

Aufgabenstellung

Ein Handlungsreisender will seine Produkte in den zehn größten Städten Deutschlands verkaufen. Er startet in Berlin und will seine Reise dort beenden.

Die zehn einwohnerreichsten Städte Deutschlands sind Berlin, Hamburg, München, Köln, Frankfurt am Main, Stuttgart, Düsseldorf, Dortmund, Essen und Bremen.

Folgende Tabelle gibt die Entfernung zwischen den Städten für eine Autoreise wieder¹:

	Berlin	Hamburg	München	Köln	Frankfurt am Main	Stuttgart	Düsseldorf	Dortmund	Essen	Bremen
nach von										
Berlin	0	288	585	575	547	633	559	494	531	392
Hamburg	289	0	775	426	493	655	400	344	365	122
München	589	775	0	577	398	220	612	604	634	748
Köln	579	426	576	0	193	369	42	95	73	320
Frankfurt a. M.	552	492	393	193	0	210	229	219	251	445
Stuttgart	637	667	231	369	203	0	404	417	426	642
Düsseldorf	563	408	611	38	228	404	0	71	37	302
Dortmund	498	346	605	95	221	418	69	0	36	240
Essen	536	374	634	74	252	427	35	37	0	267
Bremen	397	123	748	315	441	638	289	233	254	0

Welche Route sollte er wählen?

1.1 Größenordnung des Problems

Es gibt $9! = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 362.880$ mögliche Routen, da der Handlungsreisende in Berlin startet und 9 Möglichkeiten für die erste Stadt hat, 8 für die zweite, usw.

Allgemein kann man sagen, dass bei n Städten insgesamt $(n-1)!$ mögliche Routen bestehen, da die Wahl des Startpunktes egal ist.

Falls das Problem symmetrisch wäre, also die Strecke von Berlin nach München genauso lang wie die von München nach Berlin wäre, könnte man die Anzahl der Routen auf $\frac{(n-1)!}{2}$ reduzieren. Allerdings ist das Problem offensichtlich asymmetrisch (siehe München \rightarrow Berlin und München \leftarrow Berlin)

Die Tabelle ?? macht deutlich, wie schnell so etwas wächst.

n	1	$\log(n)$	$n \cdot \log(n)$	n^2	n^3	$n!$
1	1	0	0	1	1	1
2	1	0,30	0,60	4	8	2
3	1	0,48	1,43	9	27	6
4	1	0,60	2,41	16	64	24
5	1	0,70	3,49	25	125	120
6	1	0,78	4,67	36	216	720
7	1	0,85	5,92	49	343	5.040
8	1	0,90	8,13	64	512	40.320
9	1	0,95	8,59	81	729	362.880
10	1	1,00	10,00	100	1.000	3.628.800
20	1	1,30	26,02	400	8.000	$2,42 \cdot 10^{18}$

Tabelle 1: Wachstum verschiedener Funktionen

¹Mit maps.google.com ermittelte Werte. Es wurde immer auf ganze Kilometer gerundet.

1.2 Intuitive Lösung

Wenn man die Städte so auf der Karte sieht sollte man einfach mal die Route suchen, von der man denkt sie wäre gut.

Ich habe mir folgende ausgesucht:

Das ist die Route Berlin → München → Stuttgart → Frankfurt a. M. → Köln → Düsseldorf → Essen → Dortmund → Bremen → Hamburg → Berlin. Diese Route ist 1969 km lang.

1.3 Optimale Lösung

Alle Routen durchgehen

Die primitivste und langsamste Methode zum Finden der optimalen Lösung ist einfach alle möglichen Routen durch zu gehen. Dieser Algorithmus ist, was die Ausführungszeit angeht, in $\mathcal{O}(n!)$. Wie schnell diese wächst sollte mit Tabelle ?? klar geworden sein.

Dieser Ansatz ist also nur für sehr kleine Instanzen des Problems geeignet.

Eine Auswertung aller Strecken hat folgende Ergebnisse geliefert:

1969 km bzw. die von mir intuitiv gefundene Strecke ist eine von 10 optimalen Lösungen.

Die längste Route ist 4.913 km lang.

Es gibt 3.628.800 Routen, jedoch nur 2.597 verschiedene Streckenlängen. Die durchschnittliche Streckenlänge beträgt 3.838 km und wurde rot eingezeichnet, die maximale 4.913 km.

Die Verteilung der Streckenlängen sieht wie folgt aus:

Würde man eine maximale Abweichung von 5% akzeptieren, wären nur 0,002% aller Routen akzeptabel. Durch eine zufällig gewählte Strecke wird man also kaum ein gutes Ergebnis erzielen

1.4 Heuristiken

Eine Heuristik ist in diesem Zusammenhang ein vereinfachtes Verfahren, das keine optimale Lösung liefert, aber eine akzeptable Näherung daran. Dafür ist die Heuristik deutlich schneller.

