

|  |  |
| --- | --- |
| **Министерство образования и науки**  **Российской Федерации**  **Государственное образовательное учреждение высшего образования «МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ**  **ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **(МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Отчет по дисциплине «Структуры и алгоритмы обработки данных»**

**Практическая работа №5**

**«Методы поиска подстроки в строке и обработка текстовых строк»**

**Выполнил:**

Учебная группа 1бПМ\_\_\_\_\_\_\_\_\_  
ФИО Греча К. П. \_

**Принял:**

Должность Старший преподаватель

Звание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО \_ Кутейников И. А.\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

**Цель:**

Программа должна содержать следующие функции:

1 Инициализация строки с клавиатуры;

2 Инициализация строки случайными символами;

3 Вывод строки на экран;

4 Поиск подстроки;

4.1 Поиск подстроки методом последовательного доступа;

4.2 Поиск подстроки методом Кнута-Морриса-Пратта;

4.3 Поиск подстроки упрощенным методом Бойера-Мура;

4.4 Поиск подстроки встроенной функцией;

4.5 Сравнение времени работы методов;

5 Решение задачи уровня Б;

6 Решение задачи уровня С (опционально);

7 Выход из программы.

Требуется выделить в строке-предложении s все слова, разделенные

символами-разделителями «\_.,;:\n\t!?», и обработать выделенные слова в

соответствии с вариантом задания.

Определения:

Регулярное слово – слово, состоящее только из больших латинских

букв.

**Вариант**: Найти самое длинное регулярное слово и удалить из него все гласные буквы. Найти все слова, в которых имеются либо только цифры, либо только латинские буквы.

**Алгоритм:**

import java.util.Scanner;  
import java.util.Random;  
  
public class Main {  
 private static Scanner *scanner* = new Scanner(System.*in*);  
 private static Random *random* = new Random();  
 private static String *inputString*;  
  
 public static void main(String[] args) {  
 int ans;  
  
 do {  
 System.*out*.println("Выберите пункт меню: " + "\n");  
 System.*out*.println("1) Ввод в ручную ");  
 System.*out*.println("2) Ввод рандомно ");  
 System.*out*.println("3) Вывод строки на экран ");  
 System.*out*.println("4) Поиск подстроки ");  
 System.*out*.println("5) Решение задачи №10 ");  
 System.*out*.println("6) Выход из программы ");  
 System.*out*.print("Выбрать пункт меню: ");  
 ans = *scanner*.nextInt();  
 *scanner*.nextLine();  
  
 switch (ans) {  
 case 1:  
 *fillByHand*(*scanner*);  
 break;  
  
 case 2:  
 *fillByRandom*(*scanner*);  
 break;  
  
 case 3:  
 *printString*();  
 break;  
  
 case 4:  
 int searchChoice;  
 do{  
 System.*out*.println("Методы поиска подстроки: ");  
 System.*out*.println("1 Последовательный доступ ");  
 System.*out*.println("2 Метод Кнута-Морриса-Пратта ");  
 System.*out*.println("3 Поиск подстроки упрощенным методом Бойера-Мура ");  
 System.*out*.println("4 Поиск подстроки встроенной функцией ");  
 System.*out*.println("5 Сравнение времени работы методов ");  
 System.*out*.println("6 Вернуться назад ");  
 System.*out*.print("Выберите метод поиска подстроки: ");  
 searchChoice = *scanner*.nextInt();  
 *scanner*.nextLine();  
 switch (searchChoice) {  
 case 1:  
 *sequentialSearch*(*scanner*);  
 break;  
  
 case 2:  
 System.*out*.println("Введите подстроку для поиска: ");  
 String subScanner = *scanner*.next();  
 long startTimeKMP = System.*nanoTime*();  
 *KMP*(*inputString*, subScanner);  
 long endTimeKMP = System.*nanoTime*();  
 System.*out*.println("Время выполнения метода KMP: " + (endTimeKMP - startTimeKMP) + " наносекунд");  
 break;  
  
 case 3:  
 System.*out*.println("Введите подстроку для поиска: ");  
 String substring = *scanner*.next();  
 *boyerMoore*(*inputString*, substring);  
 break;  
  
