

Pathfinding

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
# import the magic code for using sql in jupyter notebook
%load_ext sql
# local database
# %sql mysql+pymysql://root:fjwwzx970814@localhost/mydb
# remote database
```

```
%sql mysql+pymysql://brickeamac:fjwWZX970814@rm-
0xih4pk94w41k3c5j8o.mysql.rds.aliyuncs.com/mydb
```

```
'Connected: brickeamac@mydb'
```

```
%sql mysql+pymysql://nity:BravoNity123@rm-
0xih4pk94w41k3c5j8o.mysql.rds.aliyuncs.com/mydb
```

```
'Connected: nity@mydb'
```

```
import sqlalchemy as sqlManager
```

```
# Create connection with database
connection =
sqlManager.create_engine('mysql+pymysql://brickeamac:fjwWZX970814@rm-
0xih4pk94w41k3c5j8o.mysql.rds.aliyuncs.com/mydb?charset=utf8')
```

```
connection =
sqlManager.create_engine('mysql+pymysql://nity:BravoNity123@rm-
0xih4pk94w41k3c5j8o.mysql.rds.aliyuncs.com/mydb?charset=utf8')
```

Procedures and Function test

```
%%sql
SELECT * FROM get_user_and_device;
```

```
* mysql+pymysql://bricke_mac:***@rm-
0xih4pk94w41k3c5j8o.mysql.rds.aliyuncs.com/mydb
15 rows affected.
```

idcustomer	firstName	lastName	iddevice	deviceType	NumConnectionsAvailable
1	first_name_0	last_name_0	2	Phone	3
1	first_name_0	last_name_0	6	Laptop	0
1	first_name_0	last_name_0	7	Ipad	0
2	first_name_1	last_name_1	3	Phone	4
2	first_name_1	last_name_1	9	Ipad	0
3	first_name_2	last_name_2	15	Phone	0
4	first_name_3	last_name_3	4	Phone	3
4	first_name_3	last_name_3	8	Ipad	0
5	first_name_4	last_name_4	5	Phone	5
6	first_name_5	last_name_5	10	Phone	4
7	first_name_6	last_name_6	11	Phone	4
8	first_name_7	last_name_7	12	Phone	5
9	first_name_8	last_name_8	13	Phone	5
10	first_name_9	last_name_9	14	Phone	4
11	first_name_9	last_name_9	1	Phone	5

```
# Get best 3 connections for a particular device
iddevice = 2
sql = 'CALL get_device_current_floor_map_points('+str(iddevice)+');'
floor_points = pd.read_sql(sql,connection)
floor_points.iloc[210:]
# floor_points
```

```

/Users/brickeawang/opt/anaconda3/lib/python3.7/site-
packages/pymysql/cursors.py:170: Warning: (3719, "'utf8' is currently an
alias for the character set UTF8MB3, but will be an alias for UTF8MB4 in a
future release. Please consider using UTF8MB4 in order to be
unambiguous.")
    result = self._query(query)

```

```

.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}

```

	idfloorMap	coordinatePoint	idfloorMap	x	y	ifpassable
210	1	211	1	10.0	10.0	1
211	1	212	1	11.0	10.0	1
212	1	213	1	12.0	10.0	1
213	1	214	1	13.0	10.0	1
214	1	215	1	14.0	10.0	1
...
415	1	416	1	15.0	0.0	1
416	1	417	1	16.0	0.0	1
417	1	418	1	17.0	0.0	1
418	1	419	1	18.0	0.0	1
419	1	420	1	19.0	0.0	1

210 rows × 6 columns

```

y_len = int(max(floor_points.x) + 1)
y_len

```

20

```
x_len = int(max(floor_points.y) + 1)
x_len
```

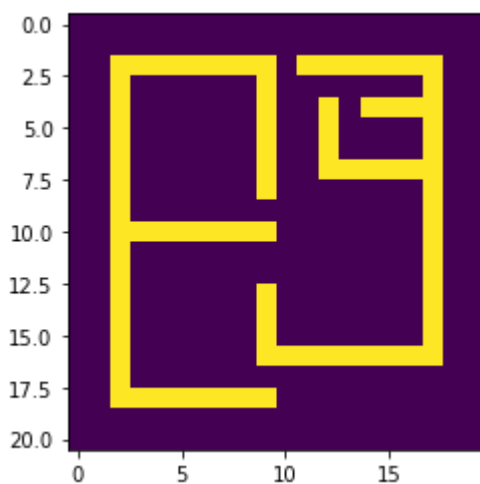
```
21
```

