Question 1

June 14, 2019

[10]:
```
import pandas as pd;
import numpy as np;
import time;
```

读取数据并生成邻接矩阵

[11]:
```
mat10 = np.zeros((10,10));
mat100 = np.zeros((100, 100));
data10 = pd.read_csv('TSP10cities.tsp', sep = ' ', header = None);
data100 = pd.read_csv('TSP100cities.tsp', sep = ' ', header = None);
```

[12]:
```
for i in range(10):
    for j in range(10):
        x_dist = float(data10.iloc[i, 1]) - float(data10.iloc[j, 1]);
        y_dist = float(data10.iloc[i, 2]) - float(data10.iloc[j, 2]);
        mat10[i, j] = np.sqrt(pow(x_dist, 2) + pow(y_dist, 2));
```

[13]:
```
for i in range(100):
    for j in range(100):
        x_dist = float(data100.iloc[i, 1]) - float(data100.iloc[j, 1]);
        y_dist = float(data100.iloc[i, 2]) - float(data100.iloc[j, 2]);
        mat100[i, j] = np.sqrt(pow(x_dist, 2) + pow(y_dist, 2));
```

[14]:
```
np.savetxt('data10.csv', mat10, delimiter = ',')
np.savetxt('data100.csv', mat100, delimiter = ',')
```

使用贪心算法解决 TSP 问题

问题规模为 n=10

[15]:
```
start = time.clock();
city_visited = [0];
total_dist = 0;
INF = float("inf");
step = 1;
nextCity = 1;
while step < 10:
    v = 1;
    current_dist = INF;
    while v < 10:
        if (v not in city_visited) and (mat10[v][city_visited[-1]] <␣
↪current_dist):
```

```
            nextCity = v;
            current_dist = mat10[v][city_visited[-1]];

        v += 1;
    city_visited.append(nextCity);
    total_dist += current_dist;
    step += 1;
total_dist += mat10[0][city_visited[-1]];
city_visited.append(0);
city_visited = [num + 1 for num in city_visited];
end = time.clock();
```

[16]:
```
print("n = 10 时：")
print("所经过的总距离为：  %f" % total_dist);
print("经过的城市序号为：", city_visited)
print("程序运行时间：  %s" % (end - start))
```

```
n = 10 时：
所经过的总距离为：  10464.183487
经过的城市序号为：  [1, 10, 9, 6, 4, 3, 2, 8, 5, 7, 1]
程序运行时间：  0.001259797635952964
```

问题规模为 n=100

[17]:
```
start = time.clock();
city_visited = [0];
total_dist = 0;
INF = float("inf");
step = 1;
nextCity = 1;
while step < 100:
    v = 1;
    current_dist = INF;
    while v < 100:
        if (v not in city_visited) and (mat100[v][city_visited[-1]] <␣
 ↪current_dist):
            nextCity = v;
            current_dist = mat100[v][city_visited[-1]];

        v += 1;
    city_visited.append(nextCity);
    total_dist += current_dist;
    step += 1;
total_dist += mat100[0][city_visited[-1]];
city_visited.append(0);
city_visited = [num + 1 for num in city_visited];
end = time.clock();
```

[19]:
```
print("n = 100 时: ")
print("所经过的总距离为:  %f" % total_dist);
print("经过的城市序号为: ", city_visited)
print("程序运行时间:  %s" % (end - start))
```

```
n = 100 时:
总距离为:  28352.214389
经过的城市序号为:  [1, 22, 73, 93, 46, 18, 33, 62, 17, 69, 53, 88, 26, 75, 39,␣
 ↪65, 51,
45, 64, 43, 15, 49, 21, 84, 61, 54, 87, 34, 89, 47, 2, 60, 71, 77, 72, 66, 57,
13, 95, 63, 79, 67, 50, 52, 40, 55, 10, 31, 58, 36, 30, 78, 28, 85, 92, 97, 27,
70, 20, 35, 4, 48, 42, 38, 25, 56, 8, 80, 44, 100, 24, 81, 98, 94, 74, 96, 29,
23, 59, 83, 91, 14, 9, 16, 76, 19, 3, 7, 41, 32, 37, 82, 12, 86, 11, 99, 90, 5,
6, 68, 1]
程序运行时间:  0.01884235150089353
```

使用动态规划解决 TSP 问题

问题规模为 n = 10

[24]:
```
start = time.clock();
num_city = mat10.shape[0]
mat = np.zeros((num_city, num_city));
dp_mat = np.zeros((num_city,1<<(num_city-1)));
mat = mat10;
for i in range(10):
    mat[i][i] = INF;
for i in range(num_city):
    dp_mat[i][0] = mat[i][0];
for j in range(1, (1<<(num_city-1))):
    for i in range(num_city):
        dp_mat[i,j]=INF;
        if i>=1:
            if ((j>>(i-1))&1)==1:
                continue;
        for k in range(1, num_city):
            if ((j>>(k-1))&1)!=0:
                if dp_mat[i,j]>float(mat[i,k])+dp_mat[k,j^(1<<(k-1))]:
                    dp_mat[i,j]=float(mat[i,k])+dp_mat[k,j^(1<<(k-1))];
end = time.clock();
```

[26]:
```
print("n = 10 时: ")
print("所经过的总距离为:  %f" % dp_mat[0,(1<<(num_city-1))-1]);
print("程序运行时间:  %s" % (end - start))
```

```
n = 10 时:
所经过的总距离为:  10127.552144
程序运行时间:  0.050572811126471606
```

