teilroute gespeichert werden kann, wird eine Klasse namens *Route* definiert, die diese Attribute hat:

* *Route.orte : list<str> -* Enthält die Namen aller besuchten Orte in der Reihenfolge, in der sie besucht werden
* *Route.jahre : list<list<str>>* - Enthält zu jedem besuchten Ort aus *Route.orte* die zum Routenpunkt gehörigen Jahre
* *Route*.*length : int* – Die Gesamtlänge der Route
* *Route.geschlossen : boolean –* Gibt an, ob bei der Route geschlossen ist (siehe hierzu Kapitel *Lösungsidee*)

Die Klasse hat die Methode *add\_punkt(ort, jahr, distance)*, die einen Punkt zur Route hinzufügt:

def add\_punkt(self, ort, jahr, distance):

"""

Fügt einen Punkt zur Route hinzu

"""

self.orte.append(ort)

self.jahre.append([jahr])

self.length += distance

Die Klasse hat die Methoden *oeffnen() und schliessen(ort)*:  
 def schliessen(self, jahr):

"""

Schließt die Route

"""

self.jahre[-1].append(jahr)

self.geschlossen = True

def oeffnen(self):

"""

Erstellt eine Kopie der Route, die geöffnet ist

"""

return Route(deepcopy(self.orte), deepcopy(self.jahre), int(self.length), geschlossen=False)

Die Klasse hat außerdem eine *print()* Methode, die die Route formatiert in der Konsole ausgibt.  
Es wird außerdem eine *\_\_lt\_\_(obj2)* Funktion implementiert, sodass zwei *Route*-Objekte mit dem „<“-Vergleichsoperator verglichen werden ka Es wird außerdem eine *\_\_lt\_\_(obj2)* Funktion implementiert, sodass zwei *Node*-Objekte mit dem „<“-Vergleichsoperator verglichen werden kann[[3]](#footnote-3). Die Funktion ist so programmiert, dass das Objekt mit dem kleineren Zeitstempel dabei das kleinere Objekt ist. Dies ermöglicht es, mit *min(list<Node>)* auf das Node-Objekt mit dem kleinsten Zeitstempel zuzugreifen.

## Finden der kürzesten verkürzten Route

Das in der Lösungsidee beschriebene Vorgehen wird umgesetzt. Hierfür wird chronologisch über alle Routenpunkte iteriert. Zuerst wird die erste Teilroute, die bis zum ersten essentiellen Routenpunkt geht, betrachtet. Es werden alle Möglichkeiten versucht, diese Teilroute zu verkürzen, und in der Liste *possible\_routes* gespeichert.  
Sobald der erste essentielle Routenpunkt erreicht wird, wird *before\_first\_essential* auf *False* gesetzt, eine Kopie von *possible\_routes* wird unter dem Namen *possible\_routes\_before\_first\_essential* gespeichert und *possible\_routes* wird auf eine leere Liste zurückgesetzt. Danach werden alle Möglichkeiten, die zweite Teilroute zu verkürzen, in *possible\_routes* gespeichert.  
So erhält man eine Liste *possible\_routes\_before\_first\_essential*, in der alle Möglichkeiten, die erste Teilroute zu verkürzen, gespeichert sind. Man erhält außerdem eine Liste *possible\_routes*, in der alle Möglichkeiten, die zweite Teilroute zu verkürzen, gespeichert sind.  
Der Programmteil, der dies durchführt:  
  
before\_first\_essential = True # Bleibt so lange True, bis der erste essentielle

Tourpunkt erreicht wird

possible\_routes = [] # In dieser Liste werden alle möglichen Routen gespeichert

length = 0

# Über alle Tourpunkte iterieren:

for tourpunkt in tourpunkte:

ort, jahr = tourpunkt[0], tourpunkt[1]

essential = tourpunkt[2] == "X"

distance = tourpunkt[3] - length # Abstand zu vorherigen Tourpunkt berechnen

length = tourpunkt[3]

next\_possible\_routes = []

if before\_first\_essential: # Falls der erste essentielle Tourpunkt noch nicht

überschritten wurde, kann beim aktuellen Tourpunkt

eine Route beginnen

next\_possible\_routes.append(Route([ort],[[jahr]],0,geschlossen=True))

for route in possible\_routes: # Über alle möglichen Routen iterieren und versuchen,

den aktuellen Tourpunkt hinzuzufügen

if essential is False:

next\_possible\_routes.append(route.oeffnen())

if route.geschlossen:

route.add\_punkt(ort, jahr, distance)

next\_possible\_routes.append(route)

else:

if ort == route.orte[-1]:

route.schliessen(jahr)

next\_possible\_routes.append(route)

if essential:

if before\_first\_essential:

