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.

Студентка гр. 8382 _____ Наконечная А. Ю.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта, используемый для поиска подстроки в тексте, и реализовать его в собственной программе для решения поставленных задач.

Постановка задачи.

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:

ab

abab

Sample Output:

0,2

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

Индивидуальное задание.

Вариант 1.

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Описание алгоритма.

Для реализации распараллеливания был написан метод void trainingText(); Метод ничего не принимает, необходим для разделения исходного текста на блоки, количество которых определяется пользователем. Метод ничего не возвращает. Распараллеливание происходит следующим образом:

1. Вычисляется минимальный и максимальный размер блока.
2. Отрез блоков происходит в цикле, где, используя значение максимального размера, происходит отрезание максимально возможной части с определённого индекса.
3. Получившиеся блоки запоминаются для дальнейшей работы.

Для получения префикс-функции был написан метод createPrefix(). Ничего не принимает, и ничего не возвращает, позволяет заполнить вектор значениями префикс-функции.

1. Создание одномерного массива (вектора), размер, которого равен длине шаблона.
2. Для первого символа образа в массив префикс-функции записывается значение ноль. Рассматриваемые символы обозначены за j и i (изначально это соответственно индексы 0 и 1).
3. Если символы, на которые указывают i и j не равны и j указывает на начало шаблона, то записываем в префикс-функцию от i-ого значения – ноль и сдвигаем i на единицу. Иначе присваиваем индексу j значение префикс-функции предыдущего символа, на который указывает j.
4. Если символы равны, тогда в префикс-функцию от i-го элемента записываем значение j + 1, и сдвигаемся по обоим индексам.
5. Шаги 3 и 4 повторяются до того момента, пока конец шаблона не будет достигнут.

С помощью метода patternSearch() происходит поиск шаблона в тексте, разделённом блоками. В метод последовательно поступают блоки, а также их индексы. Метод ничего не возвращает.

1. Заводится два “указателя”: первый указывает на индекс рассматриваемого символа текста, второй – шаблона. Оба указывают на начало.
2. Сравниваются текущие символы.
3. Если символы совпадают, то переменные-указатели сдвигаются на 1.
4. Если переменная-указатель оказалась в конце, значит было найдено вхождение, оно фиксируется в результирующем векторе.
5. Если после шага 2 символы оказались не равны и указатель на шаблон находится в начале, то ему присваивается значение префикс-функции от предыдущего для данного значения.
6. Иначе переменная-указатель передвигается на 1.
7. Если конец текста так и не достигнут, то возвращаемся к шагу 2.

Описание структур данных.

vector <int> prefix - вектор, который хранит в себе значения префикс-функции.

vector <int> result - результирующий вектор.

vector <string> blocks - вектор, используемый для хранения блоков.

string pattern/string text - строки для хранения образца и текста.

Описание функций.

void printAnswer() - функция для печати ответа;

void printPrefix() - функция для печати префикс-функции;

void printPattern() - функция для печати образца;

void printBlock() - функция для печати вектора с блоками;

void printText() - функция для печати текста;

void readData() - функция для чтения данных; считывает количество блоков, текст, образец. Проверяет количество введённых блоков на корректность.

void algorithm() - функция для организации остальных методов реализации поиска образца в тексте, а также вывода результата.

void patternSearch(string block, int index) - функция для поиска образца в тексте, принимает рассматриваемый блок, а также его индекс.

void createPrefix() - функция, используемая для создания префикса.

void trainingText() - функция, используемая для деления текста на блоки.

bool find(int elem) - функция поиска элемента elem в результирующем векторе.

void setBlock(int B)/void setText(string T)/void setPattern(string P) - функции-сеттеры.

Сложность алгоритма.

Сложность алгоритма по времени для поиска образа длины m в тексте длины n можно оценить как O(m + n), так как алгоритм можно разделить на построение префикс-функции и поиск вхождений: сложность вычисления префикс-функции по образу равна O(m), сложность прохождения по всему тексту равна O(n).

