

Mark Eatough

CSIS 2430 9:00 Class

Final Programming Project

Germany Trip Program

Assignment objective:

Utilize a shortest path algorithm to solve the traveling salesman problem to travel through 22 cities in Germany. I can only start and end in the 4 cities in Germany that have major International Airports. I used Travelocity to find all of my flights, and the Ice train how to book website for all train travel. Certain cities also had special expenses. Such as 180 euros each for new iPads for everyone in Hannover, 6,000 euros each for two watches in Basil, Switzerland, traveling 10 km to visit a castle in Koln, and taking a taxi under a river in Hamburg.

What Worked?:

I set up a weighted graph with every city as a vertex, and the weight between them as an object that contains cost in euros, time to travel, and a combination of time and euros. On Travelocity I found that it is possible to fly into one city, and out of a different city, so I included this in my calculations. Although starting and ending at different cities produced the shortest time path, it did not produce the money path, or when optimizing for both time and money. I created time train and flight objects which contained all of the information I needed, stored them in a database, and then pulled what I needed out of the database to create my weighted graph. My algorithm finds a valid path through all start and end cities I entered, and then picks out the path that optimizes with respect to all three of my weights.

What did not work?:

At the beginning of the project I way over complicated things as I was putting in data that was entirely unnecessary. It took my program a very long time to traverse through everything, so I tried to convert my recursive method into an iterative one, but I could not see a way to do this while keeping the same functionality. My shortest path algorithm I came up with took far too long, and so I had to rethink it a little bit.

Comments:

This was an interesting assignment as we worked the entire semester to build to this. I decided to use trains and taxis to complete my trip through Germany. Entering all of the data into the database was tedious to say the least, but once it was there the assignment was fairly straight forward. On Travelocity I found flights that would allow for landing in and leaving from different cities. I thought this might make my trip through Germany even more efficient, however it did not, as starting in Munich and ending in Stuttgart was the fastest way through Germany, the cost made it so that another way was better. I was surprised to see that the path that optimized for cost also optimized for both cost and time. To save 7 hours on the trip would cost 1200 dollars, and because it is rare that someone makes 171.45 dollars an hour, it seems that the path starting and ending in Frankfurt is the better choice.

```

1  '''
2  ****
3  * Discrete Structures
4  * Trip Through Germany Program
5  * Programmer: Mark Eatough
6  * Course: CSIS 2430
7  * Created Novermber 3, 2013
8
9  ****
10 '''
11
12 #class to create and manage dollars object
13 class Dollars:
14     def __init__(self, dollars):
15         self.dollars = dollars
16     #toString method equivalent
17     def __str__(self):
18         return("${0:.2f}".format(self.dollars))
19     #method to add dollars
20     def __add__(self, other):
21         myMoney = self.dollars+other.dollars
22         myDollars = Dollars(myMoney)
23         return myDollars
24     #Method to multiply dollars
25     def __mul__(self, mult):
26         myMoney = self.dollars*mult
27         myDollars = Dollars(myMoney)
28         return myDollars
29
30 #class to create and manage euros object
31 class Euros:
32     def __init__(self, euros):
33         self.euros = euros
34     #toString method equivalent
35     def __str__(self):
36         return("{0:.2f} Euros".format(self.euros))
37     #method to add Euros
38     def __add__(self, other):
39         myMoney = self.euros+other.euros
40         myEuros = Euros(myMoney)
41         return myEuros
42     #method to subtract Euros
43     def __sub__(self, other):
44         myMoney = self.euros-other.euros
45         myEuros = Euros(myMoney)
46         return myEuros
47     #Method to multiply dollars

```

```

48     def __mul__(self, mult):
49         myMoney = self.Euros*mult
50         myEuros = Euros(myEuros)
51         return myEuros
52
53 #class to create and manage time object
54 class Time:
55     def __init__(self, hours, minutes):
56         self.hours = hours
57         self.minutes = minutes
58         while(self.minutes >= 60):
59             self.hours += 1
60             self.minutes -= 60
61     #convert time to minutes for database, convert back to
time in graph
62     def timeToMinutes(self):
63         self.minutes += (self.hours*60)
64         return self.minutes
65     def __lt__(self, other):
66         return
(Time.timeToMinutes(self)<Time.timeToMinutes(other))
67     #toString method equivalent
68     def __str__(self):
69         return("{0} Hours {1}
Miniutes".format(self.hours,self.minutes))
70     #overload '+' operator
71     def __add__(self, other):
72         myHours = self.hours+other.hours
73         myMinutes = self.minutes+other.minutes
74         while(myMinutes >= 60):
75             myMinutes -= 60
76             myHours += 1
77         myTime = Time(myHours, myMinutes)
78         return myTime
79     #overload '-' operator
80     def __sub__(self, other):
81         bigMinutes = (self.hours*60)+self.minutes
82         littleMinutes = (other.hours*60)+other.minutes
83         newTime = bigMinutes-littleMinutes
84         myTime = Time(0, newTime)
85         return myTime
86     #overload '*' operator
87     def __mul__(self, mult):
88         myMinutes = (self.hours*60) + self.minutes
89         myMinutes *=mult
90         myHours = 0;
91         while(myMinutes >= 60):

```

```

92         myMinutes -= 60
93         myHours += 1
94         myTime = Time(myHours, myMinutes)
95         return myTime
96     #overload '/' operator
97     def __div__(self, div):
98         myMinutes = (self.hours*60) + self.minutes
99         myMinutes /=div
100        myHours = 0;
101        while(myMinutes >= 60):
102            myMinutes -= 60
103            myHours += 1
104        myTime = Time(myHours, myMinutes)
105        return myTime
106
107     #CONVERSTIONS
108
109     #conversion rate found on xrates.com, rounded to the nearest
cent
110     def convertToEuros(dollars):
111         euros = dollars * 0.74
112         return Euros(euros)
113     #conversion rate found on xrates.com, rounded to the nearest
cent
114     def convertToDollars(euros):
115         dollars = euros * 1.35
116         return Dollars(dollars)
117     #Converstion from miles to kilometers
118     def convertToKilometers(miles):
119         miles = kilometers * 0.62
120         return miles
121     #Conversion from kilometers to miles
122     def convertToMiles(kilometers):
123         kilometers = miles * 1.6
124         return kilometers
125     #1.96 multiplier figured by adding all times on flights and
train trips
126     #and then dividing by all sum of all costs on flights and
train trips
127     #number then rounded to 1.96
128     class Weight:
129         def __init__(self, euros, time, Transport):
130             self.euroTimeWeight = (float(euros)*1.96)+float(time)
131             self.euros = Euros(euros)
132             self.time = Time(0,time)
133             self.Transport = Transport
134             #toString method equivalent

```

```

135         def __str__(self):
136             return("{0:.2f} Euros".format(self.euros), " Time:
", self.time, " Weight: 0", self.weight, "by: ", transport)
137         #method to add two wieghts together
138         def __add__(self, other):
139             self.euros+=other.euros
140             self.time+=other.time
141             self.euroTimeWeight+=other.euroTimeWeight
142             self.Transport = "severalModes"
143             return self
144         def __sub__(self, other):
145             self.euros-=other.euros
146             self.time-=other.time
147             self.euroTimeWeight-=other.euroTimeWeight
148             self.Transport = "severalModes"
149             return self
150         def __lt__(self, other):
151             return (self.euros<other.euros)
152
153     class PathData:
154         def __init__(self, start, end, euros, time, flight):
155             self.start = start
156             self.end = end
157             self.weight = Weight(euros, time, flight)
158
159     class PathStats:
160         def __init__(self, list, weight):
161             self.list = list
162             self.weight = weight
163