# -> Es handelt sich beim aktuellen Tourpunkt um den ersten essentiellen

Tourpunkt

possible\_routes\_before\_first\_essential = deepcopy(next\_possible\_routes)

# Alle mögliche Routen vom Anfang zum ersten essentiellen

Tourpunkt werden separat gespeichert, damit später

verschiedene Anfangsorte durchprobiert werden können

next\_possible\_routes = [Route([ort], [[]], 0, geschlossen=True)]

before\_first\_essential = False # Erster essentieller Tourpunkt wurde

überschritten, also Variable auf False setzen

next\_possible\_routes = [min(next\_possible\_routes)] # Da es sich beim aktuellen

Tourpunkt um einen essentiellen Tourpunkt handelt, müssen alle Routen früher oder später durch diesen Punkt gehen, es muss also nur die bisher   
kürzeste Route weiter betrachtet werden

possible\_routes = next\_possible\_routes

Es werden anschließend alle Möglichkeiten bestimmt, die verkürzten Teilrouten aus *possible\_routes\_before\_first\_essential* mit den verkürzten Teilrouten aus *possible\_routes* zu einer Gesamtroute zusammenzuführen:

possible\_routes\_complete = []

for route2 in possible\_routes:

for route1 in possible\_routes\_before\_first\_essential:

route = Route(route1.orte[:-1]+route2.orte, route1.jahre[:-1]+[route1.jahre[-1]+route2.jahre[0]]+route2.jahre[1:], route1.length+route2.length, geschlossen=True)

if route.orte[0] == route.orte[-1]:

possible\_routes\_complete.append(route)

Die kürzeste Gesamtroute wird ausgewählt, es handelt sich bei ihr um die kürzeste verkürzte Gesamtroute, also um die Route, die der Aufgabe nach bestimmt werden soll.

route = min(possible\_routes\_complete)

## Ausgabe des Wegs

Die im vorherigen Programmteil ermittelte kürzeste Gesamtroute wird mitsamt ihrer Länge ausgegeben:

print("Die beste Route sieht so aus:")

route.print()

print("\nLänge der Route:", route.length, "LE")

# Beispiele

Die in Beispiel 1 - 5 verwendeten Eingabedateien stammen von der BWinf-Webseite. Zu Beispiel *x* gehört stets die Eingabedatei tour*x*.txt.

## Beispiel 1

**Eingabedatei:**

tour1.txt

**Inhalt der Eingabedatei:**18

Brauerei,1613,X,0

Karzer,1665,X,80

Rathaus,1678,X,150

Gründungsstein,1685, ,310

Wallanlage,1690, ,410

Rathaus,1739,X,500

Euler-Brücke,1768, ,680

Fibonacci-Gaststätte,1820,X,710

Schiefes Haus,1823, ,830

Theater,1880, ,960

Emmy-Noether-Campus,1912,X,1090

Fibonacci-Gaststätte,1923, ,1180

Hilbert-Raum,1945, ,1300

Schiefes Haus,1950, ,1400

Gauß-Turm,1952, ,1450

Emmy-Noether-Campus,1998,X,1780

Euler-Brücke,1999, ,1910

Brauerei,2012, ,2060

**Ausgabe:**

Die beste Route sieht so aus:

1) Brauerei 1613

2) Karzer 1665

3) Rathaus 1678, 1739

4) Euler-BrÃ¼cke 1768

5) Fibonacci-GaststÃ¤tte 1820

6) Schiefes Haus 1823

7) Theater 1880

8) Emmy-Noether-Campus 1912, 1998

9) Euler-BrÃ¼cke 1999

10) Brauerei 2012  
  
Länge der Route: 1020 LE

**Evalierungsdauer:**0,185 Sekunden

## Beispiel 2

**Eingabedatei:**

tour2.txt

**Inhalt der Eingabedatei:**18

Brauerei,1613, ,0

Karzer,1665,X,80

Rathaus,1678, ,150

Gründungsstein,1685, ,310

Wallanlage,1690, ,410

Rathaus,1739, ,500

Euler-Brücke,1768, ,680

Fibonacci-Gaststätte,1820,X,710

Schiefes Haus,1823, ,830

Theater,1880, ,960

Emmy-Noether-Campus,1912,X,1090

Fibonacci-Gaststätte,1923, ,1180

Hilbert-Raum,1945, ,1300

Schiefes Haus,1950, ,1400

Gauß-Turm,1952, ,1450

Emmy-Noether-Campus,1998,X,1780

Euler-Brücke,1999, ,1910

Brauerei,2012, ,2060

**Ausgabe:**

Die beste Route sieht so aus:

