МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта.

Студент гр. 8382 _____ Терехов А.Е.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм поиска образа в строке Кнута-Морриса-Пратта.

Задание.

Индивидуализация задания: 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.

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.

Сдвиг:

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

Ход работы.

Описание структур.

Структуры, использованные в лабораторной работе, такие как строки, векторы и т.д., предоставлены стандартной библиотекой языка C++.

Описание алгоритма.

Стандартный поиск:

С помощью алгоритма Кнута-Морриса-Пратта находится образ в тексте за линейное время. Основная идея заключается в использовании префикс функции. Ее значение для каждого символа определяется как длина максимального префикса, оканчивающегося на данной позиции. Исходный код программы со степика представлен в приложении А, и код оптимизированной программы по памяти в приложении Б.

Проверка является ли строка сдвигом другой строки:

Для решения данной задачи один текст рассматривается как образ, а второй как текст, в котором происходит поиск. Отличие данного алгоритма от стандартного заключается в том, что сохраняется индекс первого совпадения после череды несовпадений, в случае когда второй текст заканчивается алгоритм не заканчивает свою работу, а переходит на начало этого текста, во избежание зацикливания добавлена логическая переменная, которая ложь если проход до переноса в начало, и истина если перенос в начало уже был, таким образом по второй строке максимум будет два прохода, чего более чем достаточно.

Исходный код программы представлен в приложении В.

Сложность алгоритма.

В общем случае алгоритм требует O(m+n) памяти, где m – длина текста, а n – длина образа. Но учитывая, что при использовании алгоритма можно не хранить строку, в которой происходит поиск, а посимвольно читать ее в процессе работы алгоритма, сложность можно сократить до O(n).

По времени алгоритму необходимо пройти строку образа один раз для построения префикс функции и строку с текстом так же один раз. Из чего следует вывод, что алгоритм имеет временную сложность O(m+n).

В случае с проверкой на сдвиг аналогично.

Тестирование.

Тестирование проверки на сдвиг представлено в таблице 1.

Таблица 1. Тестирование проверки на сдвиг.

INPUT	OUTPUT
1234ABCD ABCD1234	4
asljdFKhasfkhdfgiubaenbilvjzsfizsngilusnszlirubnazlisugnjhiguzbh aijodfkg;jo;buhjongbidgbiunlbiugnjedblegiubg rubnazlisugnjhiguzbhaijodfkg;jo;buhjongbidgbiunlbiugnjedblegiub gasljdFKhasfkhdfgiubaenbilvjzsfizsngilusnszli	44
asljdFKhasfkhdfgiubaenbilvjzsfizsngilusnszlirubnazlisugnjhiguzbh aijodfkg;jo;buhjongbidgbiunlbiugnjedblegiubg rubnazlisugnjhiguzbhaijodfkg;jo;buhjongbidgbiunlbiuggjedblegiub gasljdFKhasfkhdfgiubaenbilvjzsfizsngilusnszli	-1

Тестирование стандартного алгоритма представлено в таблице 2.

Таблица 2. Тестирование алгоритма Кнута-Морриса-Пратта.

INPUT	OUTPUT
ab abab	a b 0 0 0,2
123a123 sadf123aasdfdsf24dsf123aa123a123	1 2 3 a 1 2 3 0 0 0 0 1 2 3 25
aaaaa aaaaaaaaaaaaaasdfgaasdfgaaaasdfgaaaasdfgaaaaadsf g	a a a a a 0 1 2 3 4 0,1,2,3,4,5,6,7,8,9,10,41

Вывод.

В ходе работы был реализован и протестирован алгоритм поиска подстроки в строке Кнута-Морриса-Пратта, разработан на его основе алгоритм проверки на сдвиг. Стандартный алгоритм оптимизирован по памяти. Для сборки всех

программ написан Makefile. Для запуска с использованием подготовленных тестов написаны bash-скрипты. В оптимизированной программе добавлен вывод промежуточных результатов, а именно вывод значений префикс функции.