```

```

1  '''
2  ****
3  * Discrete Structures
4  * Trip Through Germany Program
5  * Programmer: Mark Eatough
6  * Course: CSIS 2430
7  * Created Novermber 3, 2013
8
9  ****
10 '''
11
12 from TimeDistanceMoney import*
13
14 class TrainTravel:
15     def __init__(self, cityFrom, cityTo, time, dollars):
16         self.cityFrom = cityFrom
17         self.cityTo = cityTo
18         self.time = time
19         self.dollars = Dollars(dollars)
20         self.euros = convertToEuros(dollars)
21         self.travelBy = "Train"
22
23     def displayTrainTrip(self):
24         print "Leaving From: ", self.cityFrom
25         print "Comming To: ", self.cityTo
26         print "Total time: ", self.time
27         print "Total cost in euros: ", self.euros
28         print "Total cost in USD: ", self.dollars
29
30 class TaxiTravel:
31     def __init__(self, cityFrom, cityTo, distance):
32         self.cityFrom = cityFrom
33         self.cityTo = cityTo
34         self.euros = Euros(float(distance)*1.2)
35         self.dollars = convertToDollars(self.euros.euros)
36         self.time =
Time(0,int((float(distance)/float(130))*60))
37         self.travelBy = "Taxi"

```



```

1  '''
2  ****
3  * Discrete Structures
4  * Trip Through Germany Program
5  * Programmer: Mark Eatough
6  * Course: CSIS 2430
7  * Created Novermber 3, 2013
8  ****
9  '''
10 #database called GermanyDB for all objects
11 #table called train in GermanyDB
12
13 #create all needed train objects
14 #all flights found via http://www.raileurope.com/train-
15 #faq/european-trains/ice/how-to-book.h...
16 #with october 18 as travel date in economy class, prices
17 #appeared to be given in dollars
18
19 from TrainClasses import*
20 import MySQLdb as mdb
21 import sys
22
23 trainList = []
24
25 rostockToHamburg = TrainTravel("Rostock", "Hamburg",
26 Time(1,47), 82)
27 print "\n\nTrain trip from Rostock to Hamburg\n"
28 rostockToHamburg.displayTrainTrip()
29 trainList.append(rostockToHamburg)
30
31 rostockToLubeck = TrainTravel("Rostock", "Lubeck",
32 Time(2,32), 108)
33 print "\n\nTrain trip from Rostock to Hamburg\n"
34 rostockToLubeck.displayTrainTrip()
35 trainList.append(rostockToLubeck)
36
37 hamburgToLubeck = TrainTravel("Hamburg", "Lubeck",
38 Time(0,45), 26)
39 print "\n\nTrain trip from Hamburg to Lubeck\n"
40 hamburgToLubeck.displayTrainTrip()
41 trainList.append(hamburgToLubeck)
42
43 wiesbadenToKassel = TrainTravel("Wiesbaden", "Kassel",
44 Time(1,18), 104)
45 print "\n\nTrain trip from Munich to Nuremburg\n"
46 #munichToNuremburg.displayTrainTrip()
47 trainList.append(wiesbadenToKassel)

```

```

42
43 # hannoverToRostock = TrainTravel("Hannover", "Rostock",
Time(2,05), 104)
44 # print "\n\nTrain trip from Munich to Nuremburg\n"
45 # #munichToNuremburg.displayTrainTrip()
46 # trainList.append(hannoverToRostock)
47
48 hamburgToBerlin = TrainTravel("Hamburg", "Berlin",
Time(1,51), 109)
49 print "\n\nTrain trip from Hamburg to Berlin\n"
50 hamburgToBerlin.displayTrainTrip()
51 trainList.append(hamburgToBerlin)
52
53 # rostockToBerlin = TrainTravel("Rostock", "Berlin",
Time(4,47), 234)
54 # print "\n\nTrain trip from Hamburg to Berlin\n"
55 # hamburgToBerlin.displayTrainTrip()
56 # trainList.append(rostockToBerlin)
57 #
58 # lubeckToBerlin = TrainTravel("Lubeck", "Berlin",
Time(2,17), 154)
59 # print "\n\nTrain trip from Hamburg to Berlin\n"
60 # hamburgToBerlin.displayTrainTrip()
61 # trainList.append(lubeckToBerlin)
62 #
63 # bremenToBerlin = TrainTravel("Bremen", "Berlin",
Time(3,0), 152)
64 # print "\n\nTrain trip from Hamburg to Berlin\n"
65 # hamburgToBerlin.displayTrainTrip()
66 # trainList.append(bremenToBerlin)
67 #
68 # hannoverToBerlin = TrainTravel("Hannover", "Berlin",
Time(2,11), 183)
69 # print "\n\nTrain trip from Hamburg to Berlin\n"
70 # hamburgToBerlin.displayTrainTrip()
71 # trainList.append(hannoverToBerlin)
72
73 hamburgToBremen = TrainTravel("Hamburg", "Bremen",
Time(1,9), 43)
74 print "\n\nTrain trip from Hamburg to Bremen\n"
75 hamburgToBremen.displayTrainTrip()
76 trainList.append(hamburgToBremen)
77
78 rostockToBremen = TrainTravel("Rostock", "Bremen",
Time(2,56), 125)
79 trainList.append(rostockToBremen)
80

```

```

81 bremenToLubeck = TrainTravel("Bremen", "Lubeck", Time(1,54),
69)
82 print "\n\nTrain trip from Rostock to Hamburg\n"
83 bremenToLubeck.displayTrainTrip()
84 trainList.append(bremenToLubeck)
85
86 hamburgToHannover = TrainTravel("Hamburg", "Hannover",
Time(1,20), 78)
87 print "\n\nTrain trip from Hamburg to Hannover\n"
88 hamburgToHannover.displayTrainTrip()
89 trainList.append(hamburgToHannover)
90
91 hannoverToKassel = TrainTravel("Hannover", "Kassel",
Time(0,55), 65)
92 print "\n\nTrain trip from Hannover to Kassel\n"
93 hannoverToKassel.displayTrainTrip()
94 trainList.append(hannoverToKassel)
95
96 hannoverToDusseldorf = TrainTravel("Hannover", "Dusseldorf",
Time(2,26), 101)
97 print "\n\nTrain trip from Hannover to Kassel\n"
98 hannoverToDusseldorf.displayTrainTrip()
99 trainList.append(hannoverToDusseldorf)
100
101 kasselToFrankfurt = TrainTravel("Kassel", "Frankfurt",
Time(1,32), 75)
102 print "\n\nTrain trip from Kassel to Frankfurt\n"
103 kasselToFrankfurt.displayTrainTrip()
104 trainList.append(kasselToFrankfurt)
105
106 dusseldorfToKoln = TrainTravel("Dusseldorf", "Koln",
Time(0,24), 26)
107 print "\n\nTrain trip from Dusseldorf to Koln\n"
108 dusseldorfToKoln.displayTrainTrip()
109 trainList.append(dusseldorfToKoln)
110
111 dusseldorfToBonn = TrainTravel("Dusseldorf", "Bonn",
Time(0,47), 48)
112 print "\n\nTrain trip from Dusseldorf to Bonn\n"
113 dusseldorfToBonn.displayTrainTrip()
114 trainList.append(dusseldorfToBonn)
115
116 wiesbadenToMannhiem = TrainTravel("Wiesbaden", "Mannhiem",
Time(1,27), 83)
117 print "\n\nTrain trip from Wiesbaden to Mannhiem\n"
118 wiesbadenToMannhiem.displayTrainTrip()
119 trainList.append(wiesbadenToMannhiem)

