Содержание

1 Фурмолировка задачи

2 Условия упрощения алгоритма

2.1 Фурмалировка условий упрощения алгоритма

Так как мы пишем программу, что бы облегчить себе задачу, будем искать куда поставить почтамт методом переюора. Но перебор длжен быть организован так, что:

1. Точек куда можно поставить почтамт конечно, и их количество должно быть $a <$ n $< b$ $(10^5 < n < 10^6)$.
2. Для каждой точки можно определить скалярную функцию входных данных(центры активности(далее ЦА), зоны запрета полета(далее NFZ), населенности в районе).
3. Входные данные должны быть осмысленны.

2.2 Уточним первое условие

Теперь определим a, b. Переменая a отвечает за то что бы не упрастить задачу до одной точки. Я думаю a должно быть таким, что растояние между точками меньше 100метров. Площадь Маската 3500 km^2 откуда получаем $a \approx \dfrac{35 \cdot 10^8}{100^2} = 35 \cdot 10^4$, а если учесть что $1/3$ маската это малонаселенные горы получаем $a \approx 10^5$. Значение b можно определить из времени выполнения программы. Дадим напрмер на алгорим 10 минут. количество операций которое можно провести за это время можн опонять из такой программы:

```
import datetime
t = datetime.datetime.now()
i = 0
while (datetime.datetime.now() - t).seconds < 30:
    i += 1
print(i * 20)
```

Вывод программы зависит от устройства на котором она запущена. У меня выдала 792801540 или примерно $8000 \cdot 10^5$ и если на одну точку брать хотя бы 800 простых операций получаем $b \approx 10^6$

Итак первое условие $10^5 < n < 10^6$

2.3 Уточним второе условие

Для начала определим еще один важный фактор. Наша программа раставляет пачтамты последовательно и от наиболее загруженного к наименее загруженому, отсюда получаем ситуацию: стоит почтамт и не так далеко находится ЦА. Мы конечно хотим поставить пачтамт рядом с ЦА, но тут уже стоит один. Отсюда следствие: функция должна учитывать так же уже поставленные пачтамты.

2.1 функция должна опираться на уже поставленные пачтамты

2.4 Уточним третье условие

Проблема в том, что точных данных по населенности у нас нет. Поэтому мы пришли к упращению: Маскат распередлен на регионы, где население распределено равномерно. Если внутри региона видно сильное разделение, то мы искуственно разделим этот регион на "подрегионы"и пделим население так, как нам покажется правильным, что бы население в "подрегионах"было $\pm$ равномерно.

То-есть все данные должны быть проработаны на правдивость и поэтому результат программы должен быть так же осознан(должен подвергнуться критике со стороны человека)

3 Входные даные

3.1 Область обработки входных данных

Как мы понимаем, люди не будут использовать почтамат если он находится в 10km от их дома. Так же очевидно, что чем дальше почтамт от их дома, тем меньше они будут им пользоваться. Я бы взял растояние до 2km и функцию от растояния $f(R) \sim \frac{1}{R^2}$ или даже $f(R) \sim \frac{1}{R^3}$.

Так же с ЦА. люди из ЦА пойдут к пачтамту только если он очень близко, посколько в бизнес центре или складе время-деньги, а в магазинах люди веселяться так что не готовы идти долеко в этом и смысл торговых центров. Отсюда радиус < 500m а $f(R) \sim \frac{1}{R^3}$.

NFZ будут обрабатаваться просто как точки, где поставить пачтамты нельзя.

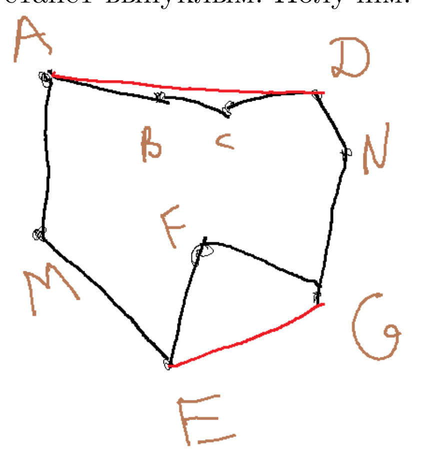
3.2 Обработка входных даных

На вход алгоритм получит точки(определяющие регионы), насеоенность регионов, точки(определяющие ЦА), рейтинг(рейтинг ЦА который создан субьективно, но пропорцианален количеству посылок, отправляемых из ЦА).

