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. 3382 _____ Копасова К. А.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить алгоритм Рабина-Карпа для поиска образца в тексте и алгоритм Грэхема для построения выпуклой оболочки и реализовать их на языке программирования Python.

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:

$1 \leq |Pattern| \leq |Text| \leq 5 \cdot 105$.

Суммарная длина всех вхождений образца в текста не превосходит 108.

Обе строки содержат только буквы латинского алфавита.

Пример:

Вход:

aba

abacaba

Выход:

0 4

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

Для начала разделим нашу задачу на два файла с алгоритмами и два файла с тестами для них: в первом файле main1.py будет реализован первый алгоритм поиска вхождения подстроки в текст (алгоритм Рабина-Карпа), а во втором файле main2.py будет реализован алгоритм построения выпуклой оболочки (алгоритм Грэхема) и его визуализация.

Начнем с файла main1.py – опишу все функции данного файла.

Основной принцип алгоритма заключается в использовании полиномиального хэширования для быстрого поиска совпадений шаблона в тексте.

Функция *polynomial_hash* вычисляет полиномиальный хэш для строки. Она итерируется по каждому символу входной строки, обновляя значение хэша по формуле hash_value = (hash_value * base + ord(char)) % mod, где ord(char) возвращает числовое представление символа. Это позволяет получить уникальное значение для строки, с минимизацией коллизий за счёт использования большого модуля mod.

Функция *cmp* выполняет побайтовое сравнение двух строк. Она проходит по каждому символу в парах строк и увеличивает счётчик совпадений при нахождении одинаковых символов. Если все символы совпадают, функция возвращает True, иначе False. Это нужно для подтверждения совпадения, так как хэширование не исключает вероятность коллизий.

Функция *get_sub_string_RK* реализует сам алгоритм Рабина-Карпа. Сначала она вычисляет длины шаблона и текста и хэш шаблона с использованием функции polynomial_hash. Затем итерируется по всем возможным подстрокам текста с той же длиной, что и шаблон. Для каждой подстроки вычисляется хэш, который сравнивается с хэшем шаблона. Если они совпадают, выполняется дополнительное посимвольное сравнение через

cmp, чтобы убедиться в полном совпадении. Все индексы совпадений записываются в список index, который затем преобразуется в строку индексов и возвращается.

Функция *main* отвечает за взаимодействие с пользователем. Она считывает шаблон и текст с помощью input() и передаёт их в функцию get_sub_string_RK(), результат которой выводится на экран.

Main1.py – опишу все функции данного файла.

Программа реализует алгоритм Грэхема для построения минимальной выпуклой оболочки множества точек на плоскости и её визуализации с помощью библиотеки matplotlib.

Функция *rotate* определяет ориентацию трёх точек на плоскости, используя векторное произведение. Она возвращает положительное значение, если поворот этих точек против часовой стрелки, ноль — если точки коллинеарны, и отрицательное — если поворот по часовой стрелке. Это используется для определения, образуют ли точки выпуклую оболочку.

Функция *grahamscan* реализует алгоритм Грэхема для построения выпуклой оболочки. Она сначала определяет самую левую точку и меняет её местами с первой точкой массива. Затем сортирует оставшиеся точки относительно этой начальной с помощью вставок по углу поворота. Алгоритм использует стек для построения оболочки, удаляя точки, которые образуют вогнутые углы, и добавляя те, которые формируют выпуклую структуру. Функция возвращает массив индексов точек, которые образуют выпуклую оболочку.

Функция *square* вычисляет площадь выпуклой фигуры, заданной массивом координат точек, по формуле площади многоугольника. Она итерирует по парам последовательных точек и вычисляет их вклад в площадь, используя формулу Гаусса для полигона. Результат возвращается как половина абсолютного значения суммы.

Функция *visualization* визуализирует построенную выпуклую оболочку и исходные точки. Она добавляет начальную точку в конец списка точек оболочки для замыкания фигуры и строит график с точками оболочки и исходными точками с использованием matplotlib. Параметры оформления, такие как маркеры и цвета, позволяют различать точки оболочки и исходные точки.

