6МИНОБРНАУКИ 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: Кнут-Моррис-Пратт

Студентка гр. 3343 _____ Малиновский А.А,

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Изучить принцип работы алгоритма Кнута-Морриса_Пратта. Написать функцию, вычисляющую для каждого элемента строки максимальное значение длины префикса и с помощью данной функции решить поставленные задачи. А именно написать программу, осуществляющую поиск вхождений подстроки в строку, а также программу, определяющую, являются ли строки циклическим сдвигом друг друга, найти индекс начала вхождения второй строки в первую.

Задание №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

Задание №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* вляется 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 алгоритмов.

Описание алгоритма.

Префикс-функция:

Алгоритм начинается с инициализации трех переменных:

- пустой список prefixes, заполненный нулями длиной строки, для которой нужно найти префикс функцию.

- i – индекс, для прохождения по строке

- j – переменная, хранящая в себе текущую длину совпадений суффикса с префиксом.

Далее, пока не достигнут конец строки, проверяем максимальное число совпадений символов, попутно увеличивая счетчики i,j. Как только мы нашли первые неравные символы, появляется два исхода: 1) совпадений не было, т. е. Можем просто продолжить перебор, оставив текущий префикс нулевым. 2) Совпадения были и нам требуется вернуть значение j на prefixes[j-1]. Откат на prefixes[j-1] символов позволяет нам эффективно продолжать поиск с

максимально возможной позиции в подстроке, не повторяя уже выполненных проверок.

Таким образом, формируется список, состоящих из максимальных длин префиксов. Далее данный список prefixes возвращается.

Алгоритм Кнута-Морриса-Пратта:

Принцип работы алгоритма КМП основан на использовании префикс-функции. При сравнении подстроки с символами строки алгоритм не перескакивает через уже проверенные символы, а использует префикс-функцию, чтобы сдвигаться вправо с наибольшей возможной позиции.

Алгоритм состоит из двух шагов:

Построение префикс-функции для подстроки.

Поиск всех вхождений подстроки в строку с использованием префикс-функции.

Шаг 1. Описан в разделе Префикс-функция

Шаг 2. Далее для поиска всех вхождений подстроки в строку мы идем по строке и подстроке одновременно, сравнивая символы на каждой позиции. Если символы совпадают, мы переходим к следующей позиции. Если символы не совпадают, мы используем префикс-функцию, чтобы определить, на какую позицию нужно сдвинуть подстроку вправо для продолжения сравнения с символами строки без потери информации о возможных совпадениях. Мы сдвигаем подстроку на значение prefixes[j-1], где j - позиция, на которой произошло несовпадение.

Алгоритм продолжает сравнение символов до тех пор, пока не найдет все вхождения подстроки в строку. Если мы достигаем конца строки, но не нашли вхождений, то возвращается пустой массив, затем выводится -1.

Оценка сложности алгоритма по памяти и операциям.

1. Сложность алгоритма поиска подстроки.

Сложность по времени линейная O(n+m), где m – длина подстроки, n – длина строки. Так как за O(m) осуществляется построение префикс-функции, а также за O(n) осуществляется проход по строке, чтобы найти индексы вхождения.

Сложность по памяти O(m), так как нужно хранить вектор префиксов данной длины.

2. Сложность алгоритма поиска циклического сдвига. Сложность по времени O(m+2n), где n – длина строки. Так как за O(m) осуществится построение префикс функции, а за O(2n) дважды будет осуществлен проход строки.

Сложность по памяти O(m).

Описание функций.

В процессе выполнения работы были написаны следующие функции:

```
std::vector<int> prefixFunction(const std::string& pattern)
```

Функция, принимающая на вход строку и вычисляющая значения максимальных длин префиксов для каждого элемента. Результат записывает в контейнер std::vector и возвращает его.

```
std::vector<int> kmp(const std::string& pattern, const std::string& text)
```

Функция, принимающая на вход подстроку pattern, вхождение которой будем искать в строке text. Возвращается строка, содержащая информацию об индексах начала вхождений подстроки в строку.

```
int findCyclicShift(const string& pattern, const string& text)
```

Функция, принимающая на вход две строки, первым аргументом принимается та строка, в которой будет осуществляться поиск сдвига, а вторым та, которую будем искать. Возвращает индекс начала вхождения второй строки в первую.

