Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики
Кафедра прикладної математики

Лабораторна робота №4
Тема: «Діаграма Вороного»

з дисципліни

“Алгоритмічні основи обчислювальної геометрії та комп’ютерної графіки”

Варіант 4

Група: КМ-22
Виконала: Бриль К.С.
Оцінка:

Київ – 2023

Завдання для варіанту 4

Мета роботи:

Розробити програмний засіб, який знаходить центри ваги зв'язаних областей множини точок заданих своїми координатами та відображує її на координатній площині і зберігає зображення в одному з графічних форматів.

Після чого будує Діаграму Вороного за знайденими точками і зберігає зображення в одному з графічних форматів.

Хід роботи

Дослідити алгорими знаходження зв'язаних областей, та алгоритми побудови діаграм Вороного

Використати датасет з лабораторної роботи №2

Скачати файл з датасетом. Файл в текстовому форматі містить пари цілих чисел які є координатами точок.

Необхідно написати програму будь якою мовою з використанням будь яких бібліотек яка

- Зчитує датасет з файлу;
- Ділить датасет на зв'язані області (множини точок однакового кольору, між якими немає точок іншого кольору):
- знаходить центри ваги зв'язаних областей (середне арифметичне по кожній координаті);
- Будує діаграму Вороного за множиною центрів ваги;
- Встановлює розміри вікна (полотна – canvas size) **960x540** пкс;
- Відображає центри ваги кругами пдіаметру 5 пкс та побудовану діаграму Вороного ;
- Відображає точки вихідного датасету на побудованому малюнку чорним кольором з насиченістю 10%;
- Виводить результати у файли будь-якого графічного формату.

Файли з результатом та звіт викласти на хмарному сховищі, текст програми на GIT. Посилання на на результат і звіт викласти на сторінці Лабораторна робота №4

Звіт повинен містити краткий опис ходу роботи із вказанням того, яка бібліотека і які методи застосовувались.

Код програми

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.spatial import Voronoi


def voronoi_finite_polygons_2d(vor, radius=None):

    if vor.points.shape[1] != 2:
        raise ValueError("Requires 2D input")

    new_regions = []
    new_vertices = vor.vertices.tolist()

    center = vor.points.mean(axis=0)
    if radius is None:
        radius = vor.points.ptp().max()

    all_ridges = {}
    for (p1, p2), (v1, v2) in zip(vor.ridge_points, vor.ridge_vertices):
        all_ridges.setdefault(p1, []).append((p2, v1, v2))
        all_ridges.setdefault(p2, []).append((p1, v1, v2))

    for p1, region in enumerate(vor.point_region):
        vertices = vor.regions[region]

        if all(v >= 0 for v in vertices):
            new_regions.append(vertices)
            continue

        ridges = all_ridges[p1]
        new_region = [v for v in vertices if v >= 0]

        for p2, v1, v2 in ridges:
            if v2 < 0:
                v1, v2 = v2, v1
            if v1 >= 0:
                continue

            t = vor.points[p2] - vor.points[p1]  # tangent
            t /= np.linalg.norm(t)
            n = np.array([-t[1], t[0]])  # normal

            midpoint = vor.points[[p1, p2]].mean(axis=0)
            direction = np.sign(np.dot(midpoint - center, n)) * n
            far_point = vor.vertices[v2] + direction * radius

            new_region.append(len(new_vertices))
            new_vertices.append(far_point.tolist())

        vs = np.asarray([new_vertices[v] for v in new_region])
        c = vs.mean(axis=0)
        angles = np.arctan2(vs[:, 1] - c[1], vs[:, 0] - c[0])
        new_region = np.array(new_region)[np.argsort(angles)]
```

```
        new_regions.append(new_region.tolist())

    return new_regions, np.asarray(new_vertices)


def get_first(s):
    for e in s:
        break
    return e


class Pixel:
    x = 0
    y = 0

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return self.x.__hash__() ^ self.y.__hash__()

    def average(self, pixels: set):
        n = len(pixels)
        self.x = 0
        self.y = 0
        for pixel in pixels:
            self.x += pixel.x
            self.y += pixel.y
        self.x = self.x // n
        self.y = self.y // n
        return self


pixels = pd.read_csv('DS4.txt', sep=" ", header=None)

