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 _____ Пьянков М.Ф.

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург
2024

Цель работы

Изучение алгоритма Рабина-Карпа для поиска вхождений подстроки в данную строку, алгоритма Грэхема для выделения минимальной выпуклой оболочки и их реализация на языке программирования python.

Задание

Задание №1

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 текста не превосходит 108. Обе строки содержат только буквы латинского алфавита.

Пример.

Вход:

aba

abacaba

Выход:

0 4

Задание №2

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). Визуализацию загружать не нужно. В данной работе первую точку нужно выбирать по наименьшей x координате. Обход производить в направлении против часовой стрелки.

Выполнение работы

Задание №1

Функция хэширования строки — полиномиальная: функция def hash(s)

1 шаг: посчитаем начальные хэши данной строки и искомой подстроки.

2 шаг: если на какой-то итерации цикла поэлементного перебора строки хэш строки сравняется с хэшем подстроки, то необходимо совершить проверку на сходство данной подстроки исходной строки с искомой подстрокой и в случае успеха добавить индекс в ответ. Далее переопределим хэш строки по формуле: hash_s = ((hash_s – s[i] * p ** (m-1)) * p + s[i + m]) % q, где p – простое число (в данном случае 31), q – большое число по модулю которого переопределяется хэш (в данном случае 10^9 + 7), m – длина искомой подстроки: функция RabinKarp(w,s).

Задание №2

Необходимая функция для определения с какой стороны от вектора лежит точка: def DotLeftSide(first, second, third) здесь векторы: second – first и third – second (считается векторное произведение и если оно положительное, то точка third лежит слева от вектора first – second иначе - справа)

Разделим алгоритм Грэхема на 4 шага

1 шаг: создадим массив индексов точек от 0 до n и произведём следующие шаги: функция def Graham(dots).

2 шаг: переместим точку с наименьшей координатой x в начало массива индексов: функция def StartPointSetting(dots, indexes).

3 шаг: отсортируем все точки по критерию расположения точки слева от вектора из предыдущей и начальной точки. Применим сортировку вставками: функция def InsertionSortByLeftSideStart(dots, indexes).

4 шаг: уберём лишние точки, которые находятся слева текущего вектора и добавим те, которые лежат справа, таким образом получим выпуклую оболочку точек, которая к тому же будет минимальной: функция def AngleSlicing(dots, indexes).

Для нахождения площади получившейся минимальной выпуклой оболочки воспользуемся методом Гаусса: функция def Area(dots, indexes).

Разработанный программный код см. в приложении А.

Визуализация и тестирование алгоритма Грэхема

Используя matplotlib.pyplot визуализируем работу алгоритма на каждом шаге:

