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 «Проектирование и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8382 _____ Нечепуренко Н.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Разработать программу для эффективного поиска подстроки в строке, используя алгоритм Кнута-Морриса-Пратта.

Постановка задачи.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|≤15000) и текста T (|T|≤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:

ab

abab

Пример выходных данных:

0,2.

Индивидуальное задание:

Вар. 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.

Алгоритм Кнута-Морриса-Пратта используется для поиска подстроки в строке за линейное время. Идея заключается в построении для строки префикс-функции, значение которой для символа определяется как длина максимального префикса, оканчивающегося на данной позиции. Положим, что необходимо найти все вхождения подстроки P в строке T. Тогда составим новую строку вида «P@T», где @ – guard, символ, которого нет в алфавите,

обеспечивающий ограничение на максимальное значение префикс функции – длину P. Вычислим значение префикс функции для полученной строки и если на каком-то символе значение функции равно длине P, то это конец вхождения и подстрока найдена. Сделать асимптотику линейной позволяет эффективный пересчёт функции на основе уже имеющихся значений функции.

Реализация алгоритма.

Для хранения значения префикс-функции можно использовать обычный массив или std::vector. В индивидуальном задании необходимо провести оптимизацию по памяти и не хранить полностью строку для поиска, поэтому будем просто хранить текущий символ и текущее значение функции. Вычислим значение префикс функции для искомой подстроки и сохраним std::vector<int> prefixFunction. Затем в цикле читаем очередной символ, вычисляем функцию и, если её значение равно длине подстроки, добавляем в ответ. Полный исходный код расположен в Приложении А.

Для решения задачи о проверки, является ли одна из строк циклическим сдвигом другой, необходимо удвоить одну из строк. Полученная строка будет содержать все циклические сдвиги исходной. Затем просто применяем алгоритм КМП. Исходный код находится в Приложении Б.

Оценка сложности.

Можно доказать, что вычисление префикс-функции для строки выполняется за O(n), где n – длина строки. Таким образом, асимптотическая сложность алгоритма КМП – O(m + n), где m и n – длины строки и подстроки.

Тестирование.

Проведём тестирование программы на нескольких наборах входных данных. Результаты представлены в таблице 1.

Таблица 1 – Тестирование программы

Входные данные	Выходные данные
acab abacababac	Input pattern: acab Prefix function for pattern is: 0 0 1 0 Input string: abacababa 2
axxa xxaaxxaxxaaaxxaa	Input pattern: axxa Prefix function for pattern is: 0 0 0 1 Input string: xxaaxxaxxaaaxxaa 3,6,11

Программа успешно прошла тестирование.

Вывод.

В результате выполнения работы был реализован алгоритм поиска подстроки в строке Кнута-Морриса-Пратта, проанализирована его асимптотика. Была применена оптимизация по памяти, а также решена задача о проверки строк и их циклических сдвигов.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

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

int main(){
    /*
    Читаем подстроку и строим массив значений префикс функции для неё
    */
    std::string needle;
    std::cout << "Input pattern:" << std::endl;
    std::cin >> needle;
    int needleSize = (int) needle.size();
    std::vector<int> prefixFunction(needleSize, 0);
    for (int i = 1; i < needleSize; i++){
        int k = prefixFunction[i-1];
        while (k > 0 && needle[k] != needle[i]){
            k = prefixFunction[k-1];
        }
        if (needle[i] == needle[k])
            k++;
        prefixFunction[i] = k;
    }

    /*
    Вывод полученной префикс-функции
    */
    std::cout << "Prefix function for pattern is:" << std::endl;
    for (const auto& value : prefixFunction){
        std::cout << value << " ";
    } std::cout << std::endl;

    //std::cout << "Input length of string cmp search in:" << std::endl;
    //int haystackSize = 0;
    //std::cin >> haystackSize;
    std::cout << "Input string:" << std::endl;

    /*
    Считываем очередной символ из буффера, ищем максимальный префикс;
    оптимизированная часть практически не отличается от обычного кмп
```

```
        */
        std::vector<int> answer;
        char currentSymbol = 65;
        int k = 0;
        int iteration = 0;
        std::cin.get();
        while(std::cin.get(currentSymbol) && currentSymbol != ' ' &&
currentSymbol != '\n') {
            while (k > 0 && currentSymbol != needle[k]) // поиск
                k = prefixFunction[k-1];                // max. префикса
            if (currentSymbol == needle[k]) // префиксы равны, увеличиваем
значение
                k++;
            if (k == needleSize) // нашли конец вхождения
                answer.push_back(iteration - needleSize + 1);
            iteration++;
        }

        /*
        Выводим ответ
        */
        int answerSize = (int) answer.size();
        if (answerSize == 0){
            std::cout << "-1" << std::endl;
            return 0;
        }
        for (int i = 0; i < answerSize; i++){
            std::cout << answer[i];
            if (i + 1 != answerSize)
                std::cout << ",";
        }
        std::cout << std::endl;

        return 0;
    }
```

ПРИЛОЖЕНИЕ Б. ПРОВЕРКА СТРОК И ЦИКЛОВ.

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

int main(){
    std::string needle, haystack;
    std::cin >> needle;
    std::cin >> haystack;
    std::swap(needle, haystack);
    int needleSize = (int) needle.size();
    int haystackSize = (int) haystack.size();
    std::string str = needle + "@" + haystack;
    int expLen = needleSize+1+2*haystackSize;
    needle.clear();
    haystack.clear();
    std::vector<int> prefixFunction(expLen, 0);
    for (int i = 1; i < expLen; i++){
        int idx = (i >= needleSize + 1 + haystackSize) ? i - haystackSize
: i;
        int k = prefixFunction[i-1];
        while (k > 0 && str[k] != str[idx]){
            k = prefixFunction[k-1];
        }
        if (str[idx] == str[k])
            k++;
        prefixFunction[i] = k;
    }
    for (int i = needleSize; i < expLen; i++){
        if (prefixFunction[i] == needleSize){
            std::cout << i - needleSize - haystackSize << std::endl;
            return 0;
        }
    }
    /*for (auto el : prefixFunction) {
        std::cout << el << " ";
    }std::cout << std::endl;*/
    std::cout << -1 << std::endl;
    return 0;
}
```