|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных.**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 6 |

**Название:**

Коллекции

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Д.Ю.Хотин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В.Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**Вариант 1, задание 6:** Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные – в начало этого списка.

Код

|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class Main {  public static void main(String[] args) {  List<Integer> list = new ArrayList<>();  list.add(-1);  list.add(2);  list.add(-3);  list.add(4);  list.add(5);  list.add(-6);  int i = 0;  int j = list.size() - 1;  while (i < j) {  while (i < j && list.get(i) >= 0) {  i++;  }  while (i < j && list.get(j) < 0) {  j--;  }  if (i < j) {  Collections.swap(list, i, j);  }  }  System.out.println(list);  }  } |

**Вариант 1, задание 7:** Ввести строки из файла, записать в список ArrayList. Выполнить сортировку строк, используя метод sort() из класса Collections.

Код

|  |
| --- |
| import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class Main {  public static void main(String[] args) {  List<String> lines = new ArrayList<>();  try (BufferedReader reader = new BufferedReader(new FileReader("C:\\University\\Java-BMSTU\\Laboratory6\\input.txt"))) {  String line;  while ((line = reader.readLine()) != null) {  lines.add(line);  }  } catch (IOException e) {  System.out.println("Error reading file: " + e.getMessage());  }  Collections.sort(lines);  System.out.println("Sorted lines:");  for (String line : lines) {  System.out.println(line);  }  }  } |

**Вариант 2, задание 7:** На плоскости задано N отрезков. Найти точку пересечения двух отрезков, имеющую минимальную абсциссу. Использовать класс TreeMap.

Код

|  |
| --- |
| import java.util.\*;  public class Main {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  System.out.print("Enter the number of line segments: ");  double n = input.nextInt();  Map<Double, List<LineSegment>> map = new TreeMap<>();  for (int i = 1; i <= n; i++) {  System.out.print("Enter the endpoints of line segment " + i + ": ");  double x1 = input.nextInt();  double y1 = input.nextInt();  double x2 = input.nextInt();  double y2 = input.nextInt();  LineSegment segment = new LineSegment(x1, y1, x2, y2);  if (map.containsKey(x1)) {  map.get(x1).add(segment);  } else {  List<LineSegment> list = new ArrayList<>();  list.add(segment);  map.put(x1, list);  }  if (map.containsKey(x2)) {  map.get(x2).add(segment);  } else {  List<LineSegment> list = new ArrayList<>();  list.add(segment);  map.put(x2, list);  }  }  List<LineSegment> currentSegments = new ArrayList<>();  double minX = Integer.MAX\_VALUE;  double minY = Integer.MAX\_VALUE;  for (Map.Entry<Double, List<LineSegment>> entry : map.entrySet()) {  double x = entry.getKey();  List<LineSegment> segments = entry.getValue();  for (int i = 0; i < segments.size(); i++) {  for (int j = i + 1; j < segments.size(); j++) {  LineSegment s1 = segments.get(i);  LineSegment s2 = segments.get(j);  Point intersection = s1.getIntersection(s2);  if ((intersection != null) && (intersection.x < minX)) {  minX = intersection.x;  minY = intersection.y;  }  }  }  }  if (minX == Integer.MAX\_VALUE && minY == Integer.MAX\_VALUE) {  System.out.println("There are no intersecting line segments.");  } else {  System.out.println("The point of intersection with the smallest abscissa is (" + minX + ", " + minY + ").");  }  }  }  class Point {  double x, y;  public Point(double x, double y) {  this.x = x;  this.y = y;  }  }  class LineSegment {  Point p1, p2;  public LineSegment(double x1, double y1, double x2, double y2) {  p1 = new Point(x1, y1);  p2 = new Point(x2, y2);  }  public Point getIntersection(LineSegment other) {  double x1 = p1.x;  double y1 = p1.y;  double x2 = p2.x;  double y2 = p2.y;  double x3 = other.p1.x;  double y3 = other.p1.y;  double x4 = other.p2.x;  double y4 = other.p2.y;  double a = y2 - y1;  double b = x1 - x2;  double c = y4 - y3;  double d = x3 - x4;  double det = a \* d - b \* c;  if (det == 0) {  System.out.println("det = 0");  return null;  }  double e = x1 \* y2 - y1 \* x2;  double f = x3 \* y4 - y3 \* x4;  double x = (e \* d - b \* f) / det;  double y = (a \* f - e \* c) / det;  return new Point(x, y);  }  } |

**Вариант 2, задание 8:** На клетчатом листе бумаги закрашена часть клеток.

\* Выделить все различные фигуры, которые образовались при этом.

\* Фигурой считается набор закрашенных клеток, достижимых друг из друга при движении в четырёх направлениях.

\* Две фигуры являются различными, если их нельзя совместить поворотом на угол, кратный 90 градусам,

\* и параллельным переносом. Используйте класс HashSet.

Код

|  |
| --- |
| import java.util.\*;  public class Main {  public static void main(String[] args) {  boolean[][] grid = {{true, false, true, false},  {true, true, false, false},  {false, true, false, true},  {false, false, false, true}  };  Set<Set<FigureExtractor.Cell>> figures = FigureExtractor.extractFigures(grid);  System.out.println("Found " + figures.size() + " figures:");  for (Set<FigureExtractor.Cell> figure : figures) {  System.out.print("Figure: ");  for (FigureExtractor.Cell cell : figure) {  System.out.print("(" + cell.i + "," + cell.j + ") ");  }  System.out.println();  }  }  }  public class FigureExtractor {  // Описание направлений движения  static final int[][] DIRECTIONS = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};  // Метод для поиска всех фигур на листе  public static Set<Set<Cell>> extractFigures(boolean[][] grid) {  Set<Set<Cell>> figures = new HashSet<>(); // Множество для хранения фигур  boolean[][] visited = new boolean[grid.length][grid[0].length]; // Матрица для отметки посещенных клеток  for (int i = 0; i < grid.length; i++) {  for (int j = 0; j < grid[0].length; j++) {  if (grid[i][j] && !visited[i][j]) { // Если клетка закрашена и не посещена ранее  Set<Cell> figure = new HashSet<>(); // Множество для хранения клеток фигуры  dfs(grid, visited, figure, i, j); // Обход в глубину для поиска клеток фигуры  figures.add(figure); // Добавление фигуры в множество  }  }  }  return figures;  }  // Метод для обхода в глубину  private static void dfs(boolean[][] grid, boolean[][] visited, Set<Cell> figure, int i, int j) {  visited[i][j] = true; // Отметка клетки как посещенной  figure.add(new Cell(i, j)); // Добавление клетки в множество фигуры  for (int[] direction : DIRECTIONS) { // Поиск соседних клеток в четырех направлениях  int ni = i + direction[0];  int nj = j + direction[1];  if (ni >= 0 && ni < grid.length && nj >= 0 && nj < grid[0].length // Проверка выхода за границы листа  && grid[ni][nj] && !visited[ni][nj]) { // Проверка наличия закрашенной и не посещенной клетки  dfs(grid, visited, figure, ni, nj); // Рекурсивный вызов для обработки соседней клетки  }  }  }  // Вспомогательный класс для хранения координат клетки  static class Cell {  int i;  int j;  Cell(int i, int j) {  this.i = i;  this.j = j;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (!(o instanceof Cell)) return false;  Cell cell = (Cell) o;  return i == cell.i && j == cell.j;  }  @Override  public int hashCode() {  return Objects.hash(i, j);  }  }  } |