 case 4:  
 *searchSubstringBuiltInFunction*(*scanner*);  
 break;  
  
 case 5:  
 System.*out*.println("Введите подстроку для поиска: ");  
 String subString = *scanner*.next();  
 *SortTimes*(*inputString*, subString);  
 break;  
 case 6:  
 System.*out*.println("Назад");  
 break;  
 default:  
 System.*out*.println("Неверный ввод");  
 break;  
 }  
 System.*out*.println();  
 }  
 while (searchChoice !=6);  
 break;  
  
  
 case 5:  
 *Exercise5*();  
 break;  
  
 case 6:  
 System.*out*.println("Программа завершена ");  
 break;  
  
 default:  
 System.*out*.println("Неверный выбор ");  
 break;  
 }  
 System.*out*.println();  
 }  
 while (ans != 6); //выход из программы  
 }  
  
 private static void fillByHand(Scanner scanner) {  
 System.*out*.print("Введите строку с клавиатуры: ");  
 *inputString* = scanner.next();  
 }  
  
  
 private static void fillByRandom(Scanner scanner) {  
 System.*out*.print("Введите длину строки для генерации случайных символов: ");  
 int length = scanner.nextInt();  
 StringBuilder sub = new StringBuilder();  
 String characters = "абвгдеёжзийклмнопростуфхцчшщыьэюя";  
 for (int i = 0; i < length; i++) {  
 int randomIndex = *random*.nextInt(characters.length());  
 char randomChar = characters.charAt(randomIndex);  
 sub.append(randomChar);  
 }  
 *inputString* = sub.toString();  
 }  
  
 private static void printString() {  
 if (*inputString* != null) {  
 System.*out*.println("Строка: " + *inputString*);  
 } else {  
 System.*out*.println("Строка не инициализирована ");  
 }  
 }  
  
  
  
 private static void sequentialSearch(Scanner scanner) {  
 System.*out*.print("Введите подстроку для поиска: ");  
 String substring = scanner.next();  
  
 long startTime = System.*nanoTime*();  
  
 int index = *inputString*.indexOf(substring);  
 while (index != -1) {  
 System.*out*.println("Подстрока найдена на позиции " + index);  
 index = *inputString*.indexOf(substring, index + 1);  
 }  
  
 long endTime = System.*nanoTime*();  
if (index==-1){  
 System.*out*.println("Подстрока не найдена на позиции " );  
}  
  
 System.*out*.println("Время выполнения: " + (endTime - startTime) + " наносекунд");  
 }  
  
  
 public static void KMP(String str, String sub) {  
 if (sub == null || sub.length() == 0) {  
 System.*out*.println("Введите подстроку: ");  
 return;  
 }  
  
 if (str == null || sub.length() > str.length()) {  
 System.*out*.println("Подстрока не инициализирована: ");  
 return;  
 }  
  
 char[] chars = sub.toCharArray();  
 int[] next = new int[sub.length() + 1];  
  
 for (int i = 1; i < sub.length(); i++) {  
 int j = next[i];  
 while (j > 0 && chars[j] != chars[i]) {  
 j = next[j];  
 }  
 if (chars[j] == chars[i]) {  
 j++;  
 }  
 next[i + 1] = j;  
 }  
  
 int j = 0;  
 for (int i = 0; i < str.length(); i++) {  
 while (j > 0 && chars[j] != str.charAt(i)) {  
 j = next[j];  
 }  
 if (chars[j] == str.charAt(i)) {  
 j++;  
 }  
 if (j == sub.length()) {  
 System.*out*.println("Подстрока найдена на позиции: " + (i - j + 1));  
 j = next[j];  
 }  
 }  
  
 if (j == 0) {  
 System.*out*.println("Подстрока не найдена.");  
 }  
 }  
  
 private static int boyerMoore(String text, String pattern) {  
 int textLength = text.length();  
 int patternLength = pattern.length();  
  
 if (patternLength == 0) {  
 return 0; // пустая подстрока всегда найдена в начале текста  
 }  
  