X = pixels[0].tolist()
Y = pixels[1].tolist()

pixels = []
for i in range(0, len(X)):
    pixels.append(Pixel(X[i], Y[i]))

pixels_set = set(pixels)


def connect_pixels(pixel, connected_area):
    connected_area.add(pixel)
    pixels_set.remove(pixel)

    pixel_left = Pixel(pixel.x - 1, pixel.y)
    if pixel_left in pixels_set:
        connect_pixels(pixel_left, connected_area)
```

```
    pixel_right = Pixel(pixel.x + 1, pixel.y)
    if pixel_right in pixels_set:
        connect_pixels(pixel_right, connected_area)

    pixel_up = Pixel(pixel.x, pixel.y + 1)
    if pixel_up in pixels_set:
        connect_pixels(pixel_up, connected_area)

    pixel_down = Pixel(pixel.x, pixel.y - 1)
    if pixel_down in pixels_set:
        connect_pixels(pixel_down, connected_area)


connected_areas = []
while len(pixels_set) != 0:

    connected_area = set()
    pixels_to_check = set()
    pixels_to_check.add(get_first(pixels_set))

    while len(pixels_to_check) != 0:
        pixel = pixels_to_check.pop()
        connected_area.add(pixel)
        pixels_set.remove(pixel)

        pixel_left = Pixel(pixel.x - 1, pixel.y)
        if pixel_left in pixels_set and pixel_left not in connected_area:
            pixels_to_check.add(pixel_left)

        pixel_right = Pixel(pixel.x + 1, pixel.y)
        if pixel_right in pixels_set and pixel_right not in connected_area:
            pixels_to_check.add(pixel_right)

        pixel_up = Pixel(pixel.x, pixel.y + 1)
        if pixel_up in pixels_set and pixel_up not in connected_area:
            pixels_to_check.add(pixel_up)

        pixel_down = Pixel(pixel.x, pixel.y - 1)
        if pixel_down in pixels_set and pixel_down not in connected_area:
            pixels_to_check.add(pixel_down)

    connected_areas.append(connected_area)


averages = []
for area in connected_areas:
    pixel_avg = Pixel()
    pixel_avg.average(area)
    averages.append([pixel_avg.x, pixel_avg.y])

points = np.array(averages)

vor = Voronoi(points)
# plot
regions, vertices = voronoi_finite_polygons_2d(vor)
```

```
dpi = plt.figure().dpi
plt.figure(figsize=(540/dpi, 960/dpi))
plt.xlim(0, 540)
plt.ylim(0, 960)

# colorize
for region in regions:
    polygon = vertices[region]
    plt.fill(*zip(*polygon), alpha=1)

plt.plot(points[:, 0], points[:, 1], 'ko', markersize=5,
markerfacecolor="red")
plt.plot(X, Y, 'o', markersize=1, color="black", alpha=0.01)
plt.axis('off')
plt.savefig('cg4.png', bbox_inches='tight', pad_inches=0)
```

Опис роботи:

Спочатку ми в масив **pixels зчитуємо наш датасет**.

Описуємо функцію **connect_pixels**(), за допомогою неї розбиваєм наш датасет на **зв'язні області**. Зберігаєм результат в масиві connected_areas.

Далі створюємо масив averages. Записуємо **центри мас** відповідних областей за допомогою функції **average()**.

Далі за цими координатами **будуємо діаграму Вороного**.

Також відображаємо точки нашого початкового датасету з прозорістю 10%.

Зберігаєм зображення у форматі png.

Опис ключових елементів програми
Клас **Pixel** використовується для роботи з пікселями. В аргументах у нього x і y, і метод, якому задається сет пікселів, і який повертає центр мас цих пікселів.

Також поза межами класу є функція **connect_pixel**, якій задається поточний піксель, і сет, в якому вона з кожним кроком додає зв'язні з першим пікселі. В кінці вона повертає сет зв'язних пікселів.

У програмі використовується 4 бібліотеки.
Pandas і **Numpy** використовуються для роботи з датасетами.
Matplotlib використовується для роботи з зображенням, додаванням точок, встановленням їхнього кольору, розміру, форми, прозорості.
scipy.**spatial** ця бібліотека використовується для побудови діаграми Вороного.