МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ по лабораторной работе №4 по дисциплине «Алгоритмы и структуры данных»

Тема: Поиск образца в тексте: алгоритм Рабина-Карпа. Построение выпуклой оболочки: алгоритм Грэхема

Студент гр. 3342 Мохамед М.Х.

Преподаватель Иванов Д. В.

Санкт-Петербург

2024

Цель работы

Изучить хэширование и разновидности алгоритмов. С помощью этих полученных навыков реализовать 2 алгоритма: 1)для нахождения всех вхождений строки в другую 2)для получения выпуклого многоугольника и нахождения его площади.

Задание

Поиск образца в тексте. Алгоритм Рабина-Карпа.

Напишите программу, которая ищет все вхождения строки Pattern в строку Text, используя алгоритм Карпа-Рабина.

На вход программе подается подстрока Pattern и текст Text. Необходимо вывести индексы вхождений строки Pattern в строку Text в возрастающем порядке, используя индексацию с нуля.

Алгоритм Грэхема

Дано множество точек, в двумерном пространстве. Необходимо построить выпуклую оболочку по заданному набору точек, используя алгоритм Грэхема.

Также необходимо посчитать площадь получившегося многоугольника.

Выпуклая оболочка - это наименьший выпуклый многоугольник, содержащий заданный набор точек.

На вход программе подается следующее:

* первая строка содержит n - число точек

* следующие n строк содержат координаты этих точек через ', '

На выходе ожидается кортеж содержащий массив точек в порядке обхода алгоритма и площадь получившегося многоугольника.

Выполнение работы

1) Поиск образца в тексте. Алгоритм Рабина-Карпа.

Алгоритм работает следующим образом.

В перемнную base сохраняется основание для хэштрования, количество символов в алфавите (256 в ASCII). Переменную prime используем для вычисления хэшей с целью минимизации коллизий и устойчивости к переполнению.

Дальше в массиве powers храним степени base. Затем при обновлении хеша для следующей подстроки используем powers[m - 1] для учета старшего символа в хеше, что исключает повторные вычисления и ускоряет процесс.

Для подстроки и первой части текста считаем хеш через такие аргументы, как base, prime и числовое значение каждого символа.

После этого идет проверка каждой подстроки с нашим паттерном для нахождения нужным нам индексов. Сначала сравниваются паттерны. Если они совпали, то дальше сравниваем строки посимвольно. В случае успешной проверки добавляем индекс данной подстроки в массив.

Дальше для перехода к следующей подстроке обновляется её хэш. Формула: hash_text = ((hash_text - ord(text[i]) * powers[m - 1]) * base + ord(text[i + m])) % prime

2) Алгоритм Грэхема

Алгоритм написан в функции graham. Сначала выбирается самая левая вершина. После этого через цикл проверяется, чтобы каждая следующая вершина лежала левее прямой, проходящей через нашу первую точку и предыдущую (это проверяет функция rotate). Алгоритм в функции rotate основан на векторном произведении. После этого точки добавляются в стек. Через эту структуру проверяется, чтобы наша фигура была выпуклая.

Функция square через специальную формулу находит площадь фигуры по получившимся вершинам.

Анализ и тестирование полученных значений

Нужно проверить работоспособность алгоритма. Тестировались различные функции через pytest(см. test.py в приложении).

Для анализа алгоритма Грэхема написана копия этой фунции, которая на каждом шаге выводит действия, проводимые с вершинами (см. visualization_graham.py).

Выводы

Были изучены принципы работы алгоритмов Рабина-Карпа и Грэхема. Их реализация была написана на языке программирования Python. Работоспособность была проверена через pytest на некоторых входных данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
def rabin_karp_search(pattern, text):
    if len(pattern) == 0 or len(pattern) > len(text):
        return []

    pattern_length = len(pattern)
    text_length = len(text)
    prime = 10**9 + 7
    base = 256

    pattern_hash = 0
    current_hash = 0
    base_power = 1

    for i in range(pattern_length - 1):
        base_power = (base_power * base) % prime

    for i in range(pattern_length):
        pattern_hash = (pattern_hash * base + ord(pattern[i])) % prime
        current_hash = (current_hash * base + ord(text[i])) % prime