 int[] last = new int[Character.*MAX\_VALUE* + 1];  
  
 for (int i = 0; i <= Character.*MAX\_VALUE*; i++) {  
 last[i] = -1;  
 }  
  
 for (int i = 0; i < patternLength; i++) {  
 last[pattern.charAt(i)] = i;  
 }  
  
 int i = patternLength - 1;  
 int j = patternLength - 1;  
  
 while (i < textLength) {  
 if (text.charAt(i) == pattern.charAt(j)) {  
 if (j == 0) {  
 return i;  
 }  
 i--;  
 j--;  
 } else {  
 i = i + patternLength - Math.*min*(j, 1 + last[text.charAt(i)]);  
 j = patternLength - 1;  
 }  
 }  
 System.*out*.println("Подстрока не найдена ");  
 return -1;  
 }  
  
 private static void searchSubstringBuiltInFunction(Scanner scanner) {  
 System.*out*.print("Введите подстроку для поиска: ");  
 String substring = scanner.next();  
  
 long startTime = System.*nanoTime*();  
  
 int index = *inputString*.indexOf(substring);  
 while (index != -1) {  
 System.*out*.println("Подстрока найдена на позиции " + index);  
 index = *inputString*.indexOf(substring, index + 1);  
 }  
  
 long endTime = System.*nanoTime*();  
  
 if (index == -1) {  
 System.*out*.println("Подстрока не найдена ");  
 }  
  
 System.*out*.println("Время выполнения: " + (endTime - startTime) + " наносекунд");  
 }  
  
 private static int sequentialSearch(String substring) {  
 return *inputString*.indexOf(substring);  
 }  
 private static void SortTimes(String inputString, String subString) {  
 long startTimeSequential = System.*nanoTime*();  
 int resultSequential = *sequentialSearch*(subString);  
 long endTimeSequential = System.*nanoTime*();  
 long sequentialTime = endTimeSequential - startTimeSequential;  
  
 long startTimeKMP = System.*nanoTime*();  
 *KMP*(inputString, subString);  
 long endTimeKMP = System.*nanoTime*();  
 long kmpTime = endTimeKMP - startTimeKMP;  
  
 long startTimeBoyerMoore = System.*nanoTime*();  
 int resultBoyerMoore = *boyerMoore*(inputString, subString);  
 long endTimeBoyerMoore = System.*nanoTime*();  
 long boyerMooreTime = endTimeBoyerMoore - startTimeBoyerMoore;  
  
 long startTimeBuiltIn = System.*nanoTime*();  
 int resultBuiltIn = inputString.indexOf(subString);  
 long endTimeBuiltIn = System.*nanoTime*();  
 long builtInTime = endTimeBuiltIn - startTimeBuiltIn;  
  
 if (resultSequential!=-1) {  
 System.*out*.println("Время выполнения поиска методом последовательного доступа: " + sequentialTime + " наносекунд");  
 } else{  
 System.*out*.println("Подстрока не найдена ");  
 }  
  
 System.*out*.println("Время выполнения поиска методом КМП: " + kmpTime + " наносекунд");  
  
 if (resultBoyerMoore!=-1){  
 System.*out*.println("Время выполнения поиска Бойера-Мура: " + boyerMooreTime + " наносекунд");  
 } else{  
 System.*out*.println("Подстрока не найдена ");  
 }  
  
 System.*out*.println("Время выполнения поиска встроенной функцией: " + builtInTime + " наносекунд");  
 }  
  
  
 public static void Exercise5() {  
 System.*out*.println("Введите текст: ");  
 String txt = *scanner*.nextLine();  
  
 String[] words = txt.split("\\s+");  
  
 String longestRegularWord = "";  
 int maxLength = 0;  
  
 System.*out*.println("\nСлова, в которых имеются либо только цифры либо только латинские буквы:");  
  
 for (String word : words) {  
 // Проверка на регулярное слово (только большие латинские буквы):  
 if (word.matches("[A-Z]+")) {  
 if (word.length() > maxLength) {  
 maxLength = word.length();  
 longestRegularWord = word;  
 }  
  