Также были созданы файлы для измерения времени выполнения КМП с различными входными данными, реализован алгоритм наивного поиска для сравнения и программа на python, создающая графики из выборки.

`void printPrefix (std::vector<int> const& prefix)`. Функция, принимающая на вход вектор и выводящая элементы вектора в консоль.

Тестирование.

Проведем тестирование.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	ab avdabhjab	Результат: 3, 7	Тест к первому заданию. Верно найдены индексы вхождения подстроки в строку.
2.	abc avdabhjab	Результат: -1	Тест ко первому заданию, когда нет ни одного вхождения подстроки в строку. Результатом в данном случае будет -1.
3.	ababab bababa	Результат: 1	Тест ко второму заданию. Верно определен индекс первого вхождения циклического сдвига (тут их несколько).
4.	abcedf abcdef	Результат: -1	Тест к второму заданию. Верно определено то, что нет циклического сдвига, результат = -1.
5.	abcedfabccaab	0 0 0 0 0 0 1 2 3 0 1 1 2	Протестирована функция prefixFunction(), верно вычислен результат.

Результат работы программы с отладочным выводом для первого задания (см. рис 1, 2, 3).

```
Введите паттерн: abbab
Введите текст: abaaba
Вычисление префикс-функции для паттерна: abbab

Итерация 1:
  i = 1, j = 0
  Сравниваем символы: pattern[1] = b и pattern[0] = a
  Устанавливаем prefixes[1] = 0
  Текущий массив prefixes: 0 0

Итерация 2:
  i = 2, j = 0
  Сравниваем символы: pattern[2] = b и pattern[0] = a
  Устанавливаем prefixes[2] = 0
  Текущий массив prefixes: 0 0 0

Итерация 3:
  i = 3, j = 0
  Сравниваем символы: pattern[3] = a и pattern[0] = a
    Символы совпали: pattern[3] = a == pattern[0] = a
    Увеличиваем j до 1
  Устанавливаем prefixes[3] = 1
  Текущий массив prefixes: 0 0 0 1

Итерация 4:
  i = 4, j = 1
  Сравниваем символы: pattern[4] = b и pattern[1] = b
    Символы совпали: pattern[4] = b == pattern[1] = b
    Увеличиваем j до 2
  Устанавливаем prefixes[4] = 2
  Текущий массив prefixes: 0 0 0 1 2

Итоговый массив префикс-функции: 0 0 0 1 2
```