    result_indices = []

    for i in range(text_length - pattern_length + 1):
        if pattern_hash == current_hash:
            if text[i:i + pattern_length] == pattern:
                result_indices.append(i)

        if i < text_length - pattern_length:
            current_hash = (current_hash - ord(text[i]) * base_power) % prime
            current_hash = (current_hash * base + ord(text[i + pattern_length])) %
prime
            current_hash = (current_hash + prime) % prime

    return result_indices

def rotate(A, B, C):
    return (B[0] - A[0]) * (C[1] - B[1]) - (B[1] - A[1]) * (C[0] - B[0])

def find_area(points):
    n = len(points)
    area = 0.0
    for i in range(n):
```

```
        j = (i + 1) % n
        area += points[i][0] * points[j][1]
        area -= points[j][0] * points[i][1]
    area = abs(area) / 2.0
    return area

def graham_scan(points):
    n = len(points)
    if n < 3:
        return points

    P = list(range(n))

    for i in range(1, n):
        if points[P[i]][0] < points[P[0]][0]:
            P[i], P[0] = P[0], P[i]

    for i in range(2, n):
        j = i
        while j > 1 and rotate(points[P[0]], points[P[j - 1]], points[P[j]]) < 0:
            P[j], P[j - 1] = P[j - 1], P[j]
            j -= 1

    S = [P[0], P[1]]
    for i in range(2, n):
        while len(S) > 1 and rotate(points[S[-2]], points[S[-1]], points[P[i]]) <
0:
            S.pop()
        S.append(P[i])

    return S

if __name__ == "__main__":
    pattern = input("Enter the pattern: ").strip()
    text = input("Enter the text: ").strip()

    occurrences = rabin_karp_search(pattern, text)
    print("Pattern found at indices:", " ".join(map(str, occurrences)))

    n = int(input("Enter the number of points: "))
    A = []
    for i in range(n):
        t = list(map(int, input("Enter point (x, y): ").split(', ')))
        A.append(t)

    convex_hull_indices = graham_scan(A)
    convex_hull_points = [A[i] for i in convex_hull_indices]
```

```
    area = find_area(convex_hull_points)

    print("Convex Hull Points:", convex_hull_points)
    print("Area of Convex Hull:", area)
```

Название файла: test.py

```
from main import rabin_karp_search, graham_scan, find_area

def test_rabin_karp():
    """Тестирование алгоритма Рабина-Карпа"""
    assert rabin_karp_search('abacaba', 'aba') == [0, 4]
    assert rabin_karp_search('xyz', 'abcdefg') == []
    assert rabin_karp_search('', 'abcdefgh') == []
    assert rabin_karp_search('abcdefgh', 'abcdefgh') == [0]
    assert rabin_karp_search('a', 'aaaaaa') == [0, 1, 2, 3, 4, 5]  # Multiple
matches

def test_graham_make_figure():
    """Тестирование алгоритма Грэхема на соединение вершин"""
    result = graham_scan([[3, 1], [6, 8], [1, 7], [9, 3], [9, 6], [9, 0]])
    assert result == [1, 0, 5, 3, 4, 2]  # Update this based on the expected indices

    # Test with collinear points
    result_collinear = graham_scan([[1, 1], [2, 2], [3, 3], [4, 4]])
    assert result_collinear == [0, 3]  # Only endpoints should be part of the hull

def test_graham_find_square():
    """Тестирование алгоритма Грэхема на нахождение площади"""
    result = graham_scan([[3, 1], [6, 8], [1, 7], [9, 3], [9, 6], [9, 0]])
    convex_hull_points = [[3, 1], [1, 7], [6, 8], [9, 6], [9, 3], [9, 0]]  # Expected
points
    assert find_area(convex_hull_points) == 47.5  # Update this based on the
expected area

    # Test with a simple square
    square_points = [[0, 0], [0, 1], [1, 1], [1, 0]]
    assert find_area(square_points) == 1.0  # Area of the square

if __name__ == "__main__":
    test_rabin_karp()
    test_graham_make_figure()
    test_graham_find_square()
    print("All tests passed!")
```