Мы хотим их привести к данным, которые проще обрабатывать. Поэтому привратим регионы в набор точек каждой из которых будет присвоено значение населености.

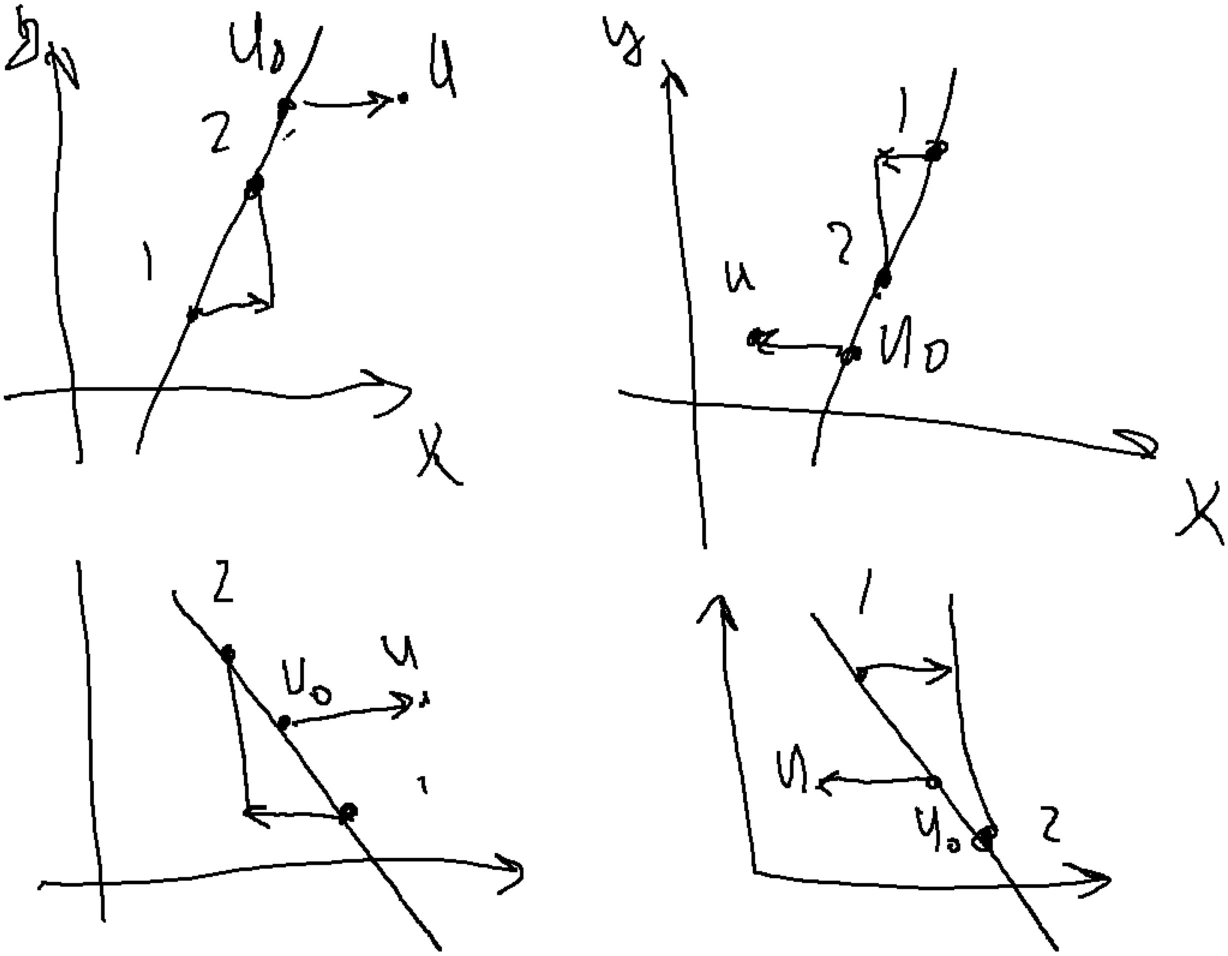
Проблема: Регионы - не всегда выпуклые многоугольники занчит определить принадлежит ли точка многоугольнику нельзя просто проверив принадлежит ли точка всем полуплоскостям образованными гранями.