Рисунок 1 – вычисление префикс функции

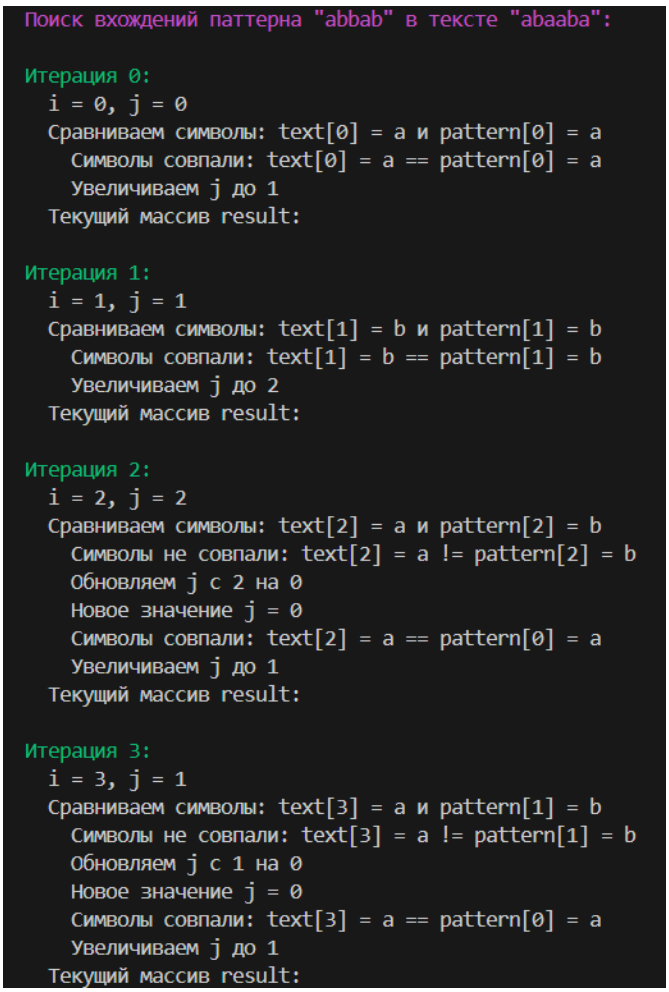

Рисунок 2 – вывод KMP

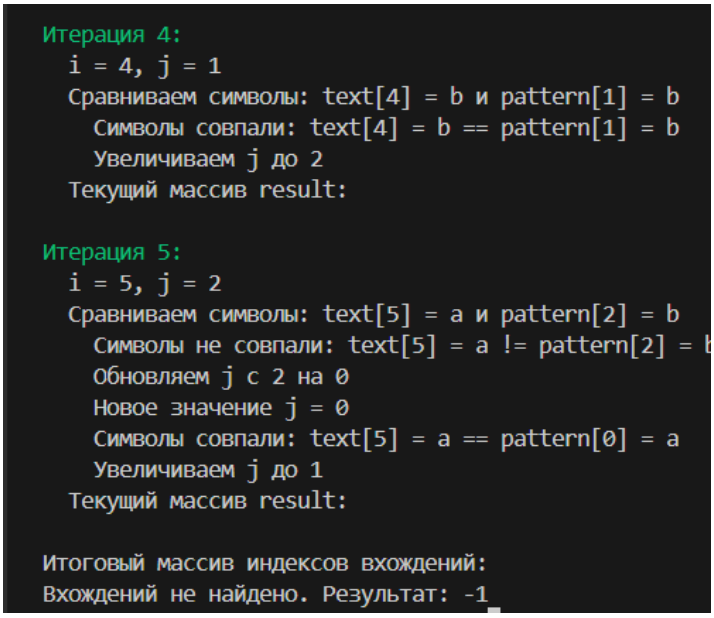

Рисунок 3 – вывод KMP

Исследование.

