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. 3384 _____ Рудаков А.Л.

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург
2024

Цель работы.

Изучить, Алгоритм Рабина-Карпина и алгоритмы Грэхема, после чего реализовать их на языке Python.

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.

Примечание: в работе запрещено использовать библиотечные реализации алгоритмов и структур.

Ограничения

$1 \leq |\text{Pattern}| \leq |\text{Text}| \leq 5 \cdot 105.$

Суммарная длина всех вхождений образца в тексте не превосходит 108. Обе строки содержат только буквы латинского алфавита.

Пример.

Вход:

aba

abacaba

Выход:

0 4

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.

Пример входных данных

6

3, 1

6, 8

1, 7

9, 3

9, 6

9, 0

Пример выходных данных

([[1, 7], [3, 1], [9, 0], [9, 3], [9, 6], [6, 8]], 47.5)

Также к очной защите необходимо подготовить визуализацию работы алгоритма, это можно сделать выводом в консоль или с помощью сторонних библиотек (например Graphviz).

Выполнение работы.

Первым был реализован алгоритм Рабина-Карпина. Вначале считываются строки *pattern* и *text*. Далее определяется количество уникальных символов, для определения мощности алфавита. Также определяется $q = 53$ – простое число, по модулю которого вычисляется хэш. Далее рассчитываются *power^i*, которые добавляются в массив, для последующего использования, чтобы не считать их каждый раз заново. Далее рассчитывается хэш паттерна, который записывается в *pattern_hash*, после чего начинается поиск паттерна в тексте.

При поиске происходит проход по всему тексту. Пока итератор меньше длины паттерна, хэш текущей подстроки *current_hash* вычисляется по формуле:

Текущий хэш = (текущий хэш + код символа * мощность алфавита ^ индекс цифры) % модуль.

Если же итератор больше либо равен длине строки, то хэш текущей подстроки вычисляется по формуле:

Текущий хэш = ((текущий хэш – код первого символа предыдущей строки * мощность алфавита ^ (длина паттерна – 1)) * мощность алфавита + код символа) % модуль.

Далее проверяется совпадение текущего хэша с хэшем паттерна, если они равны, то текущая подстрока сравнивается с паттерном посимвольно, если они равны, то индекс начала подстроки добавляется в *index_data*.

После завершения цикла индексы найденных подстрок выводятся на экран.

Далее реализован алгоритм Грэхема.

Вначале считывается количество подаваемых точек и сами точки, которые записываются в *points*. При считывании вычисляется самая левая точка (точка с наименьшей координатой x), она ставится в начало списка.

Далее точки в списке при помощи функции *def sort_point(points)* сортируются по их степени “левости” (если есть точки A, B, C, где A – начальная, то B < C, если C лежит левее вектора AB, и наоборот, B правее C, если B лежит правее вектора AC), чем точка правее, тем она ближе к началу, за основу взят алгоритм сортировки вставками. При этом используется функция *def rotate(point1, point2, point3),* выполняющая рассчет координаты z произведения векторов AB и BC, $z = a_x b_y - a_y b_x$, если z > 0 – поворот левый, иначе правый.

Далее вызывается функция *def get_convex_surface(points)*, которая вычисляет точки, принадлежащие выпуклой оболочке. За основу берутся первые 2 точки в списке, далее, проходя по начальному списку проверяется каждая точка, если при сравнении последних двух добавленных точек с текущей получается левый поворот, она добавляется в список, если правый, то предыдущие точки удаляются из списка, пока поворот не станет левым. После

прохождения она возвращает полученный список в порядке обхода против часовой стрелки.

Далее в функции *def area(points)* вычисляется площадь фигуры по формуле Гаусса.

После чего выводится результат – список точек и площадь получившейся фигуры.

Тестирование:

