

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-Морриас-Пратта

Студентка гр. 9383

Лихашва А.Д.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриас-Пратта. Написать программы, которые реализуют данный алгоритм, используя полученные знания.

Задание.

Step 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

Step 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

Описание алгоритма:

Step 1. Найти вхождение первой строки во вторую.

- В начале вычисляется префикс-функция для первой строки P .
- Префикс-функция для какого-либо символа показывает максимальную длину совпадающего префикса и суффикса подстроки, которая заканчивается этим символом.
- Далее начинается посимвольное сравнение двух строк. Если при сравнении все символы первой строки совпадают со второй, значит нашлось вхождение. В массив записывается индекс начала вхождения.
- Если при сравнении первый символ строки P не совпал с первым символом строки T , то сравнивается первый символ строки P со вторым символом строки T и так далее.
- Если при сравнении не первый символ строки P не совпадает с символом строки T , то следующим символом, с которым будет сравниваться строки P и T , будет символ строки P под индексом префикс-функции предыдущего символа.
- Алгоритм закончится, когда все символы строки будут сравнены.

Step 2. Является ли строка циклическим сдвигом второй строки.

- Используется тот же самый алгоритм, что и в Step 2, однако поиск ведётся в удвоенной первой строке, потому что удвоенная первая строка будет содержать в себе вторую строку, если она, первая строка, является циклическим сдвигом второй строки.
- Алгоритм заканчивается:
 - когда находится первое вхождение.
 - когда просматриваются все символы удвоенной первой строки.
 - когда строки изначально не равны по длине.

Сложность алгоритма :

В обеих программах сложность по памяти равна $O(m + n)$, где m – длина первой строки, n – длина второй строки.

Сложность по времени равна $O(m + n)$, где m – длина первой строки, n – длина второй строки.

Функции и структуры данных:

Структуры данных:

class KMP – класс для реализации алгоритма Кнута-Морриас-Пратта.

std::vector<int> arrPrefix — массив префикс-функций

Функции:

void KMP::Read() — функция для считывания данных.

void KMP::PrintAnswer() — функция, которая печатает ответ на экран

void KMP::Prefix() — функция, которая формирует массив значений префикс-функций символов строки P .

void KMP::AlgorithmKMP() — функция, которая реализует алгоритм Кнута-Морриса-Пратта.

void KMP::CyclicShift() — функция, которая проверят, является ли строка А циклическим сдвигом строки В.

Тестирование.

Step 1

Входные данные	Выходные данные
ab abab	0,2
qw qwerty	0
asd qwerty	-1
to rtohtoto	1,4,6

Step 2

Входные данные	Выходные данные
defabc abcdef	3
qwerty asdfgh	-1
qwerty rtyqwe	3
zxcvb vbzxc	3

Выводы.

В результате выполнения работы был изучен алгоритм Кнута-Морриас-Пратта. Основываясь на полученных знаниях, были написаны программы, которые реализуют данный алгоритм.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lab4_1.h

```
#pragma once

#include <iostream>
#include <vector>

class KMP {
private:
    std::string P, T;
    std::vector<int> arrPrefix;
    std::vector<int> answer;
public:
    KMP() {}
    void Read();
    void PrintAnswer();
    void Prefix();
    void AlgorithmKMP();
};
```

Файл lab4_1.cpp

```
#include "lab4_1.h"

void KMP::Read() {
    std::cout << "Введите первую строку P:\n";
    std::cin >> P;
    std::cout << "Введите вторую строку T:\n";
    std::cin >> T;
}

void KMP::PrintAnswer() {
    std::cout << "\n_Ответ_: \n";
    if (!answer.size()) {
        std::cout << "P не входит в T\n";
        std::cout << -1;
    }
    else {
        std::cout << "Индексы начал вхождений P в T:\n";
        for (auto a : answer) {
            if (a == answer.front())
                std::cout << a;
            else
                std::cout << ", " << a;
        }
    }
}
```

```

    }
}
std::cout << '\n';
}

void KMP::Prefix() {    //префикс-функция
    std::cout << "\n_Подсчет префикс-функции_:\n\n";
    int pLength = P.length();
    arrPrefix = std::vector<int>(pLength);
    arrPrefix[0] = 0;
    std::cout << "Значение префикс-функции для символа "<< P[0]
    << " под номером " << 0 << " равна " << 0 << '\n';
    for (int i = 1; i < pLength; i++) {
        int curLength = arrPrefix[i - 1];
        std::cout << "Вычисление значения префикс-функции для
символа " << P[i] << " под номером " << i << '\n\n';

        while (curLength > 0 && (P[curLength] != P[i])) {    //
если предыдущий суффикс нельзя расширить, то рассматриваем суф-
фикс меньшего размера
            std::cout << "Предыдущий суффикс размера " <<
curLength << " нельзя расширить.\n";
            curLength = arrPrefix[curLength - 1];
            std::cout << "Рассмотрим новый суффикс меньшего раз-
мера: " << curLength << '\n';
        }

        if (P[curLength] == P[i]) {    // проверка на расширение
            std::cout << "Суффикс длинны " << curLength << "
можно расширить.\n";
            curLength++;
            std::cout << "Новый размер суффикса: " << curLength
<< '\n';
        }

        std::cout << "Значения префикс-функции равно " <<
curLength << '\n';
        arrPrefix[i] = curLength;
    }

    std::cout << "\nЗначения префикс-функции:\n";
    for (auto j : P) {
        std::cout << j << ' ';
    }
    std::cout << '\n';
    for (auto i : arrPrefix) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
}

void KMP::AlgorithmKMP(){