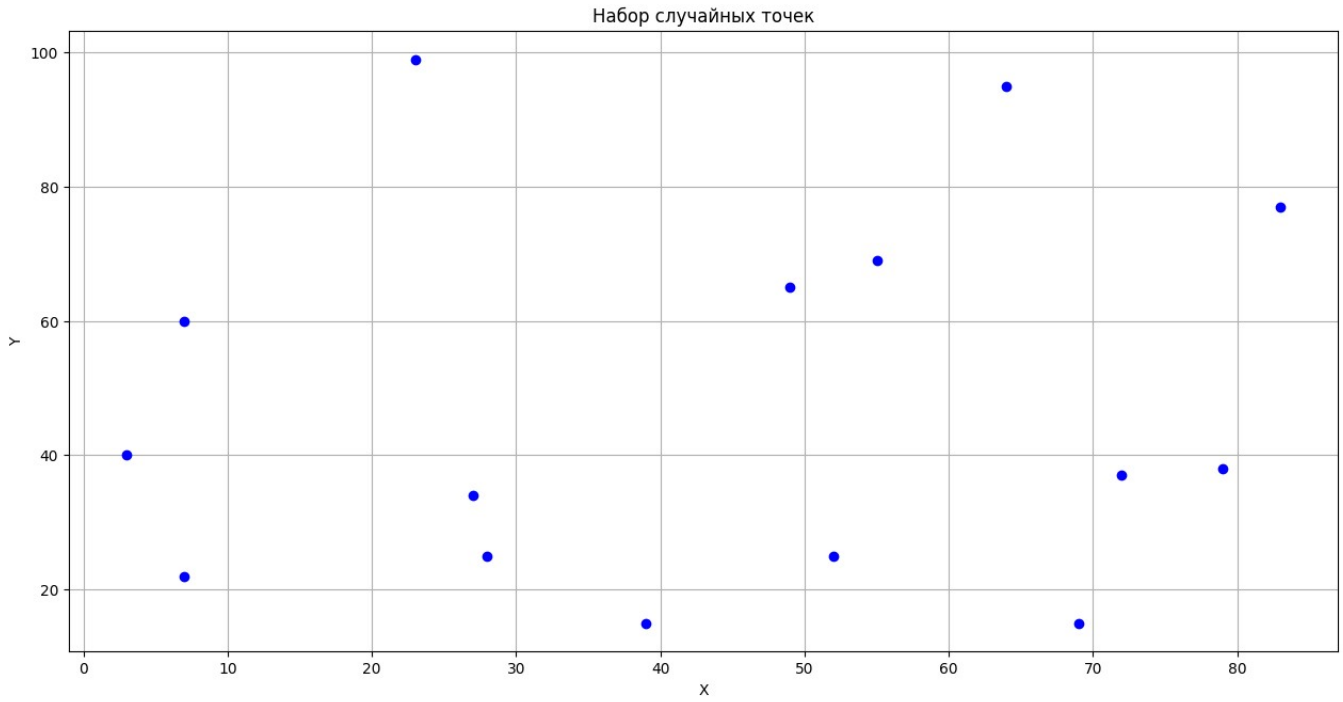

Рисунок 1 — Шаг 1 (визуализация исходных точек)

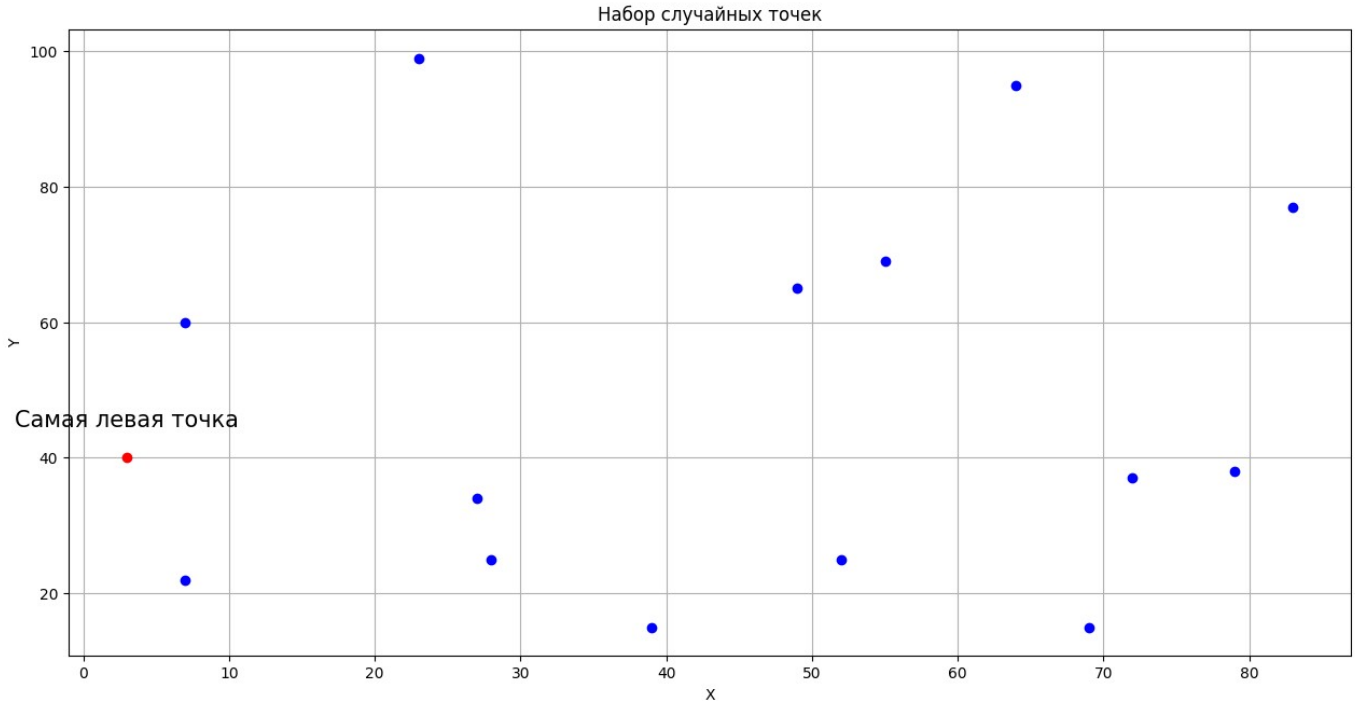

Рисунок 2 — Шаг 2 (Поиск и перемещение стартовой точки)

