Диаграмма Вороного

1. Задача

Даны наборы “вертикальных” и “горизонтальных” улиц, координаты школ и домов. Для каждого дома требуется найти наиболее близкую школу.

2. Допущения

Перед каждым походом в школу гипотетический ученик покупает кофе в ближайшем из киосков сети “Перекрёсток”, расположенных на перекрёстках дорог.

В городе существуют хотя бы одна “вертикальная” и одна “горизонтальная” дороги.

В городе существуют хотя бы одна школа и хотя бы один дом.

3. Постановка задачи

Для каждого дома требуется найти ближайший перекрёсток, откуда - ближайшую школу. Для этого построить диаграмму Вороного в формате попарных границ между школами.

Входные данные: строка координат вертикальных улиц; строка координат горизонтальных улиц; строка координат школ в формате (x,y); строка координат домов в формате (x,y).

Выходные данные:

Для каждого дома - координаты ближайшего киоска и ближайшей школы.

4. Решение задачи

Импорт используемых библиотек

```
import matplotlib.pyplot as plt
```

Определение используемых функций

def decode(string): - для трансформации строки(x,y) в список [x, y]

```
    string = string[1:len(string) - 1]
    lst = (string[:string.find(',')] + ' ' + string[string.find(',') + 1:]).split()
    for i in range(len(lst)):
        lst[i] = float(lst[i])
    return lst
```

def get_entrance(point, num, streets): - для поиска ближайшей улицы некоторого типа

```
    spec_streets = []
    spec_streets[:] = streets
    spec_streets.append(point[num])
    spec_streets.sort(key=float)
    pos = spec_streets.index(point[num])
```

if spec_streets[pos+1] == point[num]: - рассматриваются различные случаи взаимного расположения крайних улиц и точки

```
    # if house is straight on the street
    return pos
elif pos == len(spec_streets) - 1:
    return pos - 1
elif pos == 0:
    return 0
elif min(abs(float(spec_streets[pos])-float(spec_streets[pos+1])),
      abs(float(spec_streets[pos])-float(spec_streets[pos-1]))) == abs(float(spec_streets[pos])-
float(spec_streets[pos+1])):
    # if closer street is next
    return pos
else:
    return pos-1
```

def get_border(school_1, school_2, number_1, number_2, line_array): - для определения формулы границы между двумя точками в формате линии y=kx+b

```
x1, x2 = school_1[0], school_2[0]
y1, y2 = school_1[1], school_2[1]
if y1 != y2:
    k = (x2-x1)/(y1-y2)
    b = (y1+y2 - k*(x1+x2))/2
else:
    k = 'inf'
    b = (x1+x2)/2
line_array[number_1][number_2][0], line_array[number_1][number_2][1] = k, b
```

def get_furthest(house_coordinates, num1, num2, line_array, school_array): - для получения наиболее удалённой от данной точки одной из двух других точек с помощью анализа взаимного положения с границей их областей

```
k, b = line_array[num1][num2][0], line_array[num1][num2][1]
school1_x, school1_y = float(school_array[num1][0]), float(school_array[num1][1])
```

```
    if k == 'inf':
        if school1_x <= b:
            school_left = num1
            school_right = num2
        else:
            school_left = num2
            school_right = num1

        if house_coordinates[0] <= b:
            return school_right
        else:
            return school_left
    else:
        if school1_y >= k*school1_x+b:
            school_high = num1
            school_low = num2
        else:
            school_high = num2
            school_low = num1
        if float(house_coordinates[1]) >= k*float(house_coordinates[0])+b:
            return school_low
        else:
            return school_high
```

```
def visualize(massive_of_points, color, name): - для вывода на экран точек определённого рода
    for point in massive_of_points:
        plt.scatter(point[0], point[1], c=color, label=name)
```

Обработка данных:

```
streets_vertical_raw = sorted(input().split(), key=float)
streets_vertical = [float(street) for street in streets_vertical_raw]
```

```
streets_horizontal_raw = sorted(input().split(), key=float)
streets_horizontal = [float(street) for street in streets_horizontal_raw]

schools_raw = input().split()
schools = []
houses_raw = input().split()
houses = []
n_schools = len(schools_raw)
```

Создание массива для хранения границ:

```
lines = [[[i, i] for i in range(n_schools)] for j in range(n_schools)]
for school in schools_raw:
    schools.append(decode(school))
for house in houses_raw:
    houses.append(decode(house))
for i in range(n_schools):
    for j in range(n_schools):
        if i != j:
            get_border(schools[i], schools[j], i, j, lines)
del streets_horizontal_raw, streets_vertical_raw, houses_raw, schools_raw
# filled array of borders
coffee_shops = []
```

