МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 2**

*по дисциплине «Алгоритмы и структуры данных»*

Выполнил студент

группы ПИбд-12

Нгуен Т. М.

Проверил доцент кафедры

«Информационные системы»

Кулешов А.В.

Ульяновск, 2025

### Первое задание

В задаче реализован классический жадный алгоритм:

1. Заявки сортируются по времени окончания (по возрастанию).
2. Затем выбираются заявки, не перекрывающиеся с последней выбранной.
3. import java.util.\*;
4. class Request {
5. int begin;
6. int end;
7. Request(int begin, int end) {
8. this.begin = begin;
9. this.end = end;
10. }
11. public String toString() {
12. return "(" + begin + ", " + end + ")";
13. }
14. }
15. public class task1 {
16. public static List<Request> selectRequest(List<Request> requests) {
17. Collections.sort(requests, new Comparator<Request>() {
18. public int compare(Request r1, Request r2) {
19. return Integer.compare(r1.end, r2.end);
20. }
21. });
22. List<Request> selected = new ArrayList<>();
23. if (requests.isEmpty()) {
24. return selected;
25. }
26. Request lastSelected = requests.get(0);
27. selected.add(lastSelected);
28. System.out.println("Отсортированные заявки до перебора");
29. for (int i = 1; i < requests.size(); i++) {
30. Request current = requests.get(i);
32. System.out.println(current);
34. if (current.begin >= lastSelected.end) {
35. selected.add(current);
36. lastSelected = current;
37. }
38. }
39. return selected;
40. }
41. public static void main(String[] args) {
42. // Заявки в произвольном порядке (не отсортированы)
43. List<Request> requests = Arrays.asList(
44. new Request(6, 10),
45. new Request(1, 4),
46. new Request(8, 12),
47. new Request(3, 5),
48. new Request(0, 6),
49. new Request(5, 7),
50. new Request(8, 11),
51. new Request(12, 14),
52. new Request(3, 8),
53. new Request(5, 9),
54. new Request(2, 13)
55. );
56. System.out.println("Исходные заявки:");
57. for (Request request : requests) {
58. System.out.println(request);
59. }
60. List<Request> selectedActivities = selectRequest(requests);
61. System.out.println("\nВыбранные заявки:");
62. for (Request request : selectedActivities) {
63. System.out.println(request);
64. }
65. System.out.println("Общее количество: " + selectedActivities.size());
66. }
67. }

**Корректная реализация жадного алгоритма**

Сортировка по end гарантирует оптимальность выбора.

Условие current.begin >= lastSelected.end корректно отбирает непересекающиеся заявки.

**Читаемость**

Структура кода проста и понятна: отдельный класс Request, метод selectRequest для основной логики.

Используется toString() для удобного вывода заявок в консоль.

**Использование стандартных коллекций и Comparator**

Применен Collections.sort() с Comparator, что упрощает сортировку и делает её гибкой.

**Демонстрация работы**

В main выводятся как исходные заявки, так и отсортированные и отобранные, что удобно для отладки и анализа.

**Второе задание**Задача о редакционном расстоянии  
  
import java.util.\*;

public class task2 {

    public static int calculateDistance(String s1, String s2){

        int m = s1.length();

        int n = s2.length();

        int[][] dp = new int[m+1][n+1];

        for (int i = 0; i <= m; i++){

            dp[i][0] = i;

        }

        for (int j = 0; j <= n; j++){

            dp[0][j] = j;

        }

        for (int i = 1; i <= m; i++){

            for (int j = 1; j <= n; j++){

                if (s1.charAt(i-1) == s2.charAt((j-1))) {

                    dp[i][j] = dp[i-1][j-1];

                } else{

                    dp[i][j] = 1 + Math.min(dp[i-1][j], Math.min(dp[i][j-1], dp[i-1][j-1]));

                }

            }

        }

        return dp[m][n];

    }

    public static void main(String[] args) {

        String s1 = "кот";

        String s2 = "крот";

        int distance = calculateDistance(s1, s2);

        System.out.println("Редакционное расстояние: " + distance);

    }

}

### Для удобства было показано ход работы кода через Excel ****Цель задачи****

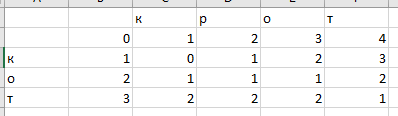
Определить минимальное количество операций, необходимых для преобразования одной строки в другую. Допустимы три типа операций:

1. Вставка символа
2. Удаление символа
3. Замена одного символа на другой

Это значение называется **редакционным расстоянием (расстоянием Левенштейна)** между двумя строками.