Сложность алгоритма по памяти также равна O(m + n), так как в процессе работы алгоритма мы храним исходный шаблон O(m), префикс функцию O(m) и текст O(n), константа отбрасывается.

Тестирование для алгоритма КМП.

№ теста	Тест	Результат
1	abcd abcdaaaaaabbb bbbccccccddd ddddabcd 2	Исходный текст: abcdaaaaaabbbbbbccccccddddddddabcd Количество блоков равно: 2 Максимальный размер блока: 20 Блок 1: abcdaaaaaabbbbbbcccc Блок 2: ccccccddddddddabcd Обрабатываемый шаблон: abcd Обработка: шаблон[0] = a и шаблон[1] = b Найдено несовпадение символовprefixFunc[1] = 0 Обработка: шаблон[0] = a и шаблон[2] = c Найдено несовпадение символовprefixFunc[2] = 0 Обработка: шаблон[0] = a и шаблон[3] = d Найдено несовпадение символовprefixFunc[3] = 0 Значения полученной префикс-функции: Шаблон[0] = a имеет prefixFunc[0] = 0 Шаблон[1] = b имеет prefixFunc[1] = 0

		Шаблон[2] = c имеет prefixFunc[2] = 0 Шаблон[3] = d имеет prefixFunc[3] = 0 Обрабатываемый блок: 1) abcdaaaaaabbbbbbcccc Шаблон: abcd Блок: abcdaaaaaabbbbbbcccc Символы для сравнения: шаблон[0] = a и блок[0] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[1] = b Найдено совпадение символов Символы для сравнения: шаблон[2] = c и блок[2] = c Найдено совпадение символов Символы для сравнения: шаблон[3] = d и блок[3] = d Найдено совпадение символов НАЙДЕНО ВХОЖДЕНИЕ ШАБЛОНА В ТЕКСТ: 0 Символы для сравнения: шаблон[3] = d и блок[4] = a Символы не совпадают

		Символы для сравнения: шаблон[0] = a и блок[4] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[5] = a Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[5] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[6] = a Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[6] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[7] = a Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[7] = a Найдено совпадение символов

		Символы для сравнения: шаблон[1] = b и блок[8] = a Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[8] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[9] = a Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[9] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[10] = b Найдено совпадение символов Символы для сравнения: шаблон[2] = c и блок[11] = b Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[11] = b Символы не совпадают

		Символы для сравнения: шаблон[0] = a и блок[12] = b Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[13] = b Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[14] = b Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[15] = b Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[16] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[17] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[18] = c Символы не совпадают

		Символы для сравнения: шаблон[0] = a и блок[19] = c Символы не совпадают Обрабатываемый блок: 2) ccccccddddddabcd Шаблон: abcd Блок: ccccccddddddabcd Символы для сравнения: шаблон[0] = a и блок[0] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[1] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[2] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[3] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[4] = c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[5] =

		c Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[6] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[7] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[8] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[9] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[10] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[11] = d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[12]

		= d Символы не совпадают Символы для сравнения: шаблон[0] = a и блок[13] = a Найдено совпадение символов Символы для сравнения: шаблон[1] = b и блок[14] = b Найдено совпадение символов Символы для сравнения: шаблон[2] = c и блок[15] = c Найдено совпадение символов Символы для сравнения: шаблон[3] = d и блок[16] = d Найдено совпадение символов НАЙДЕНО ВХОЖДЕНИЕ ШАБЛОНА В ТЕКСТ: 30 РЕШЕНИЕ: Вхождения шаблона в текст найдены: 0,30
2	abcd bnbnbnbasdnb asnd 4	-1