Функция *main* отвечает за ввод данных пользователем и выполнение алгоритма. Она считывает количество точек n и их координаты. Затем вызывает функцию grahamscan для получения индексов точек выпуклой оболочки и создает список с их координатами. Далее выводит координаты точек оболочки и её площадь, затем визуализирует результат с помощью visualization.

В файле test1.py и test2.py находятся дополнительные функции, помогающие построить графики для анализа производительности алгоритмов, отталкиваясь от подаваемого количества элементов и временем обработки данных. С помощью библиотек matplotlib и pandas был построен график зависимости.

Код программ находится в Приложении А.

Тестирование

Таблица 1 — Результаты тестирования кода программы main1.py:

№п/п	Входные данные	Выходные данные	Комментарии
1	aba abacaba	0 4	OK
2	i sdhfidgiiiisdfgisigsifigisigisifgifdgi iiiiiiifdgi	4 7 8 9 10 15 17 20 22 24 26 28 30 33 37 38 39 40 41 42 43 44 48	OK
3	dbf kfjdbfjkdkkjdbfkkdbfiuggdbfodbflk jkjklkjdbfjkdbfdbfdbf	3 12 17 24 28 40 45 48 51	OK

Таблица 2 — Результаты тестирования кода программы main2.py:

№п/п	Входные данные	Выходные данные	Комментарии
1	3 1, 4 7, 23 2, 9	([[1, 4], [7, 23], [2, 9]], 5.5)	OK
2	6 3, 1 6, 8 1, 7 9, 3 9, 6 9, 0	([[1, 7], [3, 1], [9, 0], [9, 3], [9, 6], [6, 8]], 47.5)	OK
3	5 1, 12 25, 37 3, 3 7, 35 24, 15	([[1, 12], [3, 3], [24, 15], [25, 37], [7, 35]], 559.0)	OK

Анализ полученных данных

1. График зависимости времени выполнения работы от размеров данных для алгоритма Рабина - Карпа:

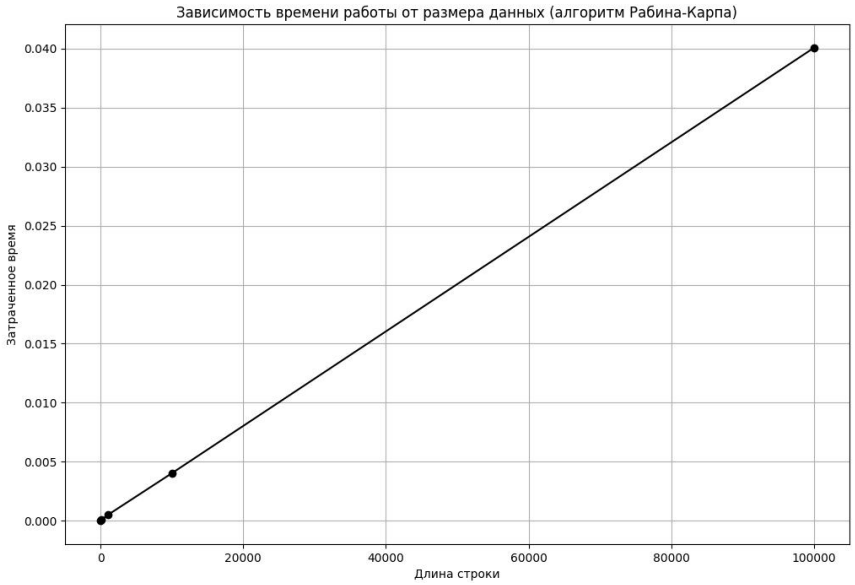
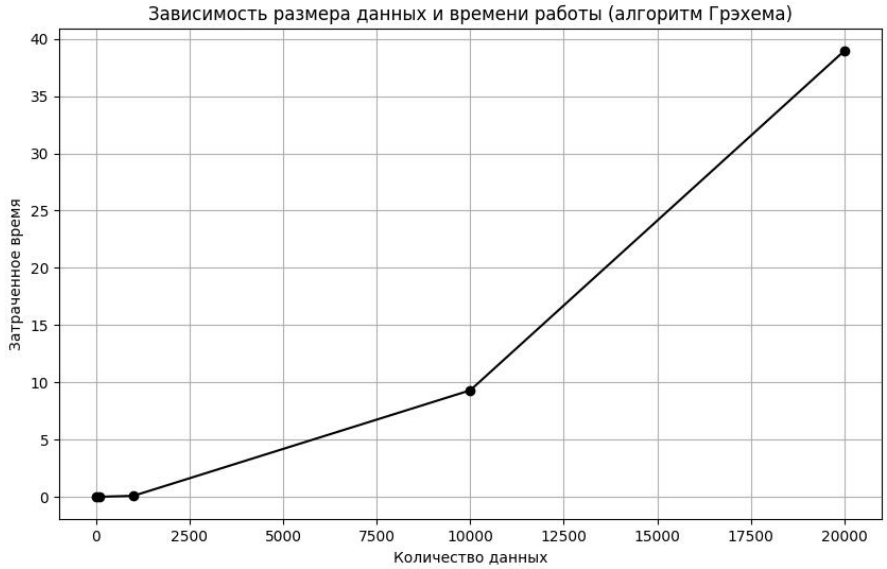