```

```

120
121 kolnToBonn = TrainTravel("Koln", "Bonn", Time(0, 23), 22)
122 print "\n\nTrain trip from Koln to Bonn\n"
123 kolnToBonn.displayTrainTrip()
124 trainList.append(kolnToBonn)
125
kolnToFrankfurt = TrainTravel("Koln", "Frankfurt", Time(1,05),
112)
127 print "\n\n Train trip from Koln to Frankfurt\n"
128 kolnToFrankfurt.displayTrainTrip()
129 trainList.append(kolnToFrankfurt)
130
131 bonnToFrankfurt = TrainTravel("Bonn", "Frankfurt",
Time(1,58), 72)
132 print "\n\n Train trip from Bonn to Frankfurt\n"
133 bonnToFrankfurt.displayTrainTrip()
134 trainList.append(bonnToFrankfurt)
135
136 frankfurtToWiesbaden = TrainTravel("Frankfurt", "Wiesbaden",
Time(0,49), 29)
137 print "\n\n Train trip from Frankfurt to Wiesbaden\n"
138 frankfurtToWiesbaden.displayTrainTrip()
139 trainList.append(frankfurtToWiesbaden)
140
141 # wiesbadenToStuttgart = TrainTravel("Wiesbaden",
"Stuttgart", Time(2,23), 130)
142 # print "\n\n Train trip from Frankfurt to Stuggart\n"
143 # #frankfurtToStuttgart.displayTrainTrip()
144 # trainList.append(wiesbadenToStuttgart)
145
146 frankfurtToMannhiem = TrainTravel("Frankfurt", "Mannhiem",
Time(0,38), 54)
147 print "\n\n Train trip from Frankfurt to Mannhiem\n"
148 frankfurtToMannhiem.displayTrainTrip()
149 trainList.append(frankfurtToMannhiem)
150
151 frankfurtToKarlsruhe = TrainTravel("Frankfurt", "Karlsruhe",
Time(1,3), 69)
152 print "\n\n Train trip from Frankfurt to Karlsruhe\n"
153 frankfurtToKarlsruhe.displayTrainTrip()
154 trainList.append(frankfurtToKarlsruhe)
155
156 # badenbadenToStuttgart = TrainTravel("Baden Baden",
"Stuttgart", Time(2,53), 179)
157 # print "\n\n Train trip from Frankfurt to Stuggart\n"
158 # #frankfurtToStuttgart.displayTrainTrip()
159 # trainList.append(badenbadenToStuttgart)

