

Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №1

Выполнили:

Шишминцев Д.В., Язев Г.А.

Проверил:

Мусаев А.А.

Санкт-Петербург

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
ПОСТАНОВКА ЗАДАЧИ.....	4
Ход работы.....	5
ЗАКЛЮЧЕНИЕ	8
СПИСОК ЛИТЕРАТУРЫ	9
ПРИЛОЖЕНИЯ	10

ВВЕДЕНИЕ

В данной работе будут представлены и реализованы различные алгоритмы поиска подстрок, а также дана оценка их эффективности.

ПОСТАНОВКА ЗАДАЧИ

Необходимо изучить и реализовать различные алгоритмы поиска подстрок (Наивный, Рабина-Карпа, Боейра-Мура и Кнута-Морриса-Пратта), а также применить их для поиска наиболее часто встречающихся двузначных чисел заданной строке и определения процента плагиата в тексте. Также необходимо сделать вывод о достоинствах и недостатках этих алгоритмов.

Ход работы

Задание 1. Поиск наиболее часто встречающихся двузначных чисел в массиве.

1.1. Наивный алгоритм

Был реализован алгоритм, основанный на последовательном сравнении шаблона с элементами строки для поиска совпадений.

Вывод: Высокая сложность алгоритма в худшем случае ($O(n^2)$ или $O(n*m)$) из-за чего выполняется крайне долго, если длина шаблона велика. Однако, он крайне прост в реализации и сохраняет приемлемую скорость при маленьком размере шаблона.

1.2. Рабина-Карпа

Был реализован алгоритм Рабина-Карпа, который основан на вычислении хэш-функции строки, по которой происходит поиск совпадений уже с помощью наивного алгоритма. Данный алгоритм является доработкой наивного алгоритма, которая может намного ускорить поиск относительно наивного алгоритма.

Вывод: Не самый шустрый алгоритм, который в худшем случае, при выборе плохой формулы для расчёта хэш-функции, имеет такую же сложность, как и у наивного алгоритма ($O(n*m)$), но в среднем выходит не самая плохая сложность ($O(n+m)$). Также можно отметить, что можно подобрать такие входные данные, что на этапе хэширования будут недопустимо большие числа, а количество «холостых» срабатываний крайне велико.

1.3. Боейра-Мура

Был реализован алгоритм Боейра-Мура, выполняющий поиск совпадений путём сдвига шаблона по строке на свою длину или до первого совпадающего символа, проверяя наличие совпадающих символов в шаблоне и строке, на основе которых и совершается сдвиг. Данный алгоритм после проведения некоторых вычислений над

шаблоном, сравнивает шаблон со строкой не во всех позициях, пропуская часть проверок, которые не дали бы результата.

Вывод: Данный алгоритм не только позволяет улучшить обработку самого плохого случая, но и неплохо справляется в среднем. Сложность алгоритма зависит от мощности алфавита и в среднем равна $O(n+m)$ (если используется таблица стоп-символов). Поскольку вычисления изначально ведутся над шаблоном, а не строкой, такой алгоритм оптимален, когда нет возможности провести обработку строки, в которой проходит поиск. Из минусов можно отметить то, что данный алгоритм может требовать много памяти, если алфавит будет слишком большой. Так, при каждом несовпадении двух символов строки и шаблона, шаблон сдвигается на всё пройденное расстояние.

