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

Студентка гр. 8382 _____ Рочева А.К.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Разработать программу, находящую все вхождения подстроки в строку и программу, определяющую, является ли одна строка циклическим сдвигом другой с помощью алгоритма Кнута-Морриса-Пратта.

Задание 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.

Вход:

Первая строка — P

Вторая строка - T

Выход:

индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести −1

Sample Input:

ab

abab

Sample Output:

0,2

Задание 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.

Вход:

Первая строка - A

Вторая строка - B

Выход:

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

Sample Input:

defabc

abcdef

Sample Output:

3

Индивидуализация.

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.

Описание алгоритма для поиска всех вхождений подстроки.

Для оптимизации по памяти подстрока и строка, в которой ведется поиск соединены в одну строку. Сначала вводится подстрока, затем, пока не наступит конец ввода (пока cin не вернет false или введенный символ не будет ‘*’ (выбран для удобства ввода)) введенный символ добавляется к строке. В итоге inputString будет выглядеть как {подстрока}+{строка, в которой ведется поиск}. Так же сразу запоминается длина подстроки.

Поиск всех вхождений идет в функции kmp(). Сначала для подстроки вычисляется значение префикс-функции (в prefix()). (Префиск-функция для i-го символа подстроки возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом). В функцию передается вся строка и размер подстроки, цикл идет от начала строки до символа с индексом размера подстроки (не включая его).

Для записи значений префикс-функции создается массив vector<int> pi. Для первого символа значение в pi равно 0. Индексы i и j указывают на

символы, которые сравниваются. Изначально i = 1, j = 0. Если i-ый символ равен j-ому, то в pi[i] записывается j+1 и оба индекса увеличиваются на один. Это значит, что происходит расширение текущего суффикса (напомню, что в pi[i] содержится максимальная длина совпадающих префикса и суффикса подстроки до символа i (включая его)). Если i-ый символ не равен j-ому, то в j записывается значение pi[j-1], т.е. индекс следующего символа за максимальным суффиксом j-1 символа. Далее будет происходить попытка расширения этого суффикса.

Результат работы функции prefix() записывается в массив vector<int> wordPI.

Сам поиск числа вхождений происходит по похожей схеме. Индексы k и i указывают на текущие символы из подстроки и строки соответственно. Если символы совпадают – происходит увеличение индексов на один (т.е. смещение их вправо по строке и подстроке), если не совпадают – из массива pi берется значение префикс-функции для последнего совпавшего символа и индекс подстроки становится равным этому значению (т.к. значение префикс-функции содержит максимальную длину совпадающих префикса и суффикса, то представляется возможным продожить сравнение, полагая, что суффикс стал префиксом). Если индекс подстроки достигает значения размера этой подстроки, то в массив результата записывается индекс i-2*wordSize+1 (т.к. строка изначально содержит подстроку в начале, то нужно два раза вычитать ее размер).

Вывод результата происходит в функции main(). Если kmp() вернула пустой массив, то выводится -1.

Описание структур данных для первого алгоритма.

1. *string inputString* – строка, содержащая подстроку и строку, в которой нужно провести поиск этой подстроки.

2. *vector<int> resultArray* – массив для хранения индексов вхождений подстроки.
3. *vector<int> wordPI* – массив со значениями префикс-функций для каждого символа подстроки.
4. *vector<int> pi* - массив со значениями префикс-функций для каждого символа подстроки (для работы в функции *prefix()*).

Описание функций.

1. *vector<int> kmp(string &inputString, int wordSize)* – функция поиска подстроки в строке. Аргументы: *string &inputString* – ссылка на строку, содержащую и подстроку, и строку, в которой ведется поиск подстроки. *int wordSize* - размер подстроки. Функция возвращает массив с индексами вхождений подстроки в строку.
2. *vector<int> prefix(string &s, int n)* – функция, вычисляющая префикс-функцию для подстроки строки s (подстрока начинается с индекса 0 и заканчивается на индексе *n-1*). Аргументы: *string &s* – ссылка на строку, содержащую подстроку, для которой вычисляются значения префикс-функции. *int n* – размер подстроки. Функция возвращает массив со значениями префикс-функции.

