



Министерство науки и высшего образования Российской Федерации  
Калужский филиал  
федерального государственного бюджетного  
образовательного учреждения высшего образования  
«Московский государственный технический университет имени Н.Э.  
Баумана (национальный исследовательский университет)»  
(КФ МГТУ им. Н.Э. Баумана)

**ФАКУЛЬТЕТ ИУК "Информатика и управление"**

**КАФЕДРА ИУК5 "Системы обработки информации"**

## **ЛАБОРАТОРНАЯ РАБОТА №5**

### **«Рекурсия и поиск подстроки.»**

**ДИСЦИПЛИНА: «Вычислительные алгоритмы»**

Выполнил: студент гр.ИУК5-41Б

\_\_\_\_\_ (\_\_\_\_ Шиндин А.О.\_\_\_\_)  
(Подпись) (Ф.И.О.)

Проверил:

\_\_\_\_\_ (\_\_\_\_ Вершинин В.Е.\_\_\_\_)  
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2023

**Цель:** получение практических навыков при реализации рекурсивных функций, типовых алгоритмов поиска подстроки в строке.

**Задачи:**

- 1) согласно варианту изучить словесную постановку задачи, выделив при этом все виды данных;
- 2) сформулировать математическую постановку задачи;
- 3) реализовать алгоритмы;
- 4) исследовать вычислительную сложность алгоритмов;

**Задание для варианта 13:**

**Рекурсия:** Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите наибольшее значение числа в этой последовательности. В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция возвращает единственное значение: максимум считанной последовательности. Гарантируется, что последовательность содержит хотя бы одно число (кроме нуля).

**Поиск подстроки:** Алгоритм Апостолико-Кроchemора (Apostolico-Crochemore algorithm)

**Ход работы:**

1. Рекурсивный алгоритм поиска максимального элемента в последовательности:

- Постановка задачи: Найти максимальный элемент в последовательности натуральных чисел.
- Формально: Пусть дана последовательность  $(a_1, a_2, \dots, a_n)$  натуральных чисел, где  $(n)$  - длина последовательности. Задача состоит в том, чтобы найти максимальное значение  $(\max(a_1, a_2, \dots, a_n))$ .
- Исходные данные: Последовательность натуральных чисел длины  $(n)$ .
- Результат: Максимальный элемент в последовательности.

2. Алгоритм поиска подстроки в тексте с использованием алгоритма Апостолико-Кроchemора:

- Постановка задачи: Найти все вхождения заданной подстроки в тексте.
- Формально: Пусть дан текст  $(T)$  и подстрока  $(P)$ . Задача состоит в том, чтобы найти все позиции в тексте, в которых встречается подстрока  $(P)$ .
- Исходные данные: Текст  $(T)$  и подстрока  $(P)$ .
- Результат: Список позиций в тексте, в которых встречается подстрока  $(P)$ .

1) Программа написанная на языке c++ для рекурсии:

```
#include <iostream>
```

```

#include <vector>
#include <cstdlib>
#include <ctime>
/*
int findMax() {
int num;
cin >> num;
if (num == 0) {
return 0;
} else {
int max_rest = findMax();
return (num > max_rest) ? num : max_rest;
}
}
*/

void generateSequence(std::vector<int>& sequence, int length) {
srand(time(NULL));
for (int i = 0; i < length - 1; ++i) {
int num = rand() % 101; // Генерация чисел в диапазоне от 0 до 100
sequence.push_back(num);
std::cout << num << std::endl;
}
std::cout << "0" << std::endl; // Завершающий 0
}

int findMax(const std::vector<int>& sequence, int index) {
if (index == sequence.size()) {
return 0;
} else {
int num = sequence[index];
int max_rest = findMax(sequence, index + 1);
return (num > max_rest) ? num : max_rest;
}
}

int main() {
std::cout << "Enter the length of the sequence of natural numbers (ending with zero):
";