3	a a 1	0
4	a aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aa 9	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35
5	Aa$aa Aaaaabbblsflksdjdlfkwwelkrjwmns,dmnflwkenkldfkfjAa$aaasldkjaskjd 13	48
6	aba abababababa 1	0,2,4,6,8
7	absdf sdsdabsdabsdasdbassdbasfabsdfbasdffssafbsabsdfabsdfbabsdffffff 1	24,41,46,52
8	absdf	24,41,46,52

	sdsdabsdabsda sdbassdbasfabs dfbasdffssafbsa bsdfabsdfbabsd ffffff 12	
9	a bbbbbbbbbbbb vvvvvvvvvvvd ddddddd 1	-1
10	asdasdasdasdas dasdasdasdasd asdasdasdasdas dasdasdasdasd asdasd asdasdasdasdas dasdasdasdasd asdasdasdasdas dasdasdasdasd asdasd 1	0

Тестирование для алгоритма поиска циклического сдвига.

№ теста	Тест	Результат
1	abcdef defabc	Начало функции циклического сдвига Поиск значений префикс-функции Происходит вычисление значения префикс-

		функции для: e Значение префикс-функции для: e = 0 Происходит вычисление значения префикс-функции для: f Значение префикс-функции для: f = 0 Происходит вычисление значения префикс-функции для: a Значение префикс-функции для: a = 0 Происходит вычисление значения префикс-функции для: b Значение префикс-функции для: b = 0 Происходит вычисление значения префикс-функции для: c Значение префикс-функции для: c = 0 Начало основной проверки на циклический сдвиг после вычисления префикс-функции d == d => Можно расширить суффикс Размер суффикса: 0 Меняем на новый размер: 1 e == e => Можно расширить суффикс Размер суффикса: 1 Меняем на новый размер: 2 f == f => Можно расширить суффикс Размер суффикса: 2 Меняем на новый размер: 3 a == a => Можно расширить суффикс Размер суффикса: 3 Меняем на новый размер: 4

		b == b => Можно расширить суффикс Размер суффикса: 4 Меняем на новый размер: 5 c == c => Можно расширить суффикс Размер суффикса: 5 Меняем на новый размер: 6 Длина текущего суффикса: 6 == Длине шаблона: 6 => Строка является циклическим сдвигом, индекс строки: 3
2	xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx	0
3	aaaa aaaaaaaa	-1
4	asfasf asfacf	-1
5	aaaaaaaabbbbbbbb aaaaaaaabbbbbbbb	0
6	agsdffasdg sdffasdgag	2

Выводы.

В результате выполнения лабораторной работы была написана программа, решающая задачу поиска в строке с помощью алгоритма Кнута-Морриса-Пратта.

ПРИЛОЖЕНИЕ А

Исходный код программы

code_with_comments_KMP.cpp

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

using std::cout;
using std::cin;
using std::string;
using std::endl;
using std::move;
using std::vector;

//класс, реализующий КМП
class Base{
private:
    //длина шаблона
    int sizePattern{};
    //длина текста
    int sizeText{};
    //кол-во блоков для разделения текста
    int numBlock{};
    //размер блока
    int sizeBlock{};
    //максимальный размер обрабатываемого блока
    int maxSizeBlock{};
    //шаблон
    string pattern;
    //текст
    string text;
    //префикс-функция
    vector <int> prefix;
    //результирующий вектор
    vector <int> result;
    //для хранения блоков
    vector <string> blocks;
    //для подсчёта кол-ва блоков при их поиске

    //функции-сеттеры
    void setPattern(string P){
        pattern = move(P);
    }

    void setText(string T){
```

```
        text = move(T);
    }

    void setBlock(int B){
        numBlock = B;
    }

    //функция поиска индекса elem в результирующем векторе
    bool find(int elem){
        for(int i : result){
            if(i == elem)
                return true;
        }
        return false;
    }