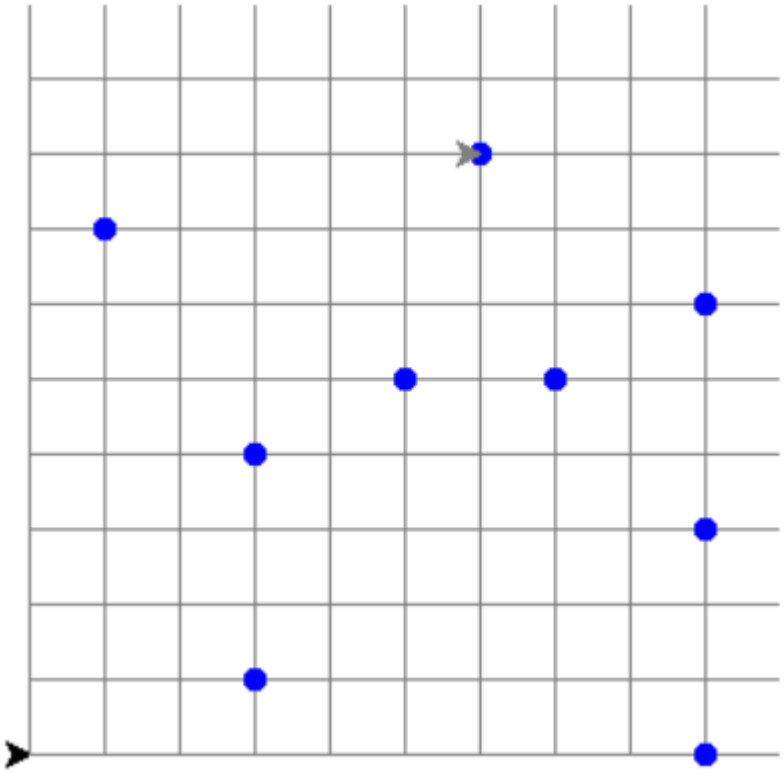

Рисунок 1 – Начальные данные

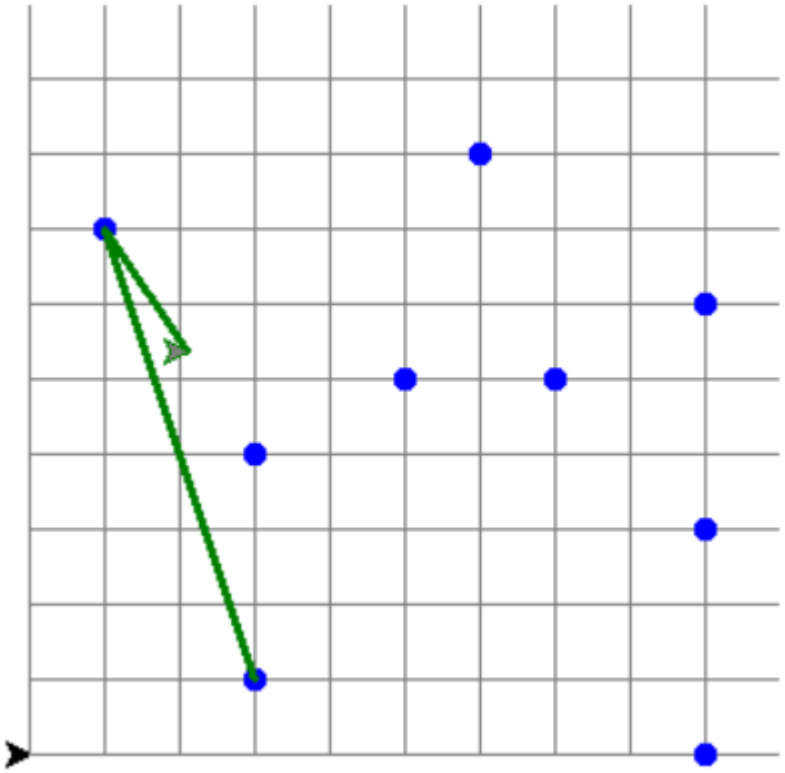

Рисунок 2 – Сортировка точек

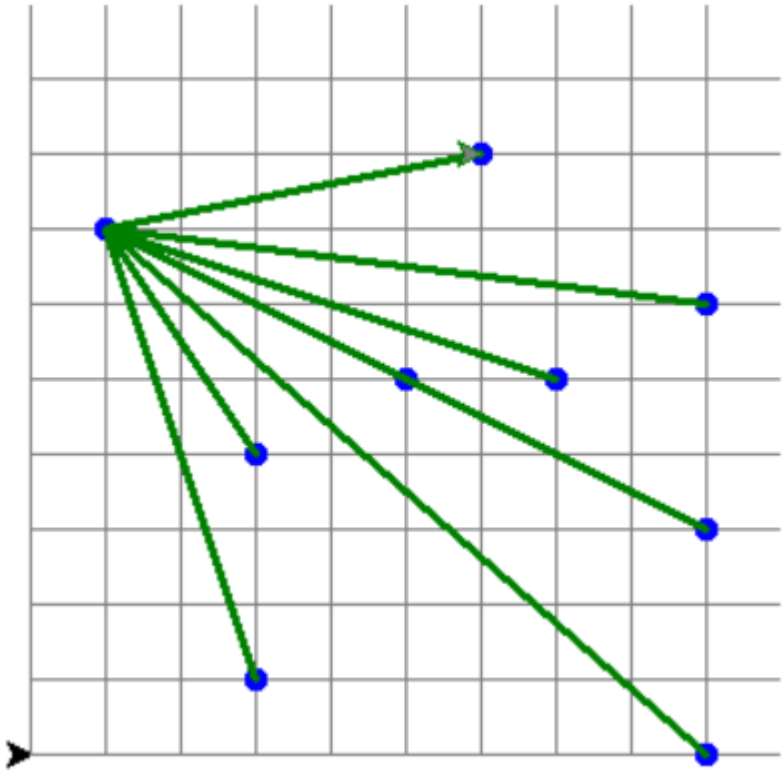

Рисунок 3 – отсортированные точки

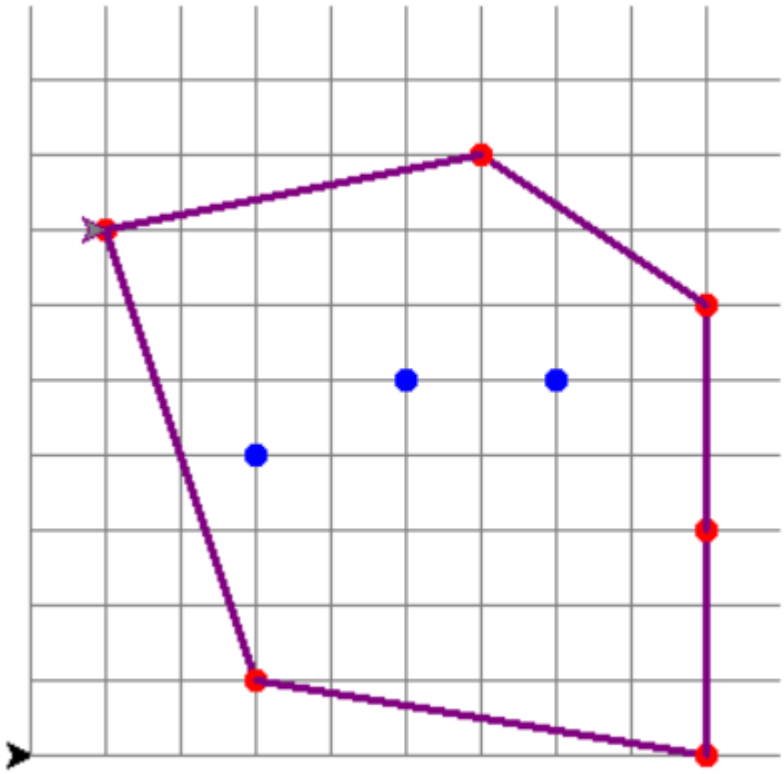

Рисунок 4 – Получившаяся поверхность

Разработанный программный код см. в приложении А.

Выводы.

Были изучены и реализованы алгоритмы Рабина-Карпа и Грэхема. В алгоритме Рабина-Карпина было использовано хэширование. В алгоритме Грэхема была использована вычислительная геометрия.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Rubin-Carp.py

```
pattern = input()
text = input()
unq_symb = []
for i in text:
    if not (i in unq_symb):
        unq_symb.append(i)
power = len(unq_symb)
q = 53
polynomial_value = []
text_length = len(text)
pattern_hash = 0
pattern_length = len(pattern)
for i in range(pattern_length):
    polynomial_value.append(power ** i)

for i in range(pattern_length):
    pattern_hash = (pattern_hash + unq_symb.index(pattern[i]) *
polynomial_value[pattern_length - i - 1]) % q

index_data = []
current_hash = 0
for i in range(text_length):
    if i < pattern_length:
        current_hash = (current_hash + unq_symb.index(text[i]) *
polynomial_value[pattern_length - i - 1]) % q
    else:
        old_hash = (unq_symb.index(text[i - pattern_length]) *
polynomial_value[pattern_length - 1]) % q
        current_hash = ((current_hash - old_hash) * power +
unq_symb.index(text[i])) % q
    if current_hash == pattern_hash:
        add_value = i - pattern_length + 1
        flag = True
        if i < pattern_length:
            start = 0
        else:
            start = i - pattern_length + 1
        for j in range(start, i + 1):