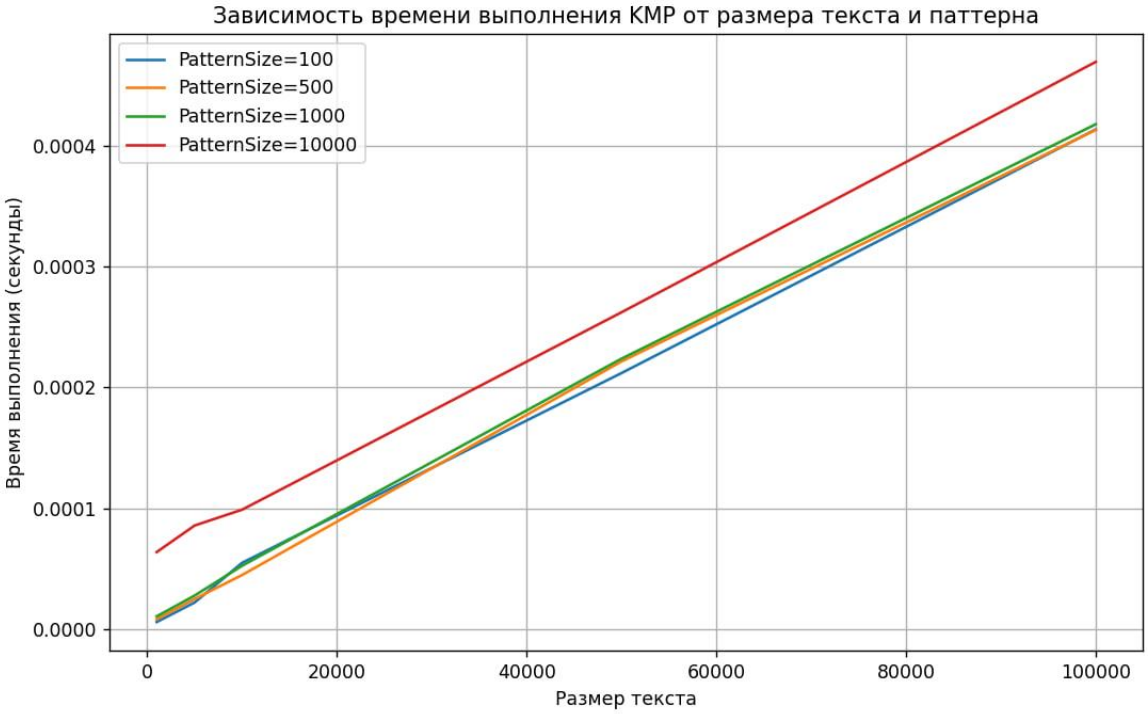

Рисунок 4 – Тестирование КМП

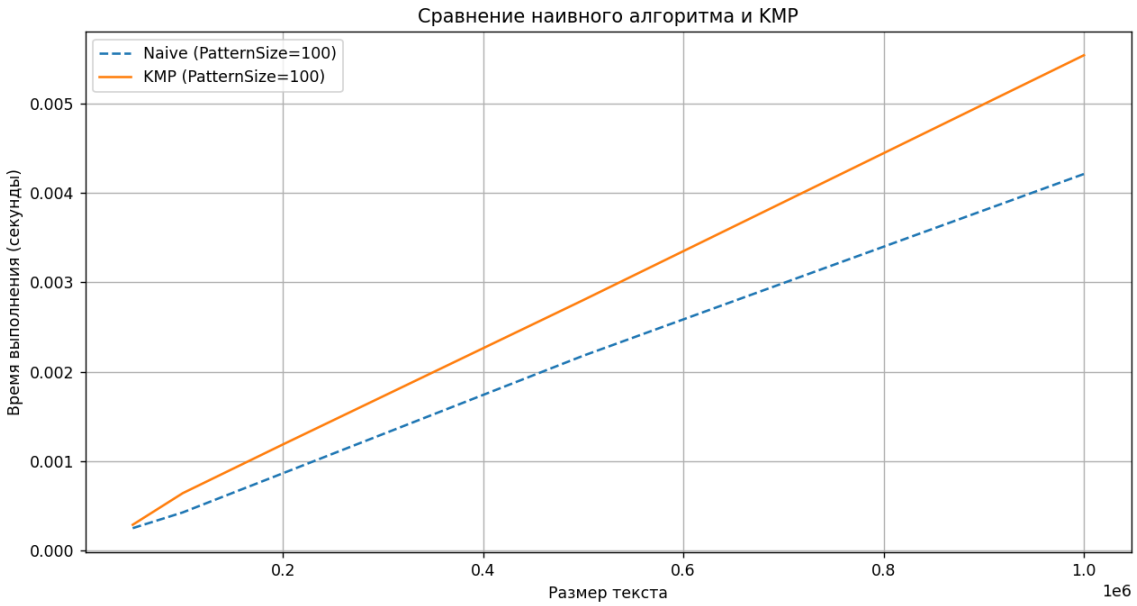

Рисунок 5 – Сравнение КМП и прямого обхода

Можно сделать вывод, что КМП выполняется быстрее, чем наивный алгоритм, показатели могут быть лучше на выборках, которые содержат много последовательностей символов входящих в подстроку.

Выводы.

Изучен принцип работы алгоритма Кнута-Морриса-Пратта. Написаны программы, корректно решающие поставленные задачи с помощью функции вычисления максимальной длины префикса для каждого символа.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: kmp.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <windows.h>

#define RESET "\x1B[0m"
#define GREEN(text) "\x1B[32m" << text << RESET
#define RED(text) "\x1B[31m" << text << RESET
#define BLUE(text) "\x1B[34m" << text << RESET
#define YELLOW(text) "\x1B[33m" << text << RESET
#define MAGENTA(text) "\x1B[35m" << text << RESET
#define CYAN(text) "\x1B[36m" << text << RESET