    //разрезание исходного текста на блоки
    void trainingText(){
        cout << "Исходный текст: ";
        printText();
        cout << "Количество блоков равно: " << numBlock << endl;
        //изменение размера вектора
        blocks.resize(numBlock);
        //минимальный размер блока
        sizeBlock = floor(sizeText / numBlock);
        //максимальный размер блока = мин размер + размер пересечений
        if(sizePattern != 1)
            maxSizeBlock = sizeBlock + (sizePattern - 1);
        //максимальный размер блока для 1 - это размер текста
        else
            maxSizeBlock = sizeBlock;
          //временные переменные, необходимые для разрезания текста на
части
        string temp;
        int index;
        cout << "Максимальный размер блока: " << maxSizeBlock << endl;
        //разрезание на блоки
        for(int i = 0; i < numBlock; i++){
            index = sizeBlock * i;
            //отрезаем от исходного текста необходимый блок
             //отрез идёт от определённого индекса, отрезается наиболь-
шая возможная часть
            for(int j = index; j < maxSizeBlock + index; j++){
                //если ещё есть, что отрезать, сохраняем часть текста к
отрезу
                if(j < sizeText)
                    temp += text[j];
                //последний блок может быть меньше максимального разме-
ра,
```

```
            // поэтому есть возможность добавить просто остаток
            else
                break;
        }
        //кладем в вектор блоков
        blocks[i] = temp;
        //очищаем временную переменную для следующей итерации
        temp.clear();
        //вывод получившихся блоков
        cout << "Блок " << i + 1 << ": ";
        printBlock(i);
        cout << endl;
    }
}

//формирование массива префикс-функции
void createPrefix(){
    cout << "Обрабатываемый шаблон: ";
    printPattern();
    //j, i - указывают на рассматриваемые символы
    int j = 0;
    int i = 1;
    //prefixFunc[0] = 0 - всегда для начального символа
    prefix.push_back(0);

    //пока не просмотрели весь шаблон
    while(i < sizePattern){
        cout << "Обработка: шаблон[" << j << "] = " <<  pattern[j];
        cout << " и шаблон[" << i << "] = " <<  pattern[i] << endl;
        //если символ повторяется
        if(pattern[i] == pattern[j]){
            cout << "Найдено совпадение символов" << endl;
            cout << "prefixFunc[" << i << "] = " << j + 1 << endl;
            //фиксируем полученное значение в префикс-функции
            prefix[i] = j + 1;
            //и двигаемся дальше
            i++;
            j++;
        }
        else{
            cout << "Найдено несовпадение символов";
            //j == 0
            if(!j){
                cout << "prefixFunc[" << i << "] = 0" << endl;
                prefix[i] = 0;
                i++;
            }
            //если j не указывает на начало суффикса
            else{
```

```
                    cout << "j = prefixFunc[" << j - 1 << "] = " <<
prefix[j-1] << endl;
                    //присваеваем значения префикс-функции предыдущего
символа, на который указывает j
                    j = prefix[j - 1];
                }
            }
        }
        cout << "Значения полученной префикс-функции: " << endl;
        printPrefix();
    }

    //поиск вхождений шаблона в текст
    void patternSearch(string block, int index){
        //T - текст P - шаблон
        //начинаем поиск из начала текста и шаблона
        int indexT = 0;
        int indexP = 0;
        cout << endl << "Шаблон: ";
        printPattern();
        cout << "Блок: ";
        printBlock(index);

        //пока не просмотрели весь текст
        while(indexT != block.length()) {
            //проверка на выход из размера шаблона indexP, если индекс
вышел за предел размера, то возврат
            //к самому началу шаблона
            if (indexP > sizePattern - 1){
                indexP = 0;
            }
            //проверка на выход за пределы текста
            if (indexT > block.length())
                break;
            cout << endl;
            cout << "Символы для сравнения: шаблон[" << indexP << "] =
" << pattern[indexP];
            cout << " и блок[" << indexT << "] = " << block[indexT];
            cout << endl;
            // если нашли совпадение
            if(pattern[indexP] == block[indexT]){
                cout << "Найдено совпадение символов" << endl;
                //двигаемся дальше
                indexT++;
                indexP++;
                //достигнут конец шаблона
                if(indexP == sizePattern){
                    cout << "НАЙДЕНО ВХОЖДЕНИЕ ШАБЛОНА В ТЕКСТ: ";
                    //индекс, на котором был найден паттерн
```