Сложность первого алгоритма.

Сложность по операциям: вычисление префикс-функции происходит за O(n), где n — размер подстроки. Цикл *for* в функции *kmp()* работает за O(m), где m — размер строки, в которой ведется поиск (в нашем случае *m = inputString.size() - wordSize*). В итоге весь алгоритм работает за O(n+m).

Сложность по памяти: в памяти хранится только строка, содержащая две входные строки и массив со значениями префикс-функции, т.е. сложность O((n+m) + n). Не учитывая память для строки сложность будет O(n), что и требовалось в задании.

Описание алгоритма для определения циклического сдвига.

Этот алгоритм очень схож с предыдущим. Чтобы определить, является ли одна строка циклическим сдвигом второй, нужно удвоить первую строку и проверить, есть ли вхождение второй строки в нее. Важным отличием является то, что теперь «подстрока» вводится после основной строки (уже после ее удвоения), поэтому цикл в kmp() начинается с индекса 0, а для проверки символов к индексу подстроки всегда добавляется число start — индекс в исходной строке, с которого начинается подстрока.

Результатом работы алгоритма является индекс вхождения подстроки в строку, либо -1, если нет вхождения (т. е. первая строка не является циклическим сдвигом второй).

Описание структур данных для второго алгоритма.

1. *string inputString* – строка, содержащая подстроку и строку, в которой нужно провести поиск этой подстроки.
2. *vector<int> wordPI* – массив со значениями префикс-функций для каждого символа подстроки.
3. *vector<int> pi* - массив со значениями префикс-функций для каждого символа подстроки (для работы в функции *prefix()*).

Описание функций.

1. *int kmp(string &inputString, int wordSize)* – функция поиска подстроки в строке. Аргументы: *string &inputString* – ссылка на строку, содержащую и подстроку, и строку, в которой ведется поиск подстроки. *int wordSize* - размер подстроки. Функция возвращает индекс вхождения подстроки в строку, либо -1, если такого нет.
2. *vector<int> prefix(string &s, int start)* – функция, вычисляющая префикс-функцию для подстроки строки s (подстрока начинается с индекса *start* и заканчивается на индексе *n-1*). Аргументы: *string &s* –

ссылка на строку, содержащую подстроку, для которой вычисляются значения префикс-функции. *int start* – иднекс начала подстроки в общей строке. Функция возвращает массив со значениями префикс-функции.

Сложность второго алгоритма.

Сложность по операциям: вычисление префикс-функции происходит за O(n), где n — размер подстроки. Цикл *for* в функции *kmp()* работает за O(2m), где m — размер первой строки (2m — потому что она была удвоена). В итоге весь алгоритм работает за O(n + 2m).

Сложность по памяти: в памяти хранится только строка, содержащая две входные строки и массив со значениями префикс-функции, т.е. сложность O((2m+n) + n). Не учитывая память для строки сложность будет O(n), что и требовалось в задании.

Тестирование

Поиск вхождений (без вывода промежуточных данных):

Входные данные	Выходные данные
qas qasqasqasqas*	0,3,6,3
hetr askljkvdhjjsdck*	-1
tw atwwtwttw*	1,4,7

(с выводом промежуточных данных):

Входные данные	Выходные данные
abab ababrabaabab*	Prefix-function for: abab first sumbol = a with max suffix = 0 current symbol: b current max suffix for b = 0 current symbol: a. It matches a letter with index 0

	now j indicates a letter <b> and suffix has increased current max suffix for a = 1 current symbol: b. It matches a letter with index 1 now j indicates a letter <a> and suffix has increased current max suffix for b = 2 Value of prefix-function: 0 0 1 2 word[0] = string[0] = a word[1] = string[1] = b word[2] = string[2] = a word[3] = string[3] = b Word was found. Start index: 0 Continue comporation from letter a word[2] != string[4] decrease k = 2. Now k = 0 word[0] = string[5] = a word[1] = string[6] = b word[2] = string[7] = a word[3] != string[8] decrease k = 3. Now k = 1 word[1] != string[8] decrease k = 1. Now k = 0 word[0] = string[8] = a word[1] = string[9] = b word[2] = string[10] = a word[3] = string[11] = b Word was found. Start index: 8 Continue comporation from letter a end of string RESULT: 0,8