// Функция для вычисления префикс-функции
std::vector<int> prefixFunction(const std::string& pattern) {
    int n = pattern.length();
    std::vector<int> prefixes(n, 0);
    int j = 0;

    std::cout << MAGENTA("Вычисление префикс-функции для паттерна: "
<< pattern )<< std::endl;

    for (int i = 1; i < n; ++i) {
        std::cout << GREEN("\nИтерация " << i << ":") << std::endl;
        std::cout << "  i = " << i << ", j = " << j << std::endl;
        std::cout << "  Сравниваем символы: pattern[" << i << "] = "
<< pattern[i]
                  << " и pattern[" << j << "] = " << pattern[j] <<
std::endl;

        while (j > 0 && pattern[i] != pattern[j]) {
            std::cout << "    Символы не совпали: pattern[" << i <<
"] = " << pattern[i]
                      << " != pattern[" << j << "] = " << pattern[j]
<< std::endl;
             std::cout << "    Обновляем j с " << j << " на " <<
prefixes[j - 1] << std::endl;
            j = prefixes[j - 1];
            std::cout << "    Новое значение j = " << j << std::endl;
        }

        if (pattern[i] == pattern[j]) {
            std::cout << "    Символы совпали: pattern[" << i << "]
= " << pattern[i]
                      << " == pattern[" << j << "] = " << pattern[j]
<< std::endl;
            j++;
            std::cout << "    Увеличиваем j до " << j << std::endl;
        }
```

```
            prefixes[i] = j;
             std::cout << "  Устанавливаем prefixes[" << i << "] = " <<
prefixes[i] << std::endl;

            // Выводим текущее состояние массива prefixes
            std::cout << "  Текущий массив prefixes: ";
            for (int k = 0; k <= i; ++k) {
                std::cout << prefixes[k] << " ";
            }
            std::cout << std::endl;
        }

        std::cout << "\nИтоговый массив префикс-функции: ";
        for (int val : prefixes) {
            std::cout << val << " ";
        }
        std::cout << std::endl;

        return prefixes;
    }

    // Функция для поиска всех вхождений pattern в text
    std::vector<int> kmp(const std::string& pattern, const std::string&
text) {
        int patternLen = pattern.length();
        int textLen = text.length();
        std::vector<int> prefixes = prefixFunction(pattern);
        std::vector<int> result;
        int j = 0;

        std::cout << MAGENTA("\nПоиск вхождений паттерна \"" << pattern
<< "\" в тексте \"" << text << "\":") << std::endl;

        for (int i = 0; i < textLen; ++i) {
            std::cout << GREEN("\nИтерация " << i << ":") << std::endl;
            std::cout << "  i = " << i << ", j = " << j << std::endl;
            std::cout << "  Сравниваем символы: text[" << i << "] = " <<
text[i]
                      << " и pattern[" << j << "] = " << pattern[j] <<
std::endl;

            while (j > 0 && text[i] != pattern[j]) {
                std::cout << "    Символы не совпали: text[" << i << "]
= " << text[i]
                          << " != pattern[" << j << "] = " << pattern[j]
<< std::endl;
                 std::cout << "    Обновляем j с " << j << " на " <<
prefixes[j - 1] << std::endl;
                j = prefixes[j - 1];
                std::cout << "    Новое значение j = " << j << std::endl;
            }

            if (text[i] == pattern[j]) {
                std::cout << "    Символы совпали: text[" << i << "] = "
<< text[i]
                          << " == pattern[" << j << "] = " << pattern[j]
<< std::endl;
```

```
                j++;
                std::cout << "    Увеличиваем j до " << j << std::endl;
            }

            if (j == patternLen) {
                std::cout << "    Найдено вхождение паттерна на позиции
" << (i - j + 1) << std::endl;
                result.push_back(i - j + 1);  // Сохраняем индекс начала
вхождения
                  std::cout << "    Обновляем j с " << j << " на " <<
prefixes[j - 1] << std::endl;
                j = prefixes[j - 1];  // Восстанавливаем индекс паттерна
                std::cout << "    Новое значение j = " << j << std::endl;
            }

            // Выводим текущее состояние массива result
            std::cout << "  Текущий массив result: ";
            for (int val : result) {
                std::cout << val << " ";
            }
            std::cout << std::endl;
        }

        std::cout << "\nИтоговый массив индексов вхождений: ";
        for (int val : result) {
            std::cout << val << " ";
        }
        std::cout << std::endl;

        return result;
    }

    int main() {
        SetConsoleOutputCP(CP_UTF8);
        SetConsoleCP(CP_UTF8);

        std::string pattern, text;
        std::cout << "Введите паттерн: ";
        std::cin >> pattern;
        std::cout << "Введите текст: ";
        std::cin >> text;

        std::vector<int> answer = kmp(pattern, text);

        if (answer.empty()) {
              std::cout << "Вхождений не найдено. Результат: -1" <<
std::endl;
        } else {
            std::cout << "Результат: ";
            for (size_t i = 0; i < answer.size(); ++i) {
                if (i > 0) {
                    std::cout << ",";
                }
                std::cout << answer[i];
            }
            std::cout << std::endl;
        }
```

