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.

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.

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.

2. Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

Определить, является ли AA циклическим сдвигом BB (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 вляется 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

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, сложность алгоритма КМП – O(m + n), где m и n – длины строки и подстроки, по времени и O(m+n) по памяти без оптимизаций или O(n) с оптимизацией.

Для проверки, является ли одна строка циклическим сдвигом другой, асимптотика будет такой же, так как константа 2 игнорируется по определению O-большого. Получаем O(m+n) по времени и O(m+n) по памяти.

Тестирование.

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

Таблица 1 – Тестирование первой программы

Входные данные	Выходные данные
acab abacababac	Input pattern: acab abacababac Build for pi function for 1 index, c symbol Candidate is: 0 Got prefixFunction[1] = 0 Build for pi function for 2 index, a symbol Candidate is: 0 a == a, at pos 0, increase value Got prefixFunction[2] = 1 Build for pi function for 3 index, b symbol Candidate is: 1 b != c, at pos 1, next candidate is: 0

	Got prefixFunction[3] = 0 Prefix function for pattern is: 0 0 1 0 Input string: 2
axxa xxaaxxaxxaaaxxaa	Input pattern: axxa xxaaxxaxxaaaxxaa Build for pi function for 1 index, x symbol Candidate is: 0 Got prefixFunction[1] = 0 Build for pi function for 2 index, x symbol Candidate is: 0 Got prefixFunction[2] = 0 Build for pi function for 3 index, a symbol Candidate is: 0 a == a, at pos 0, increase value Got prefixFunction[3] = 1 Prefix function for pattern is: 0 0 0 1 Input string: 3,6,11

Проведём тестирование программы по определению циклических сдвигов строки. Результат представлен в таблице 2.

Таблица 2 – Тестирование второй программы

Входные данные	Выходные данные
abacaba bacabaa	abacaba bacabaa Build for pi function for 1 index, a symbol Candidate is: 0 Got prefixFunction[1] = 0 Build for pi function for 2 index, c

	symbol Candidate is: 0 Got prefixFunction[2] = 0 Build for pi function for 3 index, a symbol Candidate is: 0 Got prefixFunction[3] = 0 Build for pi function for 4 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[4] = 1 Build for pi function for 5 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[5] = 2 Build for pi function for 6 index, a symbol Candidate is: 2 a != c, at pos 2, next candidate is: 0 Got prefixFunction[6] = 0 Build for pi function for 7 index, @ symbol Candidate is: 0 Got prefixFunction[7] = 0 Build for pi function for 8 index, a symbol Candidate is: 0 Got prefixFunction[8] = 0 Build for pi function for 9 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[9] = 1 Build for pi function for 10 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[10] = 2 Build for pi function for 11 index, c symbol Candidate is: 2 c == c, at pos 2, increase value

	Got prefixFunction[11] = 3 Build for pi function for 12 index, a symbol Candidate is: 3 a == a, at pos 3, increase value Got prefixFunction[12] = 4 Build for pi function for 13 index, b symbol Candidate is: 4 b == b, at pos 4, increase value Got prefixFunction[13] = 5 Build for pi function for 14 index, a symbol Candidate is: 5 a == a, at pos 5, increase value Got prefixFunction[14] = 6 Build for pi function for 15 index, a symbol Candidate is: 6 a == a, at pos 6, increase value Got prefixFunction[15] = 7 Build for pi function for 16 index, b symbol Candidate is: 7 b != @, at pos 7, next candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[16] = 1 Build for pi function for 17 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[17] = 2 Build for pi function for 18 index, c symbol Candidate is: 2 c == c, at pos 2, increase value Got prefixFunction[18] = 3 Build for pi function for 19 index, a symbol Candidate is: 3 a == a, at pos 3, increase value Got prefixFunction[19] = 4 Build for pi function for 20 index, b symbol