Определение циклического сдвига (без вывода промежуточных данных):

Входные данные	Выходные данные
aqswde qswdea*	1
zzzzzzx zzzxzzz*	3
azzaazza azaaazza*	-1

(с выводом промежуточных данных):

Входные данные	Выходные данные
defabc abcdef*	Prefix-function for: abcdef first sumbol = a with max suffix = 0 current symbol: b current max suffix for b = 0 current symbol: c current max suffix for c = 0 current symbol: d. It matches a letter with index 0 now j indicates a letter <e> and suffix has increased current max suffix for d = 1 current symbol: e. It matches a letter with index 1 now j indicates a letter <f> and suffix has increased current max suffix for e = 2 current symbol: f. It matches a letter with index 2 now j indicates a letter <a> and suffix has increased current max suffix for f = 3 Value of prefix-function: 0 0 0 0 0 0 word[0] = string[3] = d word[1] = string[4] = e word[2] = string[5] = f word[3] = string[6] = a word[4] = string[7] = b word[5] = string[8] = c Start index: 3 3

Выводы.

В ходе выполнения работы была разработана программа, выполняющая поиск вхождений подстроки в строку (приложение А) и программа, определяющая, является ли одна строка циклическим сдвигом другой (приложение В).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ФАЙЛА MAIN1.CPP

```
#include <iostream>
#include <vector>

using std::vector;
using std::string;
using std::cout;
using std::cin;
using std::endl;

// вычисление префикс-функции для строки s
vector<int> prefix(string &s, int n){
    cout << "Prefix-function for: ";
    int j = 0;
    while (j < n) {
        cout << s[j];
        j++;
    }
    cout << endl;

    vector<int> pi (n); // массив чисел pi
    j = 0; //
    cout << "first sumbol = " << s[0] << " with max suffix = 0" << endl;
    for (int i = 1; i < n; i++) {
        cout << "current symbol: " << s[i];
        while ((j > 0) && (s[i] != s[j])) { // уменьшаем суффикс
            cout << " != " << s[j] << endl;
            cout << "change j = " << j << " to ";
            // теперь рассматриваем символ с индексом, равным
максимальному суффиксу предыдущего символа (j-1)
            // т.е. в дальнейшем будем пытаться расширить этот суффикс
            j = pi[j-1];
            cout << j;
            cout << " and suffix has decreased";
        }
        if (s[i] == s[j]) { // увеличиваем суффикс
            cout << ". It matches a letter with index " << j << endl;
            j++;
            cout << "now j indicates a letter <" << s[j];
            cout << "> and suffix has increased";
        }
        cout << endl << "current max suffix for " << s[i] << " = " << j <<
endl;
        pi[i] = j;
    }
```

```
    return pi;
}

// алгоритм Кнута-Морриса-Пратта
vector<int> kmp(string &inputString, size_t wordSize) {
    vector<int> wordPI = prefix(inputString, wordSize); // значение
префикс-функции для подстроки

    cout << "Value of prefix-function:" << endl;
    for (auto & p : wordPI)
        cout << p << " ";
    cout << endl << endl;

    vector<int> resultArray;

    int k = 0;
    int stringSize = inputString.size();

    for (int i = wordSize; i < stringSize; i++) {
        while ((k > 0) && (inputString[k] != inputString[i])) {
            cout << "word[" << k << "] != string[" << i-wordSize << "]" <<
endl;
            cout << "decrease k = " << k;
            // k отодвигается назад до значения максимального суффикса
предыдущего до k символа
            k = wordPI[k-1];
            cout << ". Now k = " << k << endl;
        }

        if (inputString[k] == inputString[i]) {
            // увеличиваем k, чтобы при следующей итерации сравнить
следующие символы
            cout << "word[" << k << "] = string[" << i-wordSize << "] = "
<< inputString[k] << endl;
            k++;
        }

        if (k == wordSize){ // слово нашлось
            cout << "Word was found. Start index: " << i - (2*wordSize) +
1 << endl;
            resultArray.push_back(i - (2*wordSize) + 1);
            cout << "Continue comporation from letter ";
            k = wordPI[k-1];
            cout << inputString[k] << endl;
        }
    }
```

```
    cout << "end of string" << endl;
    return resultArray;
}

int main() {

    string inputString;

    cin >> inputString; // подстрока
    size_t wordSize = inputString.size();
    char s;
    while(true){
        if (!(cin >> s) || s == '*')
            break;
        inputString += s;
    }

    vector<int> resultArray = kmp(inputString, wordSize);
    cout << endl << "RESULT:" << endl;
    if (resultArray.empty()){
        cout << -1;
    } else {
        for (auto & index : resultArray){
            if (index != resultArray.front())
                cout << ",";
            cout << index;
        }
    }

    return 0;
}
```