Поиск ближайшей школы для каждого дома:

```
y_min = min(streets_horizontal) - вспомогательные переменные для визуализации
y_max = max(streets_horizontal)
x_min = min(streets_vertical)
x_max = max(streets_vertical)
for building in houses:
    coffee_shop = [streets_vertical[get_entrance(building, 0, streets_vertical)],
streets_horizontal[get_entrance(building, 1, streets_horizontal)]]
    print('Coffee for', building, 'at', coffee_shop)
    coffee_shops.append(coffee_shop)
```

```
    possible_schools = [i for i in range(n_schools)]
    for i in range(n_schools-1):
        if possible_schools[0] == get_furthest(coffee_shop, possible_schools[0], possible_schools[1], lines,
schools):
            possible_schools.pop(0)
        else:
            possible_schools.pop(1)
    for school_num in possible_schools:
        print('Best school for', building, 'is', schools[school_num])
    x_min = min(x_min, building[0])
    x_max = max(x_max, building[0])
    y_min = min(y_min, building[1])
    y_max = max(y_max, building[1])
for building in schools: - вспомогательный цикл для визуализации
    x_min = min(x_min, building[0])
    x_max = max(x_max, building[0])
    y_min = min(y_min, building[1])
    y_max = max(y_max, building[1])

# for every house find nearest schools by elimination method
```

Визуализация данных:

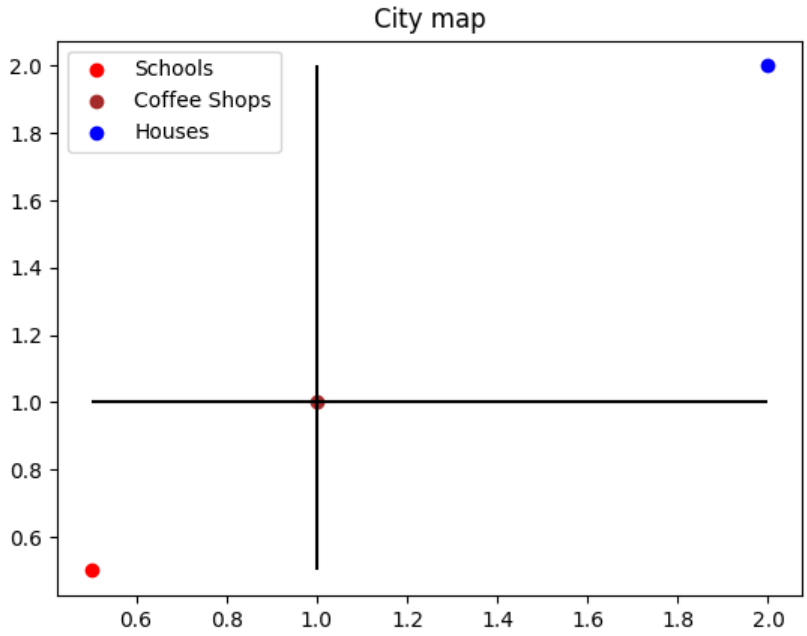
```
for street in streets_horizontal:
    plt.hlines(float(street), x_min, x_max, colors='black')

for street in streets_vertical:
    plt.vlines(float(street), y_min, y_max, colors='black')

visualize(schools, 'red', 'Schools')
visualize(coffee_shops, 'brown', 'Coffee Shops')
visualize(houses, 'blue', 'Houses')
plt.title('City map')
plt.legend()
```

plt.show()

5. Тесты:

Обозначения: синий - дома, красный - школы, коричневый - киоски.

Рис. 1: Простейший тест, все списки имеют длину 1:



Ввод: 1

1

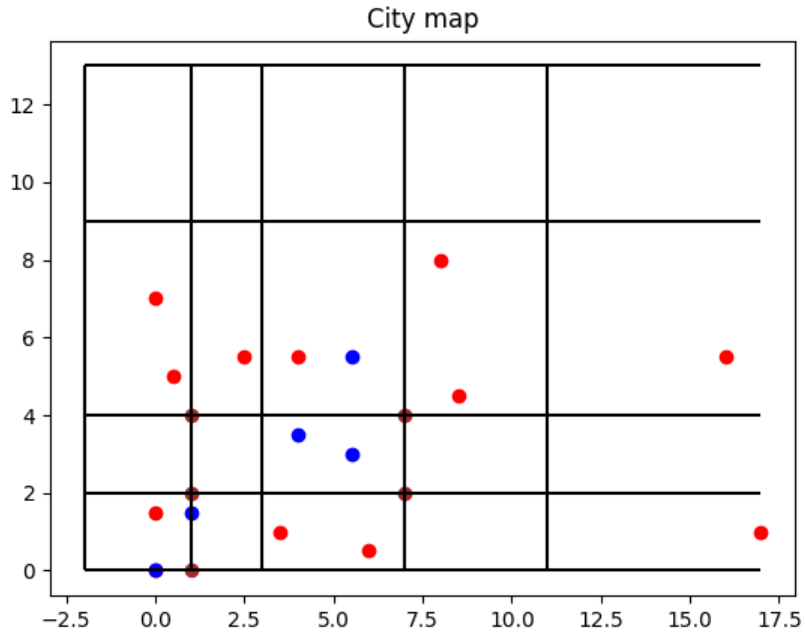
(0.5,0.5)

