

Nama : Fadlia

Nim : D0218333

Kelas : Informatika A 2018

TUGAS AKHIR INTELLIGENCE TRANSPORT SYSTEM

Senin 21 Desember 2020 kala turun hujan. Sebagai seorang programmer anda diminta merancang dan membangun aplikasi untuk menyelesaikan masalah optimasi transportasi. Anda diberi kebebasan untuk memilih algoritma terbaik dan menggunakan Bahasa pemrograman apa saja, tetapi harus dapat dijelaskan setiap detail proses di dalamnya (tidak menggunakan library algoritma). Library selain algoritma pencarian solusi misalnya library untuk membaca input output dan operasi standar misalnya untuk akar, modulo, random dan pangkat masih boleh digunakan. Tugas ini harus selesai sebelum tahun 2021.

1. DESKRIPSI PROJECT

Problem yang akan diselesaikan adalah: Bagaimana cara meminimalkan total jarak/biaya perjalanan untuk mengunjungi semua kota tepat satu kali dan kembali ke kota asal.

1. INPUT : Nama Dataset, Jumlah Kota, Label Kota dan Koordinat Kota
2. OUTPUT : Tour dengan biaya terendah untuk mengunjungi semua Kota dengan biaya minimum

2. FUNGSI OBJEKTIF

Fungsi objektif (fungsi tujuan) dari problem ini adalah meminimalkan total biaya perjalanan untuk mengunjungi semua kota tepat satu kali dan kembali ke kota asal (hanya kota asal yang dikunjungi dua kali).

3. BATASAN MASALAH

1. Semua kota dapat menjadi kandidat kota asal
2. Semua kota saling terhubung satu sama lain
3. Biaya perjalanan antar kota dihitung dari jarak titik koordinat antar kota menggunakan formula jarak euklid.
4. Program harus dapat menerima general input sesuai format, bukan hanya satu input, tetapi kita juga dapat mengganti-ganti nilai inputannya tetapi dengan format yang telah ditentukan. Misalnya untuk jumlah kota/verteks tidak harus selalu = 10 atau 11 atau 14. Kita dapat memasukkan input hingga ratusan verteks (kita batasi saja sampai 1000 verteks)
5. Dataset untuk pengujian source code dapat dilihat dan diedit dari:
<http://elib.zib.de/pub/mptestdata/tsp/tsplib/tsp/index.html>

CONTOH FORMAT INPUT:

NAME: burma14

NUM_VERTEX: 14

LABEL:

A;B;C;D;E;F;G;H;I;J;K;L;M;N

NODE COORD SECTION

X Y

16.47;96.10

16.47;94.44

20.09;92.54

22.39;93.37

25.23;97.24

22.00;96.05

20.47;97.02

17.20;96.29

16.30;97.38

14.05;98.12

16.53;97.38

21.52;95.59

19.41;97.13

20.09;94.55

Keterangan Input:

- ☐ Name = nama dataset contoh disini: burma14
- ☐ NUM_VERTEX = jumlah banyaknya kota/vertex contoh disini: 14. Berarti ada 14 kota
- ☐ LABEL = label kota. Label kota ditulis dalam satu baris dan dipisahkan oleh tanda titik koma (;). Contoh disini: A;B;C;D;E;F;G;H;I;J;K;L;M;N berarti ada 14 kota dengan label masing-masing adalah A, B, C, ..., N atau dapat dibaca sebagai Kota-A, Kota-B, Kota-C, ..., Kota-N.
- ☐ NODE_COORD_SECTION = menyatakan posisi koordinat masing-masing kota. Posisi koordinat ini dituliskan berturut-turut perbaris mewakili Label Verteks. Karena terdapat 14 Verteks maka akan ada 14 baris posisi verteks pula yang masing-masing terdiri dari posisi X dan Y dari verteks tersebut yang dipisahkan dengan penanda titik koma(;). Sebagai contoh untuk Kota-A akan berada di koordinat 16.47,96.10 atau Kota-A(16.47,96.10). ini berarti Kota-A berada pada titik X = 16.47 dan Y = 96.10. Kota-M(19.41, 97.13) berarti Kota-M berada pada titik X = 19.41 dan Y = 97.13.