            if (text[j] != pattern[j - start]):
                flag = False
                break
        if flag == True and add_value >= 0:
            index_data.append(add_value)

print(' '.join(map(str, index_data)))
```

Название файла: Graham.py

```
def rotate(point1, point2, point3):
```

```
        return       (point2[0]-point1[0])*(point3[1]-point2[1])       -
(point2[1]-point1[1])*(point3[0]-point2[0])

    def sort_point(points):
        for i in range(1, len(points)):
            for j in range(i, 1, -1):
                if rotate(points[0], points[j], points[j-1]) > 0:
                    points[j], points[j-1] = points[j-1], points[j]
        return points

    def get_convex_surface(points):
        new_points = points[:2]
        for i in range(2, len(points)):
            while  rotate(new_points[-2],  new_points[-1],  points[i])  <
0:
                new_points.pop()
            new_points.append(points[i])
        return new_points

    def area(points):
        result = 0
        length = len(points)
        for i in range(length):
            result += points[i % length][0] * points[(i+1) % length][1]
            result -= points[(i+1) % length][0] * points[i % length][1]
        result /= 2
        return result

    n = int(input())
    points = []
    min_point = []
    for i in range(n):
        current_point = list(map(int, input().split(', ')))
        points.append(current_point)
        if points[i][0] < points[0][0]:
            points[i], points[0] = points[0], points[i]

    points = sort_point(points)
    new_points = get_convex_surface(points)
    result_area = area(new_points)
    print(f'({new_points}, {result_area})')
```