 // Удаление гласных букв из регулярного слова:  
 String result = word.replaceAll("[aeiouyAEIOUY]", "");  
 System.*out*.println("Самое длинное слово после удаления гласных: " + result);  
 }  
  
 // Проверка на слова, в которых имеются либо только цифры либо только латинские буквы:  
 if (word.matches("[0-9]+") || word.matches("[a-zA-Z]+")) {  
 System.*out*.println("Слова, в которых имеются либо только цифры либо только латинские буквы: " + word);  
 }  
 }  
  
 if (!longestRegularWord.isEmpty()) {  
 System.*out*.println("Самое длинное регулярное слово: " + longestRegularWord);  
 } else {  
 System.*out*.println("Регулярные слова не найдены ");  
 }  
 }  
}

**Результат:**

Выберите пункт меню:

1) Ввод в ручную

2) Ввод рандомно

3) Вывод строки на экран

4) Поиск подстроки

5) Решение задачи №10

6) Выход из программы

Выбрать пункт меню: 2

Введите длину строки для генерации случайных символов: **100000**

Выберите пункт меню:

Выбрать пункт меню: 4

Методы поиска подстроки:

1 Последовательный доступ

2 Метод Кнута-Морриса-Пратта

3 Поиск подстроки упрощенным методом Бойера-Мура

4 Поиск подстроки встроенной функцией

5 Сравнение времени работы методов

6 Вернуться назад

Выберите метод поиска подстроки: **1**

Введите подстроку для поиска: **абв**

**Подстрока найдена на позиции 19925**

**Подстрока найдена на позиции 88659**

**Время выполнения: 9949400 наносекунд**

Методы поиска подстроки:

1 Последовательный доступ

2 Метод Кнута-Морриса-Пратта

3 Поиск подстроки упрощенным методом Бойера-Мура

4 Поиск подстроки встроенной функцией

5 Сравнение времени работы методов

6 Вернуться назад

Выберите метод поиска подстроки: **2**

Введите подстроку для поиска:

**абв**

**Подстрока найдена на позиции: 19925**

**Подстрока найдена на позиции: 88659**

**Время выполнения метода KMP: 9028800 наносекунд**

Методы поиска подстроки:

1 Последовательный доступ

2 Метод Кнута-Морриса-Пратта

3 Поиск подстроки упрощенным методом Бойера-Мура

4 Поиск подстроки встроенной функцией

5 Сравнение времени работы методов

6 Вернуться назад

Выберите метод поиска подстроки: **5**

Введите подстроку для поиска:

**абв**

**Время выполнения поиска методом последовательного доступа: 98200 наносекунд**

**Время выполнения поиска методом КМП: 3848900 наносекунд**

**Время выполнения поиска Бойера-Мура: 771200 наносекунд**

**Время выполнения поиска встроенной функцией: 108700 наносекунд**

Выберите пункт меню:

1) Ввод в ручную

2) Ввод рандомно

3) Вывод строки на экран

4) Поиск подстроки

5) Решение задачи №10

6) Выход из программы

Выбрать пункт меню: **5**

Введите текст:

**I went to SCHOOL this morning**

**Слова, в которых имеются либо только цифры либо только латинские буквы: I**

**Слова, в которых имеются либо только цифры либо только латинские буквы: went**

**Слова, в которых имеются либо только цифры либо только латинские буквы: to**

**Самое длинное слово после удаления гласных: SCHL**

**Слова, в которых имеются либо только цифры либо только латинские буквы: SCHOOL**

**Слова, в которых имеются либо только цифры либо только латинские буквы: this**

**Слова, в которых имеются либо только цифры либо только латинские буквы: morning**

**Самое длинное регулярное слово: SCHOOL**

Выберите пункт меню:

Выбрать пункт меню: 6

**Программа завершена**

**Заключение:**

Во время выполнения лабораторной работы были изучены методы работы со строками, а именно: ввод и вывод строки, удаление элементов из заданной строки, нахождение конкретных элементов строки с заданными параметрами. Также были изучены методы поиска строки, такие как: метод поиска последовательным доступом, метод КМП, метод Бойера-Мура, поиск встроенной функцией.