```

```

int length; std::cin >> length;

std::vector<int> sequence;
generateSequence(sequence, length);

int max_num = findMax(sequence, 0);
std::cout << "The maximum value in the sequence: " << max_num << std::endl;
return 0;
}

```

### Результат:

```

Enter the length of the sequence of natural numbers (ending with zero): 15
26
90
53
46
97
98
15
19
95
51
57
12
51
83
0
The maximum value in the sequence: 98
alex@fedora:~/Documents/code/alg$

```

2) Программа написанная на языке с++ для поиска подстроки:

```

#include <iostream>

#include <vector>
#include <string>
#include <fstream>

// Функция для построения таблицы смещений
void buildOffsetTable(const std::string& pattern, std::vector<int>& offsetTable) {
    int m = pattern.size(); // Длина шаблона
    offsetTable.resize(256, m); // Инициализация таблицы смещений, заполняем ее
    значением длины шаблона
}

```

```

// Заполнение таблицы смещений для символов в шаблоне
for (int i = 0; i < m - 1; ++i) {
    offsetTable[static_cast<int>(pattern[i])] = m - 1 - i; // Заполнение таблицы
    смещений для каждого символа в шаблоне
}
}

// Функция для поиска всех вхождений шаблона в тексте с использованием
алгоритма Апостолико-Крокемора
void apostolicoCrochemore(const std::string& text, const std::string& pattern) {
    int n = text.size(); // Длина текста
    int m = pattern.size(); // Длина шаблона
    std::vector<int> offsetTable; // Таблица смещений
    buildOffsetTable(pattern, offsetTable); // Построение таблицы смещений

    int i = m - 1; // Индекс в тексте
    int j = m - 1; // Индекс в шаблоне

    // Поиск вхождений шаблона в текст
    while (i < n) {
        if (text[i] == pattern[j]) { // Если символы совпадают
            if (j == 0) { // Если мы дошли до начала шаблона, то мы нашли вхождение
                std::cout << "Pattern found at index " << i << std::endl; // Выводим индекс
                вхождения
            }
            i += m; // Перемещаем индекс в тексте на длину шаблона
            j = m - 1; // Сбрасываем индекс в шаблоне
        } else {
            --i; // Перемещаемся к предыдущему символу в тексте
            --j; // Перемещаемся к предыдущему символу в шаблоне
        }
    } else {
        // Вычисляем смещение на основе таблицы смещений
        i += offsetTable[static_cast<int>(text[i])] < m - j ? m - j :
        offsetTable[static_cast<int>(text[i])];
        j = m - 1; // Сбрасываем индекс в шаблоне
    }
}

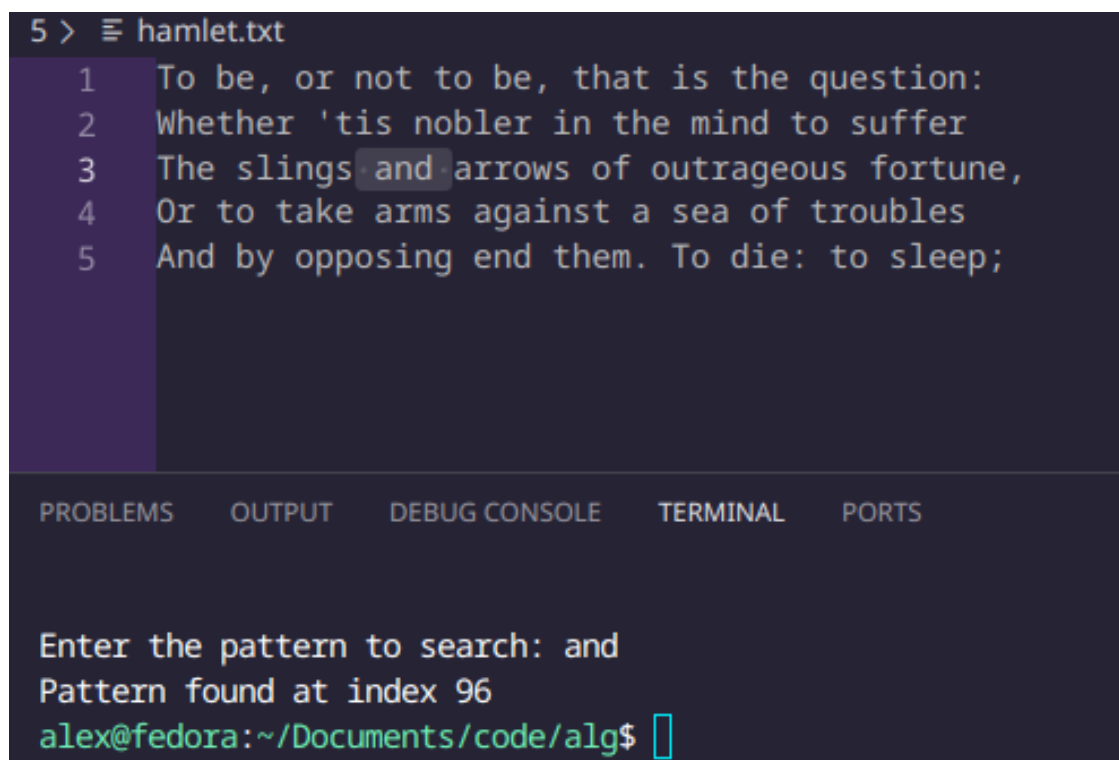
```

```

}
}
int main() {
std::ifstream file("hamlet.txt");
if (!file.is_open()) { // Если файл не открылся, выводим сообщение об ошибке
std::cerr << "Failed to open the file." << std::endl;
return 1;
}
std::string line;
std::string text;
// Считываем содержимое файла в строку text
while (std::getline(file, line)) {
text += line + "\n";
}
file.close(); // Закрываем файл
std::string pattern;
std::cout << "Enter the pattern to search: ";
std::cin >> pattern; // Запрашиваем у пользователя шаблон для поиска
apostolicoCrochemore(text, pattern); // Вызываем функцию для поиска всех
вхождений шаблона в текст
return 0;
}

