**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Кулач Д.В. |
| Преподаватель |  | Шалагинов И.В. |

Санкт-Петербург

2024

## Цель работы

Изучить и реализовать алгоритмы Рабина-Карпа и Грэхема. Представить визуализацию работы для алгоритма Грэхема. Провести исследование.

## 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

rabin\_karp(pattern, text) - Основная функция, реализующая алгоритм Рабина-Карпа для поиска всех вхождений строки pattern в строку text. Возвращает список индексов начальных позиций вхождений подстроки pattern в строку text.

\_\_init\_\_(self, x: int, y: int) -> None -Конструктор класса Point. Инициализирует точку с координатами (x, y).

\_\_eq\_\_(self, value) - Определяет, равны ли две точки. Сравнивает текущую точку с другой точкой value по координатам x и y. Генерирует исключение, если объект для сравнения не является точкой.

get\_coords(self) -> tuple[int, int] - Возвращает координаты точки в виде кортежа (x, y).

\_\_repr\_\_(self) -> str - Возвращает строковое представление точки в формате "[x, y]", удобное для отладки и вывода.

is\_left\_rotate(a: Point, b: Point, c: Point) -> bool - Определяет, находится ли точка c слева от вектора, образованного точками a и b. Возвращает True, если точка c лежит слева или на линии.

rotate(o: Point, a: Point, b: Point) -> int - Вычисляет величину векторного произведения для трёх точек: o, a, b. Используется для определения поворота: положительное значение — поворот влево, отрицательное — вправо.

graham(points: list[Point]) -> list[Point] - Реализует алгоритм построения выпуклой оболочки (алгоритм Грэхема). Сортирует точки и использует стек для добавления точек в выпуклую оболочку. Возвращает список точек, образующих выпуклую оболочку.

calculate\_area(hull: list[Point]) -> float - Вычисляет площадь многоугольника, заданного точками выпуклой оболочки, методом векторного произведения. Возвращает площадь (положительное число).

Исходный код программы см. в приложении А.

## Вывод

Были реализованы алгоритм Рабина-Карпа и Грэхема, а также визуализация для алгоритма Грэхема.

# Приложение А Исходный код программы

Название файла: graham.py

from modules.visualize import \*

def is\_left\_rotate(a: Point, b: Point, c: Point) -> bool:

u = Point(b.x - a.x, b.y - a.y)

v = Point(c.x - b.x, c.y - b.y)

return u.x \* v.y - u.y \* v.x >= 0

def rotate(o: Point, a: Point, b: Point) -> int:

return (a.x - o.x) \* (b.y - o.y) - (a.y - o.y) \* (b.x - o.x)

def graham(points: list[Point]) -> list[Point]:

stack = []

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

if points[i].x < points[0].x:

points[i], points[0] = points[0], points[i]

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

j = i

while j >= 2 and not is\_left\_rotate(points[0], points[j - 1], points[j]):

points[j], points[j - 1] = points[j - 1], points[j]

j -= 1

stack.append(points[0])

stack.append(points[1])

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

while not is\_left\_rotate(stack[-2], stack[-1], points[i]):

stack.pop()

stack.append(points[i])

return stack

def calculate\_area(hull: list[Point]) -> float:

n = len(hull)

area = 0

for i in range(n):

x1, y1 = hull[i].get\_coords()

x2, y2 = hull[(i + 1) % n].get\_coords()

area += x1 \* y2 - y1 \* x2

return abs(area) / 2

Название файла: rabin\_karp.py

def rabin\_karp(pattern, text):

p\_len = len(pattern)

t\_len = len(text)

if p\_len > t\_len:

return []

base = 256 # Размер алфавита

mod = 10\*\*9 + 7 # Большое простое число

base\_powers = [1] \* p\_len

for i in range(1, p\_len):

base\_powers[i] = (base\_powers[i - 1] \* base) % mod

pattern\_hash = 0

window\_hash = 0

for i in range(p\_len):

pattern\_hash = (pattern\_hash \* base + ord(pattern[i])) % mod

window\_hash = (window\_hash \* base + ord(text[i])) % mod

occurrences = []

for i in range(t\_len - p\_len + 1):

if pattern\_hash == window\_hash:

if text[i:i + p\_len] == pattern:

occurrences.append(i)

if i < t\_len - p\_len:

window\_hash = (window\_hash - ord(text[i]) \* base\_powers[p\_len - 1]) % mod

window\_hash = (window\_hash \* base + ord(text[i + p\_len])) % mod

window\_hash = (window\_hash + mod) % mod

return occurrences

Название файла: point.py

class Point:

def \_\_init\_\_(self, x: int, y: int) -> None:

self.x = x

self.y = y

def \_\_eq\_\_(self, value):

if not isinstance(value, Point):

raise ValueError("Point comparable only with points")

return self.x == value.x and self.y == value.y

def get\_coords(self) -> tuple[int, int]:

return self.x, self.y

def \_\_repr\_\_(self) -> str:

return f"[{self.x}, {self.y}]"

Название файла: visualize.py

import matplotlib.pyplot as plt

from modules.point import Point

def visualize(points: list[Point], hull: list[Point]) -> None:

x\_points = [p.x for p in points]

y\_points = [p.y for p in points]

x\_hull = [p.x for p in hull] + [hull[0].x]

y\_hull = [p.y for p in hull] + [hull[0].y]

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

plt.scatter(x\_points, y\_points, color='blue', label="Points", zorder=5)

plt.plot(x\_hull, y\_hull, color='red', label="Convex Hull", zorder=3)

for p in points:

plt.text(p.x + 0.1, p.y, f'({p.x}, {p.y})', fontsize=9)

plt.title("Graham Scan - Convex Hull")

plt.xlabel("X")

plt.ylabel("Y")

plt.legend()

plt.grid(True)

plt.show()

Название файла: main.py

from modules.rabin\_karp import rabin\_karp

from modules.graham import \*

def main():

print("Выберите алгоритм для выполнения:")

print("1. Алгоритм поиска подстроки Rabin-Karp")

print("2. Алгоритм выпуклой оболочки Грэма")

choice = input("Введите 1 или 2: ").strip()

if choice == "1":

pattern = input("Введите шаблон: ").strip()

text = input("Введите текст: ").strip()

result = rabin\_karp(pattern, text)

print("Шаблон найден в позициях:", " ".join(map(str, result)))

elif choice == "2":

n = int(input("Введите количество точек: "))

points = []

for \_ in range(n):

x, y = map(int, input("Введите координаты точки (x, y): ").split(", "))

points.append(Point(x, y))

hull = graham(points)

area = calculate\_area(hull)

print(f"Выпуклая оболочка: {hull}")

print(f"Площадь выпуклой оболочки: {area:.2f}")

# Визуализация

visualize(points, hull)

else:

print("Неверный выбор. Пожалуйста, введите 1 или 2.")

if \_\_name\_\_ == "\_\_main\_\_":

main()