Nächster Nachbar

Eine mögliche Heuristik ist das auswählen der Stadt, die am nächsten zur aktuellen Stadt liegt. Mit diesem Verfahren erhält man bei der gegebenen Städtekonstellation eine Route der Länge 2.162 km. Das sind 193 km oder 9,8% mehr als die optimale Lösung benötigt.

Nächster Nachbar - Dynamisch Einfügen

Die Methode des nächsten Nachbarn kann verbessert werden, indem der Nachbar nicht einfach am Ende in die Route eingefügt wird, sondern an die Stelle, an der die resultierende Route am kleinsten wäre. Damit erhält man bei der gegebenen Städtekonstellation eine optimale Route.

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 """ Dieses Script berechnet die Länge aller möglichen Routen und speichert sie
5 in "Entfernungen.txt".
6 Am Ende wird noch die optimale Route ausgegeben.
7
8 Benötigte Rechenzeit: ca. 50s """
9
10 from copy import deepcopy
11
12 # http://blog.bjrn.se/2008/04/lexicographic-permutations-using.html
13 def next_permutation(seq, pred=cmp):
14     """Like C++ std::next_permutation() but implemented as
15     generator. Yields copies of seq."""
16     def reverse(seq, start, end):
17         # seq = seq[:start] + reversed(seq[start:end]) + \
18         #     seq[end:]
19         end -= 1
20         if end <= start:
21             return
22         while True:
23             seq[start], seq[end] = seq[end], seq[start]
24             if start == end or start+1 == end:
25                 return
26             start += 1
27             end -= 1
28     if not seq:
29         raise StopIteration
30     try:
31         seq[0]
32     except TypeError:
33         raise TypeError("seq must allow random access.")
34     first = 0
35     last = len(seq)
36     seq = seq[:]
37     # Yield input sequence as the STL version is often
38     # used inside do {} while.
39     yield seq
40     if last == 1:
41         raise StopIteration
42     while True:
43         next = last - 1
44         while True:
45             # Step 1.
46             next1 = next
47             next -= 1
48             if pred(seq[next], seq[next1]) < 0:
49                 # Step 2.
50                 mid = last - 1
51                 while not (pred(seq[next], seq[mid]) < 0):
52                     mid -= 1
53                 seq[next], seq[mid] = seq[mid], seq[next]
54                 # Step 3.
55                 reverse(seq, next1, last)
56                 # Change to yield references to get rid of
57                 # (at worst) |seq|! copy operations.
58                 yield seq[:]
59                 break
60             if next == first:
61                 raise StopIteration
62         raise StopIteration
63
64 def Strecke_der_Route(Entfernungen, route):
65     """ Bestimmt die Länge der Route und gibt diese als int zurück """
66     Entfernung = 0
67     for step, index in enumerate(route):
68         index2 = route[(step+1)%len(Entfernungen)]
69         Entfernung += Entfernungen[index][index2] # von index nach index2
70     return Entfernung
71
72 def optimal_solution_with_brute_force(Entfernungen):
73     """ Findet die optimale Lösung, indem alle Routen durchgegangen werden.
74     Zurückgegeben wird eine Liste mit den Indizes """
75     liste = [i for i in xrange(0, len(Entfernungen))]
76     min_Entfernung_Route = [i for i in xrange(0, len(Entfernungen))]
77     min_Entfernung = Strecke_der_Route(Entfernungen, min_Entfernung_Route)
78     f = open('/home/moose/Entfernungen.txt', 'w')
79     for route in next_permutation(liste):
80         entfernung_tmp = Strecke_der_Route(Entfernungen, route)
81         f.write(str(entfernung_tmp) + "\n")
82         if entfernung_tmp < min_Entfernung:
83             min_Entfernung = entfernung_tmp
84             min_Entfernung_Route = deepcopy(route)
85     f.close()
86     return min_Entfernung_Route
87
88 Stadtliste = ['Berlin', 'Hamburg', 'München', 'Köln', 'Frankfurt a. M.',
89               'Stuttgart', 'Düsseldorf', 'Dortmund', 'Essen', 'Bremen']
90 Entfernungen = []
91 Entfernungen.append([0, 288, 585, 575, 547, 633, 559, 494, 531, 392])
92 Entfernungen.append([289, 0, 775, 426, 493, 655, 400, 344, 365, 122])
93 Entfernungen.append([589, 775, 0, 577, 398, 220, 612, 604, 634, 748])
94 Entfernungen.append([579, 426, 576, 0, 193, 369, 42, 95, 73, 320])

```

