

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8303

Логинов Е.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучить алгоритм Кнута-Морриса-Практа поиска подстроки в строке, а также реализовать данный алгоритм на языке программирования C++.

Вариант 2. Оптимизация по памяти: программа должна требовать $O(m)$ памяти, где m - длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

Постановка задачи.

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

Входные данные

Первая строка – P .

Вторая строка – T .

Выходные данные

Индексы начал вхождений P в T , разделенных запятой, если P не входит

в T , то вывести -1 .

Пример входных данных

ab

abab

Соответствующие выходные данные

0, 2

2. Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$). 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.

Входные данные

Первая строка – A.

Вторая строка – B.

Выходные данные

Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести –1. Если возможно несколько сдвигов вывести первый индекс.

Пример входных данных

defabc

abcdef

Соответствующие выходные данные

3

Описание алгоритмов.

1) Префикс-функция

Префикс функция `std::vector<int> prefix(std::string str)` принимает на вход строку-образец и определяет наибольшую длину префикса, который одновременно является суффиксом для этой подстроки. Таким образом, в начале подстроки длины нужно найти такой префикс максимальной длины, который был бы суффиксом данной подстроки.

В программе префикс-функция для очередного символа рассчитывается при помощи предыдущего. Например, если индекс символа совпадает со значением префикс-функции и текущие символы строк равны, то префикс-функция для текущего символа вычисляется как предыдущее значение префикс-функции + 1. Если же символы не равны, то рассматривается префикс предыдущего символа.

2) Алгоритм Кнута-Морриса-Пратта

Функция `std::vector<int> KMP(std::string str, std::string substr)` принимает на вход строку и строку-образец, вхождение которой необходимо найти. Инициализируются два счётчика для строки и подстроки. Далее, пока не будет достигнут конец строки, выполняется сравнением символов строки и подстроки. Если символы равны, то их индексы увеличиваются, и функция переходит к сравнению следующих символов. В случае если символы не равны и индекс подстроки не указывает на его начало, то новый индекс образа вычисляется при помощи префикс-функции. Если же символы не равны и индекс подстроки указывает на начало, то индекс строки увеличивается на 1.

После завершения сравнения строки и подстроки функция возвращает массив контейнер `r` с индексами начала вхождений подстроки в строку.

3) Алгоритм определения циклического сдвига

Алгоритм `int cycleShift(std::string str1, std::string str2)` получает на вход две строки, циклический сдвиг которых необходимо определить.

Инициализируются два счётчика для двух строк, происходит проверка на равенство длины строк. Если длины отличаются, то программа возвращает -1. Начинается посимвольное сравнение двух строк, если символы совпадают, то счётчики увеличиваются и выполняется переход к следующим символам. Если же совпадений еще не было, то счётчик для второй строки уменьшается.

Сдвиг будет найден в том случае, когда счётчик второй строки равен длине этой строки. Если счётчик первой строки равен длине этой строки, то происходит его обнуление и поиск происходит дальше. В результате функция возвращает индекс начала одной строки в другой строке.

Сложность алгоритмов.

1) Алгоритм Кнутта-Морриса-Пратта

Сложность по времени: $O(m + n)$, где m – длина подстроки, n – длина строки.

Сложность по памяти: $O(m)$, где m – длина подстроки. Это возможно, если не учитывать память, в которой хранится строка поиска.

2) Алгоритм определения циклического сдвига

Сложность по времени: $O(n+n) = O(n)$, где n – длина строки

Сложность по памяти: $O(2n) = O(n)$, где n – длина строки

Описание функций.

`std::vector<int> prefix(std::string substr) { return p }` – функция для вычисления префикс-функции. Получает на вход обрабатываемую строку, возвращает вектор значений p для обрабатываемой строки. Используется в обоих алгоритмах.

`std::vector<int> KMP(std::string str, std::string substr){ return result }` – функция поиска вхождений подстроки в строку при помощи алгоритма КМП. Получает на вход строку и подстроку и возвращает контейнер с индексами `result`, для которых было найдено совпадение.

`int cycleShift(std::string str1, std::string str2)` – функция проверяет, является ли строка `str2` циклическим сдвигом строки `str1`, возвращает индекс начала строки `str1` в строке `str2`. Если `str2` не является циклическим сдвигом, то функция возвращает `-1`.

Тестирование.

