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. 8303 Колосова М.П,
Преподаватель Фирсов М.А.

Санкт-Петербург
2020

Цель работы

Изучить принцип работы алгоритма КМП для поиска подстроки в строке с помощью реализации данного алгоритма.

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

Sample Input:

ab

abab

Sample Output:

0,2

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:

3

Индивидуализация

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками.

Описание алгоритма Кнута-Морриса-Пратта

В начале алгоритма для образца вычисляется префикс-функция, т.е. массив чисел $\pi[0 \ldots n-1]$: $\pi[i]$ хранит значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом. В частности, значение $\pi[0]$ полагается равным нулю.

Для единственного потока алгоритм работает следующим образом. При каждом совпадении i и j символов оба индекса увеличиваются на 1, i — индекс в тексте, j — индекс в шаблоне. Если j становится равной размеру строки-шаблона, то все символы строки-шаблона совпали, и вхождение найдено. Если очередной символ не совпал с i-ым символом, то, если j = 0, индекс i увеличивается на один, в обратном случае (j =/= 0) алгоритм обращается к префикс-функции и приравнивает j к значению префикс-функции для символа, предшествующего не совпавшему.

При разделении на потоки вместо всей строки-текста рассматриваются ее подстроки. Подстрока строится следующим образом. Сначала вычисляется flowLen - длина части текста. Она равняется частному от деления длины всего текста на количество потоков с округлением вверх, стартовая позиция строки потока определяется как f*flowLen, где f — номер потока. В худшем случае окажется, что последний символ части — первый символ вхождения и почти вся его длина находится в следующей части. Конечная позиция строки потока во всем тексте определяется сдвигом стартовой позиции на длину flowLen и прибавлением величины p-1, где p — длина шаблона, для учета вхождений на стыке частей. Для последнего потока конечная позиция — это конец строки-

текста. Повтор одного и того же индекса исключается тем, что при добавлении p-1 невозможно захватить новое вхождение.

Описание алгоритма проверки циклического сдвига

На вход алгоритму подается две строки и количество потоков, нужно проверить является ли вторая циклическим сдвигом первой. В начале алгоритма проверяется равенство длин строк. Если строки цикличны, то при склеивании двух копий одной строки, в новой строке будет вхождение второй строки. Тогда для поиска индекса логично использовать алгоритм КМП, которому на вход подается количество потоков, вторая строка как образец и новая строка-текст, полученная путем конкатенации первой строки два раза.

Описание функций

`void KMP(std::string& t, std::string& p, size_t flowCount, bool isForShift = false)` – реализация алгоритма КМП
t — строка-текст для поиска вхождений, p — строка-образец вхождения, флаг isForShift — для остановки поиска после первого вхождения, `pi` - вектор со значениями префикс-функции, flowCount — количество потоков

void **prefixFunc**(std::string& p, std::vector<size_t>& pi) — вычисление префикс-функции
p - строка-образец, pi — вектор для хранения значений префикс-функции

void **cyclicShift**(std::string& strA, std::string& strB, size_t flowCount) — функция, определяющая циклический сдвиг
strA — возможный циклический сдвиг строки strB, strB — строка, циклический сдвиг которой надо определить, flowCount — количество потоков

Описание структур данных

vector<size_t> pi — вектор для хранения значений префикс-функции образца
vector<size_t> res — вектор для хранения индексов вхождений

Сложность алгоритмов по времени

Префикс-функция образца вычисляется за o(m), где m - длина шаблона. Поиск вхождения образца в текст происходит за один цикл, максимальная длина которого равняется длине строки-текста, тогда сложность поиска — o(n), где n — эта длина. Такая справедлива оценка o(n+m).

Для второй задачи сложность алгоритма поиска циклического сдвига равна o(3n), где n - длина строки, потому что в процессе вычисляется префикс-функция для строки длины n и выполняется поиск в строке длины 2n.

Сложность алгоритмов по памяти

Для работы алгоритм вычисляет значение префикс функции для каждого символа образца и хранит эти значения в векторе, поэтому сложность алгоритма по памяти составляет o(m), где m - длина строки образца.

Тестирование

№1 Тестирования алгоритма Кнута-Морриса-Пратта

№	Input	Output
1	1 ab abab	0,2
2	2 ab abab	0, 2
3	2 ab ababfgkabf	0, 2, 7
4	5 ab ababfgkabf	0, 2, 7
5	3 aaa aaaaaaaaaaa	0,1,2,3,4,5,6,7,8
6	2 aaa aaaaaaaaaaa	0,1,2,3,4,5,6,7,8
7	2 abf aaabfgab	2
8	3	2

	abf aaabfgab	
9	1 abf aaabfgab	2
10	10 a atgklmaapr	0,6,7
11	1 a atgklmaapr	0,6,7

№2 Тестирования алгоритма поиска циклического сдвига

Корректность работы потоков была проверена при тестировании алгоритма КМП. Для удобства тестирование алгоритм поиска циклического сдвига произведено с единственным потоком.