问题规模为 n = 100

[27]:
```
num_city = mat100.shape[0]
mat = np.zeros((num_city, num_city));
dp_mat = np.zeros((num_city,1<<(num_city-1)));
start = time.clock();
num_city = mat100.shape[0]
mat = np.zeros((num_city, num_city));
dp_mat = np.zeros((num_city,1<<(num_city-1)));
mat = mat100;
for i in range(num_city):
    mat[i][i] = INF;
for i in range(num_city):
    dp_mat[i][0] = mat[i][0];
for j in range(1, (1<<(num_city-1))):
    for i in range(num_city):
        dp_mat[i,j]=INF;
        if i>=1:
            if ((j>>(i-1))&1)==1:
                continue;
        for k in range(1, num_city):
            if ((j>>(k-1))&1)!=0:
                if dp_mat[i,j]>float(mat[i,k])+dp_mat[k,j^(1<<(k-1))]:
                    dp_mat[i,j]=float(mat[i,k])+dp_mat[k,j^(1<<(k-1))];
end = time.clock();
```

```
␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-27-6b5480123e30> in <module>
         1 num_city = mat100.shape[0]
         2 mat = np.zeros((num_city, num_city));
   ----> 3 dp_mat = np.zeros((num_city,1<<(num_city-1)));
         4 start = time.clock();
         5 num_city = mat100.shape[0]


       ValueError: Maximum allowed dimension exceeded
```

使用分支界限解决 TSP 问题
问题规模为 n = 10

[63]:
```
num_city = mat10.shape[0]
mat = np.zeros((num_city, num_city));
mat = mat10;
```

```
for i in range(num_city):
    mat[i][i] = INF;
```

[64]:
```
from queue import Queue;
class node:
    def __init__(self):
        self.start = 0;
        self.end = 0;
        self.k = 1;
        self.total_dist = 0;
        self.part_low = 0;
        self.visited = [0] * num_city;
        self.path = [];
up = 0;
low = 0;
used = [0] * num_city;
q = Queue();
used[0] = 1;
```

[65]:
```
def up_bound():
    global up;
    up = dfs(0, 0, 0);
```

[66]:
```
def low_bound():
    global low;
    for i in range(num_city):
        tmp = mat[i].copy();
        tmp.sort();
        low = low + tmp[0] + tmp[1];
    low = low / 2;
```

[67]:
```
def get_part_low(v):
    result = 2 * v.total_dist;
    min1 = INF;
    min2 = INF;
    for i in range(num_city):
        if(v.visited[i] == 0 and min1 > mat[v.start][i]):
            min1 = mat[v.start][i];
    result += min1;
    for i in range(num_city):
        if(v.visited[i] == 0 and min2 > mat[v.end][i]):
            min2 = mat[v.end][i];
    result += min2;
    for i in range(num_city):
        if(v.visited[i] == 0):
            min1 = INF;
            min2 = INF;
            for j in range(num_city):
                if(min1 > mat[i][j]):
```

```
                    min1 = mat[i][j];
            for j in range(num_city):
                if(min2 > mat[i][j]):
                    min2 = mat[i][j];
            result += min1 + min2;
    return result / 2;
```

[68]:
```
def dfs(v, j, length):
    if(j == num_city - 1):
        return length + mat[0][v];
    minLength = INF;
    pos = 0;
    for i in range(num_city):
        if(used[i] == 0 and minLength > mat[i][v]):
            minLength = mat[i][v];
            pos = i;
    used[pos] = 1;
    return dfs(pos, j+1, length + minLength);
```

[69]:
```
def compute():
    global up;
    up_bound();
    low_bound();
    v = node();
    v.visited[0] = 1;
    v.path.append(0);
    v.part_low = low;
    result = INF;
    q.put(v);
    next_node = node();
    while(q.empty() == False):
        tmp = q.get();
        if(tmp.k == num_city - 1):
            pos = 0;
            for i in range(num_city):
                if(tmp.visited[i] == 0):
                    pos = i;
                    break;
            ans = tmp.total_dist + mat[pos][tmp.start] + mat[pos][tmp.end];
            if(ans <= tmp.part_low):
                result = min(ans, result);
                break;
            else:
                up = min(up, ans);
                result = min(ans, result);
                continue;
        for i in range(num_city):
            if(tmp.visited[i] == 0):
```

```
                next_node = node();
                next_node.start = tmp.start;
                next_node.total_dist = tmp.total_dist + mat[i][tmp.end];
                next_node.end = i;
                next_node.k = tmp.k + 1;
                next_node.visited = tmp.visited.copy();
                next_node.visited[i] = 1;
                next_node.path = tmp.path.copy();
                next_node.path.append(i);
                next_node.part_low = get_part_low(next_node);
                if(next_node.part_low >= up):
                    continue;
                q.put(next_node);
    return result,tmp;
```

[70]:
```
start = time.clock()
dist,vex=compute()
end = time.clock()
```

[71]:
```
print("n = 10 时: ")
print("所经过的总距离为:  %f" % dist);
print("经过的城市序号为: ", vex.path)
print("程序运行时间:  %s" % (end - start))
```

```
n = 10 时:
所经过的总距离为:  10127.552144
经过的城市序号为:  [0, 9, 8, 5, 3, 2, 1, 7, 6]
程序运行时间:  0.3125276620726254
```

问题规模为 n = 100

[84]:
```
num_city = mat100.shape[0]
mat = np.zeros((num_city, num_city));
mat = mat100;
for i in range(num_city):
    mat[i][i] = INF;
class node:
    def __init__(self):
        self.start = 0;
        self.end = 0;
        self.k = 1;
        self.total_dist = 0;
        self.part_low = 0;
        self.visited = [0] * num_city;
        self.path = [];
up = 0;
low = 0;
used = [0] * num_city;
q = Queue();
```

```
used[0] = 1;
```

一下代码无法在有效时间内运行完成

[ ]:
```
start = time.clock()
dist,vex=compute()
end = time.clock()
```