

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: Кнут-Моррис-Пратт

Студентка гр. 9383

Лапина А.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта, реализовать с его помощью 2 программы, первая из которых ищет все вхождения подстроки в строке, а вторая проверяет является ли одна строка циклическим сдвигом другой.

Основные теоретические положения.

Префикс строки – это какое-то количество, не равное длине всей строки, начальных символов строки.

Суффикс строки – это какое-то количество, не равное длине всей строки, символов конца строки.

Префикс-функция – это массив чисел, вычисляемый, как наибольшая длина суффикса, совпадающего с ее префиксом.

Алгоритм Кнута-Морриса-Пратта – эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объема входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно. Алгоритм был разработан Д. Кнутом и В. Праттом и, независимо от них, Д.Моррисом в 1977 году.

Задание.

1) Реализуйте алгоритм КМП и с его помощью для заданных шаблона Р ($|P| \leq 15000$) и текста Т ($|T| \leq 5000000$) найдите все вхождения Р в Т.

Вход:

Первая строка – Р

Вторая строка – Т

Выход:

Индексы начал вхождения Р в Т, разделенных запятой, если Р не входит в Т, то вывести -1.

Пример входных данных:

ab

abab

Пример выходных данных:

0,2

2) Заданы две строки А и В. Определить, является ли А циклическим сдвигом В (это значит, что А и В имеют одинаковую длину и А состоит из суффикса В, склеенного с префиксом В).

Вход:

Первая строка – А

Вторая строка – В

Выход:

Если А является циклическим сдвигом В, индекс начала строки В в А, иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Пример входных данных:

defabc

abcdef

Пример выходных данных:

3

Выполнение работы:

В программе реализованы 2 основные функции для реализации алгоритма КМП:

1) create_pi(std::string P) — принимает строку P, создает массив pi_, который хранит в себе значения префикс-функции для переданной строки.

Описание работы:

1) Изначально заполняем массив 0;

2) i — указывает на текущий символ, а с помощью j будем искать значение префикс-функции;

3) Пока не просмотрели всю строку (то есть пока i != <длине исходной строки>) будем проверять 1 условие - равны ли P[i] и P[j]:

*если да, то pi[i] = j+1; увеличиваем i, j;

*если нет, то проверяем условие j==0:

****если да, то записываем $pi_i = 0$; увеличиваем i ;**

****если нет, то $j = pi_j - 1$;**

В конце функция возвращает искомый массив $pi_$.

II) $Alg_KMP(std::string T, std::string P, std::vector<int> pi_)$ - ищет подстроку P в строке T , используя алгоритм КМП.

Описание работы:

1) задаем 2 счетчика $k = 0$ для строки T (в которой ищем) и $l = 0$ для строки P (которую ищем), также создаем вектор $result$ для записи результата.

2) Пока не закончилась строка T проверяем условие равенства символов в строке ($T[k] == P[l]$):

***если да, то увеличиваем k и l , если $l =$ длине строки P (значит вся подстрока P входит в строку T), то добавляем индекс, с которого строки равны в $result$;**

***если нет, то проверяем условие ($l == 0$):**

****если да, то увеличиваем k и проверяем, что строка T (в которой ищем подстроку) не закончилась и , если закончилась, то выходим из цикла**

****если нет, то $l = pi_l - 1$;**

Когда уже вся строка T просмотрена, функция возвращает $result$ — вектор с ответом (индексы начала подстроки в строке).

Алгоритм для 1 задания (поиска всех вхождений подстроки P в строку T):

1) Считываем входные данные — строки P, T ;

2) Проверяем, что длина $T >$ длины P , иначе сразу выводим -1 и завершаем выполнение программы;

3) Считаем префикс-функцию для строки P , с помощью вышеописанной функции $create_pi$;

4) Вызываем функцию Alg_KMP для поиска подстроки в строке.

5) Если алгоритм нашел вхождения, то есть вектор $result$ — не пустой, то выводим его содержимое, иначе выводим «-1».