№	Input	Output
1	1 defabc abcdef	3
2	1 aaa aaa	0
3	1 dede dede	0
4	1 abcdef klmnop	-1
5	1 babbb bbabba	-1

Вывод

В ходе выполнения лабораторной работы был реализован алгоритм КМП для поиска подстроки в строке, а также алгоритм поиска циклического сдвига на базе алгоритма КМП.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

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

using namespace std;

//префикс-функция
void prefixFunc(std::string& p, std::vector<size_t>& pi){
    size_t j = 0;
    pi[0] = 0;

    for(size_t i = 1;i < p.size();){
        if(p[j] == p[i]){
            pi[i] = j+1;
            j++;
            i++;
        }
        else{
            if(j == 0){
                pi[i]= 0;
                i++;
            }
            else{
                j = pi[j-1];
            }
        }
    }

}

//КМП-алгоритм
void KMP(std::string& t, std::string& p, size_t flowCount, bool isForShift =
false){

    vector<size_t> res;
    vector<size_t> pi (p.size());
    prefixFunc(p, pi);
    cout << endl;
    cout << "Значение префикс функции для образца:" <<endl;
    for(auto &el : pi){
        cout << el << " ";
    }
    cout << endl <<endl;

    size_t j, start, end;
    cout << "В алгоритме используется максимально возможное количество потоков."
<< endl;
    if(t.length() / flowCount < p.length())
            flowCount = t.length() / p.length();

    for(size_t f = 0; f < flowCount; f++){

        cout << "Обработка потока №" << f + 1 << ": ";
        j = 0;
        size_t r = t.length() % flowCount;
        size_t flowLen = t.length() / flowCount;
        if(r > 0)
            flowLen++;
```

```
        start = f * flowLen;//выделяем начальную позицию для текущего потока
        end = start + flowLen;//выделяем конечную позицию для текущего потока
        end += p.length() - 1;
        bool isFound = false;
        if(end > t.size())
            end = t.size();
        for(size_t s = start; s < end; s++)
            cout<<t[s];
        cout << endl << endl;

        for(size_t i = start; i < end;){
            for (size_t k = start; k < i; k++) {
                cout << t[k];
            }
            cout << " (" << t[i] << ")";
            for(size_t k = i + 1; k < end; k++) {
                cout << t[k];
            }
            cout << endl;
            string shift = "";
            for(size_t k = 0; k < i - start - j + 1; k++){
                shift += " ";
            }

            cout << shift;
            for(size_t k = 0; k < p.size(); k++){
                if( k == j){
                    cout << "(" << p[k] << ")";
                }
                else cout << p[k];
            }

            if (t[i] == p[j]) {
                cout << endl << endl;
                i++;
                j++;
                if(j == p.size()){
                    cout << "Найдено вхождение образца" << endl;
                    cout << "Индекс вхождения: ";
                    cout << i - p.size() << endl << endl;
                    res.push_back(i - p.size());
                    isFound = true;
                    j = pi[j - 1];
                    if(isForShift) {
                        cout<<endl;
                        break;
                    }
                }
            }
            else{
                cout << " - символы не совпадают." << endl << endl;
                if(j == 0) {
                    i++;
                }
                else {
                    j = pi[j-1];
                }
            }
        }
        cout<<endl;
        if(isFound == false && f == flowCount - 1 && res.empty()) {
            cout << "Вхождений не найдено" << endl;;
            cout<<-1<<endl;
            return;
        }
    }
```

```
    if(res.empty()) {
        cout << "Вхождений не найдено" <<endl;
        cout<<-1<<endl;
        return;
    }

    cout << "\nРезультат работы программы:" << endl;
    string sep = "";
    for(auto &el : res){
        cout << sep << el;
        sep = ",";
    }
    cout<<endl;

}

void cyclicShift(std::string& strA, std::string& strB, size_t flowCount){

    cout << "Строки равны:" << endl;
    if(strA == strB){
        std::cout<<0<<std::endl;
        return;
    }
    cout << "Строки разного размера:" << endl;
    if(strA.size() != strB.size()){
        std::cout<<-1<<std::endl;
        return;
    }
    strA += strA;
    KMP(strA, strB, flowCount, true);

}

int main() {
    string p, t;
    size_t flowCount;
    cin>>flowCount;
    cin>>p;
    cin>>t;
    KMP(t, p, flowCount);
    //cyclicShift(p, t, flowCount);
    return 0;
}
```