ПРИЛОЖЕНИЕ А

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

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;
// пи-функция генерирует массив переходов
vector<int> piFunction(string img){
    vector<int> pi;
    pi.insert(pi.end(), 0); // первый символ образа имеет значение пи-
функции равное нулю
    int i = 1; // счетчики
    int j = 0;
    while (i < img.size()){ // пока не дошли до конца строки первым
счетчиком
        if (img[i] == img[j]){          // если символы на iом и jом месте
равны,
            pi.insert(pi.end(), j + 1); // то добавляем в массив значений
пи-функции j+1 (значение функции для символа на iом месте)
            i++;                        // увеличиваем счетчики
            j++;                        // и переходим к следующей
итерации
        }
        else if (j == 0){               // если обнулился второй счетчик
из-за следующего else-блока, но символы на iом и jом месте не равны
            pi.insert(pi.end(), 0);     // добавляем ноль в массив
значений
            i++;                        // продолжаем движение по строке
        }
        else                            // если символы на iом и jом месте
не равны и j не равно нулю,
            j = pi[j - 1];              // то значение вычисляется на
основе уже существующего
    }
    return pi;
}

// поиск образа в строке
vector<int> kmp(string img, string haystack, vector<int> pi_vec){
    vector<int> find;   // массив ответов
    int img_ind = 0;    // счетчик для продвижения по образу
    int text_ind = 0;   // счетчик для продвижения по тексту
    while (text_ind < haystack.size()){
        if (haystack[text_ind] == img[img_ind]){    // если два одинаковых
символа двигаемся вперед и по образу и по строке
            img_ind++;
            text_ind++;
            if (img_ind == img.size()){         // если дошли до конца
образа, значит нашли вхождение
                find.insert(find.end(), text_ind - img_ind);
                img_ind = pi_vec[img_ind - 1];  // переходим не в начало
образа, а на позицию равную предпоследнему значению пи-функции
            }
        }
        else if (img_ind == 0)  // если нет совпадения, и сравнение
происходит с самым первым символом образа
            text_ind++;         // продвигаемся по тексту
        else                                // если нет совпадения, но по
```

```
образу уже продвинулись от начала
                img_ind = pi_vec[img_ind - 1];  // то в образе перескакиваем
на символ с индексом равным предыдущему значению пи-функции
            }
            return find;
        }

        int main(){
            string image;
            string haystack;
            cin >> image;   // чтение образа
            cin >> haystack;    // чтение текста
            vector<int> pi = piFunction(image); // вычисление пи-функции для
образа
            vector<int> answer = kmp(image, haystack, pi);  // вызов функции
поиска
            if (answer.empty()) // если ничего не нашли
                cout << -1 << endl;
            else{   // иначе выводим содержимое ответа
                for (int i = 0; i < answer.size() - 1; ++i)
                    cout << answer[i] << ",";
                cout << answer[answer.size() - 1] << endl;
            }
            return 0;
        }
```

ПРИЛОЖЕНИЕ Б

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

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;
// пи-функция генерирует массив переходов
vector<int> piFunction(string img){
    vector<int> pi;
    pi.insert(pi.end(), 0); // первый символ образа имеет значение пи-функции равное нулю
    int i = 1;              // счетчики
    int j = 0;
    while (i < img.size()){ // пока не дошли до конца строки первым счетчиком
        if (img[i] == img[j]){                                  // если символы на iом и jом месте равны,
            pi.insert(pi.end(), j + 1); // то добавляем в массив значений пи-функции j+1 (значение функции для символа на iом месте)
            i++;                        // увеличиваем счетчики
            j++;                        // и переходим к следующей итерации
        }
        else if (j == 0){                           // если обнулился второй счетчик из-за следующего else-блока, но символы на iом и jом месте не равны
            pi.insert(pi.end(), 0); // добавляем ноль в массив значений
            i++;                    // продолжаем движение по строке
        }
        else              // если символы на iом и jом месте не равны и j не равно нулю,
            j = pi[j - 1]; // то значение вычисляется на основе уже существующего
    }
    return pi;
}

// поиск образа в строке
vector<int> kmp(string img, vector<int> pi_vec){
    int img_s = img.size();
    char c;           // считываемый символ
    vector<int> find; // массив ответов
    int img_ind = 0;  // счетчик для продвижения по образу
    int text_ind = 0; // счетчик для продвижения по тексту
    bool correct = true;
    cin >> c;
    do{
        if (c == img[img_ind]){               // если два одинаковых символа двигаемся вперед и по образу и по строке
            text_ind++;
            if(! (cin >> c))correct = false;
            img_ind++;
            if (img_ind == img_s){ // если дошли до конца образа, значит нашли вхождение
                find.insert(find.end(), text_ind - img_ind);
                img_ind = pi_vec[img_ind - 1]; // переходим не в начало образа, а на позицию равную предпоследнему значению пи-функции
            }
```

```
            }
            else if (img_ind == 0){ // если нет совпадения, и сравнение
происходит не с самым первым символом образа
                text_ind++;
                if(! (cin >> c))correct = false;
            }
            else
                img_ind = pi_vec[img_ind - 1]; // то в образе перескакиваем на
символ с индексом равным предыдущему значению пи-функции
        } while (correct);
        return find;
    }

    int main(){
        string image;
        cin >> image;                        // чтение образа
        vector<int> pi = piFunction(image); // вычисление пи-функции для
образа
        for (int i = 0; i < image.size(); ++i){
            cout << "\t" << image[i];
        }
        cout << endl;
        for (int i = 0; i < image.size(); ++i){
            cout << "\t" << pi[i];
        }
        cout << endl;
        vector<int> answer = kmp(image, pi); // вызов функции поиска
        if (answer.empty())                   // если ничего не нашли
            cout << -1 << endl;
        else{ // иначе выводим содержимое ответа
            for (int i = 0; i < answer.size() - 1; ++i)
                cout << answer[i] << ",";
            cout << answer[answer.size() - 1] << endl;
        }
        return 0;
    }
```

