Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет прикладной информатики

**Лабораторная работа №4**

Выполнили:

Мануковская Д. М.

Сакулин И. М.

Сафронов И. С.

Проверил

Мусаев А. А.

Санкт-Петербург,

2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc181099006)

[Задание 1 4](#_Toc181099007)

[Наивный алгоритм 5](#_Toc181099008)

[Алгоритм Рабина-Карпа 6](#_Toc181099009)

[Алгоритм Бойера-Мура 7](#_Toc181099010)

[Алгоритм Кнута-Морриса-Пратта 8](#_Toc181099011)

[Сравнение алгоритмов 9](#_Toc181099012)

[ЗАКЛЮЧЕНИЕ 10](#_Toc181099013)

[СПИСОК ЛИТЕРАТУРЫ 11](#_Toc181099014)

ВВЕДЕНИЕ

В данной лабораторной работе необходимо выполнить 2 задания. В первом задача найти максимальную длину непрерывной возрастающей последовательности из массива. Второе заключается в подсчете количества наиболее часто встречающихся двузначных чисел в образовавшейся строке 4 различными алгоритмами поиска подстрок (наивный, Рабина-Карпа, Бойера-Мура, Кнута-Морриса-Пратта).

Цель работы: изучение различных алгоритмов поиска подстрок.

Задание 1

Для начала создадим массив из N случайных элементов в диапазоне от -100 до 100, используя функцию randint из библиотеки random (рисунок 1).



Рисунок 1 – Создание массива

Реализуем функцию seq для поиска максимальной непрерывной возрастающей последовательности. Её листинг представлен на Рисунок 2.

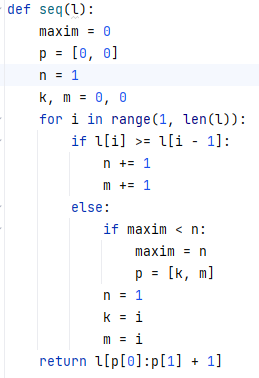


Рисунок 2 – Функция seq

Данная функция при передаче в нее массива возвращает максимальную возрастающую. последовательность.

Задание 2

Создадим функции, которые будут генерировать строки. Простые числа решетом Эратосфена, числа Фибоначчи с помощью рекурсивной формулы (Рисунок3).

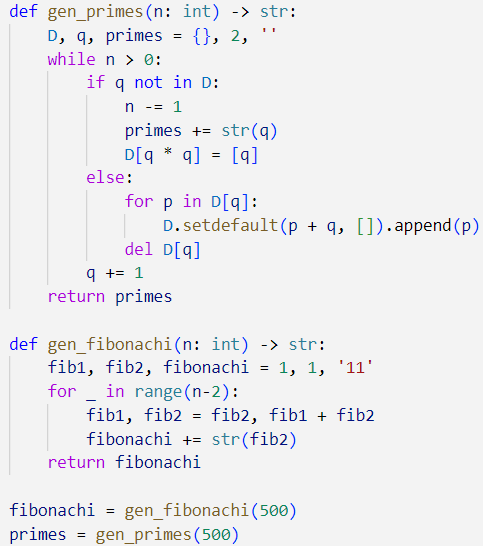


Рисунок 3 - Генерация строк

Для проверки алгоритмов ниже создадим ещё две функции. Первая, на рисунке 4, «solver», решатель, считает все вхождения каждого двузначного числа с помощью встроенной в python функции find.

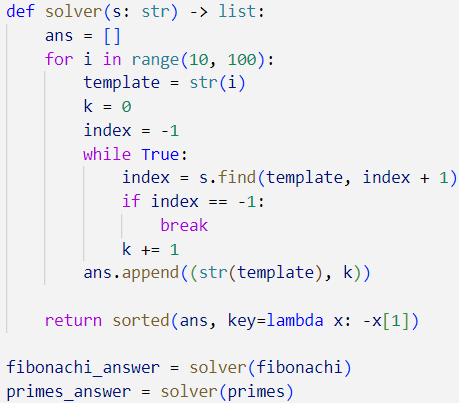


Рисунок 4 - Решатель через find

Вторая функция – «test», она запускает алгоритмы и проверяет корректность ответов, а также замеряет время исполнения (рисунок5).



