МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: " Алгоритм Кнута-Морриса-Пратта ".

Студентка гр. 8383 _____ Аверина О.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться реализовывать алгоритм Кнута-Морриса-Прата и реализовать с его помощью программу для поиска вхождений подстроки и поиска циклического сдвига строки.

Задание 1.

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

Sample Input:

ababab

Sample Output:

0,2

Задание 2.

Заданы две строки *A* (|*A*| ≤ 5000000) и *B* (|*B*| ≤ 5000000).

Определить, является ли *A* 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 -1−1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

defabcabcdef

Sample Output:

3

Индивидуализации для лаб. работы № 4:

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.

Основные теоретические положения.

Префикс-функция - это функция, которая принимает строку и возвращает максимальную длину ее префикса, совпадающего с суффиксом. Тривиальные случаи (префикс равен суффиксу и равен всей строке) не учитываются.

Алгоритм Кнута-Морриса-Пратта — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Описание алгоритма

Дана цепочка T и образец P. Требуется найти все позиции, начиная с которых P входит в T.

Построим строку S=P#T, где #— любой символ, не входящий в алфавит P и T. Посчитаем на ней значение префикс-функции p. Благодаря разделительному символу #, выполняется $\forall i$: $p[i] \leqslant |P|$. Заметим, что по определению префикс-функции при $i > |P|$ и $p[i] = |P|$ подстроки длины P, начинающиеся с позиций 0 и $i-|P|+1$, совпадают. Соберем все такие позиции $i-|P|+1$ строки S, вычтем из каждой позиции $|P|+1$, это и будет ответ. Другими словами, если в какой-то позиции i выполняется условие $p[i]=|P|$, то в этой позиции начинается очередное вхождение образца в цепочку.

Реализация алгоритма Кнута — Морриса — Пратта.

Оптимизация:

Для оптимизации алгоритма по памяти вместо того, чтобы склеивать образец(шаблон) и строку, были вычислены значения префикс-функции для префиксов образца и записаны в вектор. Дальнейшие действия будут производиться в соответствии со стандартным алгоритмом КМП, но префикс всегда будет меньше или равен образцу, разделителем выступает символ ‘\0’, который больше не встречается в строке.

Алгоритм поиска подстрок:

1. Считываются образец *T* и строка *S*.
2. Вычисляется префикс-функция для всех префиксов образца.
   2.1. Для первого элемента она равна 0. Для всех последующих сравниваются текущий символ *T*[*i*] и символ, находящийся после элемента, индекс которого равен префикс-функция элемента *T*[*i* - 1].
      2.1.1. Если они совпадают, префикс-функция *T*[*i*] равна префикс-функции *T*[*i* - 1] + 1.
      2.1.2. Если они не совпадают, элементом для сравнения становится символ, следующий за индексом префикс-функции от префикс функции *T*[*i* - 1].
3. Запускается алгоритм КМП.
   3.1. Для каждого префикса строки производится сравнение с префиксом образца таким же образом, как и в префикс-функции(пункт 2.1), но в качестве префикса выступает образец. Значение префикс-функции не может превышать длину образца.
   3.2. Если встречается значение функции, равное длине образца, индекс начала вхождения (*i* – *n* +1, где *n* – длина образца) добавляется в вектор найденных индексов.
   3.3. Если не встречается ни одного вхождения образца в строку, выводится -1.

4. Значения найденных индексов строки найденных индексов строки выводятся в консоль.

Код программы содержит следующие методы:

1) void prefix(*string *T, vector<int> *max_pref*) - префикс - функция для вычисления максимальной длины префикса, равного суффиксу в подстроке для каждого элемента.

Входное значение:

– *vector<int> *max_pref* - указатель на вектор значений префикс-функции.
– string *T - указатель на образец.

2) void KMP(*string* S, string *T, vector<int> *answer*) – функция для вычисления алгоритма КМП.

– *string *S* - указатель на строку.
– *string *T* - указатель на образец.
– *vector<int> *answer* - указатель на вектор индексов найденных вхождений образца.

3) *void inputFile*(*string *S, string *T*) - функция для ввода данных из файла.

– *string *S* - указатель на строку.
– *string *T* - указатель на образец.

Алгоритм определения циклического сдвига строки:

1. Считываются образец *T* и строка *S*.
2. Вычисляется префикс-функция для всех префиксов образца.

2.1. Для первого элемента она равна 0. Для всех последующих сравниваются текущий символ *T*[*i*] и символ, находящийся после элемента, индекс которого равен префикс-функции элемента *T* [*i* - 1].

2.1.1. Если они совпадают, префикс-функция $T[i]$ равна префикс-функции $T[i - 1] + 1$.

2.1.2. Если они не совпадают, элементом для сравнения становится символ, следующий за индексом префикс-функции от префикс функции $T[i - 1]$.

3. Запускается алгоритм определения циклического сдвига.