1) Brauerei 1613

2) Karzer 1665

3) Rathaus 1678, 1739

4) Euler-BrÃ¼cke 1768

5) Fibonacci-GaststÃ¤tte 1820

6) Schiefes Haus 1823

7) Theater 1880

8) Emmy-Noether-Campus 1912, 1998

9) Euler-BrÃ¼cke 1999

10) Brauerei 2012

Länge der Route: 1020 LE

**Evalierungsdauer:**0,188 Sekunden

## Beispiel 3

**Eingabedatei:**

tour3.txt

**Inhalt der Eingabedatei:**14

Talstation,1768, ,0

Wäldle,1805, ,520

Mittlere Alp,1823, ,1160

Observatorium,1833, ,1450

Wäldle,1841, ,1700

Bergstation,1866, ,2370

Observatorium,1874,X,2740

Piz Spitz,1898, ,3210

Panoramasteg,1912,X,3430

Bergstation,1928, ,3690

Ziegenbrücke,1935, ,3870

Panoramasteg,1952, ,4030

Ziegenbrücke,1979,X,4280

Talstation,2005, ,4560

**Ausgabe:**

Die beste Route sieht so aus:

1) Talstation 1768

2) WÃ¤ldle 1805

3) Mittlere Alp 1823

4) Observatorium 1833, 1874

5) Piz Spitz 1898

6) Panoramasteg 1912, 1952

7) ZiegenbrÃ¼cke 1979

8) Talstation 2005

Länge der Route: 2670 LE

**Evalierungsdauer:**0,177 Sekunden

## Beispiel 4

**Eingabedatei:**

tour4.txt

**Inhalt der Eingabedatei:**Siehe *A5\_Stadtfuehrung/tour4.txt*

**Ausgabe:**

Die beste Route sieht so aus:

1) Marktplatz 1562

2) Springbrunnen 1571

3) Dom 1596

4) BogenschÃ¼tze 1610, 1683

5) Schnecke 1698

6) Fischweiher 1710

7) Reiterhof 1728

8) Schnecke 1742

9) Schmiede 1765

10) GroÃŸe Gabel 1794, 1874

11) Fingerhut 1917

12) Stadion 1934

13) Marktplatz 1962

Länge der Route: 1420 LE

**Evalierungsdauer:**0,178 Sekunden

## Beispiel 5

**Eingabedatei:**

tour5.txt

**Inhalt der Eingabedatei:**Siehe *A5\_Stadtfuehrung/tour5.txt*

**Ausgabe:**

Die beste Route sieht so aus:

1) Gabelhaus 1699

2) Hexentanzplatz 1703

3) EselsbrÃ¼cke 1711

4) Dreibannstein 1724, 1752

5) Schmetterling 1760

6) Dreibannstein 1781

7) MÃ¤rchenwald 1793, 1840

8) EselsbrÃ¼cke 1855, 1877

9) Reiterdenkmal 1880

10) Riesenrad 1881, 1902

11) Dreibannstein 1911

12) Olympisches Dorf 1924

13) Haus der Zukunft 1927

14) Stellwerk 1931, 1942

15) Labyrinth 1955

16) Gauklerstadl 1961

17) Planetarium 1971

18) KÃ¤nguruhfarm 1976

19) Balzplatz 1978

20) Dreibannstein 1998

21) Labyrinth 2013

22) CO2-Speicher 2022

23) Gabelhaus 2023

Länge der Route: 2620 LE

**Evalierungsdauer:**0,178 Sekunden

# Quellcode

from copy import deepcopy

import sys

# Textdatei einlesen

# sys.argv[1] ist das erste Kommandozeilenargument

with open(sys.argv[1]) as f:

input\_lines = f.read().split("\n")

class Route:

"""

Klasse zum Speichern einer möglichen Route

"""

def \_\_init\_\_(self, orte, jahre, length, \*, geschlossen):

self.orte = orte

self.jahre = jahre

self.length = length

self.geschlossen = geschlossen

def \_\_lt\_\_(self, obj2):

"""

Diese Funktion existiert, damit mit min(list<Route>) die Route mit der kürzersten Strecke ermittelt werden kann

Returns:

boolean: Ob die Route obj2 kürzer ist als die Route self bzw. das Objekt selbst

"""

return self.length < obj2.length

def add\_punkt(self, ort, jahr, distance):

"""