```

### Результат:



The screenshot shows a code editor with a file named `hamlet.txt` open. The file contains the following text:

```

1 To be, or not to be, that is the question:
2 Whether 'tis nobler in the mind to suffer
3 The slings and arrows of outrageous fortune,
4 Or to take arms against a sea of troubles
5 And by opposing end them. To die: to sleep;

```

Below the code editor is a terminal window. The terminal shows the program's output:

```

Enter the pattern to search: and
Pattern found at index 96
alex@fedora:~/Documents/code/alg$

```

Теперь перейдем к исследованию вычислительной сложности.

1. Рекурсивный алгоритм поиска максимального элемента:

- Временная сложность (в худшем случае): ( $O(n)$ ), где ( $n$ ) - количество элементов в последовательности. Этот алгоритм имеет линейную временную сложность, так как он проходит по каждому элементу в последовательности ровно один раз.

- Пространственная сложность: ( $O(n)$ ), так как на каждом уровне рекурсии создается новый стек вызовов для хранения промежуточных значений.

2. Алгоритм поиска подстроки в тексте с использованием алгоритма Апостолико-Крокемора:

- Временная сложность: ( $O(n + m)$ ), где ( $n$ ) - длина текста, а ( $m$ ) - длина подстроки. Этот алгоритм имеет линейную временную сложность, так как он проходит по каждому символу в тексте ровно один раз, а таблица смещений строится за время ( $O(m)$ ).

- Пространственная сложность: ( $O(1)$ ), так как алгоритм требует только константное количество дополнительной памяти для хранения таблицы смещений.

**Вывод:** в результате выполнения лабораторной работы были приобретены практические навыки в реализации рекурсивных функций, типовых алгоритмов поиска подстроки в строке