```

```
160
161 frankfurtToBadenBaden = TrainTravel("Frankfurt", "Baden
Baden", Time(1,19), 78)
162 print "\n\n Train trip from Frankfurt to Baden Baden\n"
163 frankfurtToBadenBaden.displayTrainTrip()
164 trainList.append(frankfurtToBadenBaden)
165
166 frankfurtToStuttgart = TrainTravel("Frankfurt", "Stuttgart",
Time(1,34), 101)
167 print "\n\n Train trip from Frankfurt to Stuggart\n"
168 frankfurtToStuttgart.displayTrainTrip()
169 trainList.append(frankfurtToStuttgart)
170
171 frankfurtToNuremburg = TrainTravel("Frankfurt", "Nurnberg",
Time(2,06), 89)
172 print "\n\n Train trip from Frankfurt to Nuremburg\n"
173 frankfurtToNuremburg.displayTrainTrip()
174 trainList.append(frankfurtToNuremburg)
175
176 # kasselToStuttgart = TrainTravel("Kassel", "Stuttgart",
Time(2,06), 176)
177 # print "\n\n Train trip from Frankfurt to Stuggart\n"
178 # frankfurtToStuttgart.displayTrainTrip()
179 # trainList.append(kasselToStuttgart)
180 #
181 # kolnToStuttgart = TrainTravel("Koln", "Stuttgart",
Time(2,39), 213)
182 # print "\n\n Train trip from Frankfurt to Stuggart\n"
183 # frankfurtToStuttgart.displayTrainTrip()
184 # trainList.append(kolnToStuttgart)
185 #
186 # bonnToStuttgart = TrainTravel("Bonn", "Stuttgart",
Time(3,32), 173)
187 # print "\n\n Train trip from Frankfurt to Stuggart\n"
188 # frankfurtToStuttgart.displayTrainTrip()
189 # trainList.append(bonnToStuttgart)
190 #
191 # mannhiemToStuttgart = TrainTravel("Mannhiem", "Stuttgart",
Time(0,38), 55)
192 # print "\n\n Train trip from Mannhiem to Stuttgart\n"
193 # mannhiemToStuttgart.displayTrainTrip()
194 # trainList.append(mannhiemToStuttgart)
195
196 mannhiemToKarlsruhe = TrainTravel("Mannhiem", "Karlsruhe",
Time(1,07), 32)
197 print "\n\n Train trip from Mannhiem to Karlsruhe\n"
198 mannhiemToKarlsruhe.displayTrainTrip()
```

```
199 trainList.append(mannhiemToKarlsruhe)
200
201 mannhiemToBadenBaden = TrainTravel("Mannhiem", "Baden
Baden", Time(0,39), 54)
202 print "\n\n Train trip from Mannhiem to Baden Baden\n"
203 mannhiemToBadenBaden.displayTrainTrip()
204 trainList.append(mannhiemToBadenBaden)
205
206 karlsruheToBadenBaden = TrainTravel("Karlsruhe", "Baden
Baden", Time(0,20), 23)
207 print "\n\n Train trip from Karlsruhe to Baden Baden\n"
208 karlsruheToBadenBaden.displayTrainTrip()
209 trainList.append(karlsruheToBadenBaden)
210
211 karlsruheToStuttgart = TrainTravel("Karlsruhe", "Stuttgart",
Time(0,55), 36)
212 print "\n\n Train trip from Karlsruhe to Stuttgart\n"
213 karlsruheToStuttgart.displayTrainTrip()
214 trainList.append(karlsruheToStuttgart)
215
216 basilToKarlsruhe = TrainTravel("Basil", "Karlsruhe",
Time(1,48), 100)
217 print "\n\n Train trip from Dusseldorf to Koln\n"
218 dusseldorfToKoln.displayTrainTrip()
219 trainList.append(basilToKarlsruhe)
220
221 basilToMannhiem = TrainTravel("Basil", "Mannhiem",
Time(2,7), 131)
222 print "\n\n Train trip from Dusseldorf to Koln\n"
223 dusseldorfToKoln.displayTrainTrip()
224 trainList.append(basilToMannhiem)
225
226 badenBadenToBasil = TrainTravel("Baden Baden", "Basil",
Time(1,28), 77)
227 print "\n\n Train trip from Baden Baden to basil\n"
228 badenBadenToBasil.displayTrainTrip()
229 trainList.append(badenBadenToBasil)
230
231 stuttgartToNuremburg = TrainTravel("Stuttgart", "Nurnberg",
Time(2,11), 59)
232 print "\n\n Train trip from Stuttgart to Nuremburg\n"
233 stuttgartToNuremburg.displayTrainTrip()
234 trainList.append(stuttgartToNuremburg)
235
236 stuttgartToMunich = TrainTravel("Stuttgart", "Munich",
Time(2,15), 92)
237 print "\n\n Train trip from Stuttgart to Munich\n"
```

```
238 stuttgartToMunich.displayTrainTrip()
239 trainList.append(stuttgartToMunich)
240
241 munichToNuremburg = TrainTravel("Munich", "Nurnberg",
Time(1,14), 91)
242 print "\n\n Train trip from Munich to Nuremburg\n"
243 munichToNuremburg.displayTrainTrip()
244 trainList.append(munichToNuremburg)
245
246 # munichToDresden = TrainTravel("Munich", "Dresden",
Time(5,28), 204)
247 # print "\n\n Train trip from Munich to Nuremburg\n"
248 # munichToNuremburg.displayTrainTrip()
249 # trainList.append(munichToDresden)
250
251 munichToFrankfurt = TrainTravel("Munich", "Frankfurt",
Time(3,49), 192)
252 print "\n\n Train trip from Munich to Nuremburg\n"
253 munichToNuremburg.displayTrainTrip()
254 trainList.append(munichToFrankfurt)
255
256 nuremburgToDresden = TrainTravel("Nurnberg", "Dresden",
Time(4,14), 113)
257 print "\n\n Train trip from Nuremburg to Dresden\n"
258 nuremburgToDresden.displayTrainTrip()
259 trainList.append(nuremburgToDresden)
260
261 nuremburgToLeipzig = TrainTravel("Nurnberg", "Leipzig",
Time(3,36), 136)
262 print "\n\n Train trip from Nuremburg to Dresden\n"
263 nuremburgToLeipzig.displayTrainTrip()
264 trainList.append(nuremburgToLeipzig)
265
266 munichToLeipzig = TrainTravel("Munich", "Leipzig",
Time(2,51), 134)
267 print "\n\n Train trip from Munich to Nuremburg\n"
268 munichToNuremburg.displayTrainTrip()
269 trainList.append(munichToLeipzig)
270
271 dresdenToLeipzig = TrainTravel("Dresden", "Leipzig",
Time(1,37), 43)
272 print "\n\n Train trip from Dresden to Leipzig\n"
273 dresdenToLeipzig.displayTrainTrip()
274 trainList.append(dresdenToLeipzig)
275
276 dresdenToBerlin = TrainTravel("Dresden", "Berlin",
Time(2,10), 76)
```

```

277 print "\n\nTrain trip from Dresden to Berlin\n"
278 dresdenToBerlin.displayTrainTrip()
279 trainList.append(dresdenToBerlin)
280
281 leipzigToBerlin = TrainTravel("Leipzig", "Berlin",
Time(1,8), 68)
282 print "\n\nTrain trip from Leipzig to Berlin\n"
283 leipzigToBerlin.displayTrainTrip()
284 trainList.append(leipzigToBerlin)
285
kolnToStAugustin = TaxiTravel("Koln", "St. Augustin", 9.18)
287 trainList.append(kolnToStAugustin)
288 bonnToStAugustin = TaxiTravel("Bonn", "St. Augustin", 32.25)
289 trainList.append(bonnToStAugustin)
290 dusseldorfToStAugustin = TaxiTravel("Dusseldorf", "St.
Augustin", 69.9)
291 trainList.append(dusseldorfToStAugustin)
292 wiesbadenToStAugustin = TaxiTravel("Wiesbaden", "St.
Augustin", 135.9)
293 trainList.append(wiesbadenToStAugustin)
294
295 kolnToCastle = TaxiTravel("Koln", "Castle", 10.0)
296 #trainList.append(kolnToCastle)
297
298 headers = ("CREATE TABLE Train(Id INT PRIMARY KEY
AUTO_INCREMENT,"
299           "StartCity VARCHAR(25), EndCity VARCHAR(25),
Time VARCHAR(25),"
300           "Dollars VARCHAR(25), Euros VARCHAR(25), Trans
VARCHAR(25))")
301
302 connection = mdb.connect('localhost', 'Mark', 'test623',
'GermanyDB');
303 with connection:
304     cur = connection.cursor()
305     cur.execute("DROP TABLE IF EXISTS Train")
306     cur.execute(headers)
307     i = 1
308     for t in trainList:
309         cur.execute("INSERT INTO Train(StartCity) VALUES(%s)",
t.cityFrom)
310         cur.execute("UPDATE Train SET EndCity = %s WHERE Id =
%s",
311                     (t.cityTo, i))
312         cur.execute("UPDATE Train SET Time = %s WHERE Id =
%s",
313                     (Time.timeToMinutes(t.time), i))

```



```

314         cur.execute("UPDATE Train SET Dollars = %s WHERE Id =
%s",
315             (t.dollars.dollars, i))
316         cur.execute("UPDATE Train SET Euros = %s WHERE Id =
%s",
317             (t.euros.euros, i))
318         cur.execute("UPDATE Train SET Trans = %s WHERE Id =
%s",
319             (t.travelBy, i))
320         i+=1
321         cur.execute("INSERT INTO Train(StartCity) VALUES(%s)",
t.cityTo)
322         cur.execute("UPDATE Train SET EndCity = %s WHERE Id =
%s",
323             (t.cityFrom, i))
324         cur.execute("UPDATE Train SET Time = %s WHERE Id =
%s",
325             (Time.timeToMinutes(t.time), i))
326         cur.execute("UPDATE Train SET Dollars = %s WHERE Id =
%s",
327             (t.dollars.dollars, i))
328         cur.execute("UPDATE Train SET Euros = %s WHERE Id =
%s",
329             (t.euros.euros, i))
330         cur.execute("UPDATE Train SET Trans = %s WHERE Id =
%s",
331             (t.travelBy, i))
332         i+=1