Fügt einen Punkt zur Route hinzu

"""

self.orte.append(ort)

self.jahre.append([jahr])

self.length += distance

def schliessen(self, jahr):

"""

Schließt die Route

"""

self.jahre[-1].append(jahr)

self.geschlossen = True

def oeffnen(self):

"""

Erstellt eine Kopie der Route, die geöffnet ist

"""

return Route(deepcopy(self.orte), deepcopy(self.jahre), int(self.length), geschlossen=False)

def print(self):

"""

Gibt die Route formatiert in der Konsole aus

"""

for i in range(len(self.orte)):

print(str(i+1)+")", self.orte[i], ", ".join(self.jahre[i]))

# Tourpunkte einlesen und chronologisch sortieren:

num\_tourpunkte = int(input\_lines.pop(0).strip())

tourpunkte = []

for i in range(num\_tourpunkte):

line = input\_lines.pop(0).split(",")

tourpunkte.append([line[0], line[1], line[2], int(line[3].strip())])

tourpunkte = sorted(tourpunkte, key = lambda k : int(k[1]))

# Kürzeste Route finden, die den Anforderungen entspricht:

before\_first\_essential = True # Bleibt so lange True, bis der erste essentielle Tourpunkt erreicht wird

possible\_routes = [] # Alle möglichen Routen

length = 0

# Über alle Tourpunkte iterieren:

for tourpunkt in tourpunkte:

ort, jahr = tourpunkt[0], tourpunkt[1]

essential = tourpunkt[2] == "X"

distance = tourpunkt[3] - length # Abstand zu vorherigen Tourpunkt berechnen

length = tourpunkt[3]

next\_possible\_routes = []

if before\_first\_essential: # Falls der erste essentielle Tourpunkt noch nicht überschritten wurde, kann beim aktuellen Tourpunkt eine Route beginnen

next\_possible\_routes.append(Route([ort],[[jahr]],0,geschlossen=True))

for route in possible\_routes: # Über alle möglichen Routen iterieren und versuchen, den aktuellen Tourpunkt hinzuzufügen

if essential is False:

next\_possible\_routes.append(route.oeffnen())

if route.geschlossen:

route.add\_punkt(ort, jahr, distance)

next\_possible\_routes.append(route)

else:

if ort == route.orte[-1]:

route.schliessen(jahr)

next\_possible\_routes.append(route)

if essential:

if before\_first\_essential:

# -> Es handelt sich beim aktuellen Tourpunkt um den ersten essentiellen Tourpunkt

possible\_routes\_before\_first\_essential = deepcopy(next\_possible\_routes) # Alle mögliche Routen vom Anfang zum ersten essentiellen Tourpunkt werden separat gespeichert, damit später verschiedene Anfangsorte durchprobiert werden können

next\_possible\_routes = [Route([ort], [[]], 0, geschlossen=True)]

before\_first\_essential = False # Erster essentieller Tourpunkt wurde überschritten, also Variable auf False setzen

next\_possible\_routes = [min(next\_possible\_routes)] # Da es sich beim aktuellen Tourpunkt um einen essentiellen Tourpunkt handelt, müssen alle Routen früher oder später durch diesen Punkt gehen, es muss also nur die bisher kürzeste Route weiter betrachtet werden

possible\_routes = next\_possible\_routes

# Zusammensetzen aller möglichen Teilrouten von vor / nach dem ersten essentiellen Tourpunkt

possible\_routes\_complete = []

for route2 in possible\_routes:

for route1 in possible\_routes\_before\_first\_essential:

route = Route(route1.orte[:-1]+route2.orte, route1.jahre[:-1]+[route1.jahre[-1]+route2.jahre[0]]+route2.jahre[1:], route1.length+route2.length, geschlossen=True)

if route.orte[0] == route.orte[-1]:

possible\_routes\_complete.append(route)

# Kürzeste zusammengesetzte Route finden

route = min(possible\_routes\_complete)

# Gefundene Route ausgeben

print("Die beste Route sieht so aus:")

route.print()

print("\nLänge der Route:", route.length, "LE")

1. Essentielle Tourpunkte sind hier mit einem \* markiert. [↑](#footnote-ref-1)
2. Bei essentiellen Tourpunkten steht an dieser Stelle „X“, bei nicht essentiellen Tourpunkten steht an dieser Stelle „ “. [↑](#footnote-ref-2)
3. Beim Vergleichen zweier Objekte mit den „<“-Vergleichsoperator oder beim Anwenden von *min()* greift Python auf die *\_\_lt\_\_-*Funktion der Objekte zu. [↑](#footnote-ref-3)