Рисунок 5- Тестирующая функция

Основной смысл задания: с помощью четырёх алгоритмов найти все двузначные числа. Каждый из запрошенных алгоритмов находит определённый шаблон. Чтобы не дублировать код, опишем функцию, которая будет запускать каждый алгоритм 90 раз (рисунок 6)

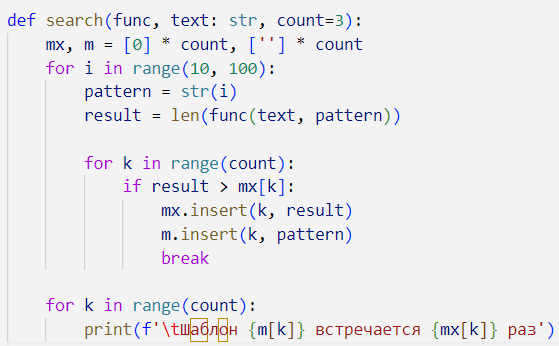


Рисунок 6 - Поиск максимальных двузначных чисел

Все алгоритмы возвращают похожий результат (рисунок 7).

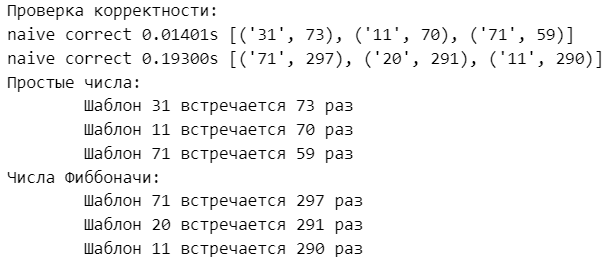


Рисунок 7 - Вывод алгоритмов

Наивный алгоритм

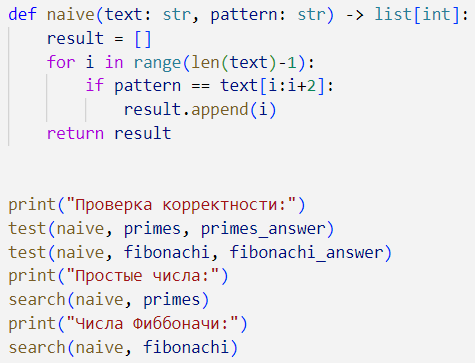


Рисунок 8 - Наивный алгоритм

Алгоритм Рабина-Карпа

Общая идея работы заключается в том, что алгоритм вычисляет хеш-значение какого-то окна строки и сравнивает его с хеш-значением искомой подстроки. Если хеш-значения совпадают, то выполняется дополнительная проверка на совпадение символов, чтобы убедиться в правильности найденной подстроки (рисунок9).

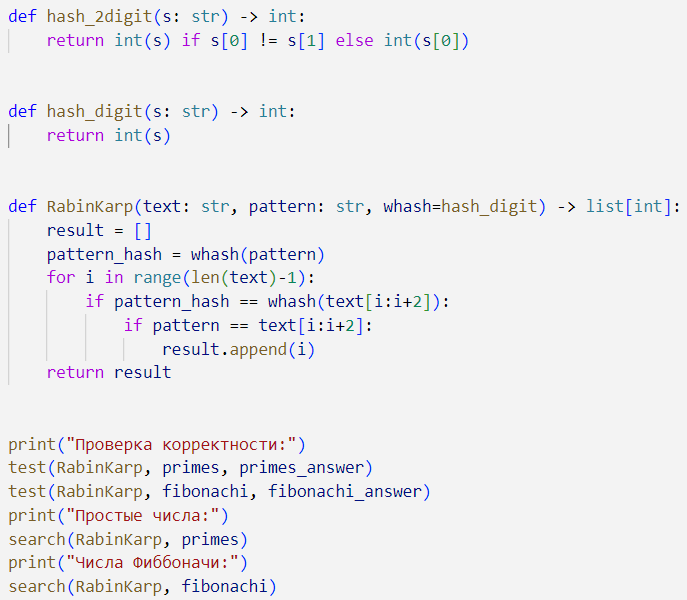


Рисунок 9 - Алгоритм Рабина-Карпа

Достоинства алгоритма:

1. В общем случае работает быстрее наивного алгоритма
2. Хорош при больших шаблонах

Недостатки алгоритма:

1. Коллизии хеш-функци
2. Большее константное значение, по сравнению с наивным алгоритмом

Алгоритм Бойера-Мура

Алгоритм Бойера-Мура имеет несколько эвристик для поиска. Реализация алгоритма с эвристикой плохого символа представлена на рисунке10.