	Candidate is: 4 b == b, at pos 4, increase value Got prefixFunction[20] = 5 Build for pi function for 21 index, a symbol Candidate is: 5 a == a, at pos 5, increase value Got prefixFunction[21] = 6 Prefix function for merged string is: 0 0 0 0 1 2 0 0 0 1 2 3 4 5 6 7 1 2 3 4 5 6 Rotated on: 1
abacaba bacbbaa	abacaba bacbbaa Build for pi function for 1 index, a symbol Candidate is: 0 Got prefixFunction[1] = 0 Build for pi function for 2 index, c symbol Candidate is: 0 Got prefixFunction[2] = 0 Build for pi function for 3 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[3] = 1 Build for pi function for 4 index, b symbol Candidate is: 1 b != a, at pos 1, next candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[4] = 1 Build for pi function for 5 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[5] = 2 Build for pi function for 6 index, a symbol Candidate is: 2 a != c, at pos 2, next candidate is: 0 Got prefixFunction[6] = 0 Build for pi function for 7 index, @ symbol

	Candidate is: 0 Got prefixFunction[7] = 0 Build for pi function for 8 index, a symbol Candidate is: 0 Got prefixFunction[8] = 0 Build for pi function for 9 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[9] = 1 Build for pi function for 10 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[10] = 2 Build for pi function for 11 index, c symbol Candidate is: 2 c == c, at pos 2, increase value Got prefixFunction[11] = 3 Build for pi function for 12 index, a symbol Candidate is: 3 a != b, at pos 3, next candidate is: 0 Got prefixFunction[12] = 0 Build for pi function for 13 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[13] = 1 Build for pi function for 14 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[14] = 2 Build for pi function for 15 index, a symbol Candidate is: 2 a != c, at pos 2, next candidate is: 0 Got prefixFunction[15] = 0 Build for pi function for 16 index, b symbol Candidate is: 0

	b == b, at pos 0, increase value Got prefixFunction[16] = 1 Build for pi function for 17 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[17] = 2 Build for pi function for 18 index, c symbol Candidate is: 2 c == c, at pos 2, increase value Got prefixFunction[18] = 3 Build for pi function for 19 index, a symbol Candidate is: 3 a != b, at pos 3, next candidate is: 0 Got prefixFunction[19] = 0 Build for pi function for 20 index, b symbol Candidate is: 0 b == b, at pos 0, increase value Got prefixFunction[20] = 1 Build for pi function for 21 index, a symbol Candidate is: 1 a == a, at pos 1, increase value Got prefixFunction[21] = 2 Prefix function for merged string is: 0 0 0 1 1 2 0 0 0 1 2 3 0 1 2 0 1 2 3 0 1 2 No rotation: -1

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++){
        std::cout << "Build for pi function for " << i << " index, " <<
needle[i] << " symbol\n";
        int k = prefixFunction[i-1];
        std::cout << "Candidate is: " << k << std::endl;
        while (k > 0 && needle[k] != needle[i]){
            std::cout << needle[i] << " != " << needle[k] << ", at pos "
<< k << ", next candidate is: ";
            k = prefixFunction[k-1];
            std::cout << k << std::endl;
        }
        if (needle[i] == needle[k]) {
            std::cout << needle[i] << " == " << needle[k] << ", at pos "
<< k << ", increase value\n";
            k++;
        }
        prefixFunction[i] = k;
        std::cout << "Got prefixFunction[" << i << "] = " << k <<
std::endl;
    }
    /*
    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 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;
        std::cout << "Build for pi function for " << i << " index, " <<
str[idx] << " symbol\n";
        int k = prefixFunction[i-1];
        std::cout << "Candidate is: " << k << std::endl;
        while (k > 0 && str[k] != str[idx]){
            std::cout << str[i] << " != " << str[k] << ", at pos " << k <<
", next candidate is: ";
            k = prefixFunction[k-1];
            std::cout << k << std::endl;
        }
        if (str[idx] == str[k]) {
            std::cout << str[i] << " == " << str[k] << ", at pos " << k <<
", increase value\n";
            k++;
        }
        prefixFunction[i] = k;
        std::cout << "Got prefixFunction[" << i << "] = " << k <<
std::endl;
    }
```

```
            std::cout << "Prefix function for merged string is:" << std::endl;
            for (auto el : prefixFunction) {
                std::cout << el << " ";
            }std::cout << std::endl;
            for (int i = needleSize; i < expLen; i++){
                if (prefixFunction[i] == needleSize){
                    std::cout << "Rotated on: " << i - needleSize - haystackSize
<< std::endl;
                    return 0;
                }
            }
            std::cout << "No rotation: " << -1 << std::endl;

            return 0;
        }
```