

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

Стукалев А.И.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить и реализовать на языке программирования C++ алгоритм Кнута-Морриса-Пратта поиска подстроки в строке, также с помощью этого алгоритма определить является ли одна строка циклическим сдвигом другой строки.

Индивидуализация.

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.

Формулировка задания для КМП.

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 для КМП.

ab

abab

Соответствующие выходные данные для КМП.

0,2

Формулировка задания для алгоритма по определению циклического сдвига.

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

Определить, является ли A циклическим сдвигом B (это значит, что AA и BB имеют одинаковую длину и AA состоит из суффикса BB , склеенного с

префиксом В). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - A

Вторая строка - ВВ

Выход:

Если A является циклическим сдвигом В, индекс начала строки В в А, иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

Пример входных данных для алгоритма по определению циклического сдвига.

defabc

abcdef

Соответствующие выходные данные для алгоритма по определению циклического сдвига.

3

Описание алгоритмов.

КМП:

Рассмотрим сравнение строк на позиции , где образец сопоставляется с частью текста . Предположим, что первое несовпадение произошло между и где . Тогда и .

При сдвиге вполне можно ожидать, что префикс (начальные символы) образца сойдется с каким-нибудь суффиксом (конечные символы) текста . Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки для индекса .

Это приводит нас к следующему алгоритму: пусть — значение префикс-функции от строки для индекса . Тогда после сдвига мы можем возобновить сравнения с места и без потери возможного местонахождения образца. Можно показать, что таблица может быть вычислена (амортизационно) за сравнений перед началом поиска. А

поскольку строка будет пройдена ровно один раз, суммарное время работы алгоритма будет равно $O(n)$, где n — длина текста.

Алгоритм для нахождения циклического сдвига.

В данном алгоритме можно обойтись без удваивания строки. В самом начале происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Инициализируются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка зацикливается и можно обойтись без удвоения строки.

Префикс функция.

Префикс-функция от строки и позиции в ней — длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки.

То есть, в начале подстроки длины нужно найти такой префикс максимальной длины, который был бы суффиксом данной подстроки.

Например, для строки префикс-функция будет такой: .

Сложность алгоритма КМП по операциям: $O(n + m)$, n — длина подстроки, m — длина строки.

Сложность алгоритма поиска циклического сдвига: $O(n + n) = O(n)$.

Сложность алгоритма КМП по памяти: $O(m)$, если не учитывать память, в которой хранится строка поиска.

Сложность алгоритма поиска циклического сдвига по памяти $O(2n) = O(n)$.

Описание структур данных алгоритма КМП и алгоритма нахождения

ЦИКЛИЧЕСКОГО СДВИГА.

1.

```
class SubStr
{
    std::vector <int> ind_lenghts;
    std::string input_string;
}
```

Класс необходимый для работы алгоритма КМП, input_string – строка-образец, ind_lenghts – массив для префиксов.

2.

```
class Cycle
{
    std::vector <int> ind_lenghts;
    std::string first_string;
    std::string second_string;
    std::string result_string;
}
```

Класс, необходимый для работы алгоритма поиска циклического сдвига. First_string – первая строка, second_string – вторая строка, result_string - строка, необходимая для конкатенации первой и второй строки, ind_lenghts – массив префиксов для result_string.

Описание функций КМП.

1. SubStr()

Конструктор класса Substr. Происходит считывание строки в input_string и инициализация нулями ind_lenghts.

2. void prefix_func()

Функция класса SubStr вычисления префикс функции для input_string и записи

результата в ind_lengths.

3. void KMP()

Функция класса SubStr нахождения подстроки в строке. Посимвольно считывает строку, в которой необходимо проводить поиск, имеет два счётчика – счётчик нахождения в input_string (j) и счётчик количества введённых символов(i), необходимый для вычисления позиция вхождения подстроки в строку. С каждым добавленным элементом второй счётчик увеличивается, первый же счётчик увеличивается лишь в том случае, если произошло равенство считанного элемента и input_string[j]. Если j равняется длине input_string, то подстрока найдена и результат выводится в консоль, j становится равным ind_lengths[j-1]. Если считанный элемент и input_string[j] не равны, j становится равным ind_lengths[j-1]. Алгоритм завершает работу, когда следующий символ считать не возможно.

4. ~SubStr()

Деструктор класса SubStr, очищающий input_string и ind_lengths.

Описание функций алгоритма нахождения циклического сдвига.

1. Cycle()

Конструктор класса Cycle. Происходит считывание первой строки в first_string, считывание второй строки в second_string, заполнение result_string следующим образом result_string = first_string + "@" + second_string, заполнение массива ind_lengths нулями.

2. void prefix_func()

Функция класса SubStr вычисления префикс функции для first_string и записи результата в ind_lengths.

3. void cycle()

Функция класса Cycle для нахождения циклического сдвига. В самом начале происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Инициализируются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы

совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается, в том случае, если он не равен нулю. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка за циклируется и можно обойтись без удвоения строки, во избежания бесконечного цикла проход по строке осуществляется не более двух раз.

4. ~Cycle()

Деструктор класса Cycle. Очищает first_string, second_string, и ind_lengths.

Способ хранения частичных решений.

Частичные решения, т.е. значения префикс функции, в массиве ind_lengths.

Тестирование.

КМП

1.

Test input:

ab
abab

Test output:

0,2

2.

Test input:

a
a

Test output:

0

3.

Test input:

```
ba
abababaaabababbababbaaaa
```

Test output:

```
1,3,5,9,11,14,16,21
```

4.