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ФАЙЛА MAIN2.CPP

```
#include <iostream>
#include <vector>

using std::vector;
using std::string;
using std::cout;
using std::cin;
using std::endl;

// вычисление префикс-функции для подстроки s, начинающеся с индекса start
vector<int> prefix(string &s, int start){
    int n = s.size();
    vector<int> pi (n-start); // массив чисел pi

    cout << "Prefix-function for: ";
    int j = start;
    while (j < n) {
        cout << s[j];
        j++;
    }
    cout << endl;

    j = 0;
    cout << "first sumbol = " << s[start] << " with max suffix = 0" <<
endl;
    for (int i = (start+1); i < n; i++) {
        cout << "current symbol: " << s[i];
        while ((j > 0) && (s[i] != s[j])) {
            cout << " != " << s[j] << endl;
            cout << "change j = " << j << " to ";
            // теперь рассматриваем символ с индексом, равным
максимальному суффиксу предыдущего символа (j-1)
            // т.е. в дальнейшем будем пытаться расширить этот суффикс
            j = pi[j-1];
            cout << j;
            cout << " and suffix has decreased";
        }

        if (s[i] == s[j]) { // увеличиваем суффикс
            cout << ". It matches a letter with index " << j << endl;
            j++;
            cout << "now j indicates a letter <" << s[j];
            cout << "> and suffix has increased";
        }
```

```
        cout << endl << "current max suffix for " << s[i] << " = " << j <<
endl;
        pi[i] = j;
    }

    return pi;
}

// алгоритм Кнута-Морриса-Пратта
int kmp(string &inputString, int wordSize) {
    int stringSize = inputString.size();
    int start = stringSize - wordSize;
    vector<int> wordPI = prefix(inputString, start); // значение префикс-
функции для подстроки

    cout << "Value of prefix-function:" << endl;
    for (auto & p : wordPI)
        cout << p << " ";
    cout << endl << endl;

    int k = 0; // индекс в подстроке (настоящий индекс = k+start)

    for (int i = 0; i < start; i++) {

        while ((k > 0) && (inputString[k+start] != inputString[i])) {
            cout << "word[" << k << "] != string[" << i << "]" << endl;
            cout << "decrease k = " << k;
            // k отодвигается назад до значения максимального суффикса
предыдущего до k символа
            k = wordPI[k-1];
            cout << ". Now k = " << k << endl;
        }

        if (inputString[k+start] == inputString[i]) {
            // увеличиваем k, чтобы при следующей итерации сравнить
следующие символы
            cout << "word[" << k << "] = string[" << i << "] = " <<
inputString[k] << endl;
            k++;
        }

        if (k == wordSize){ // слово нашлось
            cout << "Start index: " << i - wordSize + 1 << endl;
            return  i - wordSize + 1;
        }
    }

    cout << "now found." << endl;
```

```
    return -1;
}


int main() {

    string inputString;

    cin >> inputString; // эта строка – циклический сдвиг
    inputString += inputString; // удваиваем строку
    int stringSize = inputString.size();
    char s;
    while(true){ // ввод строки, для которой inputString является
циклическим сдвигом
        if (!(cin >> s) || s == '*') // '*' показывает, когда закончится
ввод
            break;
        inputString += s;
    }

    // ищем вхождение word в удвоенную строку
    int result = kmp(inputString, inputString.size() - stringSize);
    cout << result;

    return 0;
}
```