```

```

from TimeDistanceMoney import*

#class to manage flights, has many objects within it.
class Flight:
    def __init__(self, home, vacaTo, timeTo, timeFrom, dollars, vacaBack):
        self.home = home
        self.vacaTo = vacaTo
        if(vacaBack is None):
            self.vacaBack = vacaTo
        else:
            self.vacaBack = vacaBack
        self.timeTo = timeTo
        self.timeFrom = timeFrom
        self.totalTime = timeTo + timeFrom
        self.dollars = Dollars(dollars)
        self.euros = convertToEuros(dollars)

    def displayFlight(self):
        print "Flying From: ", self.home
        print "Flying To: ", self.vacaTo
        print "Flying Back From: ", self.vacaBack
        print "Total Time: ", self.totalTime
        print "total cost in USD:", self.dollars
        print "total cost in Euros:", self.euros

#class to manage concecated flights, has many objects within it.
class ConccateFlight:
    def __init__(self, vacaTo, vacaBack, dollars, avgTime):
        self.vacaTo = vacaTo
        self.vacaBack = vacaBack
        self.avgTime = avgTime
        self.dollars = dollars
        self.euros = convertToEuros(dollars.dollars)

    def displayConccateFlight(self):
        print "Flying To: ", self.vacaTo
        print "Flying Back From: ", self.vacaBack
        print "Average Time: ", self.avgTime
        print "total cost in USD:", self.dollars
        print "total cost in Euros:", self.euros

def flightConccation(flight1, numf1, flight2, numf2, flight3, numf3):
    if(flight1.vacaTo != flight2.vacaTo or flight1.vacaBack != flight2.vacaBack or
        flight1.vacaTo != flight3.vacaTo or flight1.vacaBack !=

```

```

flight3.vacaBack):
    print "Your destination cities do not match"
else:
    cityTo = flight1.vacaTo
    cityFrom = flight1.vacaBack
    totCost = (flight1.dollars*numf1) +
(flight2.dollars*numf2) + (flight3.dollars*numf3)
    avgTime = ((flight1.totalTime*numf1) +
(flight2.totalTime*numf2) +
(flight3.totalTime*numf3))/(numf1+numf2+numf3)
    combinedFlight = ConcecateFlight(cityTo, cityFrom,
totCost, avgTime)
    return combinedFlight

```

```

1  '''
2  ****
3  * Discrete Structures
4  * Trip Through Germany Program
5  * Programmer: Mark Eatough
6  * Course: CSIS 2430
7  * Created Novermber 3, 2013
8
9  ****
10 '''
11 #database called GermanyDB for all objects
12 #table called train in GermanyDB
13 import MySQLdb as mdb
14 import sys
15 #create all needed flight objects
16 #all flights found via travelocity with June 30 2014 as
departure date and August 1 2014 as return date
17 from FlightClasses import*
18
19 flightsList = []
20
21 #Flight from Salt Lake City to and from Frankfurt
22 myFrankfurt = Flight("Salt Lake City", "Frankfurt", Time(13,
4), Time(13, 15), 1727.40, None)
23
24 #Flight from Nashville to and from Frankfurt
25 parentsFrankfurt = Flight("Nashville", "Frankfurt", Time(10,
38), Time(14, 17), 1625.90, None)
26
27 #Flight from Boston to and from Frankfurt
28 grandFrankfurt = Flight("Boston", "Frankfurt", Time(9, 20),
Time(10, 35), 1080.50, None)
29
30 #Flight from Salt Lake City to and from Stuttgart
31 myStuttgart = Flight("Salt Lake City", "Stuttgart", Time(12,
55), Time(17, 23), 1778.59, None)
32
33 #Flight from Nashville to and from Stuttgart
34 parentsStuttgart = Flight("Nashville", "Stuttgart", Time(11,
10), Time(13, 2), 1616.00, None)
35
36 #Flight from Boston to and from Stuttgart
37 grandStuttgart = Flight("Boston", "Stuttgart", Time(8, 45),
Time(11, 0), 1569.90, None)
38
39 #Flight from Salt Lake City to and from Munich
40 myMunich = Flight("Salt Lake City", "Munich", Time(12, 59),

```

```

Time(18, 21), 1768.49, None)
41
42 #Flight from Nashville to and from Munich
43 parentsMunich = Flight("Nashville", "Munich", Time(11, 13),
Time(13, 26), 1607.80, None)
44
45 #Flight from Boston to and from Munich
46 grandMunich = Flight("Boston", "Munich", Time(9, 35),
Time(15, 30), 1172.80, None)
47
48 #Flight from Salt Lake City to and from Berlin
49 myBerlin = Flight("Salt Lake City", "Berlin", Time(14, 29),
Time(16, 10), 1800.10, None)
50
51 #Flight from Nashville to and from Berlin
52 parentsBerlin = Flight("Nashville", "Berlin", Time(19, 55),
Time(12, 0), 1553.49, None)
53
54 #Flight from Boston to and from Berlin
55 grandBerlin = Flight("Boston", "Berlin", Time(9, 10),
Time(10, 10), 1568.49, None)
56
57 frankfurtFlight = flightConcation(myFrankfurt, 1,
parentsFrankfurt, 2, grandFrankfurt, 2)
58 ConcecateFlight.displayConcateFlight(frankfurtFlight)
59 flightsList.append(frankfurtFlight)
60
61 stuttgartFlight = flightConcation(myStuttgart, 1,
parentsStuttgart, 2, grandStuttgart, 2)
62 ConcecateFlight.displayConcateFlight(stuttgartFlight)
63 flightsList.append(stuttgartFlight)
64
65 munichFlight = flightConcation(myMunich, 1, parentsMunich,
2, grandMunich, 2)
66 ConcecateFlight.displayConcateFlight(munichFlight)
67 flightsList.append(munichFlight)
68
69 berlinFlight = flightConcation(myBerlin, 1, parentsBerlin,
2, grandBerlin, 2)
70 ConcecateFlight.displayConcateFlight(berlinFlight)
71 flightsList.append(berlinFlight)
72
73 #Flight from Salt Lake City to Frankfurt from stuttgart
74 myFrankStutt = Flight("Salt Lake City", "Frankfurt",
Time(13, 25), Time(15, 05), 1861.09, "Stuttgart")
75 #Flight from Nashville to Frankfurt from stuttgart
76 parentsFrankStutt = Flight("Nashville", "Frankfurt",

```

```

Time(10,48),Time(14,19) , 1700.19, "Stuttgart")
77 #Flight from Boston to Frankfurt from stuttgart
78 grandFrankStutt = Flight("Boston", "Frankfurt", Time(7,0),
Time(12,25), 1594.69, "Stuttgart")
79 frankStuttFlight = flightConcation(myFrankStutt, 1,
parentsFrankStutt, 2, grandFrankStutt, 2)
80 flightsList.append(frankStuttFlight)
81
82 #Flight from Salt Lake City to Frankfurt from Munich
83 myFrankMun = Flight("Salt Lake City", "Frankfurt", Time(12,
6), Time(14, 51), 1869.59, "Munich")
84 #Flight from Nashville to Frankfurt from munich
85 parentsFrankMun = Flight("Nashville", "Frankfurt", Time(11,
7), Time(15, 44), 1641.59, "Munich")
86 #Flight from Boston to Frankfurt from munich
87 grandFrankMun = Flight("Boston", "Frankfurt", Time(9, 50),
Time(12, 45), 1256.96, "Munich")
88 frankMunFlight = flightConcation(myFrankMun, 1,
parentsFrankMun, 2, grandFrankMun, 2)
89 flightsList.append(frankMunFlight)
90
91 #Flight from Salt Lake City to Frankfurt from Berlin
92 myFrankBer = Flight("Salt Lake City", "Frankfurt", Time(12,
6), Time(17, 36), 1822.39, "Berlin")
93 #Flight from Nashville to Frankfurt from Berlin
94 parentsFrankBer = Flight("Nashville", "Frankfurt", Time(10,
48), Time(14, 14), 1709.69, "Berlin")
95 #Flight from Boston to Frankfurt from Berlin
96 grandFrankBer = Flight("Boston", "Frankfurt", Time(9, 50),
Time(12, 55), 1241.36, "Berlin")
97 frankBerFlight = flightConcation(myFrankBer, 1,
parentsFrankBer, 2, grandFrankBer, 2)
98 flightsList.append(frankBerFlight)
99
100 #Flight from Salt Lake City to Stuttgart from Frankfurt
101 myStuttFrank = Flight("Salt Lake City", "Stuttgart",
Time(12, 55), Time(14, 05), 1865.69, "Frankfurt")
102 #Flight from Nashville to Stuttgart from Frankfurt
103 parentsStuttFrank = Flight("Nashville", "Stuttgart",
Time(11, 10), Time(13, 12), 1719.59, "Frankfurt")
104 #Flight from Boston to Stuttgart from Frankfurt
105 grandStuttFrank = Flight("Boston", "Stuttgart", Time(9, 15),
Time(11, 0), 1664.59, "Frankfurt")
106 stuttFrankFlight = flightConcation(myStuttFrank, 1,
parentsStuttFrank, 2, grandStuttFrank, 2)
107 flightsList.append(stuttFrankFlight)
108