1) Алгоритм КМП

1. Входные данные из примера

```
ab
abab
0,2
```

2. Неоднократное вхождение

x

xxxxxxxxxxxx

0,1,2,3,4,5,6,7,8,9,10

3. Несовпадение

xxxxxy

xxxxxx

-1

4. Тестирование с выводом промежуточных данных

orororxor

orororororxor

<Prefix function:>

First entry: o = 0

Symbol ris not equal to o and j = 0

Prefix r = 0

Symbol o is equal to o

Prefix++ o = 1

Symbol r is equal to r

Prefix++ r = 2

Symbol o is equal to o

Prefix++ o = 3

Symbol r is equal to r

Prefix++ r = 4

Symbol x not equal to o and j != 0

Ind j x = 2

Symbol x not equal to o and j != 0

Ind j x = 0

Symbol xis not equal to o and j = 0

Prefix x = 0

Symbol o is equal to o

Prefix++ o = 1

Symbol r is equal to r

Prefix++ r = 2

Prefix function: 001234012

<KMP function:>

Symbol o is equal to o

Symbol r is equal to r

Symbol o is equal to o

Symbol r is equal to r

Symbol o is equal to o

Symbol r is equal to r

Symbol o is not equal to x

Return to previous substr symbol

Symbol o is equal to o

Symbol r is equal to r

Symbol o is not equal to x

Return to previous substr symbol

Symbol o is equal to o

Symbol r is equal to r

Symbol x is equal to x

Symbol o is equal to o

Symbol r is equal to r

Entry Start Indices orororxor in orororororxor

2) Циклический сдвиг

1. Входные данные из примера

```
defabc  
abcdef  
3
```

2. Строки разной длины

```
abd  
asfkqhbpqdnvgwe  
-1
```

3. Строки равны

```
abcabc  
abcabc  
0
```

4. Отсутствии циклического сдвига

```
abcd  
bcde  
-1
```

5. Тестирование с выводом промежуточных данных:

```
defabc  
abcdef  
<Prefix function:>  
First entry: a = 0  
Symbol b not equal to a and j = 0  
Prefix b = 0  
Symbol c not equal to a and j = 0  
Prefix c = 0  
Symbol d not equal to a and j = 0  
Prefix d = 0  
Symbol e not equal to a and j = 0  
Prefix e = 0  
Symbol f not equal to a and j = 0  
Prefix f = 0  
Prefix function: 000000  
-----  
  
Symbol d is not equal to a  
Symbol e is not equal to a  
Symbol f is not equal to a  
Symbol a is equal to a  
Symbol b is equal to b  
Symbol c is equal to c  
Symbol d is equal to d  
Symbol e is equal to e  
Symbol f is equal to f  
Cycle was founded  
3  
Process finished with exit code 0
```

Вывод.

В ходе выполнения данной лабораторной работы были получены навыки по работе с алгоритмом Кнута-Морриса-Практа для поиска всех вхождений подстроки в строку. Были написаны функция вычисления префикс-функции, функция для работы алгоритма КМП и функция проверки является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Кмп.

```
#include <iostream>
#include <vector>
//KMP
std::vector<int> prefix(std::string str){
    std::vector<int> p(str.length()); //в массиве p хранится значение префик функции

    p[0] = 0; //начальная инициализация
    int j = 0;
    int i = 1;

    while (i < str.length()){ //просмотрт всей строки
        if(str[i] == str[j]){ //символы совпадают
            p[i] = j + 1; //префикс увеличивается
            i++;
            j++;
        } else if (j == 0){ //совпадений ещё не было
            p[i] = 0;
            i++;
        } else { //нет совпадений и j != 0
            j = p[j - 1];
        }
    }
    return p;
}

std::vector<int> KMP(std::string str, std::string substr){
    std::vector<int> result;
    std::vector<int> p = prefix(substr); //вычисление префикса для подстроки
    int k = 0; //индекс для строки
    int l = 0; //индекс для подстроки

    while(k < str.length()) { //поиск по всей строке
        if(str[k] == substr[l]) { //символы совпадают
            k++;
            l++;
        }
        if (l == substr.length()) // если l == длине подстроки, то отобраз найден
```

```

        result.push_back(k - substr.length()); //возвращем индекс начала вхождения
    } else if (l != 0) {
        l = p[l - 1]; //возврат к предыдущему символу префикса в подстроке
    }
    else { //символы не совпадают
        k++;
    }
}
return result;
}

```

```

int main() {
    std::string substr;
    std::string str;
    std::cin >> substr;
    std::cin >> str;
    std::vector<int> result = KMP(str, substr);
    if (result.size() == 0) {
        std::cout << "-1";
    }
    for (int i = 0; i < result.size(); ++i) {
        std::cout << result[i];
        if(i != result.size() - 1)
            std::cout << ",";
    }

    return 0;
}

```

ЦИКЛИЧЕСКИЙ СДИВГ

```

#include <iostream>
#include <string>
#include <vector>
//cycle
std::vector<int> prefix(std::string str2){
    std::vector<int> p(str2.length()); //в массиве p хранится значение префикс функции
    p[0] = 0; //начальная инициализация
    int j = 0;
    int i = 1;
    while (i < str2.length()){ //просмотрт всей строки
        if(str2[i] == str2[j]){ //символы совпадают

```

```

        p[i] = j + 1; //префикс увеличивается
        i++;
        j++;
    } else if (j == 0){ //совпадений ещё не было
        p[i] = 0;
        i++;
    } else { //нет совпадений и j != 0
        j = p[j - 1];
    }
}

// for (i = 0; i < str2.size(); i++) std::cout << p[i];
// std::cout << std::endl;
return p;
}

int cycleShift(std::string str1, std::string str2) {
    int ind_str1=0, ind_str2=0;
    if (str1.length() != str2.length()) { //длины строк отличаются
        //std::cout << -1;
        return -1;
    }

    int lap = 0;
    //если строки не равны, то применяем алгоритм КМП
    std::vector<int> p = prefix(str2);

    while (true) {
        if (str1[ind_str1] == str2[ind_str2]) { //символы совпадают
            ind_str1++;
            ind_str2++;
        }
        if (ind_str1 == str1.length()) {
            ind_str1 = 0;
            lap++;
        }
        if (ind_str2 == str2.length()) {
            //std::cout << ind_str1;
            return ind_str1;
        }
    }
}

```

```

    else if(str1[ind_str1] != str2[ind_str2] && ind_str2 < str1.length()){
        if(ind_str2 == 0)
            ind_str1++;
        else
            ind_str2 = p[ind_str2 - 1];
    }
    if(lap > 1)
        break;
}
return -1;
}

```

```

int main() {
    std::string str1;
    std::string str2;
    std::cin>>str1;
    std::cin>>str2;
    std::cout<<sycleShift(str1, str2);
    // auto p = prefix("efefeftef");

    return 0;

}
//defabc abcdef

```