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. 7383 _____ Медведев И. С.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Содержание

Цель работы

Ознакомиться с алгоритмом нахождения подстроки в строке Кнута-Морриса-Пратта, написать программу для поиска всех вхождений подстроки в строке.

Реализация задачи

Для решения поставленной задачи был использован класс kmp. В данном классе содержатся поля: string pattern – шаблон, вхождения которого мы ищем, vector<size_t> prefix – вектор в котором хранятся значения префикс функций, size_t last_prefix – переменная, в которой хранится значение префикс функции для последнего рассмотренного символа. Так же в классе kmp объявлена friend int Shift (string A, string B) – функция, которая рассматривает две строки, и если одну из них можно получить циклическим сдвигом, то функция вернет количество сдвигов вправо, если нет, то функция вернет -1. Данная функция склеивает две строки и вычисляет значения префикс функции, тем самым последнее значение в векторе со значениями префикс функции будет показывать сколько сдвигов надо сделать (последние *n* символов второй строки совпали с первыми n символами первой строки).

Так же класс kmp содержит конструктор, который на вход принимает строку-шаблон и проходится по всем символам и задает значения префикс функции с помощью метода set_prefix. Метод bool set_prefix принимает на вход символ рассматриваемой строки и сравнивает ее с символом шаблона, имеющий индекс last_prefix. Если они равны, то увеличивает значение last_prefix, иначе присваивает переменной last_prefix значение prefix[last_prefix-1] пока она не станет равна нулю или символ шаблона, имеющий индекс last_prefix, не станет равен рассматриваемому символу. Возвращает true если last_prefix равен длине шаблона.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Сложность алгоритма

В начале метод kmp задает значения префикс функции для шаблона. Он проходит по всем символам один раз, следовательно, временная сложность O(|N|), где N – количество символов шаблона. Затем мы проходимся по всем символам текста, вычисляя значения префикс функции, следовательно, временная сложность O(|M|), где M – количество символов текста. Значит общая временная сложность алгоритма O(|M| + |N|).

Память в данном алгоритме выделяется под значения префикс функции шаблона, поэтому по памяти сложность алгоритма можно оценить, как O(|N|), где N – количество символов шаблона.

Вывод

В процессе выполнения данной лабораторной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта поиска подстроки в строке с помощью префикс-функции. Так же с использованием этого алгоритма реализована программы, с помощью префикс-функции проверяющей, является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <string>
#include <iostream>
#include <vector>
#include <cstring>

using namespace std;

class kmp {
    friend int shift(string A, string B);
    string pattern;
    vector<size_t> prefix;
    size_t last_prefix;

public:
    kmp(string tmp) :pattern(tmp), prefix(pattern.length()) {
        last_prefix = prefix[0] = 0;
        for (size_t i=1; i<pattern.length(); i++) {
            set_prefix(pattern[i]);
            prefix[i] = last_prefix;

        }
        last_prefix = 0;
    }

    bool set_prefix(char temp) {
        while (last_prefix && pattern[last_prefix] != temp)
            last_prefix = prefix[last_prefix-1];

        if (pattern[last_prefix] == temp)
            last_prefix++;

        return last_prefix == pattern.length();
    }


};

int shift(string A, string B) {
    if (A == B)
        return 0;
    kmp e(A + "$" + B);
    size_t index = e.prefix[e.prefix.size() - 1];
    for(size_t i = index; i < A.size(); i++){
        if(A[i] != B[i - index])
            return -1;
    }
    return index;
}

int main() {

    string pattern = "";
    string text = "";
    string ans = "";
    cin >> pattern >> text;
    kmp tmp(pattern);
    for (size_t i = 0; i<text.length(); i++) {
```

```
        if (tmp.set_prefix(text[i])) {
            if(!ans.empty())
                ans += ',';

            ans += to_string(i + 1 - pattern.length());
        }

    }
    if (ans == ""){
        cout << "-1" <<endl;
        return 0;
        }

     cout << ans << endl;
     return 0;
}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

Ввод	Вывод
aaa aaaaaaa	0,1,2,3,4
ilo iloveanimeiloiilo	0,10,14
qqwwrqwrr adasdasd	-1
ab abab	0,2
qwertyuiop qwertyuiopasdfghjkl;qwertyuioprfvt5gbyhnujmerwtgwqwertyuiop	0,20,49