ПРИЛОЖЕНИЕ В

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

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

// пи-функция генерирует массив переходов
vector<int> piFunction(string img){
    vector<int> pi;
    pi.insert(pi.end(), 0); // первый символ образа имеет значение пи-
функции равное нулю
    int i = 1; // счетчики
    int j = 0;
    while (i < img.size()){ // пока не дошли до конца строки первым
счетчиком
        if (img[i] == img[j]){          // если символы на iом и jом месте
равны,
            pi.insert(pi.end(), j + 1); // то добавляем в массив значений
пи-функции j+1 (значение функции для символа на iом месте)
            i++;                        // увеличиваем счетчики
            j++;                        // и переходим к следующей
итерации
        }
        else if (j == 0){               // если обнулился второй счетчик
из-за следующего else-блока, но символы на iом и jом месте не равны
            pi.insert(pi.end(), 0);     // добавляем ноль в массив
значений
            i++;                        // продолжаем движение по строке
        }
        else                            // если символы на iом и jом месте
не равны и j не равно нулю,
            j = pi[j - 1];              // то значение вычисляется на
основе уже существующего
    }
    return pi;
}

int kmp(string image, string haystack, vector<int> pi_vec) {
    if (image.size() != haystack.size()) // проверка на равенство длин
        return -1;
    int first_equal = -1;   // индекс первого совпадения после серии
несовпадений
    int i_img = 0;             // индексы для первого и второго текста
соответственно
    int i_hay = 0;
    bool loop = false;      // переменная для контроля зацикливания, false
если еще ни разу не дошли до конца haystack
    while (true) {
        if (image[i_img] == haystack[i_hay]) {    // если два
рассматриваемых символа совпадают
            if (i_img == 0)                    // и это произошло впервые,
то обновляем индекс первого совпадения
                first_equal = i_hay;
            i_img++;   // независимо от того в какой раз символы совпали
продвигаемся вперед по обеим строкам
            i_hay++;
```

```
                if (i_img == image.size())  // если дошли до конца первой
строки, значит нашли индекс во второй строке с которого началось совпадение
                    return first_equal;     // возвращаем индекс первого
совпадения
            }
            else if (i_img == 0){           //если символы не совпали но
рассматриваем первую строку с самого начала
                i_hay++;                    // то продвигаемся по второй
строке вперед
            }
            else{                           // если символы не совпали, но
сравнивали уже не первый символ первой строки,
                i_img = pi_vec[i_img - 1];  // то в в ней переходим на символ
с индексом равным предыдущему значению пи-функции
            }
            if(i_hay == haystack.size()){   // если дошли до конца второй
строки
                if (loop)                   // дважды
                    return -1;              // значит ничего не нашли
                loop = true;    // если в первый раз
                i_hay = 0;      // переходим в начало второй строки
            }
        }
    }

    int main() {
        string text1;   // первый текст как haystack
        string text2;   // второй текст как needle, образ
        cin >> text1;
        cin >> text2;
        vector<int> pi = piFunction(text2);
        int answer = kmp(text2, text1, pi);
        cout << answer << endl;
        return 0;
    }
```