```
                                cout << indexT + (sizeBlock * index) -
(sizePattern);
                    //в результирующем векторе уже есть данный индекс
                                if(find(indexT+(sizeBlock * index) -
(sizePattern)))
                        cout << "Вхождение уже записано" << endl;
                    //фиксируем индекс вхождения найденного шаблона
                    else
                          result.push_back(indexT+(sizeBlock * index) -
(sizePattern));
                      //indexT возвращаем на следующий индекс после на-
хождения последнего паттерна, чтобы не пропустить
                        //ничего. Пример, тест: pattern - "aba", text -
"abababababa", numBlock = 1
                    if(sizePattern != 1)
                              indexT = indexT + (sizeBlock * index) -
(sizePattern) + 1;
                    }
            }
            else{
                cout << "Символы не совпадают" << endl;
                //indexP == 0
                if(!indexP)
                    //сдвигаемся по тексту
                    indexT++;
                 //перемещаемся на элемент с индексом = префикс-функции
предыдущего элемента
                else
                    indexP = prefix[indexP - 1];
            }
        }
    }

     //промежуточная функция для работы с остальными элементами кода,
организатор алгоритма
    void algorithm() {
        //разбиение текста
        trainingText();
           //изменение размера массива для хранения значений префикс-
функции
        prefix.resize(pattern.length() - 1);
        //вычисление префикс-функции
        createPrefix();
        //для всех блоков
        for (int i = 0; i < numBlock; i++) {
            cout << endl << "Обрабатываемый блок: " << i + 1 << ") ";
            printBlock(i);
            //поиск шаблона в текущем блоке текста
            patternSearch(blocks[i], i);
```

```
        }
        cout << endl;
        cout << endl << endl << "РЕШЕНИЕ: "<< endl;
        printAnswer();
    }

public:
    void readData(){
        //ввод информации
        string P;
        string T;
        cout << "Введите шаблон: " << endl;
        cin >> P;
        cout << "Введите текст: " << endl;
        cin >> T;
        //инициализация паттерна и текста
        setPattern(P);
        setText(T);
        //вычисление размера паттерна и текста
        sizePattern = (int)pattern.length();
        sizeText = (int)text.length();

        //считывание кол-ва блоков
        int B;
        cout << "Введите количество блоков: " << endl;
        cin >> B;
        int flag = 0;

        //проверка кол-ва блоков
        while(!flag){
            //если кол-во блоков меньше= 0;
              //или размер паттерна = 1, тогда кол-во блоков не может
быть больше размера текста
            if(B <= 0 || (sizePattern == 1 && B > sizeText)){
                cout << "Некорректное количество блоков" << endl;
                cin >> B;
            }
            //ввод блоков оказался верным
            else{
                flag = 1;
            }
        }
        //инициализация кол-ва блоков
        setBlock(B);
        //вызов алгоритма
        algorithm();
    }

    //функция для печати исходного текста
```

```
    void printText(){
        for(char i : text){
            cout << i;
        }
        cout << endl;
    }

    //функция для печати вектора с блоками
    void printBlock(int index){
        cout << blocks[index];
    }

    //функция для печати шаблона
    void printPattern(){
        for(char i : pattern){
            cout << i;
        }
        cout << endl;
    }

    //функция для печати префикс-функции
    void printPrefix(){
        for(int i = 0; i < prefix.size(); i++){
              cout <<"Шаблон[" << i << "] = " << pattern[i] << " имеет
prefixFunc[" << i << "] = " << prefix[i] << endl;
        }
        cout << endl;
    }