```
map_matirx = np.zeros((x_len,y_len))
```

```
for i in range(len(floor_points.coordinatePoint)):
    if floor_points.iloc[i].ifpassable == 0:
        # means this point is inside a wall
        matirx_x = int(20-floor_points.iloc[i].y)
        matirx_y = int(floor_points.iloc[i].x)
        map_matirx[matirx_x][matirx_y] = 1
```

```
plt.imshow(map_matirx)
```

```
<matplotlib.image.AxesImage at 0x11c360810>
```



```
def get_floor_map_points(iddevice):
    # This function is to get the device floor and load the data into
    matirx
    # Get the floor points data
    sql = 'CALL get_device_current_floor_map_points('+str(iddevice)+');'
    floor_points = pd.read_sql(sql,connection)
```

```

# Create matirx
y_len = int(max(floor_points.x) + 1)
x_len = int(max(floor_points.y) + 1)
map_matirx = np.zeros((x_len,y_len))

for i in range(len(floor_points.coordinatePoint)):
    if floor_points.iloc[i].ifpassable == 0:
        # means this point is inside a wall
        matirx_x = int(20-floor_points.iloc[i].y)
        matirx_y = int(floor_points.iloc[i].x)
        map_matirx[matirx_x][matirx_y] = 1
plt.imshow(map_matirx)

return map_matirx

```

Positioning simulationmap_matirx

```

# Trilateration Positioning Algorithm
def trilateration(beacons=None,distance=None):
    A = 2*(beacons.iloc[2].y - beacons.iloc[0].y)
    A_ = 2*(beacons.iloc[1].y - beacons.iloc[0].y)
    B = 2*(beacons.iloc[2].x - beacons.iloc[0].x)
    B_ = 2*(beacons.iloc[1].x - beacons.iloc[0].x)

    delta_1 = np.square(distance[0]) - np.square(distance[2]) +
    np.square(beacons.iloc[2].y) - np.square(beacons.iloc[0].y) +
    np.square(beacons.iloc[2].x) - np.square(beacons.iloc[0].x)
    delta_2 = np.square(distance[0]) - np.square(distance[1]) +
    np.square(beacons.iloc[1].y) - np.square(beacons.iloc[0].y) +
    np.square(beacons.iloc[1].x) - np.square(beacons.iloc[0].x)

    device_x = (delta_1 * A_ - delta_2 * A)/(B * A_ - B_ * A)
    device_y = (delta_1 * B_ - delta_2 * B)/(B_ * A - B * A_)

    return (round(device_x),round(device_y))

```

```

# Get best 3 connections for a particular device
iddevice = 2
sql = 'CALL
get_current_best_3_connections('+str(iddevice)+' ,@number,@c1,@c2,@c3);'
best_3_connection = pd.read_sql(sql,connection)
best_3_connection

```

```

.dataframe tbody tr th {
    vertical-align: top;
}

```

```
.dataframe thead th {
    text-align: right;
}
```

	num_available_connections	connection_1	connection_2	connection_3
0	3	6	8	7

```
# get connected 3 beacons coordinate and distance
sql = 'CALL get_3_connected_beacons_coordinate_and_distance('\
      +str(best_3_connection.connection_1[0])+',' +'\
      +str(best_3_connection.connection_2[0])+',' +'\
      +str(best_3_connection.connection_3[0])+')'
coordinate_distance = pd.read_sql(sql,connection)
coordinate_distance
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y	distance	idfloor
0	11	14	8	1
1	7	15	12	1
2	4	13	14	1