1.4. Кнута-Морриса-Пратта

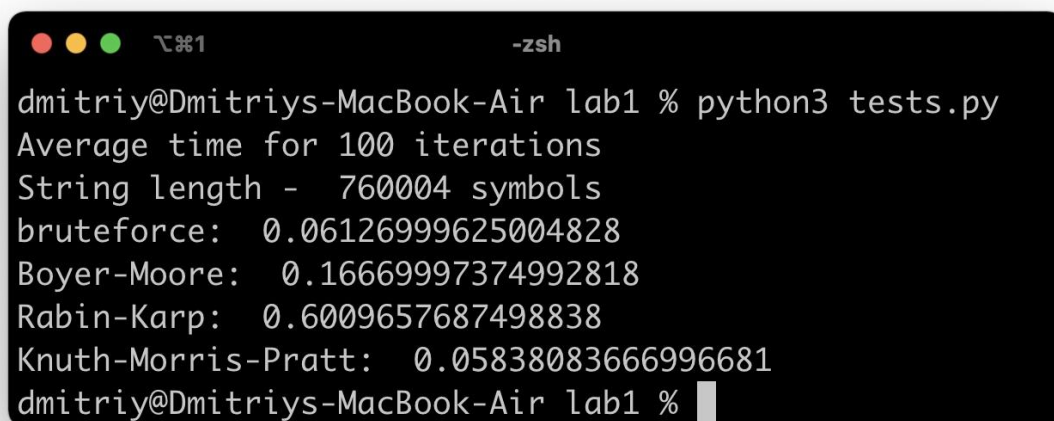
Был реализован алгоритм Кнута-Морриса-Пратта, основывающийся на использовании префикс-функции, благодаря которой высчитывается наиболее выгодная длина пропуска при несовпадении.

Вывод: У данного алгоритма одна из наименьших сложностей ($O(P+T)$ P - сложность префикс функции, T – сложность самой функции), которая в идеальных обстоятельствах может принимать линейные значения. Поскольку алгоритм довольно сложен в понимании и исполнении, выигрыш от его использования на небольших текстах крайне незначителен, также как и в текстах с большим алфавитом, где количество совпадений будет гораздо меньше количества несовпадений. Также данный алгоритм будет эффективен в строках с большим количеством повторяющихся символов.

1.5. Вывод

Анализируя особенности и сложности алгоритмов, был сделан вывод, что наиболее эффективным и довольно универсальным алгоритмом можно считать алгоритм Кнута-Морриса-Пратта, который будет

наиболее эффективен для поиска совпадений в строках большого размера с большим количеством повторяющихся символов. Ниже представлено время выполнения всех алгоритмов на произвольной строке из 500 чисел:

A terminal window with a dark background and light-colored text. The window title bar shows three colored circles (red, yellow, green) and the text "zsh". The terminal content shows a user running a Python script. The output lists the average time for 100 iterations for four different algorithms on a string of 760,004 symbols. The results are: brute force (0.06126999625004828), Boyer-Moore (0.16669997374992818), Rabin-Karp (0.6009657687498838), and Knuth-Morris-Pratt (0.05838083666996681). The prompt shows the user is at the "lab1" directory.

```
dmitriy@Dmitriys-MacBook-Air lab1 % python3 tests.py
Average time for 100 iterations
String length - 760004 symbols
bruteforce: 0.06126999625004828
Boyer-Moore: 0.16669997374992818
Rabin-Karp: 0.6009657687498838
Knuth-Morris-Pratt: 0.05838083666996681
dmitriy@Dmitriys-MacBook-Air lab1 %
```

Рисунок 1. Результаты выполнения алгоритмов

Задание 2. Алгоритм для поиска процента плагиата в тексте.

В данном задании описан алгоритм, определяющий оригинальность текста, сравнивая его с текстом статьи на сайте Wikipedia. Данное задание было реализовано за счёт алгоритма Кнута-Морриса-Пратта, поскольку по результатам Задания 1 этот алгоритм оказался наиболее эффективным для больших текстов с маленьким алфавитом (Частыми повторениями символов).

Вывод: Была реализована программа-антиплагиат, за основу которой был выбран алгоритм Кнута-Морриса-Пратта.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены различные алгоритмы для поиска подстрок и сделаны выводы об их эффективности. Также были написаны программы, на языке Python, реализующие эти алгоритмы.

СПИСОК ЛИТЕРАТУРЫ

1. Хабр. Алгоритмы поиска по строке. [Электронный ресурс].
[Сайт]URL: <https://clck.ru/33bBiQ>

ПРИЛОЖЕНИЯ

A. [I7uBO3ABP/lab1 \(github.com\)](https://github.com/I7uBO3ABP/lab1)