    //функция для печати результата
    void printAnswer(){
        cout << "Вхождения шаблона в текст ";
        if(result.empty())
            cout << "не найдены: -1" << endl;
        else{
            cout << "найдены: " ;
            for(int k = 0; k < result.size(); k++){
                if(result.size() - 1 != k)
                    cout << result[k] << ",";
                else
                    cout << result[k] << endl;
            }
        }
    }
};

int main() {
    //класс, реализующий КМП
    Base start;
```

```
    //начало считывания данных
    start.readData();
    return 0;
}
```

ПРИЛОЖЕНИЕ В

Исходный код программы

stepik1.cpp

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

using std::cout;
using std::cin;
using std::string;
using std::endl;
using std::move;
using std::vector;

//класс, реализующий КМП
class Base{
private:
    //длина шаблона
    int sizePattern{};
    //длина текста
    int sizeText{};
    //кол-во блоков для разделения текста
    int numBlock{};
    //размер блока
    int sizeBlock{};
    //максимальный размер обрабатываемого блока
    int maxSizeBlock{};
    //шаблон
    string pattern;
    //текст
    string text;
    //префикс-функция
    vector <int> prefix;
    //результирующий вектор
    vector <int> result;
    //для хранения блоков
    vector <string> blocks;

    //функции-сеттеры
    void setPattern(string P){
        pattern = move(P);
    }

    void setText(string T){
        text = move(T);
```

```
    }

    void setBlock(int B){
        numBlock = B;
    }

//разрезание исходного текста на блоки
    vector<string> trainingText(){
        //минимальный размер блока
        sizeBlock = floor(sizeText / numBlock);
        //максимальный размер блока = мин размер + размер пересечений
        if(pattern.length() != 1)
            maxSizeBlock = sizeBlock + (pattern.length() - 1);
            //максимальный размер блока для 1 - это размер текста
        else
            maxSizeBlock = sizeBlock;
          //временные переменные, необходимые для разрезания текста на
части
        string temp;
        int index;
        //разрезание на блоки
        for(int i = 0; i < numBlock; i++){
            index = sizeBlock * i;
            //отрезаем от исходного текста необходимый блок
             //отрез идёт от определённого индекса, отрезается наиболь-
шая возможная часть
            for(int j = index; j < maxSizeBlock + index; j++){
                //если ещё есть, что отрезать, сохраняем часть текста к
отрезу
                if(j < text.length())
                    temp += text[j];
                       //последний блок может быть меньше максимального
размера,
                    // поэтому есть возможность добавить просто остаток
                else
                    break;
            }
            //кладем в вектор блоков
            blocks[i] = temp;
            //очищаем временную переменную для следующей итерации
            temp.clear();
        }
        return blocks;
    }

    //формирование массива префикс-функции
    void prefixFunction(){
        //j, i - указывают на рассматриваемые символы
        int j = 0;
```

```
        int i = 1;
        //prefixFunc[0] = 0 - всегда для начального символа
        prefix.push_back(0);

        //пока не просмотрели весь шаблон
        while(i < pattern.length()){
            //если символ повторяется
            if(pattern[i] == pattern[j]){
                //фиксируем полученное значение в префикс-функции
                prefix[i] = j + 1;
                //и двигаемся дальше
                i++;
                j++;
            }
            else{
                //j == 0
                if(!j){
                    prefix[i] = 0;
                    i++;
                }
                    //если j не указывает на начало суффикса
                else{
                     //присваеваем значения префикс-функции предыдущего
символа, на который указывает j
                    j = prefix[j - 1];
                }
            }
        }
    }

    // Вывод индексов вхождений pattern в text, либо -1, если их нет
    void patternSearch(string block) {
        int currentLength = 0;
        bool chechOccurrence = false;
        int indexT = 0;
        //пока не просмотрели весь текст
        while (indexT != block.length()) {
             //проверка на выход из размера шаблона indexP, если индекс
вышел за предел размера, то возврат
            //к самому началу шаблона
                while (currentLength > 0 && pattern[currentLength] !=
block[indexT]) {
                currentLength = prefix[currentLength - 1];
            }
            if (pattern[currentLength] == block[indexT]) {
                currentLength++;
            }
            if (currentLength == sizePattern) {
                if (!chechOccurrence) {
```

```
                    cout << indexT - currentLength + 1;
                    chechOccurrence = true;
                }
                else {
                    cout << "," << indexT - currentLength + 1;
                }
            }
            indexT++;
        }
        if (!chechOccurrence) {
            cout << -1;
        }
    }

     //промежуточная функция для работы с остальными элементами кода,
организатор алгоритма
    void algorithm() {
        blocks.resize(numBlock);
        //разбиение текста
        trainingText();
           //изменение размера массива для хранения значений префикс-
функции
        prefix.resize(pattern.length() - 1);
        //вычисление префикс-функции
        prefixFunction();
        //для всех блоков
        for (int i = 0; i < numBlock; i++) {
            //поиск шаблона в текущем блоке текста
            patternSearch(blocks[i]);
        }
    }

public:
    void readData(){
        //ввод информации
        string P;
        string T;
        cin >> P;
        cin >> T;
        int B = 1;
        //инициализация паттерна и текста
        setPattern(P);
        setText(T);
        //вычисление размера паттерна и текста
        sizePattern = (int)pattern.length();
        sizeText = (int)text.length();
        //инициализация кол-ва блоков
        setBlock(B);
        //вызов алгоритма
```

```
        algorithm();
    }
};

int main() {
    //класс, реализующий КМП
    Base start;
    //начало считывания данных
    start.readData();
    return 0;
}
```

