**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. 8304 |  | Ястребов И.М. |
| 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.

**Задание.**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P и текста T 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.

**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.

При сдвиге вполне можно ожидать, что префикс образца P сойдется с каким-нибудь суффиксом текста T. Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки P для индекса j. Пусть [j] — значение префикс-функции от строки P для индекса j. Тогда после сдвига мы можем возобновить сравнения с места T[i + j] и P[[j]] без потери возможного местонахождения образца.

Сложность алгоритма O(m + n), где

m – длина образца,

n – длина строки в которой мы ищем.

Однако, следует сделать замечание о том, что к сложности можно добавить сложность хранения ответа. Это легко регулируется — и если мы не хотим помнить ответ, то, конечно, сложность будет такая, как описано выше, но если мы храним ответ, то потенциально занимаем O(m + n + (n-m) = O(2n).

## Описание основных структур данных и функций.

void prefixFunction(const std::string& srcString, std::vector<int>& prefixFunctionValues); // Префикс-функция

- префикс-функция, выдающая массив размера строки, в котором содержатся размеры максимальных бордеров соответствующих префиксов. Принимает строку и массив, куда записывает результат обработки строки.

void KMP(std::istream& input, std::ostream& output) // Алгоритм КМП

- алгоритм Кнута-Морриса-Пратта, модернизированный согласно индивидуальному заданию. Принимает два потока — входной и выходной.

**Тестирование.**

Таблица 1 – Результат работы.

|  |  |
| --- | --- |
| **Ввод** | **Вывод** |
| fc  kdheojsdcfc;jflkhfcfcj;klkhffc | Result: 9, 17, 19, 28 |
| ultralord  ultralordksdhflsabultra3lordcabcdultralordlsdafjabclsdjfkultralord | Result: 0, 33, 57 |
| k  dskghjkdfhkdgdhgkasjghxdvkj | Result: 2, 6, 10, 16, 25 |
| ddd  akshfsdhjkahfa | Result: -1 |

# Вывод.

В ходе выполнения данной работы был реализован алгоритм Кнута-Морриса-Пратта для поиска вхождений подстроки P в строку T а также префикс-функция для строк.

**ПРИЛОЖЕНИЕ А.  
ИСХОДНЫЙ КОД**

#include "pch.h"

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

std::string inPath = "./input.txt";

std::string outPath = "./output.txt"; // Пути к файлам ввода и вывода

void prefixFunction(const std::string& srcString, std::vector<int>& prefixFunctionValues); // Префикс-функция

void KMP(std::istream& input, std::ostream& output); // Алгоритм КМП

void prefixFunction(const std::string& srcString, std::vector<int>& prefixFunctionValues) {

prefixFunctionValues[0] = 0; // Инициализируем начальное значение префикс-функции

for (size\_t i = 1; i < srcString.size(); ++i) {

int expectedValue = prefixFunctionValues[i - 1]; // Запоминаем предыдущее значение,

// оно фигурирует в теоретическких расчетах для КМП

while (expectedValue > 0 && srcString[expectedValue] != srcString[i]) // Смотрим в предыдущие значения, пока не встретим 0 или пока не получим совпадение

expectedValue = prefixFunctionValues[expectedValue - 1];// Шаг назад

if (srcString[expectedValue] == srcString[i]) // Если цикл завершился, потому что есть совпадение, то

expectedValue += 1; // увеличиваем на 1 и пишем в ответ. В случае, когда дошли до 0, запишется 0

prefixFunctionValues[i] = expectedValue;

}

}

void KMP(std::istream& input, std::ostream& output) {

std::string substring;

std::cout << "Enter pattern(substring)" << std::endl;

input >> substring; // Считываем шаблон (подстроку)

std::vector<int> prefixFunctionValues(substring.size()); // Вектор для хранения значений префикс-функции

prefixFunction(substring, prefixFunctionValues); // Вычисляем значение префикс-функции

output << "Prefix function values: ";

for (auto i : prefixFunctionValues)

output << i << " ";

output << std::endl;

int expectedValue = 0;

int resultCount = -1; // Если совпадений не обнаружится, то ответ так и останется -1

char current(0); // Для посимвольного считывания строки

std::cout << "Enter the text string" << std::endl;

input.get(current); // Считываем символ переноса строки, оставшийся после ввода шаблона

input.get(current); // Считываем первый символ строки-текста

std::vector<int> result; // Массив с ответом - индексами вхождений

for(int i =0; (current != '\n' && !input.eof()); ++i) { // Пока каретка не дошла до конца входных данных

output << "Current index = " << i << " Expected value = " << expectedValue << std::endl;

while (expectedValue > 0 && current != substring[expectedValue]) { // Пока не совпадут последние символы

expectedValue = prefixFunctionValues[expectedValue - 1]; // Пробуем слудющее значение

output << " Expected value = " << expectedValue << std::endl;

}

if (current == substring[expectedValue]) { // Если есть совпадение

expectedValue += 1; // Засчитываем вхождение

output << " Expected value = " << expectedValue << std::endl;

}

if (expectedValue == substring.size()) { // Если совпали длины совпадений и шаблона

resultCount = i - substring.size() + 1; // Формируем индекс

result.push\_back(resultCount);

output << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

output << "Loop iteration #" << i << " Expected value was (equal to substr length) = " << expectedValue

<< " Found index = " << resultCount << std::endl;

output << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

input.get(current);

}

output << std::endl << "Result: ";

if (!result.empty()) {

for (size\_t i = 0; i < result.size() - 1; ++i) {

output << result[i] << ",";

}

output << result[result.size() - 1] << std::endl;

}

output << "Total entries : " << (int)(((result.size()) != 0) ? (result.size()) : (-1)) << std::endl;

}

int main() {

std::cout << "Choose input mode" << std::endl << std::endl

<< "1 - console" << std::endl

<< "2 - file" << std::endl; // Выбор режима ввода

int mode = 0;

std::cin >> mode;

if (mode == 1) { // Консоль

std::cout << std::endl << "Choose output mode" << std::endl << std::endl

<< "1 - console" << std::endl

<< "2 - file" << std::endl; // Выбор режима вывода

std::cin >> mode;

if (mode == 1) { // Консоль

KMP(std::cin, std::cout);

}

else if (mode == 2) { // Файл

std::ofstream fileout;

fileout.open(outPath);

if (!fileout.is\_open()) {

std::cout << "Can't open file!\n";

}

KMP(std::cin, fileout);

}

}

else if (mode == 2) { // Файл

std::ifstream filein;

filein.open(inPath);

std::cout << std::endl << "Choose output mode" << std::endl << std::endl

<< "1 - console" << std::endl // Выбор режим вывода

<< "2 - file" << std::endl;

std::cin >> mode;

if (mode == 1) { // Консоль

KMP(filein, std::cout);

}

else if (mode == 2) { // Файл

std::ofstream fileout;

fileout.open(outPath);

KMP(filein, fileout);

}

}

return 0;

}