Test input:

```
baaa
baabbbabababababab
```

Test output:

```
-1
```

Алгоритм нахождения циклического сдвига

1.

Test input:

```
defabc
abcdef
```

Test output:

```
3
```

2.

Test input:

```
a
a
```

Test output:

```
0
```

3.

Test input:

```
aaa
aaaa
```

Test output:

```
-1
```

4.

Test input:
defabc abcdec
Test output:
-1

Выводы.

В ходе выполнения лабораторной работы были изучены и реализованы на языке программирования C++ алгоритм КМП нахождения подстроки в строке и алгоритм нахождения циклического сдвига. Также алгоритм КМП был модифицирован в соответствии с индивидуализацией, для этого хранились только подстрока и массив со значениями префикс функции для подстроки, а строка, в которой необходимо было производить поиск, считывалась посимвольно.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

КМП.

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

class SubStr
{
    std::vector <int> ind_lengths;
    std::string input_string;
public:
    SubStr()
    {
        std::cin >> input_string;
        std::vector <int> tmp(input_string.length()); // инициализируем нулями вектор длин префиксов
        ind_lengths = tmp;
    }
    void prefix_func()
    {
        for (size_t i=1; i<input_string.length(); ++i)
        {
            // ищем, какой префикс можно расширить
            size_t j = ind_lengths[i-1]; // длина предыдущего префикса-суффикса, возможно нулевая
            while ((j > 0) && (input_string[i] != input_string[j])) // если нельзя расширить
                j = ind_lengths[j-1]; // уменьшаем значение префикса

            if (input_string[i] == input_string[j])
                ++j; // расширяем найденный префикс
            ind_lengths[i] = j;
        }
    }
}

void KMP()
{
    char c;
    int j = 0;
    int i = 0;
    bool no_one_digit = false;
    prefix_func(); // считается префикс функция для образца
    std::cin >> c;
    while(!std::cin.fail()) // пока возможно считать символ
    {
        if(input_string[j] == c) // соответствие найдено
        {
            //std::cout << "Equal elements:  " << input_string[j] << "==" << c << "\n";
            j++;
            i++;
            std::cin >> c;
        }
        if(j == input_string.length()) // подстрока найдена
        {
            //std::cout << "Sub string founded: " << input_string[j] << ":@" << c << "\n";
            if(no_one_digit) // если более одной цифры, то между ними ставится запятая
                std::cout << ',';
            std::cout << i - j;
            j = ind_lengths[j-1];
        }
    }
}
```

```

        no_one_digit = true;
    }
    else if(input_string[j] != c && !std::cin.fail())//совпадения не обнаружено или конец ввода
    {
        //std::cout << "Non equal:      " << input_string[j] << "!=" << c << "\n";
        if(j != 0)
            j = ind_lengths[j - 1];
        else
        {
            i++;
            std::cin >> c;
        }
    }

}

if(!no_one_digit)//если подстроки найдено не было
    std::cout << -1;
}

~SubStr()
{
    ind_lengths.clear();
    input_string.clear();
}
};

int main()
{
    SubStr* tmp = new SubStr();
    tmp->KMP();
    delete tmp;
}

```

Алгоритм нахождения циклического сдвига.

```

#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

class Cycle
{
    std::vector <int> ind_lengths;
    std::string first_string;
    std::string second_string;
public:
    Cycle()
    {
        std::cin >> first_string;
        std::cin >> second_string;
        std::vector <int> tmp(second_string.length());
        ind_lengths = tmp;
        tmp.clear();
    }

    void prefix_func()
    {
        for (size_t i=1; i<second_string.length(); ++i)
        {

```

```

        // ищем, какой префикс можно расширить
        size_t j = ind_lengths[i-1]; //длина предыдущего префикса, может быть нулевой
        while ((j > 0) && (second_string[i] != second_string[j])) //если нельзя расширить,
            j = ind_lengths[j-1]; //уменьшаем значение префикса

        if (second_string[i] == second_string[j])
            ++j; //расширяем найденный префикс
        ind_lengths[i] = j;
    }
}

void cycle()
{
    if(first_string.length() != second_string.length())//если длины строк не равны - выход
    {
        std::cout << "Non-equal lengths: " << first_string.length() << "!=" << second_string.length() << "\n";
        std::cout << -1;
        return;
    }
    this->prefix_func();// вычисление префикс функции для second_string
    int laps = 0;
    for (int ind_f = 0, ind_s = 0;;)
    {
        if(first_string[ind_f] == second_string[ind_s])
        {
            //std::cout << "Equal elements: " << first_string[ind_f] << "==" << second_string[ind_s] << " index: " << ind_f <<
            " " << ind_s << "\n";
            ind_f++;
            ind_s++;
        }
        if(ind_f == first_string.length())
        {
            ind_f = 0;
            laps++;
        }
        if(ind_s == second_string.length())
        {
            //std::cout << "Cycle founded: ";
            std::cout << ind_f;
            return;
        }
        else if(first_string[ind_f] != second_string[ind_s] && ind_s < second_string.length())
        {
            //std::cout << "Nonequal elements: " << first_string[ind_f] << "!=" << second_string[ind_s] << " index: " << ind_f
            << " " << ind_s << "\n";
            if(ind_s == 0)
                ind_f++;
            else
                ind_s = ind_lengths[ind_s - 1];
        }
        if(laps > 1)
            break;
    }
    std::cout << -1;
}

~Cycle()
{
    ind_lengths.clear();
    first_string.clear();
    second_string.clear();
}
};

```

```
int main()
{
    Cycle* tmp = new Cycle();
    tmp->cycle();
    delete tmp;
}
```