Рисунок 10 - Алгоритм Бойера-Мура

Общая оценка вычислительной сложности современного варианта алгоритма Бойера — Мура – O(n+m), где n – длина текста, m – длина шаблона

Проверка идёт по тексту слева-направо, по шаблону справа-налево. Для прохождения по строке используются наборы правил – эвристики. Эти эвристики позволяют уменьшить количество проверок символов и, таким образом, повысить производительность алгоритма поиска.

1. Эвристика "сдвига": данный подход основан на том, что при каждом несовпадении символов в строке и подстроке существует возможность пропустить несколько символов в строке при поиске. Это позволяет значительно ускорить процесс поиска.
2. Эвристика "хороших суффиксов": эта эвристика используется для определения того, насколько можно сдвинуть подстроку вправо относительно строки, когда происходит несовпадение символов. Поиск "хороших суффиксов" помогает избежать лишних сравнений.
3. Эвристика "плохих символов": данная эвристика основана на том, что при несовпадении символов в строке и подстроке можно использовать информацию о том, где встречается последний символ, который необходимо проверить, в данной строке. Это позволяет пропустить лишние сравнения.

Достоинства алгоритма:

1. Хорошая скорость при правильной эвристике.
2. Эффективен при поиске, когда шаблона нет в строке или он встречается несколько раз.

Недостатки алгоритма:

1. Требует предварительной обработки поисковой строки
2. Высокая сложность реализации в некоторых эвристиках

Алгоритм Кнута-Морриса-Пратта

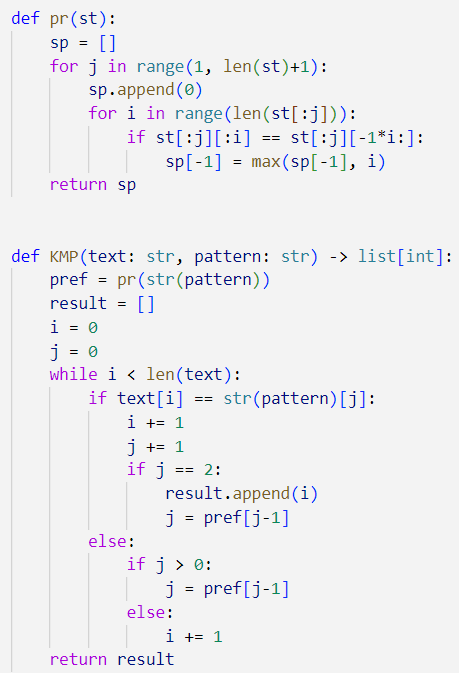


Рисунок 11 - Алгоритм Кнута — Морриса — Пратта

Сравнение алгоритмов

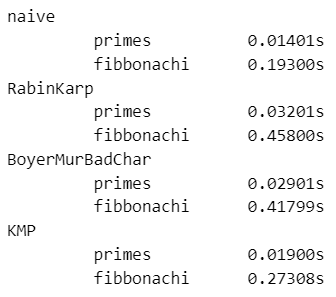


Рисунок 12 - Замеры времени исполнения

ЗАКЛЮЧЕНИЕ

Цель работы достигнута. В ходе работы научились реализовать алгоритмы поиска подстрок. Была создана программа для нахождения максимальной возрастающей подпоследовательности массива. Реализовали 4 алгоритма поиска подстрок (наивный, Рабина-Карпа, Бойера-Мура, Кнута-Морриса-Пратта) для подсчёта количества наиболее часто встречающихся двузначных чисел в образовавшейся строке. Сравнили изученные алгоритмы поиска подстрок. Сделали вывод о их достоинствах и недостатках.

СПИСОК ЛИТЕРАТУРЫ