ПРИЛОЖЕНИЕ С

Исходный код программы

stepik2.cpp

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


using std::cout;
using std::endl;
using std::vector;
using std::string;
using std::cin;


// Вычисление префикс-функции для строки text, возвращает вектор того же размера, что и строка
vector<int> createPrefix(string& text) {
    vector<int> prefix(text.length());
    prefix[0] = 0;
    for (int i = 1; i < text.length(); i++) {
        int currentLength = prefix[i - 1];
         // Если предыдущий суффикс нельзя расширить, нужно попытаться взять суффикс меньшего размера
         // Этот потенциальный размер суффикса по сути является значением prefix[currentLength - 1]
        while (currentLength > 0 && text[currentLength] != text[i]) {
            currentLength = prefix[currentLength - 1];
        }
        // Если символы справа от префикса и суффикса совпадают, суффикс расширяется
        if (text[currentLength] == text[i]) {
            currentLength++;
        }
        prefix[i] = currentLength;
    }
    return prefix;
}


// Если строка pattern является сдвигом text, выводится индекс начала вхождения, иначе -1
void shift(string& text, string& pattern) {
    if (text.length() != pattern.length()) {
        cout << -1;
        return;
    }
    vector<int> prefix = createPrefix(pattern);
    int textLength = text.length();
```

```
    int curLength = 0;
    for (int i = 0; i < textLength * 2; i++) {
        // j используется для циклического прохода по строке
        int j = i % textLength;
         // Если предыдущий суффикс нельзя расширить, нужно попытаться взять суффикс меньшего размера
         // Этот потенциальный размер суффикса по сути является значением pattern[currentLength - 1]
        while (curLength > 0 && pattern[curLength] != text[j]) {
            curLength = prefix[curLength - 1];
        }
        // Если символы справа от префикса и суффикса совпадают, суффикс расширяется
        if (pattern[curLength] == text[j]) {
            curLength++;
        }
          // Если длина текущего суффикса равна длине шаблона, найдено вхождение
        if (curLength == textLength) {
            cout << i - curLength + 1;
            return;
        }
    }
    cout << -1;
}