График имеет заметный линейный рост и очевидную зависимость - чем больше данных, тем больше времени затрачивает алгоритм. Но все же алгоритм тратит минимум времени для обработки больших строк (для 100 тыс символов - всего 0.04 секунды), что является хорошим показателем работы программы.

2. График зависимости времени выполнения работы от размеров данных для алгоритма Грэхема:



Данный график так же имеет очевидную зависимость - чем больше данных, тем больше времени затрачивает алгоритм. По сравнению с прошлым алгшоритмом, данный тратит гораздо больше времени для обработки всех элементов в каждом случае. Происходи это потому, что main2.py «сложнее» (выполняется в целом дольше, да и алгоритм сложнее), чем main1.py, поэтому данные совершенно разные, но в целом они и не должны быть похожими.

Выводы

В ходе лабораторной работы удалось изучить алгоритм Рабина-Карпа для поиска образца в тексте и алгоритм Грэхема для построения выпуклой оболочки и реализовать их на языке программирования Python. Также удалось провести анализ сравнения времени выполнения программ от количества операций для двух разных алгоритмов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main1.py:

```
"""Реализация алгоритма Рабина-Карпа"""

def polynomial_hash(input_string, base=26, mod=1_000_000_007):
    """Вычисление полиномиального хэша строки."""
    hash_value = 0
    for char in input_string:
        hash_value = (hash_value * base + ord(char)) % mod
    return hash_value

def cmp(str1, str2):
    """Сравнение двух строк"""
    counter = 0
    for val1, val2 in zip(str1, str2):
        if val1 == val2:
            counter += 1
        else:
            return False

    if counter == len(str1):
        return True

def get_sub_string_RK(pattern, text):
    """Алгоритм Рабина-Карпа"""
    len_pattern = len(pattern)
    len_text = len(text)
    hash_pattern = polynomial_hash(pattern)

    index = []
    for i in range(len_text-len_pattern+1):
        if polynomial_hash(text[i:i+len_pattern]) == hash_pattern:
            # посимвольное сравнение
            if cmp(text[i:i+len_pattern], pattern):
                index.append(i)
    res = [str(i) for i in index]
    return " ".join(res)

def main():
    """Головная функция"""
    pattern = input()
    text = input()
    print(get_sub_string_RK(pattern, text))

main()
```

Файл test1.py:

```
import time
import random
import string
import matplotlib.pyplot as plt
from main1 import get_sub_string_RK

def generate_random_string(length):
    return ''.join(random.choices(string.ascii_lowercase, k=length))

def data_generation_test():
```

```
    n = [10, 100, 1000, 10000, 100000]
    pattern_length = random.randint(1, 10)
    all_time = []

    for size in n:
        text = generate_random_string(size)
        pattern = generate_random_string(pattern_length)

        start = time.time()
        result = get_sub_string_RK(pattern, text)
        end = time.time()

        all_time.append(end - start)

    plt.figure(figsize=(12, 8))
    plt.plot(n, all_time, linestyle='-', color='black', marker='o')
    plt.title("Зависимость времени работы от размера данных (алгоритм Рабина-
Карпа)")
    plt.xlabel('Длина строки')
    plt.ylabel('Затраченное время')
    plt.grid(True)
    plt.show()

data_generation_test()
```