```

95 Entfernungen.append([552, 492, 393, 193, 0, 210, 229, 219, 251, 445])
96 Entfernungen.append([637, 667, 231, 369, 203, 0, 404, 417, 426, 642])
97 Entfernungen.append([563, 408, 611, 38, 228, 404, 0, 71, 37, 302])
98 Entfernungen.append([498, 346, 605, 95, 221, 418, 69, 0, 36, 240])
99 Entfernungen.append([536, 374, 634, 74, 252, 427, 35, 37, 0, 267])
100 Entfernungen.append([397, 123, 748, 315, 441, 638, 289, 233, 254, 0])
101
102 print("Berechnung aller Routen wurde begonnen.")
103 route = optimal_solution_with_brute_force(Entfernungen)
104 print("Berechnung aller Routen wurde abgeschlossen.")
105 print("Länge der optimalen Route: ", Strecke_der_Route(Entfernungen, route))
106 for index in route:
107     print(Stadtliste[index])

```

Handlungsreisender-in-Deutschland-Alle-Routen.py

```

1 #!/usr/bin/python
2 #-*- coding: utf-8 -*-
3
4 """ Dieses Script analysiert die Verteilung der Längen der Routen. Diese müssen
5     durch "Entfernungen.txt" gegeben sein.
6
7     """
8
9 print("Beginne alle Routen zu lesen.")
10 f = open('/home/moose/mathe/Entfernungen.txt', 'r')
11 lines = f.readlines()
12 f.close()
13 print("Alle Routen wurden eingelesen.")
14
15 number = length = 0
16 minimum = int(lines[0])
17 maximum = int(lines[0])
18 liste = []
19 dict_entfernungen = {}
20 for line in lines:
21     line = int(line)
22     if line in dict_entfernungen:
23         dict_entfernungen[line] += 1
24     else:
25         dict_entfernungen[line] = 1
26     liste.append(line)
27     number += 1
28     length += line
29     if line < minimum:
30         minimum = line
31     if line > maximum:
32         maximum = line
33 print("Minimum: %i" % minimum)
34 print("Maximum: %i" % maximum)
35 print("Durchschnitt: %i" % (length/number))
36 print("Anzahl der Routen: %i" % number)
37
38 less_than2069 = 0.0
39 for length, count in dict_entfernungen.items():
40     if length <= 2068:
41         less_than2069 += count
42
43 print("maximale Abweichung von 5%%: %f %% aller Strecken" % (less_than2069/number*100))
44
45 EntfernungsSet = set(liste)
46 print("Anzahl der verschiedenen Streckenlängen: %i" % len(dict_entfernungen))
47
48 import math, pylab
49
50 x_list = []
51 y_list = []
52 for x, y in dict_entfernungen.items():
53     y_list.append(y)
54     x_list.append(x)
55
56 pylab.xlabel("Streckenlänge in km")
57 pylab.ylabel("Anzahl der Routen")
58 pylab.plot(x_list, y_list, 'b')
59
60 # Einzeichnen des Durchschnitts:
61 y_max = max(y_list)
62 x_list = []
63 y_list = []
64 y_list.append(0)
65 x_list.append((length/number))
66 y_list.append(y_max)
67 x_list.append((length/number))
68 pylab.plot(x_list, y_list, 'r')
69
70 pylab.savefig('Fig1.png')
71
72 pylab.show()

```

Handlungsreisender-in-Deutschland-analyse.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 def nearest_neighbour(Entfernungen):
5     """ Entfernungen: Entfernungen[x][y] gibt die Länge der Strecke von x nach
6         y als integer an.
7         return: route als Liste, z.B. [0,3,4,1,2]
8         Es wird immer mit der Stadt 0 begonnen. """
9     route = []
10    cities = len(Entfernungen)
11    maxEntfernung = max([item for sublist in Entfernungen for item in sublist])
12    aktuelleCity = routenLaenge = 0
13    for i in xrange(0, cities):
14        route.append(aktuelleCity)
15        for sublist in Entfernungen:
16            sublist[aktuelleCity] = maxEntfernung + 1
17            aktuelleCity = Entfernungen[aktuelleCity].index( min(Entfernungen[aktuelleCity]) )
18            routenLaenge += Entfernungen[route[-1]][aktuelleCity]
19
20    print("Routenlänge: %i" % routenLaenge)
21    return route
22
23 Entfernungen = []
24 Entfernungen.append([ 0, 288, 585, 575, 547, 633, 559, 494, 531, 392])
25 Entfernungen.append([289, 0, 775, 426, 493, 655, 400, 344, 365, 122])
26 Entfernungen.append([589, 775, 0, 577, 398, 220, 612, 604, 634, 748])
27 Entfernungen.append([579, 426, 576, 0, 193, 369, 42, 95, 73, 320])
28 Entfernungen.append([552, 492, 393, 193, 0, 210, 229, 219, 251, 445])
29 Entfernungen.append([637, 667, 231, 369, 203, 0, 404, 417, 426, 642])
30 Entfernungen.append([563, 408, 611, 38, 228, 404, 0, 71, 37, 302])
31 Entfernungen.append([498, 346, 605, 95, 221, 418, 69, 0, 36, 240])
32 Entfernungen.append([536, 374, 634, 74, 252, 427, 35, 37, 0, 267])
33 Entfernungen.append([397, 123, 748, 315, 441, 638, 289, 233, 254, 0])
34 Stadtliste = ['Berlin', 'Hamburg', 'München', 'Köln', 'Frankfurt a. M.',
35              'Stuttgart', 'Düsseldorf', 'Dortmund', 'Essen', 'Bremen']
36
37 for city in nearest_neighbour(Entfernungen):
38     print("%s" % Stadtliste[city])