3.1. Для каждого элемента производится сравнение с префиксом образца таким же образом, как и в префикс-функции(пункт 2.1), но в качестве префикса выступает образец. Значение префикс-функции не может превышать длину образца.

3.1.1. Если был рассмотрен последний префикс строки и значение префикс-функции в нем больше нуля, значит сохраняем индекс, с которого началось вхождение исходной строки (*start_i*). Выходим из цикла.

3.1.2. Если не встречается ни одного вхождения образца в строку, выводится -1.

3.1.3. Если было найдено значение префикс-функции, равное длине исходной строки, значит проверяемая строка равна исходной и функция возвращает индекс ее начала.

3.2. Проверяем с начала префиксы строки от 0 до индекса вхождения(*start_i*).

3.3. Если значение префикс-функции оказывается равно длине исходной строки, значит циклический сдвиг найден. Функция возвращает индекс *start_i*.

3.4. Иначе возвращает -1.

4. Индекс выводится в консоль.

Код программы содержит следующие методы:

1) *void prefix*(*string *T, vector<int> *max_pref*) - префикс - функция для вычисления максимальной длины префикса, равного суффиксу в подстроке для каждого элемента.

Входное значение:

– *vector<int> *max_pref* - указатель на вектор значений префикс-функции.
– string *T - указатель на образец.

2) *int offset*(*string* B, string *A*) – функция для определения циклического сдвига в строке.
– *string *A*- указатель на проверяемую строку.
– *string *B* - указатель на исходную строку.

Оценка сложности алгоритма поиска вхождений подстроки.

Сложность по времени: O(|S| + |T|), где |S| - длина строки для поиска в ней вхождений образца, а |T| - длина строки образца. Сложность вычисляется из того, что сначала нужно пройти по всем префиксам образца для вычисления их префикс-функции, а затем по всем префиксам строки для поиска вхождений.

Сложность по памяти: O(T), где |T| - длина строки образца. Такая сложность получается, если учитывать только дополнительную память для хранения значений префикс-функций символов образца, без памяти для хранения исходных данных.

Оценка сложности алгоритма поиска циклического сдвига.

Сложность по времени: O(|S| + |T|), где |S| - длина строки для поиска в ней вхождений образца, а |T| - длина строки образца. Сначала нужно пройти по всем префиксам образца для вычисления их префикс-функции, а затем по всем префиксам строки в худшем случае 2 раза для поиска вхождений, т.е. |T| + |T|.

Сложность по памяти: O(T), где |T| - длина строки образца. Такая сложность получается, если учитывать только дополнительную память для хранения значений префикс-функций символов образца, без памяти для хранения исходных данных.

Тестирование алгоритма поиска вхождений подстроки.

№	Входное значение	Результат
1	asd asdasdasdasdaadasdadaaaddasaaadsdasdddadasdsdsd	0,3,6,9,15,33,40
2	ab asbasbasdabababababdkaakdjakjabbda	9,11,13,15,27
3	as fdgdfgdgfdfgdfgdf	-1
4	abab abababababbababababbabbab	0,2,4,9,11,16

Подробный тест.

```
Test 1:   Tests1/Test1.txt
1
asd
aesdfeasd


Getting started with the substring occurrence search algorithm.

T[0] = a
S[0] = a
The symbols are the same.

T[0] = a
S[1] = e

T[0] = a
S[2] = s

T[0] = a
S[3] = d

T[0] = a
S[4] = f

T[0] = a
S[5] = e

T[0] = a
S[6] = a
The symbols are the same.

T[1] = s
S[7] = s
The symbols are the same.

T[2] = d
S[8] = d
The symbols are the same.
Pattern detected, index 6

Answer: 6
```

Тестирование алгоритма поиска циклического сдвига.

№	Входное значение	Результат
1	asd das	2
2	asd asd	0
3	qwertyuiop uiopqwerty	6
4	qwertyuiopp uiopqwerty	-1
5	qwertytiop uiopqwerty	-1

Подробный тест.

```
Test 1:   Tests2/Test1.txt
2
asd das


Getting started with the cyclic string shift search algorithm.

B[0] = d
A[0] = a

B[0] = d
A[1] = s

B[0] = d
A[2] = d
The symbols are the same.
Possible occurrence Index: 2

B[1] = a
A[0] = a
The symbols are the same.

B[2] = s
A[1] = s
The symbols are the same.

It is cycle shift.
Answer: 2
```

Вывод.

Было получено теоретическое представление об алгоритме Кнута-Морриса-Пратта и на основе него были реализованы программы для поиска вхождений подстроки и поиска циклического сдвига строки.

ПРИЛОЖЕНИЕ А

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

//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.