Файл main2.py:

```
"""Реализация и визуализация алгоритма Грэхема"""
import matplotlib.pyplot as plt

def rotate(elem_a, elem_b, elem_c):
    """Функция вычисления ориентации трех точек"""
    return (elem_b[0] - elem_a[0]) * (elem_c[1] - elem_b[1]) - (elem_b[1] -
elem_a[1]) * (elem_c[0] - elem_b[0])

def grahamscan(arr_point, n):
    """Алгорим Грэхема"""
    point = [int(i) for i in range(n)]
    for i in range(1, n):
        if arr_point[point[i]][0] < arr_point[point[0]][0]: # если p[i] точка
лежит левее p[0] точки
            point[i], point[0] = point[0], point[i] # меняем местами номера
этих точек

    for i in range(2, n): # сортировка вставкой
        j = i
        while j > 1 and (rotate(arr_point[point[0]], arr_point[point[j - 1]],
arr_point[point[j]]) < 0):
            point[j], point[j - 1] = point[j - 1], point[j]
            j -= 1

    stack = [point[0], point[1]] # создаем стек
    for i in range(2, n):
        while        rotate(arr_point[stack[-2]],        arr_point[stack[-1]],
arr_point[point[i]]) < 0:
            del stack[-1]
        stack.append(point[i])

    return stack

def square(arr):
```

```
    """Нахождение площади по координатам фигуры"""
    area = 0
    for i in range(len(arr)):
        x1, y1 = arr[i]
        x2, y2 = arr[(i + 1) % len(arr)]
        area += x1 * y2 - y1 * x2
    return abs(area) / 2


def visualization(arr_points_new, arr_points_old):
    """Визуализация минимальной выпуклой области, полученной с помощью
алгоритма Грэхема"""
    arr_points_new.append(arr_points_new[0])
    arr_points_old.append(arr_points_old[0])

    x_points, y_points = zip(*arr_points_new)
    x_poi_old, y_poi_old = zip(*arr_points_old)
    plt.plot(x_points, y_points, marker = 'o', linestyle = '-', color =
'black')
    plt.plot(x_poi_old, y_poi_old, marker = 'o', linestyle='None',  color =
'r')

    plt.title("Точки и выпуклая фигура")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid(True)
    plt.show()

def main():
    """Головная функция"""
    n = int(input())
    arr_points = []
    for _ in range(n):
        x, y = map(int, input().split(', '))
        arr_points.append([x, y])

    true_posled = grahamscan(arr_points, n)
    new_arr_points = []
    for i in true_posled:
        new_arr_points.append(arr_points[i])

    print((new_arr_points, square(new_arr_points)))
    visualization(new_arr_points, arr_points)

main()
```

Файл test2.py:

```
import time
import random
import pandas as pd
import matplotlib.pyplot as plt
from main2 import grahamscan

def data_generation_test():
    n = [10, 100, 1000, 10000, 20000]
    all_time = []
    for i in range(len(n)):
        start = time.time()
        arr_points = []
```

```
        for _ in range(n[i]):
            x = random.uniform(-100, 100)
            y = random.uniform(-100, 100)
            arr_points.append([x, y])

        true_posled = grahamscan(arr_points, n[i])
        new_arr_points = []
        for i in true_posled:
            new_arr_points.append(arr_points[i])

        # print((new_arr_points, square(new_arr_points)))
        # visualization(new_arr_points, arr_points)

        end = time.time()
        all_time.append(end-start)

    plt.figure(figsize=(10, 6))
    plt.plot(n, all_time, linestyle = '-', color = 'black',  marker='o')
    plt.title("Зависимость размера данных и времени работы (алгоритм
Грэхема)")
    plt.xlabel('Количество данных')
    plt.ylabel('Затраченное время')
    plt.grid(True)
    plt.show()

data_generation_test()
```