Решение: Изменим многоугольник вырезая вершины так, что он станет выпуклым. Получим:



в даном случае остается проверить для точки $U = (x, y)$, $U \in ADGEM \cap U \notin FEG \cap U \notin ABCD$.

Как же найти такие EFG? Как мы знаем уравнение прямой выглядит так: $\dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1}$. Теперь как определить что точка лежит справа? Можно просто посмоттеть где прямая через т U пересекает ось X например:



Подставим у от U в уравнение получим:$x_3 = \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} + x_1$ тогда $U_0 = (x_3, y_U)$ тогда если $\dfrac{x_1 - x_2}{y_1 - y_2} > 0$ то U должна лежать правее точки U_0 получаем по знаку $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv x_2 - x_1$ иначе U должна лежать левее точки U_0 получаем по знак $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv -(x_2 - x_1)$ соеденяем получаем $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv$

$\dfrac{x_1 - x_2}{y_1 - y_2} \cdot (x_2 - x_1) \implies (x_U - x_1) - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} \equiv \dfrac{(x_2 - x_1)^2}{y_2 - y_1} \implies (x_U - x_1)(y_2 - y_1) - (y_U - y_1)(x_2 - x_1) \equiv (x_2 - x_1)^2 \implies (x_U - x_1)(y_2 - y_1) - (y_U - y_1)(x_2 - x_1) \equiv 1 \implies (x_U - x_1)(y_2 - y_1) > (y_U - y_1)(x_2 - x_1)$.
Случай $U \in \dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1}$ нужно расмотреть отдельно но пусть в таком счучае будем считать, что принадлежит.

3.3 Итоговые Входные даные

Итак мы упрастили ввод до точек каждой из которой принадлежит ярлык ЦА или просто регион и какя то цифра хорактерезующая количество отправок.

4 Преобразование векторный графики в точечную

4.1 Точка

Для начала хорошо бы создать класс точка с полями x, y. Так же прописать функцию поиска растояния и функции отрезка. Функцию отрезка можно получить вида $y = f(x)$ из ранее уже выведеной.

```
import math
class point:
    def __init__(self, x:float, y:float):
    self.x = x
    self.y = y

    def lenf(self, other):
    return math.sqrt((self.x - other.x)** 2 + (self.y - other.y)** 2)

    def equatian(self, other, **printer):
    x_1, y_1 = self.x, self.y
    x_2, y_2 = other.x, other.y
    equatin = f"x * {(y_2 - y_1) / (x_2 - x_1)} - {x_1 * (y_2 - y_1) / (x_2 -
    if printer:
```

```
        print(f"y = {equatin}")
    return equatin
```

И для красоты добавим функции print-a.

```
    def __str__(self):
        return f"({self.x}, {self.y})"

    def __repr__(self):
        return f"<<{self.x}, {self.y}>>"
```

4.2 Лежит точка справа от отрезка?

Функция принисает 3 точки 2 из которых задают направленую прямую и 3-я это проверяемая точка. Все необходимые функции мы уже вывели.

```
def point_right_line(point1, point2, u) -> bool:
    return (u.x - point1.x) * (point2.y - point1.y) > (u.y - point1.y) * (poin
```