Алгоритм для 2 задания (определение является ли A циклическим сдвигом B):

- 1) Считываем входные данные — строки A и B;
- 2) Удваиваем строку A и реализуем с помощью этого алгоритм КМП, где A будет строкой в которой ищем B, а B, соответственно, будет являться подстрокой.
- 3) Считаем префикс-функцию для строки B, с помощью вышеописанной функции create_pi;
- 4) Вызываем функцию Alg_KMP для поиска подстроки в строке.
- 5) Далее нужно проверить, что сдвиг был найден, то есть result не пуст и при этом нужно проверить, что длина исходных строк была равна, тогда выводим 1 значение, хранящееся в векторе ответа, так как по условию задания необходимо вывести наименьший (в нашем случае первый найденный) сдвиг, если условие не выполнено выводим «-1».

Тестирование

Для задания 1:

- 1) Входные данные:

ab

abab

Выходные данные:

0,2

- 2) Входные данные:

qwer

asdfqwerjhq

Выходные данные:

4

- 3) Входные данные:

HELLO

wHELLOrrrHELbbfHELLOHELLO

Выходные данные:

1,15,20

4) Входные данные:

YES

YEhhtrSEYsvsYYYEFSYS

Выходные данные:

-1

Для задания 2:

1) Входные данные:

defabc

abcdef

Выходные данные:

3

2) Входные данные:

qweqwe

qweqwe

Выходные данные:

0

3) Входные данные:

abcdef

bcdefga

Выходные данные:

-1

4) Входные данные:

baaaaa

abaaaa

Выходные данные:

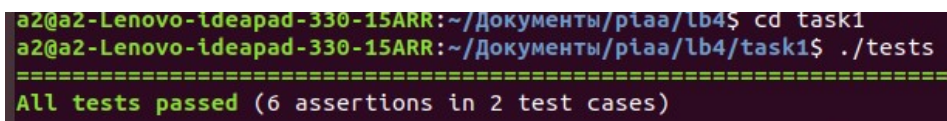
5

Также были реализованы тесты:

Для 1 задания:

- 1) Для проверки работы функции `create_pi` — задается строку `P`, вызываем тестируемую функцию и сравниваем возвращаемое значение.
- 2) Реализован тест для функции `Alg_KMP`— которая ищет индексы вхождений подстроки `P` в строку `T`. Проверяем возвращаемое значение.

Результаты работы написанных тестов предоставлены на рисунке 1



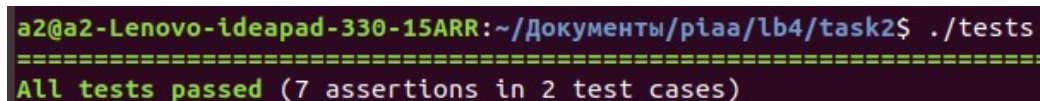
```
a2@a2-Lenovo-ideapad-330-15ARR:~/Документы/p1aa/lb4$ cd task1
a2@a2-Lenovo-ideapad-330-15ARR:~/Документы/p1aa/lb4/task1$ ./tests
=====
All tests passed (6 assertions in 2 test cases)
```

Рисунок 1 — Результаты тестов программы

Для 2 задания:

- 1) Для проверки работы функции `create_pi` — задается строку `B`, вызываем тестируемую функцию и сравниваем возвращаемое значение.
- 2) Реализован тест для функции `Alg_KMP`— которая ищет индекс начала циклического сдвига (если `A` таковым является для `B`). Рассмотрены случаи возвращения пустого вектора - если `A` не является циклическим сдвигом, а также случаи, когда найден 1 и более вариантов сдвига.

Результаты работы написанных тестов предоставлены на рисунке 2



```
a2@a2-Lenovo-ideapad-330-15ARR:~/Документы/p1aa/lb4/task2$ ./tests
=====
All tests passed (7 assertions in 2 test cases)
```

Рисунок 2 — Результаты тестов программы

Разработанный программный код см. в приложении А.

Вывод.

В процессе выполнения лабораторной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта. С помощью данного алгоритма построены программы для поиска всех вхождений подстроки в строку и для определения циклического сдвига двух строк.

