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 Кнутта-Морриса-Пратта

Студент гр. 8382 _____ Ершов М.И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучение работы алгоритма Кнутта-Морриса-Пратта для поиска подстроки в строке, а также изучение способа нахождения значений префикс-функции.

Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 1500$) 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

Sample Input:

ab

abab

Sample Output:

0,2

Задание 2

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:

Индивидуализация

Вариант 2

Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца

Описание префикс-функции

Префикс-функция F(S, i) (в дальнейшем F(i)) – это функция, которая по заданному индексу строки, возвращает длину максимального суффикса, который равен префиксу в срезе строки до переданного индекса.

Пусть дана строка S. Необходимо вычислить F(i):

1) Считывать значения префикс-функции F(i) будем по очереди: от i=1 к i=n - 1 (значение F(0) просто присвоим равным нулю).
2) Для подсчёта текущего значения F(i) мы заводим переменную k, обозначающую длину текущего рассматриваемого образца. Изначально k = F(i-1).
3) Тестируем образец длины k, для чего сравниваем символы S[k] и S[i]. Если они совпадают — то полагаем F(i) = k + 1 и переходим к следующему индексу i + 1. Если же символы отличаются, то уменьшаем длину k, полагая её равной F(k-1), и повторяем этот шаг алгоритма с начала.
4) Если мы дошли до длины k = 0 и так и не нашли совпадения, то останавливаем процесс перебора образцов и полагаем F(i) = 0 и переходим к следующему индексу i +1.

Описание алгоритма задания 1

В программе используется алгоритм Кнутта-Морриса-Пратта. Он заключается в том, что для образа (P) выполняется вычисление значений префикс-функции, после чего эти значения используются при сравнении P и T по следующему правилу:

1. Сравниваем P[k] и K[i] символы P и T (изначально k=0, i = 0).
2. Если P[k] равно T[i], то сравниваем P[k+1] и T[k+1].

3. Если P[k] не равно T[i], то в массиве значений префикс-функций ищем предшествующий P[i] символ и устанавливаем значение k, равное значению префикс-функции предшествующего символа. Продолжаем с шага 1.
4. В случае, если k становится равен длине образа, то мы нашли вхождение – записываем индекс вхождения в результат и обнуляем k.

Описание алгоритма задания 2

Задание 2 сводим к заданию 1, удваивая циклически сдвинутую строку. После чего, ищем вхождение исходной строки в удвоенной, используя алгоритм задания 1.

Сложность алгоритма

Сложность алгоритма вычисления префикс функции – $O(n)$, где n – длина строки. Обусловлена тем, что K (значение префикс-функции) на каждой итерации цикла либо увеличивается на 1, либо обнуляется.

Сложность алгоритма 1 по времени составляет - $O(m + n)$, где m - длина P, а n – длина T.

Сложность алгоритма 2 по времени составляет – $O(n)$ (аналогично алгоритму 1)

Сложность по памяти для обоих алгоритмов составляет – $O(m)$, где m – длина образа. Эта сложность обусловлена тем, что для вычисления i-го значения префикс-функции, ей необходимо значение $F(S, i - 1)$ и первые m элементов префикс-функции (нет необходимости сохранять значение префикс функции, которое превышает длину паттерна).

Описание функций и структур данных

Class FullString – структура, для хранения склейки строк без использования доп. памяти (использование ссылок на них).

Поля FullString:

- const string &smp - паттерн;
- const string &dst - текст;

Методы FullString:

- FullString(const string &sample, const string &dest) – конструктор для заполнения образа и строки, в которой ведётся поиск;

- char operator[](size_t i)– возвращает по заданному индексу значение строки p + t;

Class PrefixFunction – структура, для вычисления префикс функции от образа.

Поля PrefixFunction:

- ConcatStrings full_str – строка, склейка образа и строки;
- size_t sample_size – длина паттерна;
- size_t dst_index – текущий индекс;
- vector <size_t> prefixes – вектор, в котором хранятся значения префикс-функции образца;

Методы PrefixFunction:

- PrefixFunction(const string &sample, const string &dest) – конструктор, в котором вычисляется значение префикс-функции образца;
- size_t nextValue()– метод, возвращающая следующее значение префикс-функции;

Функции main.cpp:

- vector <size_t> getSubstringIndices(const string &sample, const string &dest) – функция, возвращающая вектор индексов, на которых sample встречается в dest.
- int getIndexShifted(const string &a, const string &b) – функция, возвращающая индекс, на котором a встречается в b + b.

Тестирование

Задание 1	
Ввод	Вывод
kbk kbkkbkbbakbabababk	0,3
abc aaaaaaaabc	7
baba abbabbabba	-1
aaak aabaaaaakabaaaaa	5
abbaa abbabbvabbaabbaahsabaavas	7,11
Задание 2	
defabc abcdef	3
giabcdefh	-1

abcdefghi	
leapp appel	-1

Вывод

В ходе выполнения лабораторной работы была изучена работа алгоритма Кнутта-Морриса-Пратта для нахождения подстроки в строке, а также префикс-функция для нахождения наибольшего суффикса равного префиксу в срезе строки.