```

```

Prefix();
std::cout << "\n\n Алгоритм Кнута-Морриса-Пратта_:\n\n";
int tSize = T.size();
int pSize = P.size();
int pIndex = 0;
int tIndex = 0;
while (tIndex < tSize) {
    if (P[pIndex] == T[tIndex]) { //если символы равны
        std::cout << "Найдено совпадение " << pIndex << "
символа образца " << P[pIndex] << "и " << tIndex << " символа
текста " << T[tIndex] << '\n';
        pIndex++;
        tIndex++;
        if (pIndex == pSize) { //если вхождение найдено
            std::cout << "Вхождение найдено, индекс равен "
<< tIndex - pIndex << "\n\n";
            answer.push_back(tIndex - pIndex);
            pIndex = arrPrefix[pIndex - 1]; //переход на по-
зицию равную предпоследнему значению префикс-функции
        }
    }
    else if (pIndex == 0) // если сравнение с первым симво-
лом
        tIndex++;
    else //если же по образцу продвинулись
        pIndex = arrPrefix[pIndex - 1];
}

int main() {
    setlocale(LC_ALL, "Russian");
    KMP k;
    k = KMP();
    k.Read();
    k.AlgorithmKMP();
    k.PrintAnswer();
    return 0;
}

```

Файл lab4_2.h

```

#pragma once

#include <iostream>
#include <vector>

class KMP {
private:
    std::string A, B;
    std::vector<int> arrPrefix;
    int answer;

```



```

public:
    KMP() {};
    void Read();
    void PrintAnswer();
    void Prefix();
    void CyclicShift();
};

```

Файл lab4_2.cpp

```

#include "lab4_2.h"

```

```

void KMP::Read() {
    std::cout << "Введите первую строку A:\n";
    std::cin >> A;
    std::cout << "Введите вторую строку B:\n";
    std::cin >> B;
}

void KMP::PrintAnswer() {
    std::cout << "\n\n_Ответ_: \n";
    if (answer == -1) {
        std::cout << "A Не является циклическим сдвигом B.\n";
        std::cout << answer << '\n';
    }
    else {
        std::cout << "A является циклическим сдвигом B.\n";
        std::cout << "Индекс начала строки B в A: " << answer <<
'\n';
    }
}

void KMP::Prefix() { //префикс-функция
    std::cout << "\n Подсчет префикс-функции_: \n\n";
    int bLength = B.length();
    arrPrefix = std::vector<int>(bLength);
    arrPrefix[0] = 0;
    std::cout << "Значение префикс-функции для символа " << B[0]
<< " под номером " << 0 << " равна " << 0 << '\n';
    for (int i = 1; i < bLength; i++) {
        int curLength = arrPrefix[i - 1];
        std::cout << "Вычисление значения префикс-функции для
символа " << B[i] << " под номером " << i << "\n\n";

        while (curLength > 0 && (B[curLength] != B[i])) { //
если предыдущий суффикс нельзя расширить, то рассматриваем суф-
фикс меньшего размера
            std::cout << "Предыдущий суффикс размера " <<
curLength << " нельзя расширить.\n";

```

```

        curLength = arrPrefix[curLength - 1];
        std::cout << "Рассмотрим новый суффикс меньшего раз-
мера: " << curLength << '\n';
    }

    if (B[curLength] == B[i]) {        // проверка на расширение
        std::cout << "Суффикс длинны " << curLength << "
можно расширить.\n";
        curLength++;
        std::cout << "Новый размер суффикса: " << curLength
<< '\n';
    }

    std::cout << "Значение префикс-функции равно " <<
curLength << '\n';
    arrPrefix[i] = curLength;
}

std::cout << "\nЗначения префикс-функции:\n";
for (auto j : B) {
    std::cout << j << ' ';
}
std::cout << '\n';
for (auto i : arrPrefix) {
    std::cout << i << ' ';
}
std::cout << '\n';
}

void KMP::CyclicShift() {
    if (A.length() != B.length()) {
        std::cout << "Длины строк не равны, значит это не цик-
лический сдвиг.\n";
        answer = -1;
        return;
    }

    Prefix();
    std::cout << "\n\n_Определение, является ли A циклическим
сдвигом B_:\n\n";
    int aLength = A.length();
    int curLength = 0;
    for (int i = 0; i < aLength * 2; i++) { //поиск по удвоен-
ной первой строке
        int j = i % aLength;
        if (B[curLength] != A[j]) //если символы не равны
            std::cout << "Несовпадение " << j << " символа стро-
ки A " << '(' << A[j] << ')' << " и " << curLength << " символа стро-
ки B " << '(' << B[curLength] << ")";
        while (curLength > 0 && B[curLength] != A[j]) {
            curLength = arrPrefix[curLength - 1];
        }
    }
}

```

```

        std::cout << '\n';
        if (B[curBLength] == A[j]) { //если символы равны
            std::cout << "Найдено совпадение " << j << " символа
строки A " << '(' << A[j] << ')' << " и " << curBLength << "
символа строки B " << '(' << B[curBLength] << ")";
            curBLength++;
        }

        if (curBLength == aLength) { //если нашлось вхождение
            answer = i - curBLength + 1;
            std::cout << "\n\nВхождение нашлось. Индекс равен "
<< answer << '\n';
            return;
        }
    }

    answer = -1;
    return;
}

int main() {
    setlocale(LC_ALL, "Russian");
    KMP k;
    k = KMP();
    k.Read();
    k.CyclicShift();
    k.PrintAnswer();
    return 0;
}

```