Приложение А

Исходный код программы

Задание 1:

Название файла: main.cpp

```
#include "lb4_1.hpp"

int main(){
    std::string P, T;
    std::cin >> P;
    std::cin >> T;
    if (P.length() > T.length()){
        std::cout << "-1";
        return 0;
    }

    std::vector <int> pi_ = create_pi(P);
    std::vector <int> result = Alg_KMP(T, P, pi_);
    if (result.empty())
        std::cout << "-1";
    else{
        for (int i = 0; i != result.size(); i++){
            if (i == result.size()-1)
                std::cout << result[i];
            else
                std::cout << result[i] << ",";
        }
    }
    return 0;
}
```

Название файла: lb4_1.cpp

```
#include "lb4_1.hpp"

std::vector <int> create_pi(std::string P){
    std::vector <int> pi_;
    for (int i = 0; i != P.length(); i++){
        pi_.push_back(0);
    }
    int i = 1;
    int j = 0;
    while (i != P.length()){
        if (P[i] == P[j]){
            pi_[i] = j+1;
            i++;
            j++;
        }
        else{
            if (j == 0){
                pi_[i] = 0;
                i++;
            }
            else{
                j = pi_[j-1];
            }
        }
    }
}
```

```

        }
    }
    return pi_;
}

std::vector<int> Alg_KMP(std::string T, std::string P, std::vector<int> pi_){
    int k = 0;
    int l = 0;
    std::vector<int> result;
    while (k!=T.length()){
        if (T[k] == P[l]){
            k++;
            l++;
            if (l==P.length()){
                result.push_back(k - P.length());
            }
        }
        else{
            if(l==0){
                k++;
                if (k == T.length()){
                    break;
                }
            }
            else{
                l = pi_[l-1];
            }
        }
    }
    return result;
}

```

Название файла: lb4_1.hpp

```

#include <iostream>
#include <vector>
#include <cstring>

std::vector<int> create_pi(std::string P);
std::vector<int> Alg_KMP(std::string T, std::string P, std::vector<int> pi_);

```

Название файла: tests_programm.cpp

```

#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include "../source/lb4_1.hpp"

TEST_CASE("Тест для функции create_pi") {

    SECTION("1 случай") {
        std::string P = "ab";
        std::vector<int> check;
    }
}

```

```

    check.push_back(0);
    check.push_back(0);
    CHECK(create_pi(P) == check);
}

SECTION("2 случай") {
    std::string P = "abbaabbab";
    std::vector<int> check;
    check.push_back(0);
    check.push_back(0);
    check.push_back(0);
    check.push_back(1);
    check.push_back(1);
    check.push_back(2);
    check.push_back(3);
    check.push_back(4);
    check.push_back(2);
    CHECK(create_pi(P) == check);
}

SECTION("3 случай") {
    std::string P = "efefeftef";
    std::vector<int> check;
    check.push_back(0);
    check.push_back(0);
    check.push_back(1);
    check.push_back(2);
    check.push_back(3);
    check.push_back(4);
    check.push_back(0);
    check.push_back(1);
    check.push_back(2);
    CHECK(create_pi(P) == check);
}

TEST_CASE("Тест для функции Alg_KMP") {

    SECTION("1 случай") {
        std::string P = "ab";
        std::string T = "abab";
        std::vector<int> pi_ = create_pi(P);
        std::vector<int> check;
        check.push_back(0);
        check.push_back(2);
        CHECK(Alg_KMP(T, P, pi_) == check);
    }

    SECTION("2 случай") {
        std::string P = "ab";
        std::string T = "dasabdbab";
        std::vector<int> pi_ = create_pi(P);
        std::vector<int> check;
        check.push_back(3);
        check.push_back(6);
        CHECK(Alg_KMP(T, P, pi_) == check);
    }
}

```

```

SECTION("3 случай") { //когда подстрока не найдена
    std::string P = "qwe";
    std::string T = "ababqknwegkrqe";
    std::vector<int> pi_ = create_pi(P);
    std::vector<int> check;
    CHECK(Alg_KMP(T, P, pi_) == check);
}

}

```

Задание 2:

Название файла: main.cpp

```
#include "lb4_2.hpp"
```

```

int main(){
    std::string A, B;
    std::cin >> A;
    std::cin >> B;

    A = A+A;

    std::vector<int> pi_ = create_pi(B);
    std::vector<int> result = Alg_KMP(A, B, pi_);
    if (result.empty() || (2 * B.length() != A.length()))
        std::cout << "-1\n";
    else{
        std::cout << result[0] << "\n";
    }
    return 0;
}

```

Название файла: lb4_2.cpp

```
#include "lb4_2.hpp"
```

```

std::vector<int> create_pi(std::string B){
    std::vector<int> pi_;
    for (int i = 0; i != B.length(); i++){
        pi_.push_back(0);
    }
    int i = 1;
    int j = 0;
    while (i != B.length()){
        if (B[i] == B[j]){
            pi_[i] = j+1;
            i++;
            j++;
        }
        else{
            if (j==0){
                pi_[i] = 0;
                i++;
            }
        }
    }
}

```

```

        else{
            j = pi_[j-1];
        }
    }
}
return pi_;
}

std::vector <int> Alg_KMP(std::string A, std::string B, std::vector
<int> pi_){
    int k = 0;
    int l = 0;
    std::vector <int> result;
    while (k!=A.length()){
        if (A[k] == B[l]){
            k++;
            l++;
            if (l==B.length()){
                result.push_back(k - B.length());
                break;
            }
        }
        else{
            if(l==0){
                k++;
                if (k == A.length()){
                    break;
                }
            }
            else{
                l = pi_[l-1];
            }
        }
    }
    return result;
}

```

Название файла: lb4_2.hpp

```

#include <iostream>
#include <vector>
#include <cstring>

```

```

std::vector <int> create_pi(std::string B);
std::vector <int> Alg_KMP(std::string A, std::string B, std::vector
<int> pi_);

```

Название файла: tests_programm.cpp

```

#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include "../source/lb4_2.hpp"

```

```

TEST_CASE("Тест для функции create_pi") {

```

```

SECTION("1 случай") {
    std::string P = "ab";
    std::vector<int> check;
    check.push_back(0);
    check.push_back(0);
    CHECK(create_pi(P) == check);
}

SECTION("2 случай") {
    std::string P = "abbaabbab";
    std::vector<int> check;
    check.push_back(0);
    check.push_back(0);
    check.push_back(0);
    check.push_back(1);
    check.push_back(1);
    check.push_back(2);
    check.push_back(3);
    check.push_back(4);
    check.push_back(2);
    CHECK(create_pi(P) == check);
}

SECTION("3 случай") {
    std::string P = "efefeftef";
    std::vector<int> check;
    check.push_back(0);
    check.push_back(0);
    check.push_back(1);
    check.push_back(2);
    check.push_back(3);
    check.push_back(4);
    check.push_back(0);
    check.push_back(1);
    check.push_back(2);
    CHECK(create_pi(P) == check);
}

}

TEST_CASE("Тест для функции Alg_KMP") {

    SECTION("1 случай") {
        std::string A = "defabc";
        std::string B = "abcdef";
        A = A+A;
        std::vector<int> pi_ = create_pi(B);
        std::vector<int> check;
        check.push_back(3);
        CHECK(Alg_KMP(A, B, pi_) == check);
    }

    SECTION("1 случай") {
        std::string A = "yqwert";
        std::string B = "qwerty";
        A = A+A;
        std::vector<int> pi_ = create_pi(B);
        std::vector<int> check;
        check.push_back(1);
    }
}

```

```

CHECK(Alg_KMP(A, B, pi_) == check);
}

SECTION("3 случай") { //когда найдено несколько вариантов
    std::string P = "abcdabc";
    std::string T = "abcdabc";
    std::vector<int> pi_ = create_pi(P);
    std::vector<int> check;
    check.push_back(0);
    CHECK(Alg_KMP(T, P, pi_) == check);
}

SECTION("4 случай") { //когда A не является циклическим сдвигом B
    std::string P = "qweabc";
    std::string T = "abcaew";
    std::vector<int> pi_ = create_pi(P);
    std::vector<int> check;
    CHECK(Alg_KMP(T, P, pi_) == check);
}
}

```