Приложение А

Исходный код main.cpp

```
#include <iostream>
#include <vector>
#include <cstdio>

using namespace std;

// Класс для "склеивания" строк
class FullString {
    const string &smp;
    const string &dst;
public:
    // Конструктор
    FullString(const string &sample, const string &dest) : smp(sample), dst(dest)
{}

    // Доступ к "склеенной" строке через оператор []
    char operator[](size_t i) {
        if (i < smp.size()) {
//            cout << "p[" << i << "]: " << sbstr[i] << endl;
            return smp[i];
        }
//        cout << "t[(" << i << " - " << sbstr.size() << ") % " << dst.size() <<
"]: " << dst[(i - sbstr.size()) % dst.size()] << endl;
        return dst[(i - smp.size()) % dst.size()];
//        return t[(i - p.size()) % t.size()];
    }
};

class PrefixFunction {
    ConcatStrings full_str; // Склеенные sample и dest
    size_t sample_size; // Размер образца
    size_t dst_index; // Текущий индекс строки
    vector<size_t> prefixes; // Значения префикс-функции для образца
public:
    // Функция, рассчитывающая значения префикс-фукнции для образца
    PrefixFunction(const string& sample, const string& dest) : full_str(sample,
dest),

sample_size(sample.size()),

dst_index(sample.size())
    {
        cout << "Вычисляем значения префикс-функции для образца" << endl;
        prefixes.resize(sample_size); // Выделяем память под размер паттерна
        int i = 0;
        cout << sample.substr(0, i+1) << ": 0" << endl;
        prefixes[i++] = 0; // Для первого элемента значение префикс-функции - 0

        while (i < sample_size) {
            auto k = prefixes[i - 1]; // Берем предыдущее значение

            while (k > 0 && sample[i] != sample[k]) // Пока конец суффикса !=
концу префикса
                k = prefixes[k - 1]; // Сохраняем предыдущее значение префикс
функции

            // Конец префикса == концу суффикса - увеличиваем предыдущее значение
на 1, иначе обнуляем
            prefixes[i] = (sample[i] == sample[k]) ? k + 1 : 0;
            cout << sample.substr(0, i+1) << ": " << prefixes[i] << endl;
```

```
            i++;
        }
        cout << endl;
    }

    size_t nextValue() {
        static size_t sample_index = 0; // static - сохраняет свое значение после
выхода из блока

        if (full_str[dst_index] != full_str[sample_index]) {
            cout << "string_index: " << dst_index << ", sample_index: " <<
sample_index << "; ";
            cout << full_str[dst_index] << "!=" <<  full_str[sample_index] <<
endl;
            cout << "Ищем новую позицию для продолжения поиска" << endl;
        }
        while (sample_index > 0 && full_str[dst_index] != full_str[sample_index])
{// Пока конец суффикса != концу префикса
            cout  << "sample_index - было: " << sample_index << ", стало: ";
            int tmp = sample_index - 1;
            sample_index = prefixes[sample_index - 1]; // // Сохраняем предыдущее
значение префикс функции
            cout << sample_index << " (prefixes[" << tmp << "])" << endl;
            if (full_str[dst_index] == full_str[sample_index]) {
                cout << "string_index: " << dst_index << ", sample_index: " <<
sample_index << "; ";
                cout << full_str[dst_index] << "==" <<  full_str[sample_index] <<
endl;
                cout << "Позиция найдена, продолжаем!" << endl;
                cout << endl;
            }
        }
        if (full_str[dst_index] != full_str[sample_index]) {
            cout << "Вернулись в начало образца, продолжаем поиски" << endl;
            cout << endl;
        }

        // Конец префикса == концу суффикса - увеличиваем предыдущее значение на 1
        if (full_str[dst_index] == full_str[sample_index]) {
            cout << "string_index: " << dst_index << ", sample_index: " <<
sample_index << "; ";
            cout << full_str[dst_index] << "==" << full_str[sample_index] << endl;
            sample_index++;
        }
        dst_index++;

        // Дошли до конца образца - вхождение найдено!
        if (sample_index == sample_size) {
            cout << "Вхождение найдено!" << endl;
            cout << "sample_prefix = " << prefixes[sample_index - 1] << "
(prefixes[" << sample_index - 1 << "])";
            cout << endl;
            sample_index = prefixes[sample_index - 1];
            return sample_size;
        }

        cout << "Символы совпали, проверяем следующие" << endl;
        return sample_index;
    }
};

vector<size_t> getIndices(const string &sample, const string &dest) {
    PrefixFunction prefixFunction(sample, dest);
    vector<size_t> result;
```

```
    const string str = sample + dest;

    int i = 0;
    cout << "Начинаем искать индексы вхождений" << endl;
    while (i < dest.size()) {
        size_t sub_position = prefixFunction.nextValue();

        // Если дошли до конца образца - вхождение найдено, пушим в result
        if (sub_position == sample.size())
            result.push_back(i - sample.size() + 1); // i - pi(k) + 1

        i++;
    }

    return result;
}


int getIndexShifted(const string &a, const string &b)
{
    if (a.size() != b.size()) return -1;
    PrefixFunction prefixFunction(a, b);

    const string str = a + b + b;

    int i = 0;
    while (i < 2*b.size()) {
        size_t sub_position = prefixFunction.nextValue();

        // Дошли до конца A - значит, B является циклическим сдвигом A, возвращаем
индекс
        if (sub_position == a.size()) return int(i - a.size() + 1); // i - pi(k) +
1
        i++;
    }

    return -1;
}

int main()
{
    system("chcp 65001");

    string p, t;
    cin >> p >> t;

    auto result = getIndices(p, t);
//    auto result_shift = getIndexShifted(t, p);

    if (result.empty()) cout << -1;
    else {
        int i = 0;
        while (i < result.size())
            cout << result[i] << ((++i != result.size()) ? ',' : ' ');

    }
//    cout << result_shift << endl;

    return 0;
}
```