// Считывание из файла
void inputFile(string *S, string *T)
{
    ifstream file;
//    file.open("/home/olyaave/CLionProjects/PAA_LAB4/input.txt");
    file.open("input.txt");
    if (file.is_open()) {
        file >> *T  >> *S;
        file.close();
    } else {
        cout << "File isn't open!";
    }
}

void prefix(string *T, vector<int> *max_pref){

    size_t n = T->length();

    int prev_pref = (*max_pref)[0] = 0;

    for (int i = 1; i < n; ++i)
    {
        while(prev_pref > 0 && (*T)[prev_pref] != (*T)[i])  // Поиск
максимального префикса-суффикса
            prev_pref = (*max_pref)[prev_pref - 1];
```

```
        if((*T)[prev_pref] == (*T)[i])  // Если символы совпадают,
расширяем префикс
            prev_pref++;

        (*max_pref)[i] = prev_pref;  // Сохраняем значение префикс-
функции для текущего префикса
    }

}

// Функция для поиска подстроки
void KMP(string* S, string *T, vector<int> *answer)  // S - строка,
T - шаблон
{
    cout << "\nGetting started with the substring occurrence search
algorithm.\n";
    size_t n = T->length();
    vector<int> max_pref(n);

    prefix(T, &max_pref);

    int prev_pref = 0;
    for (int i = 0; i < S->length(); ++i)
    {
        while(prev_pref > 0 && (*T)[prev_pref] != (*S)[i])  // Поиск
максимального префикса-суффикса
        {
            prev_pref = max_pref[prev_pref - 1];
        }
        cout << "\nT[" << prev_pref << "] = " << (*T)[prev_pref] <<
endl;
        cout << "S[" << i << "] = " << (*S)[i] << "\n";

        if((*T)[prev_pref] == (*S)[i]) {    // Если символы
совпадают, расширяем префикс
            prev_pref++;
            cout << "The symbols are the same.\n";
        }
```

```
        if(prev_pref == n) {    // Если найден шаблон, сохраняем в
векторе
            answer->push_back(i - n + 1 );
            cout << "Pattern detected, index " << i - n + 1 <<"\n";
        }
    }
    if(answer->empty()) {
        cout << "\nNo occurrences of pattern.\n";
        cout << "-1";
    }
    else {
        cout << "\nAnswer: ";
        for (int i = 0; i < answer->size() - 1; ++i) {
            cout << (*answer)[i] << ",";
        }
        cout << (*answer)[answer->size() - 1] << endl;
    }
}


int offset(string* B, string *A) // ищем B в A
{
    cout << "\nGetting started with the cyclic string shift search
algorithm.\n";

    if((*A).size() != (*B).size())
        return -1;

    int sizeA = A->length();
    int sizeB = B->length();

    vector<int> max_pref(sizeB);

    int prev_pref = 0;
    int start_i = -1;

    for (int i = 0; i < sizeA; ++i)
    {
```

```
        while(prev_pref > 0 && (*B)[prev_pref] != (*A)[i])  // Поиск
максимального префикса-суффикса
            prev_pref = max_pref[prev_pref - 1];

        cout << "\nB[" << prev_pref << "] = " << (*B)[prev_pref] <<
endl;
        cout << "A[" << i << "] = " << (*A)[i] << "\n";

        if((*B)[prev_pref] == (*A)[i])  // Если символы совпадают,
расширяем префикс
        {
            prev_pref++;
            cout << "The symbols are the same.\n";
        }
        if(i == sizeA- 1 && prev_pref > 0) {  // Если встречен
крайний префикс,
                                              // сохраняем индекс, с
которого возможно начался циклический сдвиг
            start_i = sizeA - prev_pref;
            cout << "Possible occurrence Index: " << start_i <<
"\n";
        }
        if(prev_pref == sizeB)   // Если значение префикс-функции
равно длине исходной строки
        {                        // до конца проверяемой, значит
исходная совпадает с проверяемой
            cout << "\nThe string being checked matches the original
one.\n";
            return 0;
        }
    }


    for (int i = 0; i < start_i; ++i)
    {
        while(prev_pref > 0 && (*B)[prev_pref] != (*A)[i])
            prev_pref = max_pref[prev_pref - 1];
```

```
        cout << "\nB[" << prev_pref << "] = " << (*B)[prev_pref] <<
endl;
        cout << "A[" << i << "] = " << (*A)[i] << "\n";

        if((*B)[prev_pref] == (*A)[i]) {
            prev_pref++;
            cout << "The symbols are the same.\n";
        }

        if(prev_pref == B->size()) {
            cout << "\nIt is cycle shift.\n"
                    "Answer: ";
            return start_i;
        }
    }
    cout << "It isn't cycle shift.\n";
    return -1;
}


int main() {
    string S, T;
    vector<int> answer;

//    inputFile(&S, &T);
    int type;
    cin >> type >>T >> S;
    if(type == 1)
        KMP(&S, &T, &answer);
    else
        cout << offset(&S, &T);


    return 0;
}
```