```

```

109 #Flight from Salt Lake City to Stuttgart from Munich
110 myStuttMun = Flight("Salt Lake City", "Stuttgart", Time(12,
111 55), Time(14, 20), 1965.19, "Munich")
112 #Flight from Nashville to Stuttgart from Munich
113 parentsStuttMun = Flight("Nashville", "Stuttgart", Time(11,
114 10), Time(13, 37), 1700.99, "Munich")
115 #Flight from Boston to Stuttgart from Munich
116 grandStuttMun = Flight("Boston", "Stuttgart", Time(9, 15),
117 Time(10, 55), 1654.09, "Munich")
118 stuttMunFlight = flightConcation(myStuttMun, 1,
119 parentsStuttMun, 2, grandStuttMun, 2)
120 flightsList.append(stuttMunFlight)
121
122 #Flight from Salt Lake City to Stuttgart from Berlin
123 myStuttBer = Flight("Salt Lake City", "Stuttgart", Time(12,
124 55), Time(14, 25), 1847.69, "Berlin")
125 #Flight from Nashville to Stuttgart from Berlin
126 parentsStuttBer = Flight("Nashville", "Stuttgart", Time(11,
127 10), Time(15, 27), 1725.99, "Berlin")
128 #Flight from Boston to Stuttgart from Berlin
129 grandStuttBer = Flight("Boston", "Stuttgart", Time(9, 15),
130 Time(10, 30), 1649.59, "Berlin")
131 stuttBerFlight = flightConcation(myStuttBer, 1,
132 parentsStuttBer, 2, grandStuttBer, 2)
133 flightsList.append(stuttBerFlight)
134
135 #Flight from Salt Lake City to Munich from Frankfurt
136 myMunFrank = Flight("Salt Lake City", "Munich", Time(12,
137 59), Time(14, 55), 1839.19, "Frankfurt")
138 #Flight from Nashville to Munich from Frankfurt
139 parentsMunFrank = Flight("Nashville", "Munich", Time(11,
140 35), Time(13, 12), 1711.69, "Frankfurt")
141 #Flight from Boston to Munich from Frankfurt
142 grandMunFrank = Flight("Boston", "Munich", Time(9, 35),
143 Time(10, 35), 1246.89, "Frankfurt")
144 munFrankFlight = flightConcation(myMunFrank, 1,
145 parentsMunFrank, 2, grandMunFrank, 2)
146 flightsList.append(munFrankFlight)
147
148 #Flight from Salt Lake City to Munich from Stuttgart
149 myMunStutt = Flight("Salt Lake City", "Munich", Time(12,
150 59), Time(15, 55), 1937.69, "Stuttgart")
151 #Flight from Nashville to Munich from Stuttgart
152 parentsMunStutt = Flight("Nashville", "Munich", Time(11,
153 35), Time(13, 2), 1675, "Stuttgart")
154 #Flight from Boston to Munich from Stuttgart
155 grandMunStutt = Flight("Boston", "Munich", Time(9, 25),

```

```

Time(10, 30), 1564.99, "Stuttgart")
142 munStuttFlight = flightConcation(myMunStutt, 1,
parentsMunStutt, 2, grandMunStutt, 2)
143 flightsList.append(munStuttFlight)
144
145 #Flight from Salt Lake City to Munich from Berlin
146 myMunBer = Flight("Salt Lake City", "Munich", Time(12, 59),
Time(16, 10), 1945.29, "Berlin")
147 #Flight from Nashville to Munich from Berlin
148 parentsMunBer = Flight("Nashville", "Munich", Time(11, 33),
Time(14, 14), 1709.69, "Berlin")
149 #Flight from Boston to Munich from Berlin
150 grandMunBer = Flight("Boston", "Munich", Time(7, 25),
Time(9, 55), 1681.29, "Berlin")
151 munBerFlight = flightConcation(myMunBer, 1, parentsMunBer,
2, grandMunBer, 2)
152 flightsList.append(munBerFlight)
153
154
155 #Flight from Salt Lake City to Berlin from Frankfurt
156 myBerFrank = Flight("Salt Lake City", "Berlin", Time(13, 5),
Time(14, 5), 1858.39, "Frankfurt")
157 #Flight from Nashville to Berlin from Frankfurt
158 parentsBerFrank = Flight("Nashville", "Berlin", Time(12, 0),
Time(14, 35), 1761.69, "Frankfurt")
159 #Flight from Boston to Berlin from Frankfurt
160 grandBerFrank = Flight("Boston", "Berlin", Time(10, 20),
Time(13, 20), 1378.66, "Frankfurt")
161 berFrankFlight = flightConcation(myBerFrank, 1,
parentsBerFrank, 2, grandBerFrank, 2)
162 flightsList.append(berFrankFlight)
163
164 #Flight from Salt Lake City to Berlin from Stuttgart
165 myBerStutt = Flight("Salt Lake City", "Berlin", Time(14,
45), Time(15, 5), 1844.79, "Stuttgart")
166 #Flight from Nashville to Berlin from Stuttgart
167 parentsBerStutt = Flight("Nashville", "Berlin", Time(12, 0),
Time(15, 45), 1565.09, "Stuttgart")
168 #Flight from Boston to Berlin from Stuttgart
169 grandBerStutt = Flight("Boston", "Berlin", Time(9, 15),
Time(12, 25), 1631.99, "Stuttgart")
170 berStuttFlight = flightConcation(myBerStutt, 1,
parentsBerStutt, 2, grandBerStutt, 2)
171 flightsList.append(berStuttFlight)
172
173 #Flight from Salt Lake City to Berlin from Munich
174 myBerMun = Flight("Salt Lake City", "Berlin", Time(14, 29),