int main() {
    string A;
    string B;
    cin >> A;
    cin >> B;
    shift(A, B);
    return 0;
}
```

ПРИЛОЖЕНИЕ D

Исходный код программы

code_with_comments_shift.cpp

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

using std::cout;
using std::endl;
using std::vector;
using std::string;
using std::cin;

// Вычисление префикс-функции для строки text, возвращает вектор того
же размера, что и строка
vector<int> createPrefix(string& text) {
    cout << "Поиск значений префикс-функции" << endl;
    vector<int> prefix(text.length());
    prefix[0] = 0;
    for (int i = 1; i < text.length(); i++) {
        int currentLength = prefix[i - 1];
         // Если предыдущий суффикс нельзя расширить, нужно попытаться
взять суффикс меньшего размера
         // Этот потенциальный размер суффикса по сути является значе-
нием prefix[currentLength - 1]
         cout << "Происходит вычисление значения префикс-функции для: "
<< text[i] << endl;
        while (currentLength > 0 && text[currentLength] != text[i]) {
             cout << "Предыдущий суффикс нельзя расширить, нужно взять
суффикс меньшего размера" << endl;
            cout << "Размер суффикса: " << currentLength << endl;
            currentLength = prefix[currentLength - 1];
               cout << "Меняем на новый размер: " << currentLength <<
endl;
        }
        // Если символы справа от префикса и суффикса совпадают, суффи-
кс расширяется
        if (text[currentLength] == text[i]) {
            cout << "Можно расширить суффикс" << endl;
            cout << "Размер суффикса: " << currentLength << endl;
            currentLength++;
               cout << "Меняем на новый размер: " << currentLength <<
endl;
        }
        cout << "Значение префикс-функции для: " << text[i] << " = " <<
currentLength << endl;
        prefix[i] = currentLength;
```

```
    }
    return prefix;
}

// Если строка pattern является сдвигом text, выводится индекс начала вхождения, иначе -1
void shift(string& text, string& pattern) {
    if (text.length() != pattern.length()) {
            cout << "A не является циклическим сдвигом B, поэтому результат: ";
        cout << -1;
        return;
    }
    vector<int> prefix = createPrefix(pattern);
    int textLength = text.length();
    int curLength = 0;
    cout << "Начало основной проверки на циклический сдвиг после вычисления префикс-функции" << endl;
    for (int i = 0; i < textLength * 2; i++) {
        // j используется для циклического прохода по строке
        int j = i % textLength;
         // Если предыдущий суффикс нельзя расширить, нужно попытаться взять суффикс меньшего размера
         // Этот потенциальный размер суффикса по сути является значением pattern[currentLength - 1]
        while (curLength > 0 && pattern[curLength] != text[j]) {
              cout << pattern[curLength] << " != " << text[j] << " => Нельзя расширить суффикс, уменьшаем его" << endl;
            cout << "Размер суффикса: " << curLength << endl;
            curLength = prefix[curLength - 1];
            cout << "Меняем на новый размер: " << curLength << endl;
        }
        // Если символы справа от префикса и суффикса совпадают, суффикс расширяется
        if (pattern[curLength] == text[j]) {
              cout << pattern[curLength] << " == " << text[j] << " => Можно расширить суффикс" << endl;
            cout << "Размер суффикса: " << curLength << endl;
            curLength++;
            cout << "Меняем на новый размер: " << curLength << endl;
        }
         // Если длина текущего суффикса равна длине шаблона, найдено вхождение
        if (curLength == textLength) {
             cout << "Длина текущего суффикса: " << curLength << " == " << "Длине шаблона: " << textLength <<
             " => Строка является циклическим сдвигом, индекс строки: " << i - curLength + 1 << endl;
            return;
```

```
        }
    }
    cout << -1;
}

int main() {
    string A;
    string B;
    cout << "Введите строку A" << endl;
    cin >> A;
    cout << "Введите строку B" << endl;
    cin >> B;
    cout << "Начало функции циклического сдвига" << endl;
    shift(A, B);
    return 0;
}
```