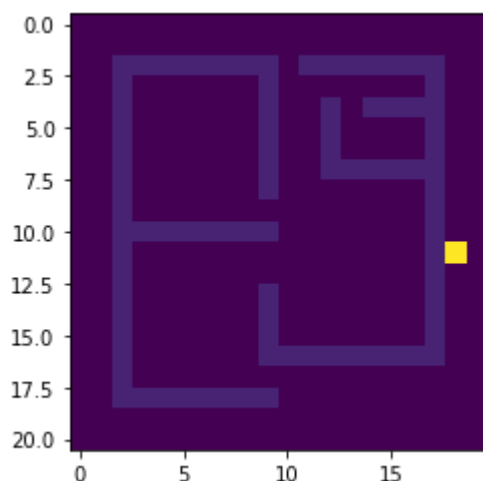
```
current_x,current_y = trilateration(beacons = coordinate_distance,distance
= coordinate_distance.distance)
(current_x,current_y)
```

```
(18.0, 9.0)
```

```
matirx_current_x = int(20 - current_y)
matirx_current_y = int(current_x)
map_matirx[matirx_current_x,matirx_current_y] = 10
```

```
plt.imshow(map_matirx)
```

```
<matplotlib.image.AxesImage at 0x11c425f50>
```



```
# Trilateration Positioning Algorithm
def trilateration(beacons=None,distance=None):
    A = 2*(beacons.iloc[2].y - beacons.iloc[0].y)
    A_ = 2*(beacons.iloc[1].y - beacons.iloc[0].y)
    B = 2*(beacons.iloc[2].x - beacons.iloc[0].x)
    B_ = 2*(beacons.iloc[1].x - beacons.iloc[0].x)

    delta_1 = np.square(distance[0]) - np.square(distance[2]) +
np.square(beacons.iloc[2].y) - np.square(beacons.iloc[0].y) +
np.square(beacons.iloc[2].x) - np.square(beacons.iloc[0].x)
    delta_2 = np.square(distance[0]) - np.square(distance[1]) +
np.square(beacons.iloc[1].y) - np.square(beacons.iloc[0].y) +
np.square(beacons.iloc[1].x) - np.square(beacons.iloc[0].x)

    device_x = (delta_1 * A_ - delta_2 * A)/(B * A_ - B_ * A)
    device_y = (delta_1 * B_ - delta_2 * B)/(B_ * A - B * A_)

    return (round(device_x),round(device_y))

def get_current_position(iddevice):
    # This function will return the device current position coordinate
    data

    # Get best 3 connections for a particular device
    sql = 'CALL
get_current_best_3_connections('+str(iddevice)+',@number,@c1,@c2,@c3);'
    best_3_connection = pd.read_sql(sql,connection)
```

```

# get connected 3 beacons coordinate and distance
sql = 'CALL get_3_connected_beacons_coordinate_and_distance('\
      +str(best_3_connection.connection_1[0])+','+\
      +str(best_3_connection.connection_2[0])+','+\
      +str(best_3_connection.connection_3[0])+')'
coordinate_distance = pd.read_sql(sql,connection)

current_x,current_y = trilateration(beacons =
coordinate_distance,distance = coordinate_distance.distance)

return (current_x,current_y)

```

Single Floor Pathfinding

Breadth First Search Algorithm

```

# 这里我使用的是 广度优先搜索策略
# 为了方便用类来实现了

class BFS_PathFind(object):
    def __init__(self,start,end,current_map):
        # Start,end should be a tuple with x,y
        # current_map should be a 2-D array
        self.start = start
        self.end = end
        self.current_map = current_map.copy()
        self.map_shape = current_map.shape
        self.came_from = [[()] for i in range(current_map.shape[1])] for j
in range(current_map.shape[0])]

    def is_have_came_from(self,point):
        return not self.came_from[point[0]][point[1]] == ()

    def calculate_came_from(self):
        frontier = []
        frontier.append(self.start)
        self.came_from[self.start[0]][self.start[1]] = self.start
        while len(frontier)!=0:
            current = frontier.pop(0)

            if current == self.end:
                # If we found the end point the exit the algorithm
                return self.came_from

            # Neiborhood
            top = (current[0]-1,current[1])
            left = (current[0],current[1]-1)
            buttom = (current[0]+1,current[1])
            right = (current[0],current[1]+1)

            # Top path within map and is not a wall

```



```

        if top[0] > -1 and self.current_map[top[0]][top[1]]!=1:
            if not self.is_have_came_from(top):
                # If we dont have came from for this point
                self.came_from[top[0]][top[1]] = current
                frontier.append(top)

        # Left path within map and is not a wall
        if left[1] > -1 and self.current_map[left[0]][left[1]]!=1:
            if not self.is_have_came_from(left):
                # If we dont have came from for this point
                self.came_from[left[0]][left[1]] = current
                frontier.append(left)

        # Bottom path within map and is not a wall
        if bottom[0] < self.map_shape[0] and
self.current_map[bottom[0]][bottom[1]]!=1:
            if not self.is_have_came_from(bottom):
                # If we dont have came from for this point
                self.came_from[bottom[0]][bottom[1]] = current
                frontier.append(bottom)

        # Right path within map and is not a wall
        if right[1] < self.map_shape[1] and self.current_map[right[0]]
[right[1]]!=1:
            if not self.is_have_came_from(right):
                # If we dont have came from for this point
                self.came_from[right[0]][right[1]] = current
                frontier.append(right)
        return self.came_from

    def find_path(self):
        current = self.end
        while current != self.start:
            self.current_map[current[0]][current[1]] = 10
            if current == self.end:
                self.current_map[current[0]][current[1]] = 20
            current = self.came_from[current[0]][current[1]]
            self.current_map[current[0]][current[1]] = 5
        return self.current_map

