**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 Грэхэма**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Лодыгин И.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Реализовать и исследовать алгоритм Рабина-Карпа и алгоритм Грэхема, а также привести визуализацию алгоритма Грэхема.

## 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 ≤ |Pattern| ≤ |Text| ≤ 5 · 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

**Подсказки:**

1. Будьте осторожны с операцией взятия подстроки — она может оказаться дорогой по времени и по памяти.

2. Храните степени x \*\* p в списке - тогда вам не придется вычислять их каждый раз заново.

**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**

## Выполнение работы

**Алгоритм Рабина-Карпа.**

Задача заключалась в реализации алгоритма поиска подстрок на основе хэширования. В процессе работы были созданы две функции: get\_hash для вычисления хэша строки и search\_patterns для выполнения поиска подстрок в заданном тексте. Используются константы 𝑄=256 (размер хэш-таблицы) и B=13 (основное значение для расчета хэша). Ниже приводится описание функционала и алгоритмов.

1. Функция get\_hash

Эта функция отвечает за вычисление хэш-значения строки с использованием алгоритма, основанного на методе Полиномиального Хэширования.

Порядок работы:

* Инициализируется переменная res, которая хранит текущее хэш-значение.
* Для каждого символа строки:
* Текущее хэш-значение умножается на константу B и прибавляется ASCII-код символа, после чего вычисляется остаток по модулю Q.
* Возвращается итоговое хэш-значение.

2. Функция search\_patterns

Эта функция реализует поиск всех вхождений заданного шаблона (pattern) в текст (text) с использованием алгоритма Рабина-Карпа.

Порядок работы:

1. Инициализируется список result, куда будут записываться индексы всех найденных совпадений.
2. Вычисляются длины текста (text\_length) и шаблона (pattern\_length).
3. Рассчитывается множитель multiplier. Этот множитель используется для обновления хэша текста на каждом шаге.
4. Вычисляются хэши для шаблона (pattern\_hash) и начальной подстроки текста (text\_hash) с использованием функции get\_hash.
5. Выполняется основной цикл:

* На каждом шаге сравниваются хэши подстроки текста и шаблона.
* Если хэши совпадают, выполняется дополнительная проверка содержимого подстроки для устранения коллизий.
* Если подстрока совпадает с шаблоном, её начальный индекс записывается в result.
* Хэш обновляется для следующей подстроки, используя формулу.
* Если обновлённое хэш-значение отрицательно, к нему прибавляется Q для нормализации.

1. По завершении цикла возвращается список индексов совпадений.

**Алгоритм Грэхэма.**

Реализована программа для построения выпуклой оболочки множества точек на плоскости с использованием алгоритма Грэма. Для достижения цели было создано три функции: area, orientation и graham. Каждая из них выполняет свою задачу, обеспечивая корректное выполнение алгоритма. Ниже описан ход выполнения.

1. Функция area

Функция предназначена для вычисления площади многоугольника, заданного массивом его вершин.

Ход выполнения:

* Инициализируется переменная value, в которой накапливается результат.
* Выполняется обход всех точек, где для каждой точки рассчитывается произведение координат текущей и следующей точки (с учетом замыкания многоугольника).
* Суммируется разность произведений.
* Итоговое значение делится пополам и берётся модуль для исключения отрицательных значений.
* Возвращается площадь многоугольника.

2. Функция orientation

Функция определяет ориентацию трёх точек, что необходимо для построения выпуклой оболочки.

Ход выполнения:

* Вычисляется ориентирующее значение val с использованием определителя, который отражает расположение точки third относительно вектора (first,second).
* В зависимости от значения val:
* Возвращается 0, если точки лежат на одной прямой.
* Возвращается 1, если поворот направлен по часовой стрелке.
* Возвращается -1, если поворот против часовой стрелки.
* Эта функция помогает проверять выпуклость при добавлении новых точек в оболочку.

3. Функция graham

Функция реализует алгоритм Грэма для построения выпуклой оболочки.

Ход выполнения:

* Ищется точка с минимальной x-координатой (при равных x — с минимальной y-координатой). Эта точка становится начальной.
* Сортируются остальные точки относительно начальной по их полярному углу.
* Инициализируется стек points\_stack, куда помещаются начальная точка и следующая точка из отсортированного списка.
* Для каждой последующей точки:

1. Проверяется ориентация последних двух точек в стеке и текущей точки.
2. Если ориентация показывает, что добавление текущей точки нарушает выпуклость (поворот вправо), последняя точка удаляется из стека.
3. После удаления нарушающих точек текущая точка добавляется в стек.

* По завершении цикла в стеке остаются вершины выпуклой оболочки в порядке обхода.