(2,2)

Вывод: Coffee for [2.0, 2.0] at [1.0, 1.0]

Best school for [2.0, 2.0] is [0.5, 0.5]

Рис 2: Тест с использованием случайных координат школ и домов, полученных с помощью генераторов списков и функции random:



Ввод (координаты улиц):

1 3 11 -2 7

9 0 4 2 13

Вывод: Schools: [[0.0, 0.0], [0.0, 1.5], [3.5, 1.0], [2.5, 5.5], [0.0, 7.0], [0.5, 5.0], [8.5, 4.5], [6.0, 0.5], [4.0, 5.5], [17.0, 1.0], [8.0, 8.0], [16.0, 5.5]]

Houses: [[0.0, 0.0], [1.0, 1.5], [4.0, 3.5], [5.5, 3.0], [5.5, 5.5]]

Coffee for [0.0, 0.0] at [1.0, 0.0]

Best school for [0.0, 0.0] is [0.0, 0.0]

Coffee for [1.0, 1.5] at [1.0, 2.0]

Best school for [1.0, 1.5] is [0.0, 1.5]

Coffee for [4.0, 3.5] at [1.0, 4.0]

Best school for [4.0, 3.5] is [0.5, 5.0]

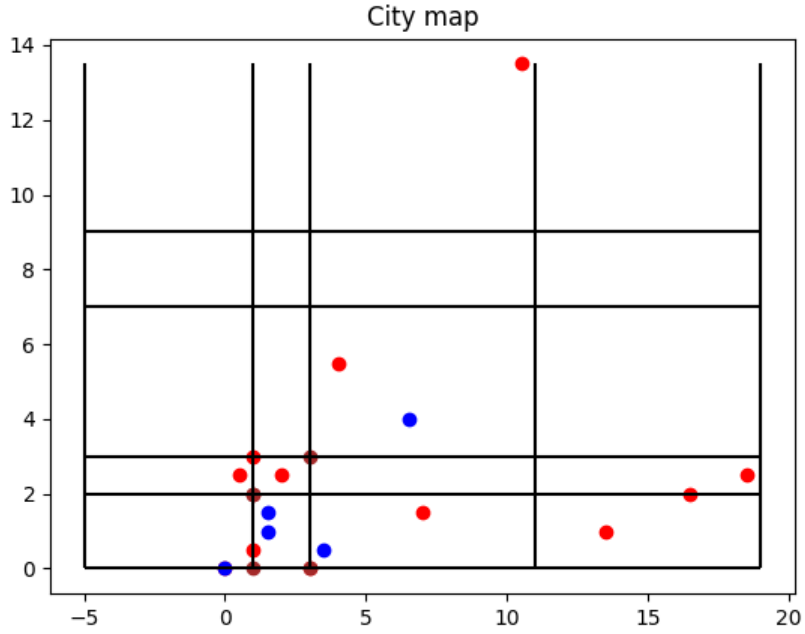
Coffee for [5.5, 3.0] at [7.0, 4.0]

Best school for [5.5, 3.0] is [8.5, 4.5]

Coffee for [5.5, 5.5] at [7.0, 2.0]

Best school for [5.5, 5.5] is [6.0, 0.5]

Рис. 3: Тест на случайных данных:



Ввод: 1 3 11 19 -5

0 7 2 3 9

Вывод: Schools: [[0.0, 0.0], [1.0, 0.5], [1.0, 3.0], [2.0, 2.5], [3.0, 0.0], [4.0, 5.5], [0.5, 2.5], [10.5, 13.5], [7.0, 1.5], [16.5, 2.0], [18.5, 2.5], [13.5, 1.0]]

Houses: [[0.0, 0.0], [1.5, 1.5], [1.5, 1.0], [3.5, 0.5], [6.5, 4.0]]

Coffee for [0.0, 0.0] at [1.0, 0.0]

Best school for [0.0, 0.0] is [1.0, 0.5]

Coffee for [1.5, 1.5] at [1.0, 2.0]

Best school for [1.5, 1.5] is [0.5, 2.5]

Coffee for [1.5, 1.0] at [1.0, 2.0]

Best school for [1.5, 1.0] is [0.5, 2.5]

Coffee for [3.5, 0.5] at [3.0, 0.0]

Best school for [3.5, 0.5] is [3.0, 0.0]

Coffee for [6.5, 4.0] at [3.0, 3.0]

Best school for [6.5, 4.0] is [2.0, 2.5]

6. Выводы

Алгоритм с указанными допущениями реализован, в указанных рамках вырожденные случаи проверены. Верность алгоритма проверена прямым сравнением с результатами распределения домов по евклидову расстоянию до школ на нескольких тестах (прямое сравнение расстояний без построения диаграмм Вороного).