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. 8304 _____ Щука А. А.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы вхождения подстроки в строку, а также разработать алгоритм проверки двух строк на циклический сдвиг.

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.

Задание.

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.

Пример входных данных

aba

ababa

Пример выходных данных

0, 2

Описание алгоритма КМП.

На вход алгоритма передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения.

Оптимизация – строка-текст считывается посимвольно, в памяти хранится текущий символ.

Алгоритм сначала вычисляет префикс-функцию строки-образца.

Далее посимвольно считывается строка-текст. Переменная-счетчик изначально k = 0. При каждом совпадении k-го символа образца и i-го символа текста счетчик увеличивается на 1. Если k = размер образца, значит вхождение найдено. Если очередной символ текста не совпал с k-ым символом образца, то сдвигаем образец, причем точно знаем, что первые k символов образца

совпали с символами строки и надо сравнить k+1-й символ образца (его индекс k) с i-м символом строки.

Сложность алгоритма по операциям: O (m + n), m – длина образца, n – длина текста.

Сложность алгоритма по памяти: O (m), m – длина образца.

Описание алгоритма проверки циклического сдвига.

Для того, чтобы вычислить, является ли одна строка циклическим сдвигом другой, можно воспользоваться префикс функцией.

Сначала алгоритм сравнивает размеры строк, если они не совпадают – строки не могут являть циклическим сдвигом.

Затем алгоритм сравнивает строки, если они равны, то они циклический сдвиг друг друга со смещением 0.

Если алгоритм не завершил работу, складывается первая строка с двумя вторыми (лексикографически), далее вычисляется префикс-функция от строки результата. Строка результат имеет вид ABB (A – первая строка, B – вторая строка). Если в B_2 у какого-нибудь символа префикс-функция равна длине строки (size(A) = size(B) = prefix(i)), то строки являются циклическим сдвигом.

Сложность алгоритма по операциям: O (n), n – длина строки

Сложность алгоритма по памяти: O (n), n – длина строки

Описание функций и структур данных.

`std::vector<size_t> prefixFunction (const std::string& string)`

Функция вычисления префикс-функции строки. Принимает на вход строку, возвращает массив со значениями префикс-функции.

`std::vector<int> KMP(std::istream& input, const std::string& pattern)`

Функция, реализующая алгоритм КМП. Принимает на вход поток ввода текста и образец, вхождения которого нужно найти. Возвращает массив вхождений (-1 если вхождений не найдено).

`void cyclicShift(const std::string& firstString, const std::string& secondString)`

Функция, реализующая алгоритм проверки строк на циклический сдвиг. Принимает на вход две строки для проверки, выводит либо индекс вхождения одной строки в другую со сдвигом, либо -1, если строки не являются циклическим сдвигом.

`void writeRes(std::ostream& output, std::vector<int>& result)`

Функция печати результата. Принимает на вход поток вывода и массив-результат. Выводит содержимое массива через запятую.

Тестирование.

Входные данные:

aba

ababa

Результат работы программы:

```
Input from: 0 - console, 1 - file: 0
Write to: 0 - console, 1 - file: 0
Enter pattern: aba
Calculating prefix
Text: aba
Prefix 0 = 0
Prefix 1 = 0
Prefix 2 = 1
Enter text and press enter and ctrl + z: ababa
Char: a
a ==  a
Size of pattern: 3. k = 1
Char: b
b ==  b
Size of pattern: 3. k = 2
Char: a
a ==  a
Size of pattern: 3. k = 3
Pattern found! Index: 0
Char: b
  !=  b
b ==  b
Size of pattern: 3. k = 2
Char: a
a ==  a
Size of pattern: 3. k = 3
Pattern found! Index: 2
^Z
Char:
  ==
Size of pattern: 3. k = 4
Indices: 0,2
Press <RETURN> to close this window...
```

Входные данные:

ab

aababaaabaaba

Результат работы программы:

```
Input from: 0 - console, 1 - file: 0
Write to: 0 - console, 1 - file: 0
Enter pattern: ab
Calculating prefix
Text: ab
Prefix 0 = 0
```

```
Prefix 1 = 0
Enter text and press enter and ctrl + z: aababaaabaaba
Char: a
a == a
Size of pattern: 2. k = 1
Char: a
b != a
a == a
Size of pattern: 2. k = 1
Char: b
b == b
Size of pattern: 2. k = 2
Pattern found! Index: 1
Char: a
  != a
a == a
Size of pattern: 2. k = 1
Char: b
b == b
Size of pattern: 2. k = 2
Pattern found! Index: 3
Char: a
  != a
a == a
Size of pattern: 2. k = 1
Char: a
b != a
a == a
Size of pattern: 2. k = 1
Char: a
b != a
a == a
Size of pattern: 2. k = 1
Char: b
b == b
Size of pattern: 2. k = 2
Pattern found! Index: 7
Char: a
  != a
a == a
Size of pattern: 2. k = 1
Char: a
b != a
a == a
Size of pattern: 2. k = 1
Char: b
b == b
Size of pattern: 2. k = 2
Pattern found! Index: 10
Char: a
  != a
a == a
Size of pattern: 2. k = 1
^Z
Char:
b !=
Size of pattern: 2. k = 0
Indices: 1,3,7,10
Press <RETURN> to close this window...
```

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм КМП и алгоритм проверки двух строк на циклический сдвиг, а также функция вычисления префикса строки.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