**Программный код:** Разработанный программный код для алгоритмов Рабина-Карпа и Грэхэма приведён в приложении А.

## Тестирование

Название файла: tests.py

from modules.rabin\_karp import search\_patterns

from modules.graham import graham, area

def test\_rabin\_karp():

pattern = "lar"

text = "lar fff lar lar"

assert search\_patterns(pattern, text) == [0, 8, 12], "Test failed for Rabin-Karp"

def test\_graham():

dots = [[1, 1], [3, 4], [5, 2], [2, 2], [4, 4], [0, 0]]

expected\_hull = [[0, 0], [5, 2], [4, 4], [3, 4]]

expected\_area = 8.0

hull, hull\_area = graham(dots), area(expected\_hull)

assert hull == expected\_hull, f"Expected {expected\_hull}, got {hull}"

assert abs(hull\_area - expected\_area) < 1e-6, f"Expected area {expected\_area}, got {hull\_area}"

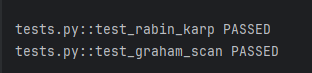
test\_rabin\_karp()

print("Rabin-Karp test passed")

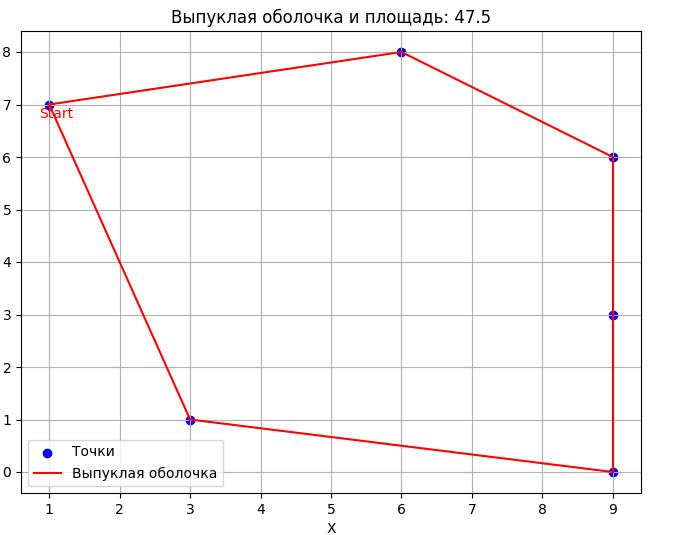
test\_graham()

print("Graham-Scan test passed")

Результат работы программы:



Визуализация результата работы алгоритма Грэхэма:



Для визуализации самого алгоритма запустить файл main.py.

## Выводы

Были изучены и применены на практике алгоритм Рабина-Карпа и алгоритм Грэхэма.

# Приложение А Исходный код программы

Название файла: main.py

from modules.rabin\_karp import search\_patterns

from modules.graham import graham

from modules.drawer import visualize\_convex\_hull

pattern = input()

text = input()

result = search\_patterns(pattern, text)

print(\*result)

quantity = int(input())

dots = list()

for \_ in range(quantity):

dots.append(list(map(int, input().split(","))))

hull, area = graham(dots)

print(hull, area)

visualize\_convex\_hull(dots, hull, area)

Название файла: rabin\_karp.py

Q = 256

B = 13

def get\_hash(string: str) -> int:

length = len(string)

res = 0

for i in range(length):

res = (B\*res + ord(string[i])) % Q

return res

def search\_patterns(pattern: str, text: str) -> list:

result = list()

text\_length = len(text)

pattern\_length = len(pattern)

multiplier = 1

for \_ in range(1, pattern\_length):

multiplier = (multiplier \* B) % Q

pattern\_hash = get\_hash(pattern)

text\_hash = get\_hash(text[:pattern\_length])

for i in range(text\_length - pattern\_length + 1):

if(pattern\_hash == text\_hash):

if text[i: i + pattern\_length] == pattern:

result.append(i)

if(i < text\_length - pattern\_length):

text\_hash = ((text\_hash - ord(text[i]) \* multiplier) \* B + ord(text[i + pattern\_length])) % Q

if text\_hash < 0:

text\_hash += Q

return result

Название файла: graham.py

def area(points):

value = 0.0

for i in range(len(points)):

next\_i = (i + 1) % len(points)

value += points[i][0] \* points[next\_i][1] - points[i][1] \* points[next\_i][0]

return abs(value) / 2

def orientation(first, second, third):

val = (second[1] - first[1]) \* (third[0] - second[0]) - (second[0] - first[0]) \* (third[1] - second[1])

if val == 0:

return 0

elif val > 0:

return 1

else:

return -1

def graham(data: list) -> list:

for i in range(1, len(data)):

if data[0][0] > data[i][0]:

data[0], data[i] = data[i], data[0]

for i in range(2, len(data)):

tmp = i

while tmp > 1 and orientation(data[0], data[tmp - 1], data[tmp]) == 1:

data[tmp], data[tmp - 1] = data[tmp - 1], data[tmp]

tmp -= 1

points\_stack = [data[0], data[1]]

for i in range(2, len(data)):

while len(points\_stack) > 1 and orientation( points\_stack[-2], points\_stack[-1], data[i] ) == 1:

points\_stack.pop()

points\_stack.append(data[i])

return points\_stack

Название файла: drawer.py

import matplotlib.pyplot as plt

def visualize\_convex\_hull(points, convex\_hull, convex\_area):

plt.figure(figsize=(8, 6))

x\_coords, y\_coords = zip(\*points)

plt.scatter(x\_coords, y\_coords, color="blue", label="Точки")

plt.annotate("Start", points[0], xytext=(6, -10), textcoords="offset points",

color="red", ha="center")

for idx in range(len(convex\_hull)):

next\_idx = (idx + 1) % len(convex\_hull)

plt.plot(

[convex\_hull[idx][0], convex\_hull[next\_idx][0]],

[convex\_hull[idx][1], convex\_hull[next\_idx][1]],

color="red", label="Граница оболочки" if idx == 0 else ""

)

plt.title(f"Визуализация выпуклой оболочки\nПлощадь: {convex\_area}")

plt.xlabel("Ось X")

plt.ylabel("Ось Y")

plt.legend()

plt.grid(visible=True)

plt.show()