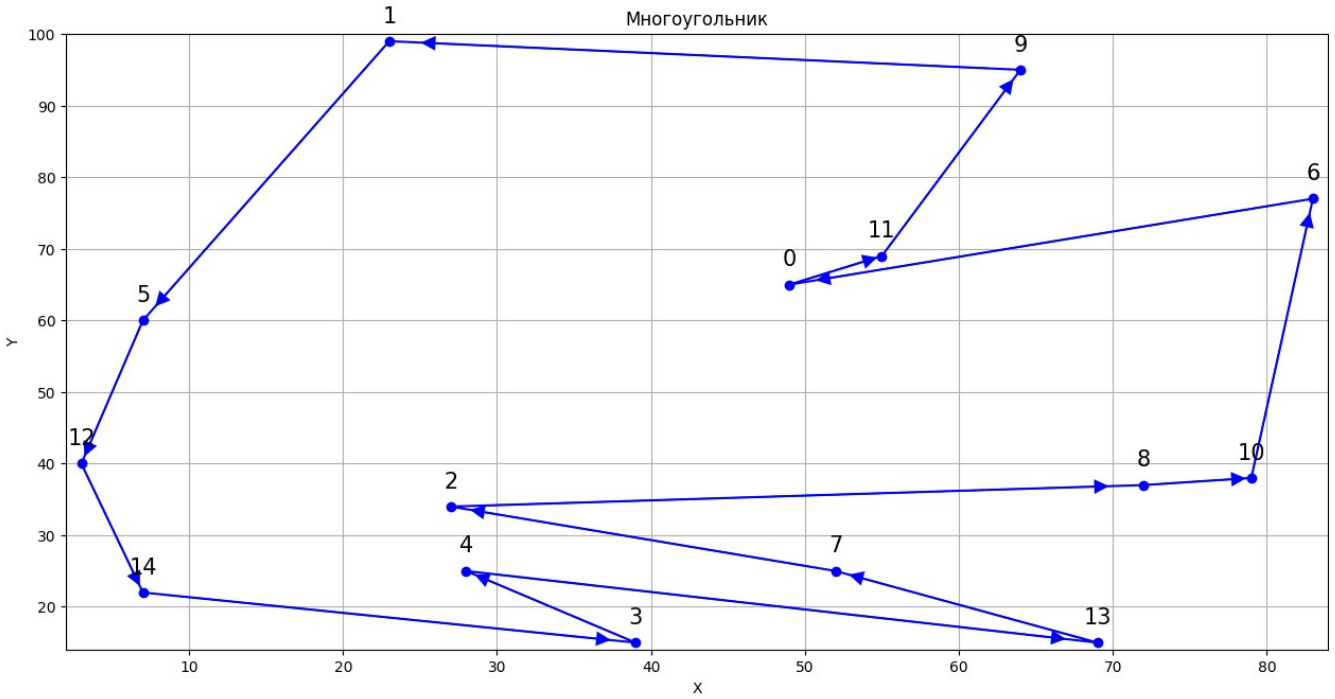

Рисунок 3 — Шаг 3 (Сортировка точек по критерию расположения с левой стороны)

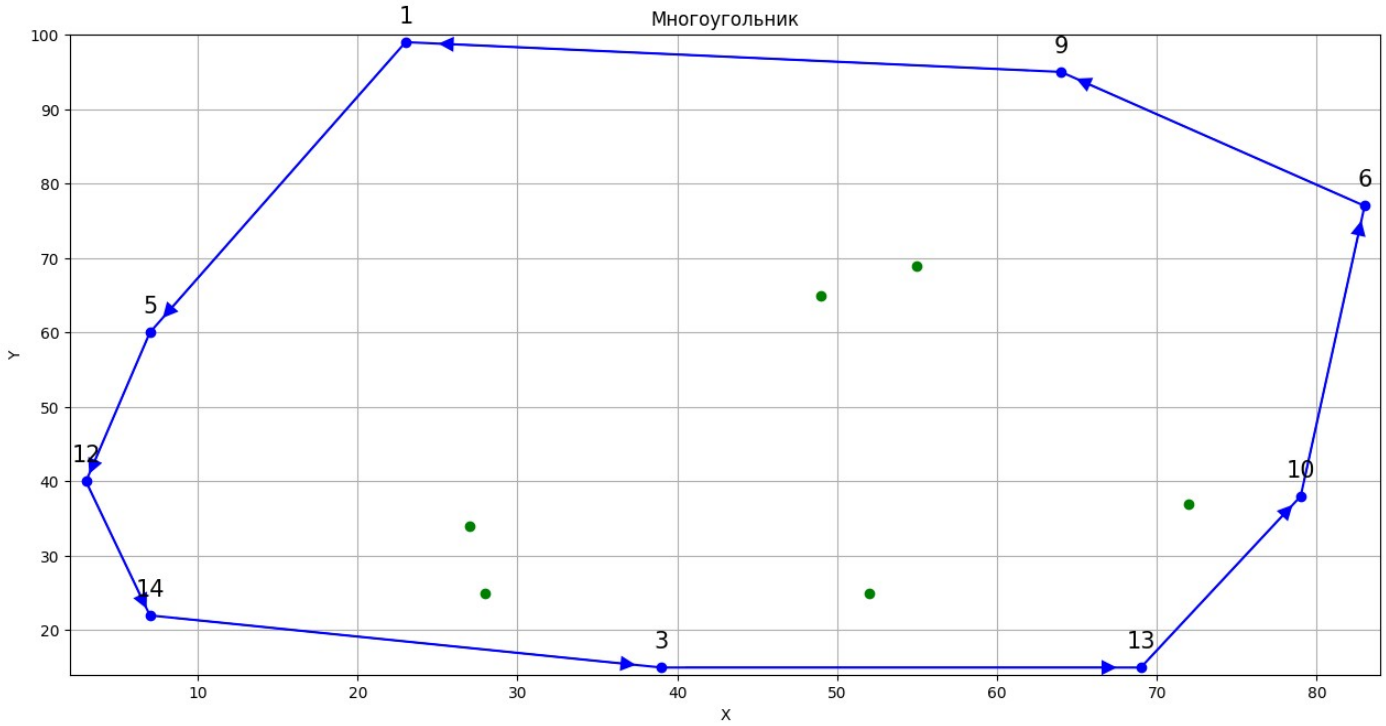

Рисунок 4 — Шаг 4 (Удаление лишних точек, которые лежат слева и добавление точек, которые лежат справа)

Как видно, алгоритм отработал корректно.

Выводы

Исследованы и успешно реализованы алгоритмы Рабина-Карпа и Грэхема для решения поставленных задач, которые были успешно выполнены. Произведены визуализация и тестирование алгоритма Грэхема.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.py

```
def DotLeftSide(first, second, third):
    return (second[0] - first[0]) * (third[1] - second[1]) -
(second[1] - first[1]) * (third[0] - second[0])


def StartPointSetting(dots, indexes):
    for i in range(1, len(dots)):
        if dots[indexes[i]][0] < dots[indexes[0]][0]:
            indexes[i], indexes[0] = indexes[0], indexes[i]
    return indexes


def InsertionSortByLeftSideStart(dots, indexes):
    for i in range(2, len(indexes)):
        j = i
        while j > 1 and DotLeftSide(dots[indexes[0]], dots[indexes[j -
1]], dots[indexes[j]]) < 0:
            indexes[j], indexes[j - 1] = indexes[j - 1], indexes[j]
            j -= 1
    return indexes


def AngleSlicing(dots, indexes):
    result = [indexes[0], indexes[1]]
    for i in range(2, len(dots)):
        while DotLeftSide(dots[result[-2]], dots[result[-1]],
dots[indexes[i]]) < 0:
            result.pop()
        result.append(indexes[i])
    return result


def Graham(dots):
    n = len(dots)
    indexes = [i for i in range(n)]
    indexes = StartPointSetting(dots, indexes)
    indexes = InsertionSortByLeftSideStart(dots, indexes)
    return AngleSlicing(dots, indexes)


def Area(dots, indexes):
    square = 0
    for i in range(len(indexes)):
        x1, y1 = dots[indexes[i]]
        x2, y2 = dots[indexes[(i + 1) % len(indexes)]]
        square += x1 * y2 - x2 * y1
    return abs(square) / 2


if __name__ == "__main__":
```

```
n = int(input())
dots = []
for i in range(n):
    x, y = list(map(int, input().split(', ')))
    dots.append([x, y])
indexes = Graham(dots)
print(([dots[i] for i in indexes], Area(dots, indexes)))
```

Файл RabinKarp.py

```
#python
def hash(s):
    q = (10**9 + 7)
    result = ord(s[0])
    base = 31
    for i in range(len(s) - 1):
        result = result * base + ord(s[i+1])
    return result % q
def RabinKarp(w,s):
    base = 31
    q = (10 ** 9 + 7)
    n = len(s)
    m = len(w)
    hash_s = hash(s[0:m])
    hash_w = hash(w)
    res = []
    i = 0
    while True:
        if hash_s == hash_w:
            if w == s[i:i+m]:
                res.append(i)
        if i + m >= n:
            break
        hash_s = ((hash_s - ord(s[i]) * base ** (m-1)) * base +
ord(s[i + m])) % q
        i += 1
    return ' '.join(map(str, sorted(res)))

if __name__ == "__main__":
    m = input()
    s = input()
    print(RabinKarp(m, s))
```