4.3 Работа с выпуклой фигуры

Тут будут 3 функции для читаемости. 1-ая будет проверять лежит ли точка внутри фигуры. 2-ая вернет все точки внутри заданой фигуры.

```
def point_in_normal_figure(pl:List[Point], u:Point):
    if len(pl) < 3: print("errore!!!!! Beckend  line")
    orintation_right = point_right_line(pl[0], pl[1], pl[2])
    point1 = pl[0]
    for point2 in pl[1:]:
        if orintation_right != point_right_line(point1, point2, u):
            return False
        point1 = point2
    if orintation_right != point_right_line(pl[-1], plt[0], u):
        return False
    return True
```

```
def normal_figure_to_points(pl:List[Point]):
    global delta_point
    x, y = list(map(lambda a: a.x ,pl)),list(map(lambda a: a.y ,pl))
    x,y = ([i for i in range(min(x), max(x), delta_point)], [i for i in range(m
    all_points = list(map(lambda x:Point(x[0], x[1]), itertools.product(x,y)))
    del x, y
    new_all_points = []
    for i in all_points:
        if point_in_normal_figure(pl, i):
            new_all_points += [i]
    del all_points
    return new_all_points
```

4.4 Разбиение впуклого многоугольника

Самое сложное определить находится внутряняя часть фигуры справа или слева. Я предлагаю способ, который возможно будет работать не вегда, но в большинстве способов. Просто определить для каждого отрезка сколько точек лежит справа сколько слева. И так для всех и сложить. Получим два числа например r, l(r-спрва l-слева). И тогда если $r > l$ то спарава иначе слева.

Тогда код будет выглядеть так:

```
def orintation_righ(main_fig:List[Point]):
    r, l = 0, 0
    temp_ln = len(main_fig)
    for i in range(temp_ln):
        for j in range(temp_ln):
            if j != i and j != (i + 1) % temp_ln:
                if point_right_line(main_fig[i], main_fig[(i + 1) % temp_ln], m
                r += 1
            else:
                l += 1
    return r > l
```

```
def bad_figure_to_points(main_fig:List[Point]):
    added_fig = []
    orintation_right = orintation_righ(main_fig)
    count = 0
    i = 0
    while True:
        points = [main_fig[i], main_fig[(i + 2)% len(main_fig)], main_fig[(i +
        if (point_right_line(points[0], points[1], points[2]) and (not orintati
            count += 1
        else:
            added_fig += [points]
            main_fig.remove(main_fig[(i + 1)% len(main_fig)])
            count = 0
        i += 1
        if count + 3 == len(main_fig):      #number 3 is random num for skip mi
            break
    main_poins = normal_figure_to_points(main_fig)
    added_points = []
    for i in added_fig:
        added_points += normal_figure_to_points(i)
        for i in main_poins:
            if i in added_points:
                main_poins.remove(i)
    return main_poins
```

5 Алгоритм

5.1 Функция точки

Опредлим эту функцию из логики того, что на нее влияет. Итак она будет зависить от населености района, где она стоит и от ближайших ЦА. Пусть $peple(x, y)$ это население в точки x, y, $Bcentr(x, y)$ вернет роходимость Бизнес центра в точке x, y, если там нет бизнес центра вернет 0, $Tcenter(x, y), storegge(x, y)$ аналогично для торогого цента и склада. Получаем что то типо:

$$f(x,y) = \left(\sum_{-R_1<i<R_1} \sum_{-R_1<j<R_1} \frac{k_1}{\sqrt{i^2+j^2}^2} \cdot peple(i,j) \right) +$$

$$+ \left(\sum_{-R_2<i<R_2} \sum_{-R_2<j<R_2} \frac{k_2}{\sqrt{i^2+j^2}^3} \cdot Bcentr(i,j) \right) +$$

$$+ \left(\sum_{-R_2<i<R_2} \sum_{-R_2<j<R_2} \frac{k_3}{\sqrt{i^2+j^2}^3} \cdot Tcenter(i,j) \right) +$$

$$+ \left(\sum_{-R_2<i<R_2} \sum_{-R_2<j<R_2} \frac{k_4}{\sqrt{i^2+j^2}^3} \cdot storegge(i,j) \right)$$

R_1, R_2 мы договорились взять 1 km и 500m, а k_1, k_2, k_3, k_4 зависят от входных данных, они будут определны дополнительными расчетами.Получаем: