**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Алгоритм Кнута-Морриса-Пратта

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Усачева Д.В. |
| Преподаватель |  | Шевелева А. М. |

Санкт-Петербург

2023

## Цель работы.

Изучить реализацию эффективного алгоритма поиска подстрок в

строке – алгоритма Кнута-Морриса-Пратта. На его основе решить два задания.

## Задание 1.

Реализуйте алгоритм КМП с его помощью для заданных шаблона P (|P| ≤ 15000) и текста T (|T| ≤ 5000000) найдите все вхождения P в T.

Вход:

Первая строка – P;

Вторая строка – T.

Выход:

Индексы начал вхождений P в T, разделенных запятой, если P не входит T, то вывести -1.

Sample Input:

ab

abab

Sample Output:

0,2

## Задание 2.

Заданы две строки A (|A| ≤ 5000000) и B (|B| ≤ 5000000). Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка – A;

Вторая строка – B.

Выход:

Если A циклическим сдвигом B, индекс начала строки B в А, иначе вывести -1. Если возможны несколько сдвигов, вывести первый индекс.

Sample Input:

defabc

abcdef

Sample Output:

3

## Выполнение работы.

Для решения обоих задач была реализована функция prefix\_function, на вход которого принимается строка в этой функции список prefix заполняется значениями длины максимального префикса для каждого символа исходной строки. Первый элемент списка prefix всегда равен нулю, так как нет префиксов и суффиксов для первого символа.

Для решения первой задачи была написана программа на языке C++, результатом выполнения которой является вывод всех вхождений подстроки в строку, найденных по алгоритму Кнута-Морриса-Пратта.

Была реализована функция task1, представляющая собой реализацию алгоритма Кнута-Морриса-Пратта. Она вычисляет значение префикс функции для строки (first\_string + '#' + second\_string, где first\_string – искомая строка, second\_string – строка в которой ведется поиск вхождений, # – символ, не входящий в алфавит двух строк). Далее цикл от начала second\_string в массиве префиксов и до его конца. Если на текущей итерации максимальная длина префикса равна длине first\_string, значит, перед нами вхождение first\_string в second\_string.

Также для этой задачи была реализована функция Print, обеспечивающая корректный вывод для результата, полученного в task1.

Для решения второй задачи была написана программа на языке C++, которая определяет, является ли строка циклическим сдвигом другой строки.

Была реализована функция task2. Она вычисляет значение префикс функции для строки (first\_string + '#' + second\_string + second\_string) если длины строк first\_string и second\_string одинаковы. Далее идет поиск первого вхождения строки first\_string в second\_string. Если в массиве префиксов обнаружена максимальная длина префикса равная длине first\_string, то выводится индекс вхождения. Если вхождения не было найдено или длины строк не совпадают, функция возвращает -1.

Разработанный программный код см. в приложении А.

## Выводы.

В ходе проделанной работы был изучен один из алгоритмов эффективного поиска подстроки в строке - алгоритм Кнута-Морриса-Пратта. Для его реализации была написана функция на C++, вычисляющая значение максимальных длин префиксов для каждого символа. Данный алгоритм был применен для решения двух задач:

* поиск вхождений строки в тексте (задание 1);
* определение циклического сдвига одной строки во второй строки, при наличии такого сдвига определить на индекс начала второй строки в первой (задание 2).

Данные программы были успешно протестированы на тестах в курсе Stepik.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lb4.cpp

#include <iostream>

#include <vector>

#include <string>

'''

Функция считает значение префикс-функции для строки

Возвращаемый список prefix - список элементы которого содержат

информацию о маскимальной длине префикса для каждого сивола.

'''

std::vector<int> prefix\_function(std::string& string) {

std::vector<int> prefix(string.length(), 0);

for (int i = 1; i < string.length(); i++) {

int j = prefix[i - 1];

while (j > 0 && string[i] != string[j])

j = prefix[j - 1];

if (string[i] == string[j]) {

prefix[i] = j + 1;

}

else

prefix[i] = j;

}

return prefix;

}

'''

Функция для корректного вывода результата задания 1

если позиций, начиная с которых идет строка first\_string в строке second\_string нет, то выводится -1

'''

void Print(std::vector<int> result) {

if (result.size() == 0)

std::cout << -1;

else {

for (int r = 0; r < result.size() - 1; r++) {

std::cout << result[r] << ',';

if (r == result.size() - 2)

std::cout << result[r + 1];

}

}

std::cout << "\n";

}

'''

Функция, которая реализует алгоритм КМП

result - вектор позиций, начиная с которых идет строка first\_string в строке second\_string

'''

void task1(std::vector<int> prefix, int len\_first\_string, int len\_second\_string) {

std::vector<int> result;

for (int i = 0; i < len\_second\_string; i++) {

if (prefix[i + len\_first\_string + 1] == len\_first\_string)

result.push\_back(i - len\_first\_string + 1);

}

Print(result);

}

'''

Функция, которая определяет, является ли строка first\_string циклическим сдвигом second\_string

Результатом выполнения являеется число, указывающее на сколько символов вправо был совершен сдвиг

Если строка first\_string не являеется циклическим сдвигом second\_string, выводится -1

'''

int task2(std::vector<int> prefix, int len\_first\_string, int len\_second\_string) {

if (len\_first\_string != len\_second\_string) {

std::cout << "-1" << "\n";

return 0;

}

for (int i = 0; i < 2 \* len\_first\_string; i++) {

if (prefix[i + len\_first\_string + 1] == len\_first\_string) {

std::cout << i - len\_first\_string + 1 << "\n";

return 0;

}

}

std::cout << "-1" << "\n";

return 0;

}

int main() {

std::string first\_string, second\_string, string\_sum\_task1, string\_sum\_task2;

std::cin >> first\_string >> second\_string;

string\_sum\_task1 = first\_string + '#' + second\_string;

string\_sum\_task2 = first\_string + '#' + second\_string + second\_string;

task1(prefix\_function(string\_sum\_task1), first\_string.length(), second\_string.length());

task2(prefix\_function(string\_sum\_task2), first\_string.length(), second\_string.length());

return 0;

}