


第5回 経路の最適化

1922074 木村太紀



1.空間的距離計算を実現する

任意のスタート地点を入力する事で、それ以外の全地点への距離を出力可能

出力結果

```
武蔵国分寺から武蔵国分寺跡までの距離
1.1141130658575487km
-----
武蔵国分寺から都立武蔵国分寺公園までの距離
0.15453625763876822km
-----
武蔵国分寺からお鷹の道・真姿の池湧水群までの距離
0.05440833418085095km
-----
武蔵国分寺から都立小金井公園までの距離
3.1272822169149106km
-----
武蔵国分寺から江戸東京たてもの園までの距離
2.434626101368746km
-----
武蔵国分寺からけやき公園までの距離
1.1351460161541524km
-----
武蔵国分寺から昭和記念公園までの距離
4.420672840577104km
-----
武蔵国分寺から玉川上水緑道までの距離
3.4103306392378583km
-----
武蔵国分寺から谷保天満宮までの距離
1.9214893671464357km
-----
```

[作成したスクリプトのリンク](#)

```
from dataaccess import DataAccess
import itertools
import math

hoge = DataAccess()
count = 0
list_latitude = {}
list_longitude = {}
#赤緯、経半径
red_rad = 6378.137
pol_rad = 6356.752
name = {}
name_ex = {}
#化成緯度、経度それぞれ辞書型に格納
for id_n in range(1,11):
    x = list(itertools.chain.from_iterable(hoge.get_spots(id_n)))
    #化成緯度のリスト
    list_latitude[count] = x[2]
    list_longitude[count] = x[3]
    name[count] = x[0]
    name_ex[count] = x[0]
    count += 1

def make_dist(point_A,point_B):
    print(point_A + "から" + point_B + "までの距離")
    #適当なidを取得
    for i in name:
        if name[i] == point_A:
            id_A = i
        elif name[i] == point_B:
            id_B = i

    param_rati_A = math.atan(red_rad/pol_rad * math.tan(list_latitude[id_A]))
    param_rati_B = math.atan(red_rad/pol_rad * math.tan(list_latitude[id_B]))

    dist_sphere = math.acos(math.sin(param_rati_A) * math.sin(param_rati_B) + math.cos(param_rati_A) * math.cos(param_rati_B))
    s(list_longitude[id_A]-list_longitude[id_B])

    corr_1 = (math.sin(dist_sphere)-dist_sphere) * (math.sin(param_rati_A)+math.sin(param_rati_B))**2 / math.cos(dist_sphere/2)**2
    corr_2 = (math.sin(dist_sphere)+dist_sphere) * (math.sin(param_rati_A)-math.sin(param_rati_B))**2 / math.cos(dist_sphere/2)**2
    dist_corr = ((red_rad - pol_rad) / red_rad)/8 * (corr_1 - corr_2)

    final_dist = red_rad * (dist_sphere + dist_corr)
    print(str(final_dist) + "メートル")
    〇〇〇

def set_point(point_A):
    for i in name:
        if name[i] == point_A:
            point_A_id = i
    del name_ex[point_A_id]
    for i in name_ex:
        point_B = name[i]
        make_dist(point_A,point_B)
        print("-----")

point_A = "武蔵国分寺"
set_point(point_A)
```

2.時速4km/hとして、2点間の移動時間を算出

出力結果

```
武蔵国分寺から武蔵国分寺跡までの時間
16.71169598786323分
-----
武蔵国分寺から都立武蔵国分寺公園までの時間
2.318043864581523分
-----
武蔵国分寺からお鷹の道・真姿の池湧水群までの時間
0.8161250127127642分
-----
武蔵国分寺から都立小金井公園までの時間
46.909233253723656分
-----
武蔵国分寺から江戸東京たてもの園までの時間
36.51939152053119分
-----
武蔵国分寺からけやき公園までの時間
17.027190242312287分
-----
武蔵国分寺から昭和記念公園までの時間
66.31009260865656分
-----
武蔵国分寺から玉川上水緑道までの時間
51.15495958856788分
-----
武蔵国分寺から谷保天満宮までの時間
28.822340507196536分
-----
```