```



```

1 #Weighted Graph file
2
3 #Vertex class
4 class Vertex:
5     def __init__(self, key):
6         self.id = key
7         self.connectedTo = {}
8         self.connectionList = []
9     #add connection with weight to a vertex
10    def addConnection(self, nbr, weight=0):
11        self.connectedTo[nbr] = weight
12        if nbr not in self.connectionList:
13            self.connectionList.append(nbr)
14    #toString method to return
15    def __str__(self):
16        return self.id# + ' connectedTo: ' + str([x.id for x
in self.connectedTo])
17    #returns all connections
18    def getConnections(self):
19        return self.connectedTo.keys()
20    #returns Id
21    def getId(self):
22        return self.id
23    #return weight
24    def getWeight(self, nbr):
25        return self.connectedTo[nbr]
26
27 #Wiegthed Graph class
28 class WiegthedGraph:
29     def __init__(self):
30         self.vertList = {}
31         self.numVertices = 0
32     #add a new vertex to the graph
33     def addVertex(self, key):
34         if key not in self.vertList:
35             self.numVertices = self.numVertices + 1
36             newVertex = Vertex(key)
37             self.vertList[key] = newVertex
38             return newVertex
39     #return vertex of graph
40     def getVertex(self, n):
41         if n in self.vertList:
42             return self.vertList[n]
43         else:
44             return None
45     #check to see if graph contains vertex
46     def __contains__(self, n):

```

```
47         return n in self.vertList
48     #add a new edge to the graph
49     def addEdge(self,f,t,cost=0):
50         if f not in self.vertList:
51             nv = self.addVertex(f)
52         if t not in self.vertList:
53             nv = self.addVertex(t)
54         self.vertList[f].addConnection(self.vertList[t],
cost)#1, cost2, cost3)
55     #get vertices of graph
56     def getVertices(self):
57         return self.vertList.keys()
58
59     def __iter__(self):
60         return iter(self.vertList.values())
61
```

```

1 from WiegthedGraphADT import*
2 from TimeDistanceMoney import*
3 #Shortest path file
4 def toVertex(graph, city):
5     for v in graph:
6         if(v.getId() == city):
7             city = v
8     return city
9 #method to get next city for stack
10 def addNextCity(currentCity, endCity, stack, weight, graph,
check):
11     while(currentCity!=endCity):
12         if endCity in currentCity.connectionList:
13             weight+=currentCity.getWeight(endCity)
14             stack.append(endCity)
15             return endCity
16         for i in currentCity.connectionList:
17             if i not in stack and i not in check:
18                 weight+=currentCity.getWeight(i)
19                 stack.append(i)
20                 return i
21         removedCity = stack.pop()
22         check.append(removedCity)
23         weight-=stack[len(stack)-1].getWeight(removedCity)
24         currentCity = stack[len(stack)-1]
25         print len(check)
26     #return removeCity(currentCity, stack)
27 #method to remove a city from stack
28 def removeCity(currentCity, stack):
29     stack.pop()
30     return stack[len(stack)-1]
31 #method to return a path
32 def getPath(graph, cityStart, cityEnd):
33     stack = []
34     check = []
35     currentWeight = Weight(0,0,0)
36     cityStart = toVertex(graph, cityStart)
37     cityEnd = toVertex(graph, cityEnd)
38     tempCity = cityStart
39     stack.append(cityStart)
40     while cityEnd not in stack:
41         tempCity = addNextCity(tempCity, cityEnd, stack,
currentWeight, graph, check)
42
43     print "A valid path from",cityStart, "to", cityEnd, "is:"
44     for i in stack:
45         print i

```

```
46
47     print "total time = ", currentWeight.time
48     print "total money = ", currentWeight.euros
```

```

1 from WiegthedGraphADT import*
2 from TimeDistanceMoney import*
3 from ShortestPath import*
4 import MySQLdb as mdb
5 import sys
6
7 #file to import from database into graph
8
9
10 germanTrainGraph = WiegthedGraph()
11
12 connection = mdb.connect('localhost', 'Mark', 'test623',
13 'GermanyDB');
14
15 with connection:
16     cur = connection.cursor()
17     cur.execute("SELECT * FROM Train")
18     rows = cur.fetchall()
19     for row in rows:
20         germanTrainGraph.addEdge(row[1], row[2],
21 Weight(float(row[5]),int(row[3]), row[6]))
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 from ShortestPath import*
2 import os
3 import time
4
5 #Shortest Circuit file
6 def addCity(currentCity, endCity, stack, weight, i,
pathSize):
7     while(i < len(currentCity.connectionList)):
8         if(len(stack) == 0):
9             return
10        if len(stack) == pathSize-1 and endCity in
currentCity.connectionList:
11            weight+=currentCity.getWeight(endCity)
12            stack.append(endCity)
13            return endCity
14        if currentCity.connectionList[i] not in stack and
currentCity.connectionList[i] != endCity:
15            currentCity.connectionList[i]
16            stack.append(currentCity.connectionList[i])
17
18 weight+=currentCity.getWeight(currentCity.connectionList[i])
19         return currentCity.connectionList[i]

```

```

19         i+=1
20     removeCurrentCity(stack, endCity, weight, pathSize)
21
22 def removeCurrentCity(stack, startCity, weight, pathSize):
23     previousCity = stack.pop()
24     if(len(stack) == 0):
25         return
26     currentCity = stack[len(stack)-1]
27     i = currentCity.connectionList.index(previousCity)+1
28     weight-=currentCity.getWeight(previousCity)
29     addCity(currentCity, startCity, stack, weight, i,
pathSize)
30
31 def hamiltonCirciut(graph, startCity, endCity, weight):
32     stack = []
33     startCity = toVertex(graph, startCity)
34     endCity = toVertex(graph, endCity)
35     stack.append(startCity)
36     currentWeight = weight
37     if(startCity == endCity):
38         pathSize = len(graph.vertList)+1
39     else:
40         pathSize = len(graph.vertList)
41
42     while(len(stack)<pathSize):
43         i=0
44         if(len(stack) == 0):
45             break
46         currentCity = stack[len(stack)-1]
47         addCity(currentCity, endCity, stack, currentWeight, i,
pathSize)
48
49     if(len(stack) == 0):
50         currentWeight = Weight(0,0,0)
51
52     return PathStats(stack, currentWeight)

```

```

1 from WiegthedGraphADT import*
2 from TimeDistanceMoney import*
3 from ShortestPath import*
4 from CreateGraph import*
5 from GraphTests import*
6 from HamiltonCircuit import*
7 #Shortest path application
8 castle = 20
9 river = 10
10
11 flightList = []
12
13 connection = mdb.connect('localhost', 'Mark', 'test623',
'GermanyDB');
14
15 with connection:
16     cur = connection.cursor()
17     cur.execute("SELECT * FROM Flights")
18     rows = cur.fetchall()
19     for row in rows:
20         flightList.append(PathData(row[1], row[2],
float(row[5]), int(row[3]), "flight"))
21
22     #for f in flightList:
23     #    print f.start, f.end
24
25 #allConnections1(germanTrainGraph)
26 #getPath(germanTrainGraph, "Koln", "Dresden")
27 shortestStack = []
28 shortestWeight = Weight(0,0,0)
29 shortestTimeStack = []
30 shortestTimeWeight = Weight(0,0,0)
31 shortestMoneyStack = []
32 shortestMoneyWeight = Weight(0,0,0)
33
34
35 for f in flightList:
36     temp = hamiltonCirciut(germanTrainGraph, f.start, f.end,
f.weight)
37     if(shortestWeight.euroTimeWeight == 0 or
temp.weight.euroTimeWeight<shortestWeight.euroTimeWeight and
temp.weight.euroTimeWeight!=0):
38         shortestStack = temp.list
39         shortestWeight = temp.weight
40         #print "Shortest weight = ",
shortestWeight.euroTimeWeight
41     if(shortestTimeWeight.time.hours == 0 or