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


constexpr bool DEBUG = true;
constexpr const char* PATH_INPUT = "D:/test.txt";
constexpr const char* PATH_OUTPUT = "D:/result.txt";


std::vector<size_t> prefixFunction (const std::string& string) {
    if (DEBUG) {
        std::cout << "Calculating prefix\nText: " << string << "\n";
        std::cout << "Prefix 0 = 0\n";
    }
    // в i-м элементе (его индекс i-1) количество совпавших
    // символов в начале образца и в конце подстроки длины i.
    // p[0]=0 всегда
    std::vector<size_t> pi(string.size());
    pi[0] = 0;

    // заполняем массив длин префиксов для образца
    for (size_t i = 1; i < string.size(); ++i) {
        size_t j = pi[i-1];

        while ((j > 0) && (string[i] != string[j])) {
            j = pi[j-1];
        }

        if (string[i] == string[j]) {
            ++j;
        }

        if (DEBUG) {
            std::cout << "Prefix " << i << " = " << j << "\n";
        }

        pi[i] = j;
    }

    return pi;
}


std::vector<int> KMP(std::istream& input,
                     const std::string& pattern) {
    std::vector<int> result;
    size_t size = pattern.size();

    auto pi = prefixFunction(pattern);

    size_t k = 0;
    size_t i = 0;

    if (DEBUG) {
        std::cout << "Enter text and press enter and ctrl + z: ";
    }
```

```
    while (!input.eof()) {
        char ch = 0;
        input >> ch;

        if (DEBUG) {
            std::cout << "Char: " << ch << "\n";
        }

        while ((k > 0) && (pattern[k] != ch)) {
            // Очередной символ строки не совпал с символом в образце. Сдвигаем
образец,
            // причем точно знаем, что первые k символов образца совпали с символами
строки
            // и надо сравнить k+1-й символ образца (его индекс k) с i-м символом
строки.
            if (DEBUG) {
                std::cout << pattern[k] << " != " << ch << "\n";
            }

            k = pi[k-1];
        }

        if (pattern[k] == ch) {
            if (DEBUG) {
                std::cout << pattern[k] << " == " << ch << "\n";
            }
            // есть совпадение очередного символа
            // увеличиваем длину совпавшего фрагмента на 1
            k++;
        }

        if (DEBUG) {
            std::cout << "Size of pattern: " << size << ". k = " << k << "\n";
        }

        if (k == size) {
            if (DEBUG) {
                std::cout << "Pattern found! Index: " << i + 1 - size << "\n";
            }
            // образец найден
            result.push_back(i + 1 - size);
        }
        ++i;
    }

    if (result.size() == 0) {
        result.push_back(-1);
    }

    return result;
}


void cyclicShift(const std::string& firstString,
                 const std::string& secondString) {
    if (DEBUG) {
        std::cout << "CyclicShift\n";
    }

    if (firstString.size() != secondString.size()) {
        //если размеры строк не совпадают
```

```
        if (DEBUG) {
            std::cout << "Sizes not equal\n";
        }
        std::cout << -1;
        return;
    }
    else if (firstString == secondString) {
        if (DEBUG) {
            std::cout << "Strings are equal\n";
        }
        //если строки равны
        std::cout << 0;
        return;
    }

    size_t size = firstString.size();

    char* buff = new char[3 * size];

    //контатенация одной первой строки и двух вторых
    size_t i = 0;
    for (; i < size; ++i) {
        buff[i] = secondString[i];
    }
    for (int k = 0; k < 2; ++k) {
        for (size_t j = 0; j < size; ++j) {
            buff[i++] = firstString[j];
        }
    }

    //вычисление префикс функции для итоговой строки
    auto pi = prefixFunction(buff);

    for (size_t i = 2 * size - 2; i < 3 * size; ++i) {

        if (pi[i] == size) {
            //если в каком-то месте префикс функция равна размеру первой строки,
            //то вторая строка является циклическим сдвигом
            if (DEBUG) {
                std::cout << "It's cyclicShift!\n";
            }

            std::cout << i + 1 - 2 * size;
            delete [] buff;
            return;
        }
    }

    //если не является
    if (DEBUG) {
        std::cout << "It's not cyclicShift!\n";
    }
    std::cout << -1;
    delete [] buff;
}


void writeRes(std::ostream& output, std::vector<int>& result) {
    if (DEBUG) {
        output << "Indices: ";
    }
```

```
    for (size_t i = 0; i < result.size(); ++i) {
        output << result[i];
        if (i != result.size() - 1) {
            output << ',';
        }
    }
    output << "\n";
}


int main() {
    int chooseInput = 0;
    int chooseOutput = 0;
    std::string pattern;
    std::vector<int> result;

    if (DEBUG) {
        std::cout << "Input from: 0 - console, 1 - file: ";
        std::cin >> chooseInput;
        std::cout << "Write to: 0 - console, 1 - file: ";
        std::cin >> chooseOutput;
    }

    if (chooseInput == 0) {
        if (DEBUG) {
            std::cout << "Enter pattern: ";
        }
        std::cin >> pattern;
        result = KMP(std::cin, pattern);
    }
    else if (chooseInput == 1) {
        std::ifstream file;
        file.open(PATH_INPUT);

        if (file.is_open()) {
            file >> pattern;
            result = KMP(file, pattern);
            file.close();
        }
        else {
            std::cout << "Can't open file!\n";
            return 0;
        }
    }
    else {
        std::cout << "Incorrect input!\n";
        return 0;
    }

    if (chooseOutput == 0) {
        writeRes(std::cout, result);
    }
    else if (chooseOutput == 1) {
        std::ofstream file;
        file.open(PATH_OUTPUT);
        writeRes(file, result);
        file.close();
    }
    else {
        std::cout << "Incorrect input!\n";
```

```
            return 0;
        }

        return 0;
    }
```