В коде применяется классический **динамический алгоритм**:

1. Создается двумерная таблица dp[m+1][n+1], где m и n — длины строк s1 и s2.
2. Ячейка dp[i][j] содержит минимальное количество операций для преобразования первых i символов s1 в первые j символов s2.
3. Инициализация:
   * Преобразование пустой строки в строку из j символов требует j вставок.
   * Аналогично, из i символов в пустую — i удалений.
4. Основная формула:
   * Если символы равны: dp[i][j] = dp[i-1][j-1]
   * Иначе: dp[i][j] = 1 + min(удаление, вставка, замена)

Вот пример работы для 2 слов Кот и Крот  
  
  
  
  
  
Третья задача

Даны асимптомы log n, n^2, k^n

Для первого я взял алгоритм бинарного поиска

public class O\_nlog {

    public static void quickSort(int[] arr, int low , int high){

        if (low < high) {

            int pivotIndex = partition(arr, low, high);

            quickSort(arr, low, pivotIndex -1);

            quickSort(arr, pivotIndex+1, high);

        }

    }

    private static int partition(int[] arr, int low, int high){

        int pivot = arr[high];

        int i = low - 1;

        for (int j = low; j <= high; j++){

            if (arr[j] < pivot) {

                i++;

                swap(arr, i , j);

            }

        }

        swap(arr, i+1 , high);

        return i+1;

    }

    private static void swap(int[] arr, int i, int j){

        int temp = arr[i];

        arr[i] = arr[j];

        arr[j] = temp;

    }

}

Для второго сортировка вставками

public class O\_n2 {

    public static int[] Insert(int[] array){

        for(int i = 0; i < array.length; i++){

            int element = array[i], j = i;

            while (j > 0 && element < array[j-1]){

                array[j] = array[j-1];

                j--;

            }

            array[j] = element;

        }

        return array;

    }

}

И для третьего жадный алгоритм, работающий рекурсием  
  
import java.util.\*;

public class O\_kn {

    public static List<List<Integer>> findAllSubsets(int[] elements) {

        List<List<Integer>> allSubsets = new ArrayList<>();

        generateSubsets(elements, 0, new ArrayList<>(), allSubsets);

        return allSubsets;

    }

    private static void generateSubsets(int[] elements, int index, List<Integer> currentSubset, List<List<Integer>> allSubsets) {

        if (index == elements.length) {

            allSubsets.add(new ArrayList<>(currentSubset));

            return;

        }

        generateSubsets(elements, index + 1, currentSubset, allSubsets);

        currentSubset.add(elements[index]);

        generateSubsets(elements, index + 1, currentSubset, allSubsets);

        currentSubset.remove(currentSubset.size() - 1);

    }

}

Для сравнения скорости выполнения, в главное файл где вызываются все коды алгоритмов, где при выполнения кода мы увидим разницу log n и n^2, а также как медленно работает жадный алгоритм для 20 элементах массива  
  
  
import java.util.\*;

public class Main {

    public static void main(String[] args){

        System.out.println("Задание 3 O(n^2), введите количество элементов массива и введите массивы");

        int a;

        Scanner num = new Scanner(System.in);

        a = num.nextInt();

        int[] arr = new int[a];

        for (int i = 0; i < a; i++){

            arr[i] = (int)(Math.random() \* 10000);

        }

        for (int i = 0; i < a; i++){

            System.out.printf("%d ", arr[i]);

        }

        long start = System.nanoTime();

        int[] arrNew = O\_n2.Insert(arr);

        long end = System.nanoTime();

        System.out.println();

        for (int i = 0; i < a; i++){

            System.out.printf("%d ", arrNew[i]);

        }

        System.out.println();

        System.out.printf("Время сортировки - %f ms\n", (float)(end-start) / 1000000);

        System.out.println("Задание 3 O(n\*log(n)), введите количество элементов массива и введите массивы");

        int a1;

        a1 = num.nextInt();

        int[] arr1 = new int[a1];

        for (int i = 0; i < a; i++){

            arr1[i] = (int)(Math.random() \* 10000);

        }

        System.out.printf(Arrays.toString(arr1));

        long start1 = System.nanoTime();

        O\_nlog.quickSort(arr1, 0, arr1.length-1);

        long end1 = System.nanoTime();

        System.out.println();

        System.out.printf(Arrays.toString(arr1));

        System.out.println();

        System.out.printf("Время сортировки - %f ms\n", (float)(end1-start1) / 1000000);

        int[] elements = {1, 2, 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

        long start2 = System.nanoTime();

        List<List<Integer>> subsets = O\_kn.findAllSubsets(elements);

        long end2 = System.nanoTime();

        for (List<Integer> subset : subsets) {

            System.out.println(subset);

        }

        System.out.printf("Время сортировки - %f ms\n", (float)(end2-start2) / 1000000);

    }

}

Результат  