```
        return 0;
    }
}
```

Название файла: cycle.cpp

```
    #include <iostream>
    #include <vector>
    #include <string>
    #include <windows.h>

    using namespace std;

    // Функция для вычисления префикс-функции
    vector<int> prefixFunction(const string& pattern) {
        int n = pattern.length();
        vector<int> prefixes(n, 0);
        int j = 0;

        cout << "Вычисление префикс-функции для паттерна: " << pattern
<< endl;

        for (int i = 1; i < n; ++i) {
            cout << "\nИтерация " << i << ":" << endl;
            cout << "  Сравниваем символы: pattern[" << i << "] = " <<
pattern[i]
                 << " и pattern[" << j << "] = " << pattern[j] << endl;

            while (j > 0 && pattern[i] != pattern[j]) {
                cout << "    Символы не совпали. Обновляем j с " << j <<
" на " << prefixes[j - 1] << endl;
                j = prefixes[j - 1];
            }

            if (pattern[i] == pattern[j]) {
                cout << "    Символы совпали. Увеличиваем j с " << j <<
" на " << j + 1 << endl;
                ++j;
            }

            prefixes[i] = j;
              cout << "  Устанавливаем prefixes[" << i << "] = " <<
prefixes[i] << endl;

            // Выводим текущее состояние массива prefixes
            cout << "  Текущий массив prefixes: ";
            for (int k = 0; k <= i; ++k) {
                cout << prefixes[k] << " ";
            }
            cout << endl;
        }

        cout << "\nИтоговый массив префикс-функции: ";
        for (int val : prefixes) {
            cout << val << " ";
        }
        cout << endl;
```

```
        return prefixes;
    }

    // Функция для поиска циклического сдвига
    int findCyclicShift(const string& pattern, const string& text) {
        int patternLen = pattern.length();
        int textLen = text.length();
        vector<int> prefixes = prefixFunction(pattern);
        int j = 0;

        cout << "\nПоиск циклического сдвига для паттерна: " << pattern
<< " в тексте: " << text << endl;

        for (int i = 0; i < 2 * textLen; ++i) {
            int idx = i % textLen;  // Используем индекс по модулю длины
строки
            cout << "\nИтерация " << i << ":" << endl;
             cout << "  Индекс в тексте: " << idx << ", символ: " <<
text[idx] << endl;
            cout << "  Текущее значение j: " << j << endl;

            while (j > 0 && text[idx] != pattern[j]) {
                cout << "    Символы не совпали: text[" << idx << "] = "
<< text[idx]
                     << " != pattern[" << j << "] = " << pattern[j] <<
endl;
                cout << "    Обновляем j с " << j << " на " << prefixes[j
- 1] << endl;
                j = prefixes[j - 1];
                cout << "    Новое значение j: " << j << endl;
            }

            if (text[idx] == pattern[j]) {
                cout << "    Символы совпали: text[" << idx << "] = " <<
text[idx]
                     << " == pattern[" << j << "] = " << pattern[j] <<
endl;
                ++j;
                cout << "    Увеличиваем j до " << j << endl;
            }

            if (j == patternLen) {
                 cout << "    Найдено полное совпадение! Индекс начала
сдвига: " << (i - j + 1) << endl;
                return i - j + 1;
            }
        }

        cout << "    Совпадение не найдено." << endl;
        return -1;
    }

    int main() {
        SetConsoleOutputCP(CP_UTF8);
        SetConsoleCP(CP_UTF8);

        string text, pattern;
```

```
        cout << "Введите текст: ";
        cin >> text;
        cout << "Введите паттерн: ";
        cin >> pattern;

        if (text.length() != pattern.length()) {
            cout << "Длины текста и паттерна не совпадают. Результат: -
1" << endl;
        } else {
            int result = findCyclicShift(pattern, text);
            cout << "Результат: " << result << endl;
        }

        return 0;
    }
```