```

```

def coordinate_to_matirx(coordinate_data,matirx):
    return (int(matirx.shape[0]-1-
coordinate_data[1]),int(coordinate_data[0]))

```

```

# Single Floor Pathfinding Simulation
iddevice = 2
destination = (5,5)

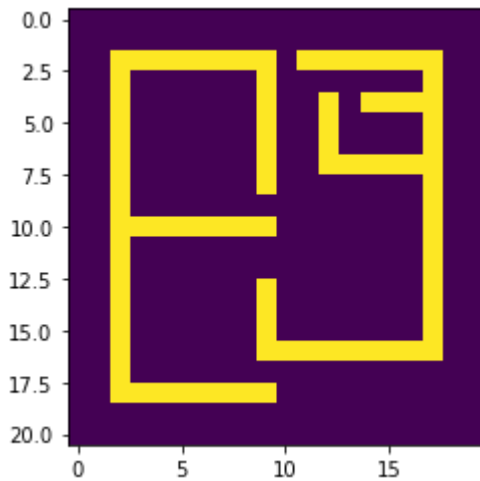
# get current floor map data and transfer it into matirx

```

```

current_map_matirx = get_floor_map_points(iddevice)
# transfer the destination coordinate into matirx
destination_matirx = coordinate_to_matirx(destination,current_map_matirx)
# get current coordinate point
current_position = get_current_position(iddevice)
# transfer the current coordinate point into matirx data
current_position_matirx =
coordinate_to_matirx(current_position,current_map_matirx)

```



```
current_position_matirx
```

```
(11, 18)
```

```
destination_matirx
```

```
(15, 5)
```

```
%matplotlib auto
```

```
Using matplotlib backend: MacOSX
```

```

%%time
# 最后这里我用魔术方法计算了一下时间开销，然后下面是模拟的路径，其数值为1
test =

```

```
BFS_PathFind(current_position_matirx,destination_matirx,current_map_matirx
)
test.calculate_came_from()
plt.imshow(test.find_path())
```

CPU times: user 119 ms, sys: 40 ms, total: 159 ms
Wall time: 205 ms

<matplotlib.image.AxesImage at 0x11c6ea7d0>

```
current_position_changing = []
for i in range(11,0,-1):
    current_position_changing.append((i,18))
for i in range(18,9,-1):
    current_position_changing.append((1,i))
for i in range(1,12):
    current_position_changing.append((i,10))
current_position_changing
```

```
[(11, 18),
 (10, 18),
 (9, 18),
 (8, 18),
 (7, 18),
 (6, 18),
 (5, 18),
 (4, 18),
 (3, 18),
 (2, 18),
 (1, 18),
 (1, 18),
 (1, 17),
 (1, 16),
 (1, 15),
 (1, 14),
 (1, 13),
 (1, 12),
 (1, 11),
 (1, 10),
 (1, 10),
 (2, 10),
```

```
(3, 10),
(4, 10),
(5, 10),
(6, 10),
(7, 10),
(8, 10),
(9, 10),
(10, 10),
(11, 10)]
```

```
for current_position_matirx in current_position_changing:
    test =
    BFS_PathFind(current_position_matirx,destination_matirx,current_map_matirx
    )
    test.calculate_came_from()
    plt.imshow(test.find_path())
    plt.pause(1)
```

```
-----
-

NameError                                Traceback (most recent call
last)

<ipython-input-1-6f63fc1e2e58> in <module>
----> 1 for current_position_matirx in current_position_changing:
      2     test =
    BFS_PathFind(current_position_matirx,destination_matirx,current_map_matirx
    )
      3     test.calculate_came_from()
      4     plt.imshow(test.find_path())
      5     plt.pause(1)

NameError: name 'current_position_changing' is not defined
```