```

Handlungsreisender-in-Deutschland-Nearest-Neighbour.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from copy import deepcopy
5
6 def compute_length(Entfernungen, route):
7     """ Berechnet die Länge der Route und gibt diesen Integer zurück. """
8     entfernung = 0
9     for i in xrange(0, len(route)-1):
10         entfernung += Entfernungen[route[i]][route[i+1]]
11     return entfernung
12
13 def nearest_insertion(Entfernungen):
14     """ Entfernungen: Entfernungen[x][y] gibt die Länge der Strecke von x nach
15         y als integer an.
16         return: route als Liste, z.B. [0,3,4,1,2]
17         Es wird immer mit der Stadt 0 begonnen. """
18     route = [0]
19     cities = len(Entfernungen)
20     maxEntfernung = max([item for sublist in Entfernungen for item in sublist])
21     aktuelleCity = 0
22     Entfernungen_read = deepcopy(Entfernungen)
23     for i in xrange(0, cities):
24         for sublist in Entfernungen:
25             sublist[aktuelleCity] = maxEntfernung + 1
26             aktuelleCity = Entfernungen[aktuelleCity].index( min(Entfernungen[aktuelleCity]) )
27             minInsert = None
28             routenLaenge = len(route)*(maxEntfernung+1)
29             for insertIndex in xrange(0, len(route) ):
30                 route_tmp = deepcopy(route)
31                 route_tmp.insert(insertIndex, aktuelleCity)
32                 if compute_length(Entfernungen_read, route_tmp) < routenLaenge:
33                     routenLaenge = compute_length(Entfernungen_read, route_tmp)
34                     minInsert = insertIndex
35             route.insert(minInsert, aktuelleCity)
36             routenLaenge += Entfernungen[route[-1]][aktuelleCity]
37
38     #Drehen der Route, sodass Berlin am Anfang und am Ende steht
39     route = route[route.index(0)+1:] + route[0:route.index(0)+1]
40
41     print("Routenlänge: %i" % compute_length(Entfernungen_read, route))
42     return route
43
44 Entfernungen = []
45 Entfernungen.append([ 0, 288, 585, 575, 547, 633, 559, 494, 531, 392])
46 Entfernungen.append([289, 0, 775, 426, 493, 655, 400, 344, 365, 122])
47 Entfernungen.append([589, 775, 0, 577, 398, 220, 612, 604, 634, 748])
48 Entfernungen.append([579, 426, 576, 0, 193, 369, 42, 95, 73, 320])
49 Entfernungen.append([552, 492, 393, 193, 0, 210, 229, 219, 251, 445])
50 Entfernungen.append([637, 667, 231, 369, 203, 0, 404, 417, 426, 642])
51 Entfernungen.append([563, 408, 611, 38, 228, 404, 0, 71, 37, 302])
52 Entfernungen.append([498, 346, 605, 95, 221, 418, 69, 0, 36, 240])
53 Entfernungen.append([536, 374, 634, 74, 252, 427, 35, 37, 0, 267])
54 Entfernungen.append([397, 123, 748, 315, 441, 638, 289, 233, 254, 0])
55 Stadtliste = ['Berlin', 'Hamburg', 'München', 'Köln', 'Frankfurt a. M.',
56               'Stuttgart', 'Düsseldorf', 'Dortmund', 'Essen', 'Bremen']
57
58 for city in nearest_insertion(Entfernungen):
59     print("%s" % Stadtliste[city])

```

Handlungsreisender-in-Deutschland-Nearest-Insertion.py