```



```

temp.weight.time.hours<shortestTimeWeight.time.hours and
temp.weight.time.hours!=0):
42     shortestTimeStack = temp.list
43     shortestTimeWeight = temp.weight
44     #print "Shortest Time = ", shortestTimeWeight.time
45     if(shortestMoneyWeight.euros.euros == 0 or
temp.weight.euros.euros<shortestMoneyWeight.euros.euros and
temp.weight.euros.euros!=0):
46         shortestMoneyStack = temp.list
47         shortestMoneyWeight = temp.weight
48         #print "Shortest money = ", shortestMoneyWeight.euros
49
50 print "The shortest path optomizing for both is: "
51 j = 0
52 for i in shortestStack:
53     message = ""
54     if(i == toVertex(germanTrainGraph,"Hannover")):
55         message = "Buy 5 new iPads at 180 euros each"
56     if(i == toVertex(germanTrainGraph,"Basil")):
57         message = "Buy 2 new watches at 6,000 euros each"
58     if(i == toVertex(germanTrainGraph,"Koln")):
59         message = "Take a taxi 20 km round trip to see the
castle"
60     if(i == toVertex(germanTrainGraph,"Hamburg")):
61         message = "Take a taxi under the river"
62     if(i == toVertex(germanTrainGraph,"Baden Baden")):
63         message = "Spend the day at the spa"
64     j+=1
65     print j,".",i, "\t", message
66
67 shortestWeight.euros.euros+=(2100)
68 shortestWeight+=Weight(float(castle)*1.2,
int(float(castle)/float(130)*60), "taxi")
69 shortestWeight+=Weight(float(river)*1.2,
int(float(river)/float(130)*60), "taxi")
70 print "total time = ", shortestWeight.time
71 print "total euros = ", shortestWeight.euros
72 print "total weight = ", shortestWeight.euroTimeWeight
73 print "cost in dollars = ",
convertToDollars(shortestWeight.euros.euros)
74
75 print "\n\n\nThe shortest path optomizing for time is: "
76 j = 0
77 for i in shortestTimeStack:
78     message = ""
79     if(i == toVertex(germanTrainGraph,"Hannover")):
80         message = "Buy 5 new iPads at 180 euros each"

```

```

81     if(i == toVertex(germanTrainGraph, "Basil")):
82         message = "Buy 2 new watches at 6,000 euros each"
83     if(i == toVertex(germanTrainGraph, "Koln")):
84         message = "Take a taxi 20 km round trip to see the
castle"
85     if(i == toVertex(germanTrainGraph, "Hamburg")):
86         message = "Take a taxi under the river"
87     j+=1
88     print j, ".", i, "\t", message
89     shortestTimeWeight.euros.euros+=(2100)
90     shortestTimeWeight+=Weight(float(castle)*1.2,
int(float(castle)/float(130)*60), "taxi")
91     shortestTimeWeight+=Weight(float(river)*1.2,
int(float(river)/float(130)*60), "taxi")
92     print "total time = ", shortestTimeWeight.time
93     print "total euros = ", shortestTimeWeight.euros
94     print "total weight = ", shortestTimeWeight.euroTimeWeight
95     print "cost in dollars = ",
convertToDollars(shortestTimeWeight.euros.euros)
96
97     print "\n\n\nThe shortest path optomizing for money is: "
98     j = 0
99     for i in shortestMoneyStack:
100         message = ""
101         if(i == toVertex(germanTrainGraph, "Hannover")):
102             message = "Buy 5 new iPads at 180 euros each"
103         if(i == toVertex(germanTrainGraph, "Basil")):
104             message = "Buy 2 new watches at 6,000 euros each"
105         if(i == toVertex(germanTrainGraph, "Koln")):
106             message = "Take a taxi 20 km round trip to see the
castle"
107         if(i == toVertex(germanTrainGraph, "Hamburg")):
108             message = "Take a taxi under the river"
109         j+=1
110         print j, ".", i, "\t", message
111     print "total time = ", shortestMoneyWeight.time
112     print "total euros = ", shortestMoneyWeight.euros
113     print "total weight = ", shortestMoneyWeight.euroTimeWeight
114     print "cost in dollars = ",
convertToDollars(shortestMoneyWeight.euros.euros)

```

```
jGRASP Wedge2

The shortest path optomizing for both is:
1 . Frankfurt
2 . Kassel
3 . Wiesbaden
4 . Mannhiem
5 . Karlsruhe
6 . Basil          Buy 2 new watches at 6,000 euros each
7 . Baden Baden    Spend the day at the spa
8 . Stuttgart
9 . Nurnberg
10 . Munich
11 . Dresden
12 . Leipzig
13 . Berlin
14 . Hamburg       Take a taxi under the river
15 . Lubeck
16 . Bremen
17 . Rostock
18 . Hannover       Buy 5 new iPads at 180 euros each
19 . Dusseldorf
20 . Koln           Take a taxi 20 km round trip to see the casle
21 . St. Augustin
22 . Bonn
23 . Frankfurt
total time = 72 Hours 12 Miniutes
total euros = 8762.24 Euros
total weight = 17389.99824
cost in dollars = $11829.03
```

```
jGRASP Wedge2

The shortest path optomizing for time is:
1 . Munich
2 . Nurnberg
3 . Dresden
4 . Leipzig
5 . Berlin
6 . Hamburg       Take a taxi under the river
7 . Lubeck
8 . Bremen
9 . Rostock
10 . Hannover      Buy 5 new iPads at 180 euros each
11 . Kassel
12 . Wiesbaden
13 . St. Augustin
14 . Koln          Take a taxi 20 km round trip to see the casle
15 . Dusseldorf
16 . Bonn
17 . Frankfurt
18 . Mannhiem
19 . Karlsruhe
20 . Basil         Buy 2 new watches at 6,000 euros each
21 . Baden Baden
22 . Stuttgart
total time = 64 Hours 54 Miniutes
total euros = 9652.87 Euros
total weight = 18697.628728
cost in dollars = $13031.38
```

jGRASP Wedge2

The shortest path optomizing for money is:

```
1 . Frankfurt
2 . Kassel
3 . Wiesbaden
4 . Mannhiem
5 . Karlsruhe
6 . Basil      Buy 2 new watches at 6,000 euros each
7 . Baden Baden
8 . Stuttgart
9 . Nurnberg
10 . Munich
11 . Dresden
12 . Leipzig
13 . Berlin
14 . Hamburg   Take a taxi under the river
15 . Lubeck
16 . Bremen
17 . Rostock
18 . Hannover  Buy 5 new iPads at 180 euros each
19 . Dusseldorf
20 . Koln      Take a taxi 20 km round trip to see the casle
21 . St. Augustin
22 . Bonn
23 . Frankfurt
total time = 72 Hours 12 Miniutes
total euros = 8762.24 Euros
total weight = 17389.99824
cost in dollars = $11829.03
```