Информатика. Задания на кластеризацию. Повторение материала.

Р-00. (демо-2025) Учёный решил провести кластеризацию некоторого множества звёзд по их расположению на карте звёздного неба. Кластер звёзд – это набор звёзд (точек) на графике, лежащий внутри прямоугольника высотой H и шириной W. Каждая звезда обязательно принадлежит только одному из кластеров.

Истинный центр кластера или центроид, – это одна из звёзд на графике, сумма расстояний от которой до всех остальных звёзд кластера минимальна. Под расстоянием понимается расстояние Евклида между двумя точками $A(x_1, y_1)$ и $B(x_2, y_2)$ на плоскости, которое вычисляется по формуле:

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

В файле *А* хранятся данные о звёздах двух кластеров, где $H = 3$, $W = 3$ для каждого кластера. В каждой строке записана информация о расположении на карте одной звезды: сначала координата x, затем координата y. Значения даны в условных единицах. Известно, что количество звёзд не превышает 1000.

В файле *Б* хранятся данные о звёздах трёх кластеров, где $H = 3$, $W = 3$, для каждого кластера. Известно, что количество звёзд не превышает 10000. Структура хранения информации о звездах в файле аналогична файлу *Б*.

Для каждого файла определите координаты центра каждого кластера, затем вычислите два числа: P_x – среднее арифметическое абсцисс центров кластеров, и P_y – среднее арифметическое ординат центров кластеров. В ответе запишите четыре числа: в первой строке сначала целую часть произведения $P_x \cdot 10000$, затем целую часть произведения $P_y \cdot 10000$ для файла *А*, во второй строке – аналогичные данные для файла *Б*.

Возможные данные одного из файлов иллюстрированы графиком.

Внимание! График приведён в иллюстративных целях для произвольных значений, не имеющих отношения к заданию. Для выполнения задания используйте данные из прилагаемого файла.

Решение

Разобьем нашу задачу на подзадачи и для начала реализуем функцию для алгоритма $DBSCAN$:

```
from math import dist  # Функция для вычисления расстояния между точками
from turtle import *
def dbscan(a, r):
    clusters = [] # Список для хранения кластеров
    while a: # Пока в списке есть элементы
# Создаем новый кластер и добавляем в него первую точку из списка a
        clusters.append([a.pop(0)])
# Проходим по всем точкам в последнем добавленном кластере
        for j in clusters[-1]:
            for i in a: # Проходим по оставшимся точкам в списке a
# Если расстояние между точками j и i меньше радиуса r
                if dist(j, i) < r:
# Добавляем точку i в текущий кластер
                    clusters[-1].append(i)
                    a.remove(i) # Удаляем точку i из списка a
    return clusters # Возвращаем список кластеров
```

Далее считаем данные из файла и применим созданную функцию для кластеризации точек. В конце отберем только кластеры, которые содержат более 5 точек, чтобы избавиться от мусорных кластеров:

```
f = open('27B.txt')
s = f.readline()
a = [list(map(float, i.replace(',', '.').split())) for i in f]
# Выполняем кластеризацию с радиусом 0.5
c = dbscan(a, 0.5)
# Отбираем только кластеры с более чем 5 точками
clusters = [i for i in c if len(i) > 5]
```

Следующим шагом проверим, точно ли все точки правильно распределились по кластерам. Для этого используем библиотеку *turtle* и прорисуем все кластеры разными цветами для визуального различия:

```
m = 50 # Масштабный коэффициент
tracer(0) # Отключаем анимацию для мгновенного отображения
# Цвета для разных кластеров
col = ['orange', 'black', 'blue', 'red', 'green', 'purple']
pu() # Поднимаем перо черепашки
# Отрисовываем точки кластеров разными цветами
for i in range(len(clusters)):
    for j in clusters[i]:
        x, y = j
        goto(x * m, y * m) # Перемещаемся к координатам точки
        dot(3, col[i]) # Рисуем точку размером 3 пикселя цветом кластера
done() # Завершаем работу с графикой
```

Все точки распределены верно, поэтому последним шагом можно найти центроиды каждого кластера и вывести их на экран:

```
for j in clusters:
    mn = 10000000005000000000 # Инициализируем минимальную сумму
    for star in j:
        s = 0
# Считаем сумму расстояний от текущей точки до всех других в кластере
        for i in j: s += dist(star, i)
        if s < mn: # Если сумма меньше текущего минимума,
            mn = s # обновляем минимум
            white_star = star # и запоминаем точку
    print(white_star) # Выводим координаты центральной точки кластера
```

Получаем полный код программы (без визуализации):

```
from math import dist  # Функция для вычисления расстояния между точками
from turtle import *
def dbscan(a, r):
    clusters = [] # Список для хранения кластеров
    while a: # Пока в списке есть элементы
        clusters.append([a.pop(0)])
        for j in clusters[-1]:
            for i in a: # Проходим по оставшимся точкам в списке a
                if dist(j, i) < r:
                    clusters[-1].append(i)
                    a.remove(i) # Удаляем точку i из списка a
    return clusters # Возвращаем список кластеров

f = open('27B.txt')
s = f.readline()
a = [list(map(float, i.replace(',', '.').split())) for i in f]
c = dbscan(a, 0.5) # Выполняем кластеризацию с радиусом 0.5
# Отбираем только кластеры с более чем 5 точками
clusters = [i for i in c if len(i) > 5]

for j in clusters:
    mn = 10000000005000000000 # Минимальная сумма
    for star in j:
        s = 0
# Считаем сумму расстояний от текущей точки до всех других в кластере
        for i in j: s += dist(star, i)
        if s < mn: # Если сумма меньше текущего минимума,
            mn = s # обновляем минимум
            white_star = star # и запоминаем точку
    print(white_star) # Выводим координаты центральной точки кластера
```