CONTOH FORMAT OUTPUT

TOUR: I - J - A - B - N - C - D - E - F - L - G - M - H - K - I

Total Distance: 30.878503892588

Keterangan Output

- ☐ TOUR: I - J - A - B - N - C - D - E - F - L - G - M - H - K - I berarti bahwa Tour optimal yang berhasil ditemukan adalah berawal dari Kota-I dan berakhir di Kota-I dengan lintasan: I - J - A - B - N - C - D - E - F - L - G - M - H - K - I = dari Kota-I ke Kota-J, kemudian ke Kota-A, kemudian ke Kota-B, kemudian ke Kota-N, ... dan seterusnya hingga kembali ke Kota-I.

□ Total Distance: 30.878503892588 berarti total jarak optimum yang berhasil diperoleh adalah 30.878503892588. sesuai dengan fungsi objektif kita maka semakin kecil total jarak ini akan semakin baik. Jika seandainya ada algoritma yang dapat menemukan nilai total jarak yang lebih kecil lagi meskipun dengan TOUR berbeda, maka nilai total jarak terkecil itu yang akan kita pilih.

REQUIREMENT

1. Program dapat menerima input dan menghasilkan output dalam format TEXT. (10%)

Jawaban :

```
import os ,dataset,math,random #memanggil library yang akan digunakan
dataset=input("masukkan nama dataset : ") #menginputkan nama dataset dari keyboard
cities=input("masukkan jumlah kota : ") #menginputkan nama kota dari keyboard
label=input("masukkan label kota : ") #menginputkan label dari keyboard
kordinat=input("masukkan koordinat kota : ") #menginputkan tour dari keyboard
```

```
import os ,dataset,math,random #memanggil library yang akan digunakan
dataset=input("masukkan nama dataset : ")#menginputkan nama dataset dari keyboard
cities=input("masukkan jumlah kota : ")#menginputkan nama kota dari keyboard
label=input("masukkan label kota : ")#menginputkan label dari keyboard
kordinat=input("masukkan koordinat kota : ")#menginputkan tour dari keyboard
def Distance(v1,u1,v2,u2):
```

2. Terdapat penjelasan/deskripsi Problem yang akan diselesaikan. (10%)

Jawaban : Travelling Salesman Problem (TSP) merupakan salah satu permasalahan optimasi klasik yang sulit untuk dipecahkan secara konvensional. Penyelesaian eksak terhadap persoalan ini akan melibatkan algoritma yang mengharuskan mencari kemungkinan semua solusi yang ada, Sehingga akan terjadi ledakan kombinasi dan membuat kompleksitas waktu dari eksekusi algoritma sangat tinggi. Masalah TSP melibatkan seorang sales yang harus melakukan kunjungan ke sejumlah kota dalam menjajakan produknya. Rangkaian kota-kota yang dikunjungi harus membentuk suatu jalur sedemikian sehingga kota-kota tersebut hanya boleh dilewati tepat satu kali dan kemudian kembali lagi ke kota awal. Kasus seperti ini sering diistilahkan dengan sirkuit hamilton, representasinya dikenal dengan istilah hamiltonian. Penyelesaian terhadap permasalahan TSP ini adalah untuk memperoleh jalur terpendek. Penyelesaian eksak terhadap masalah TSP mengharuskan untuk melakukan perhitungan terhadap semua kemungkinan rute yang dapat diperoleh, kemudian memilih salah satu rute yang terpendek. Untuk itu jika terdapat kota yang harus dikunjungi, maka terdapat $n!$ kombinasi kota yang akan dibandingkan jarak masing-masing. Dengan cara ini waktu komputasi yang dibutuhkan akan jauh meningkat seiring dengan bertambahnya jumlah kota yang harus dikunjungi. Proses optimasi menggunakan data input berupa kota dan jarak antar kota kemudian diolah menggunakan algoritma greedy dengan penanda sehingga didapatkan jalur terpedek dan total biayanya. Penanda dalam algoritma digunakan agar setiap kota hanya sekali dikunjungi. Pendekatan algoritma greedy dengan penanda memberikan solusi yang mempunyai kompleksitas waktu

komputasi jauh lebih singkat dibandingkan dengan sejumlah algoritma lain seperti Algoritma Brute Force dan Dynamic Programming.

3. Terdapat studi literature tentang masalah yang akan diselesaikan (minimal 10 paper internasional 10 tahun terakhir dari IEEE, ELSEVIER, ACM, springer). (10%)

Jawaban : Studi Literature Tentang Masalah Yang Akan Diselesaikan:

1. Applegate, D. L., Bixby, R. E., Chvatal, V dan Cook, W. J. (2007), The Implementation of Greedy Algorithm for Solving Travelling Salesman Problem in a Distribution Company
2. Applegate, D. L., Bixby, R. E., Chvatal, V dan Cook, W. J. (2006), The Travelling Salesman Problem: A Computational Study, Princeton University Press, New Jersey.
3. Taha, H. A. (2007), Operations Research An Introduction Edisi 8, Pearson Prentice Hall, New Jersey.
4. Thomas H Cormen, "Introduction to Algorithms, Second Edition", The MIT Press, London, 2009.
5. Christian Nilsson, "Heuristic for Travelling Salesman Problem", Linköping University, 2010.
6. Simon Harris, "Beginning Algorithms", Wiley Publishing Inc., Indianapolis, 2010.
7. Gregory Gutin, "The Greedy Algorithm for the Symmetric TSP", Royal Holloway, University of London, 2009.
8. Iman, N. (2018). *Greedy Algorithm for Solving Travelling Salesman Problem in a Distribution Company* -Electronic Commerce Research and Applications, 30(May), 72–82.
9. Kang, J. (2018). *Implementation of Greedy Algorithm for Solving Travelling Salesman Problem Human-Centric Computing and Information Sciences*, 8(1), 32.
10. Kauffman, R. J., & Ma, D. (2015). Special issue: Contemporary research on Travelling Salesman Problem in the global revolution.

4. Terdapat Penjelasan Algoritma yang digunakan. (20%)

Jawaban : Algoritma Greedy Algoritma greedy merupakan algoritma yang memecahkan masalah langkah demi langkah dengan mengambil pilihan yang terbaik yang dapat diperoleh saat itu yang diistilahkan dengan optimum local. Algoritma ini berharap bahwa dengan memilih optimum lokal pada setiap langkah akan mencapai optimum global. Prinsipnya take what you can get now!, tidak ada waktu untuk balik mengecek ke belakang atau ke depan. Di dalam algoritma greedy prinsip Pencarian jalur terpendek memakai fungsi seleksi dan itu sangat berguna untuk menentukan jalan tersingkat untuk menuju suatu tempat sesuai dengan asumsi diatas. dalam penerapannya, algoritma ini tidak selalu mendapatkan solusi optimal namun pasti menemukan solusi. Kelebihan algoritma ini adalah kemampuannya menemukan solusi dengan jumlah node yang banyak dilakukan dengan sangat cepat. Algoritma greedy biasanya digunakan untuk menentukan memecahkan masalah lintasan linier dimana asal dan tujuan merupakan node yang berbeda. Hal ini menjadi kendala apabila diterapkan pada kasus TSP, dimana jalur yang ada merupakan jalur hamiltonian. Hal ini dapat diatasi dengan menggunakan penanda. B. Penanda TSP merupakan jalur Hamiltonian, sehingga setiap node dimungkinkan memiliki dua arah. Jika kita menggunakan algoritma greedy pada kasus ini, maka

harus dilengkapi dengan kemampuan untuk mengenali kota yang telah dilewati agar tidak terjadi perulangan. Dengan kata lain, satu kota hanya di kunjungi sekali. Untuk menyelesaikan hal tersebut, dibutuhkan sebuah penanda setiap kota yang telah dikunjungi. Kota atau node yang telah dikunjungi harus ditandai, misalnya inisialisasi setiap kota mempunyai tanda = 0, jika kota telah dikunjungi, maka tandanya berubah menjadi 1. Ketika proses pelacakan jalur terjadi, kota dengan tanda = 1 tidak perlu dilewati/ diproses. Dengan adanya penanda pada algoritma greedy, proses pelacakan dapat dilakukan tanpa ada kota yang dikunjungi berulang. Hal ini memudahkan pelacakan untuk kembali ke kota asal. Pseudo codenya adalah sebagai berikut :

- a. Input jumlah node
- b. Input jarak antar kota, dimana jarak yang diinput cukup sekali, misalnya cukup menginput jarak antara kota A ke kota B, secara otomatis, program akan mengisi jarak antara kota B ke kota A.
- c. Setiap yang telah dilewati diberi penanda = 1, sementara kota yang belum dilewati secara default mempunyai penanda = 0.
- d. Gunakan algoritma greedy untuk mencari optimum local step by step dengan tidak memproses kota yang telah dilewati (yaitu ditandai dengan penanda=1).
- e. Outpunya berupa jalur optimal sesuai prinsip greedy dan total distance yang dibutuhkan.

5. Terdapat Flowchart untuk Algoritma yang akan digunakan. (10%)

Jawaban : adapun bentuk flowchart nya yaitu



6. Terdapat Implementasi Source Code untuk menyelesaikan masalah. (20%)

Jawaban:

Implementasi Source Code untuk menyelesaikan masalah TSP yaitu menghitung tour dengan biaya terendah untuk mengunjungi semua kota dengan biaya minimum dengan algoritma greedy dengan bahasa pemrograman python :

```
import os ,dataset,math,random #memanggil library yang akan digunakan
dataset=input("masukkan nama dataset : ") #mengimputkan nama dataset dari keyboard
cities=input("masukkan jumlah kota : ") #mengimputkan nama kota dari keyboard
label=input("masukkan label kota : ") #mengimputkan label dari keyboard
kordinat=input("masukkan koordinat kota : ") #mengimputkan tour dari keyboard
def distance(x1,y1,x2,y2):
    return math.sqrt((x1-x2)**2+(y1-y2)**2) #fungsi menghitung jarak dua titik kota a-
                                         b;b-c,dst.
def totaldistancetour(tour):
    d=0 #fungsi menghitung total jarak yang akan terakumulasi dengan nilai awal =0
    print(cities) #mencetak nama kota yang dikunjungi
    print(tour) #mencetak jumlah kota yang dikunjungi
    for i in range(1,len(tour)): #perulangan yang akan terjadi akumulasi secara terus menerus
        hingga mencapai batas yang telah di tentukan
        x1=cities[tour[i-1]][0] #x1 merupakan kota ke 0 dari cities yang ke tour i-1(0) dengan dua
        kolom dengan koordinat x kolom 0
        y1=cities[tour[i-1]][1] #y1 merupakan kota ke 1 dari cities yang ke tour i-1(0) dengan dua
        kolom dengan koordinat y kolom 1
        x2=cities[tour[i]][0] #x1 merupakan kota ke 0 dari cities yang ke tour 1 dengan dua kolom
        dengan koordinat x kolom 0
        y2=cities[tour[i]][1] #x1 merupakan kota ke 0 dari cities yang ke tour 1 dengan dua kolom
        dengan koordinat x kolom 1
        d=d+distance(x1,y1,x2,y2) #menghitung jarak sementara dari satu kota ke satu kota
        lainnya
    print("jarak seentara" ,d)
    x1=cities[tour[len(tour)-1]][0]
    y1=cities[tour[len(tour)-1]][1]
    x2=cities[tour[0]][0]
    y2=cities[tour[0]][1]
    d=d+distance(x1,y1,x2,y2) #menghitung total jarak yang telah di lewati dan kembali ke
    kota awal
    return d
```

```

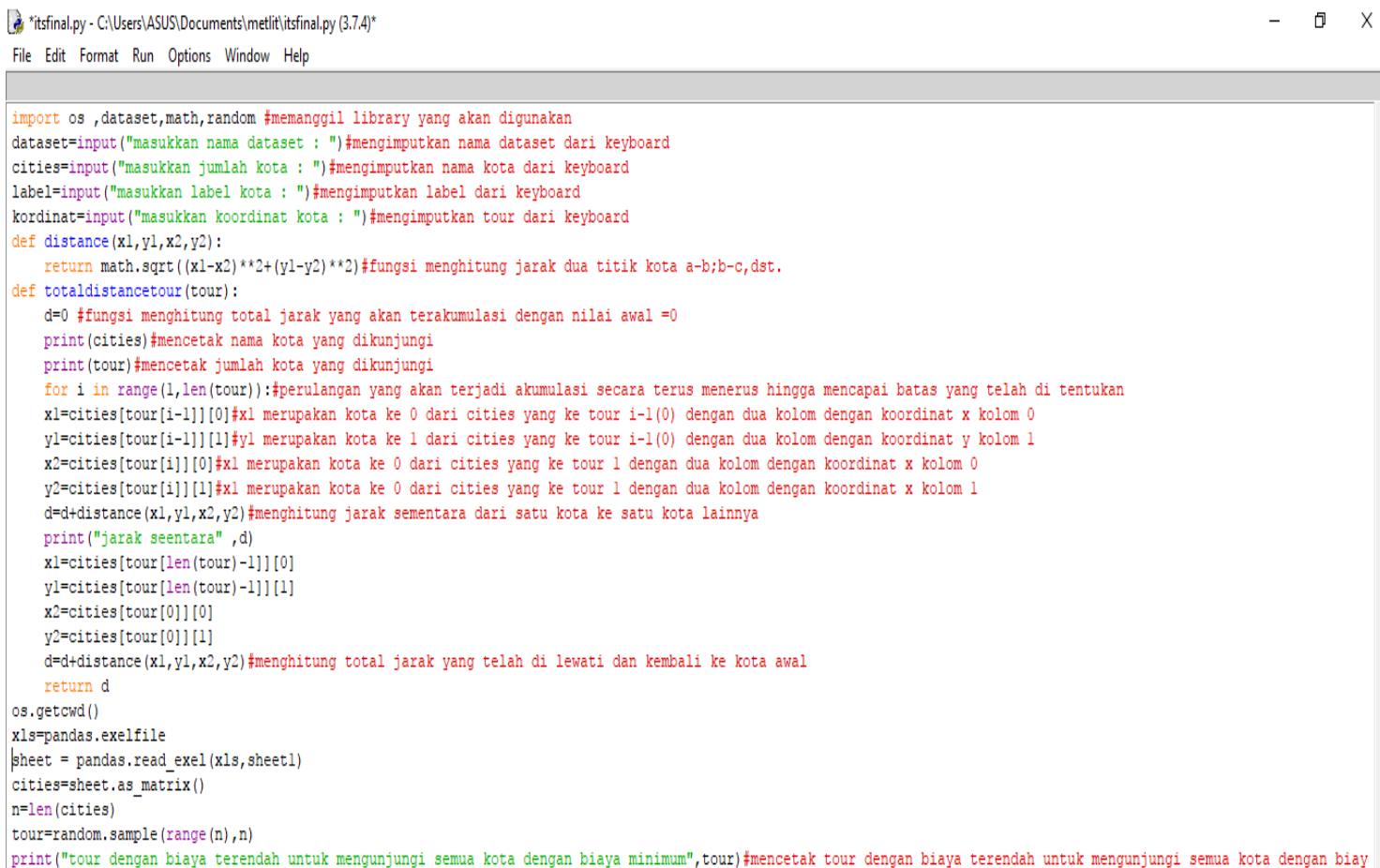
os.getcwd()
xls=pandas.exelfile
sheet = pandas.read_excel(xls,sheet1)
cities=sheet.as_matrix()
n=len(cities)
tour=random.sample(range(n),n)
print("tour dengan biaya terendah untuk mengunjungi semua kota dengan biaya minimum",tour) #mencetak
tour dengan biaya terendah untuk mengunjungi semua kota dengan biaya minimum

```

7. Terdapat Komentar Source Code yang mampu membuat programmer lain mudah memahami source code yang dibuat. (10%)

Jawaban:

Keterangan source code diawali dengan hastag (#)



```

*itsfinal.py - C:\Users\ASUS\Documents\metit\itsfinal.py (3.7.4)*
File Edit Format Run Options Window Help

import os ,dataset,math,random #memanggil library yang akan digunakan
dataset=input("masukkan nama dataset : ")#menginputkan nama dataset dari keyboard
cities=input("masukkan jumlah kota : ")#menginputkan nama kota dari keyboard
label=input("masukkan label kota : ")#menginputkan label dari keyboard
kordinat=input("masukkan koordinat kota : ")#menginputkan tour dari keyboard
def distance(x1,y1,x2,y2):
    return math.sqrt((x1-x2)**2+(y1-y2)**2)#fungsi menghitung jarak dua titik kota a-b;b-c,dst.
def totaldistancetour(tour):
    d=0 #fungsi menghitung total jarak yang akan terakumulasi dengan nilai awal =0
    print(cities)#mencetak nama kota yang dikunjungi
    print(tour)#mencetak jumlah kota yang dikunjungi
    for i in range(1,len(tour)):#perulangan yang akan terjadi akumulasi secara terus menerus hingga mencapai batas yang telah di tentukan
        x1=cities[tour[i-1]][0]#x1 merupakan kota ke 0 dari cities yang ke tour i-1(0) dengan dua kolom dengan koordinat x kolom 0
        y1=cities[tour[i-1]][1]#y1 merupakan kota ke 1 dari cities yang ke tour i-1(0) dengan dua kolom dengan koordinat y kolom 1
        x2=cities[tour[i]][0]#x2 merupakan kota ke 0 dari cities yang ke tour i dengan dua kolom dengan koordinat x kolom 0
        y2=cities[tour[i]][1]#y2 merupakan kota ke 1 dari cities yang ke tour i dengan dua kolom dengan koordinat x kolom 1
        d=d+distance(x1,y1,x2,y2)#menghitung jarak sementara dari satu kota ke satu kota lainnya
        print("jarak seantara" ,d)
        x1=cities[tour[len(tour)-1]][0]
        y1=cities[tour[len(tour)-1]][1]
        x2=cities[tour[0]][0]
        y2=cities[tour[0]][1]
        d=d+distance(x1,y1,x2,y2)#menghitung total jarak yang telah di lewati dan kembali ke kota awal
    return d
os.getcwd()
xls=pandas.exelfile
sheet = pandas.read_excel(xls,sheet1)
cities=sheet.as_matrix()
n=len(cities)
tour=random.sample(range(n),n)
print("tour dengan biaya terendah untuk mengunjungi semua kota dengan biaya minimum",tour)#mencetak tour dengan biaya terendah untuk mengunjungi semua kota dengan biay

```

8. Terdapat Pengujian Program. (10%)

Jawaban: Screenshote Pengujian Program

```

import os, itertools, math, random, pandas

def distance(x1,y1,x2,y2):
    return math.sqrt((x1-x2)**2+(y1-y2)**2)

def totaldistancetour(tour):
    d=0

```

```
print(cities)
```

```
print(tour)
```

```
for i in range(1, len(tour)):
```

```
    x1=cities[tour[i-1]][0]
```

```
    y1=cities[tour[i-1]][1]
```

```
    x2=cities[tour[i]][0]
```

```
    y2=cities[tour[i]][1]
```

```
    d=d+distance(x1,y1,x2,y2)
```

```
    print("jarak sementara".d)
```

```
x1=cities[tour[len(tour)-1]][0]
```

```
y1=cities[tour[len(tour)-1]][1]
```

```
x2=cities[tour[0]][0]
```

```
y2=cities[tour[0]][1]
```

```
d=d+distance(x1,y1,x2,y2)
```

```
return d
```

```
os.getcwd ()
```

```
xls = pandas.ExcelFile('Uji coba import coordinate 15 titik.xls')
```

```
xls = pandas.read_excel(xls, 'Sheet1')
```

```
cities=sheet.as_matrix()
```

```
n=len(cities)
```

```
tour = random.sample(range(n),n);
```

```
1 import os, itertools, math, random, pandas
2
3 def distance(x1,y1,x2,y2):
4     return math.sqrt((x1-x2)**2+(y1-y2)**2)
5
6 def totaldistancetour(tour):
7     d=0
8     print(cities)
9     print(tour)
10    for i in range(1,len(tour)):
11        x1=cities[tour[i-1]][0]
12        y1=cities[tour[i-1]][1]
13        x2=cities[tour[i]][0]
14        y2=cities[tour[i]][1]
15        d=d+distance(x1,y1,x2,y2)
16    print("jarak sementara",d)
17    x1=cities[tour[len(tour)-1]][0]
18    y1=cities[tour[len(tour)-1]][1]
19    x2=cities[tour[0]][0]
20    y2=cities[tour[0]][1]
21    d=d+distance(x1,y1,x2,y2)
22    return d
23
24 os.getcwd()
25 xls = pandas.ExcelFile('Uji coba import coordinate 15 titik
26 sheet = pandas.read_excel(xls, 'Sheet1')
27 cities=sheet.as_matrix()
28 n=len(cities)
29 tour = random.sample(range(n),n);
```

The screenshot shows the output of the code execution in a Jupyter Notebook. The top part displays the output of the `print(cities)` and `print(tour)` statements. The `cities` variable is a 15x2 array of coordinates, and the `tour` variable is a list of indices representing the order of cities visited. Below the output, the 'Variable explorer' panel shows the variables `allpossiblestour` (a list of size 362880) and `cities` (a 15x2 array). The 'Python console' panel at the bottom shows the Python version (3.6.0) and the Anaconda version (4.3.1).

Name	Type	Size	Value
allpossiblestour	list	362880	[[1, 6, ...]]
cities	int64	(15, 2)	array([...])

Python 3.6.0 [Anaconda 4.3.1 (64-bit)]
(default, Dec 23 2016, 11:57:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.