作成したスクリプトのリンク

```
from dataaccess import DataAccess
import itertools
import math

hoge = DataAccess()
count = 0
list_latitude = {}
list_longitude = {}
#経度、緯度
red_rad = 6378.137
pol_rad = 6356.752
name = {}
name_ex = {}
#地点の距離、経度それぞれ辞書型に格納
for id_n in range(1,11):
    x = list(itertools.chain.from_iterable(hoge.get_spots(id_n)))
    #地点の経度
    list_latitude[count] = x[2]
    list_longitude[count] = x[3]
    name[count] = x[0]
    name_ex[count] = x[8]
    count += 1

def make_dist(point_A, point_B):
    print(point_A + "から" + point_B + "までの時間")
    #地点の名称取得
    for i in name:
        if name[i] == point_A:
            id_A = i
        elif name[i] == point_B:
            id_B = i

    rad_lat_A = math.radians(list_latitude[id_A])
    rad_long_A = math.radians(list_longitude[id_A])
    rad_lat_B = math.radians(list_latitude[id_B])
    rad_long_B = math.radians(list_longitude[id_B])

    param_rati_A = math.atan(pol_rad/red_rad * math.tan(rad_lat_A))
    param_rati_B = math.atan(pol_rad/red_rad * math.tan(rad_lat_B))

    dist_sphere = math.acos(math.sin(param_rati_A) * math.sin(param_rati_B) + math.cos(param_rati_A) * math.cos(param_rati_B)) + math.cos(rad_long_A - rad_long_B)

    corr_1 = (math.sin(dist_sphere)-dist_sphere) * (math.sin(param_rati_A)-math.sin(param_rati_B))*2 / math.cos(dist_sphere/2)**2
    corr_2 = (math.sin(dist_sphere)+dist_sphere) * (math.sin(param_rati_A)-math.sin(param_rati_B))*2 / math.cos(dist_sphere/2)**2
    f = (red_rad - pol_rad) / red_rad
    dist_corr = f/8 * (corr_1 - corr_2)

    final_dist = red_rad * (dist_sphere + dist_corr)
    time = final_dist/4 *60
    print(str(time) + "分")

def set_point(point_A):
    for i in name:
        if name[i] == point_A:
            point_A_id = i
            del name_ex[point_A_id]
    for i in name_ex:
        point_B = name[i]
        make_dist(point_A, point_B)
        print("-----")

#地点を指定する事で、それ以外の地点との距離、時間を計算
point_A = "武蔵国分寺"
set_point(point_A)
```

3.各スポットにおいて推奨滞在時間をDBに設定

ID	spot_name	spot_location	spot_latitude	spot_longitude	spot_type	nearest_station	when_build	open	close	rating_5Lv	history	culture	nature	religion	amusement	stay_time
1	武蔵国分寺跡	国分寺	35.423802	139.2828026	寺社	国分寺	750	24	24	3	1	1	0	1	0	10
2	都立武蔵国分寺公園	国分寺	35.4151912	139.2818347	公園	西国分寺	2002	24	24	3	0	0	1	0	1	30
3	お鷹の道・真姿の池湧水群	国分寺	35.413903	139.2825565	自然	国分寺	847	24	24	5	0	0	1	0	0	30
4	武蔵国分寺	国分寺	35.4138058	139.2819694	寺社	国分寺	1333	24	24	4	1	1	0	1	0	15
5	都立小金井公園	小金井	35.4300291	139.3100976	公園	小金井	1954	24	24	5	0	0	1	0	1	60
6	江戸東京たてもの園	小金井	35.4256399	139.3045236	博物館	小金井	1993	9	16	3	1	1	0	0	1	45
7	けやき公園	国分寺	35.4239893	139.2828434	公園	国分寺	1980	24	24	5	0	0	1	0	1	40
8	昭和記念公園	立川	35.424197	139.2349866	公園	立川	1983	24	24	5	0	0	1	0	1	70
9	玉川上水緑道	国分寺・小金井・小平・立川	35.4347169	139.2544989	遊歩道	国分寺・小金井	1653	24	24	4	0	0	1	0	1	30
10	谷保天満宮	国立	35.4048379	139.2638867	寺社	国立	903	24	24	4	1	1	0	1	0	20

- ・武蔵国分寺跡:10分
- ・都立武蔵国分寺公園:30分
- ・お鷹の道/真姿の池湧水群:30分
- ・武蔵国分寺:15分
- ・都立小金井公園:60分
- ・江戸東京たてもの園:45分
- ・けやき公園:40分
- ・昭和記念公園:70分
- ・玉川上水緑道:30分
- ・谷保天満宮:20分

[使用CSVのリンク](#)

[DB作成用リンク\(追加に使用したSQLは別\)](#)

4. スポットを最短で巡るルートを算出

出力結果

```
~最適な道順~
-----
武蔵国分寺
↓
お鷹の道・真姿の池湧水群
↓
都立武蔵国分寺公園
↓
武蔵国分寺跡
↓
けやき公園
↓
江戸東京たてももの園
↓
都立小金井公園
↓
谷保天満宮
↓
昭和記念公園
↓
玉川上水緑道
↓
終了！
-----
```

主な関数

```
def salesman(point_abc):
    loops = len(name_box)
    print("~最適な道順~")
    print("-----")
    while loops > 0:
        point_A = point_abc
        min_count = 0
        set_point(point_A)
        print(point_A)
        print("↓")
        if loops > 1:
            for get_min in time_box:
                if min(time_box) == get_min:
                    name_box.remove(point_A)
                    point_abc = name_box[min_count]
                    loops = loops - 1
            else:
                min_count += 1
        else:
            print("終了!")
            print("-----")
    time_box.clear()
```

使用データは10件/[作成したスクリプトのリンク](#)