Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №3
«Полиморфизм на основе интерфейсов в языке Java»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Приобретение навыков реализации интерфейсов для обеспечения возможности полиморфной обработки объектов класса.

Задачи

55: Класс булевских матриц размера m*n с порядком на основе суммарного количества строк и столбцов, все элементы которых равны между собой.

Решение

Для поиска суммарного количества строк и столбцов, все элементы которых равны между собой, реализовано префиксное дерево (бор).

BoolMatrix.java

```
import java.util.Random;

public class BoolMatrix implements Comparable<BoolMatrix> {
    private final int n;
    private final int m;
    private int countEqual;
    private byte[][] matrix;

    private byte[][] generateRandomMatrix() {
        byte[][] res = new byte[n][m];
        Random random = new Random();
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                res[i][j] = (random.nextBoolean() ? (byte)1
 ↪  : (byte)0);
            }
        }
        return res;
    }
```

```java
public BoolMatrix(int n, int m) {
    this.n = n;
    this.m = m;
    this.matrix = generateRandomMatrix();
    this.countEqual = countEqualRows() +
     ↪  countEqualColumns();
}

@Override
public String toString() {
    String res = "Одинаковых строк/столбцов: " +
     ↪  countEqual + "\n";
    for (byte[] row : this.matrix) {
        for (byte el : row) {
            res += el + " ";
        }
        res += "\n";
    }
    return res;
}

@Override
public int compareTo(BoolMatrix obj) {
    return this.countEqual - obj.countEqual;
}

private int countEqualRows() {
    int ans = 0;
    BoolArrayMultiset set = new BoolArrayMultiset();
    for (byte[] row : this.matrix) {
        int count = set.insert(row);
        if (count == 2) {
            ans += 2;
        } else if (count > 2) {
            ans++;
        }
    }
    return ans;
}
```

```
    private int countEqualColumns() {
        int ans = 0;
        BoolArrayMultiset set = new BoolArrayMultiset();
        for (int j = 0; j < m; ++j) {
            byte[] column = new byte[n];
            for (int i = 0; i < n; ++i) {
                column[i] = this.matrix[i][j];
            }
            int count = set.insert(column);
            if (count == 2) {
                ans += 2;
            } else if (count > 2) {
                ans++;
            }
        }
        return ans;
    }
};

class BoolArrayMultiset {
    private class Node {
        private Node[] next;
        private int count;

        Node() {
            this.count = 0;
            this.next = new Node[] {null, null};
        }
    };

    private Node root;

    public BoolArrayMultiset() {
        this.root = new Node();
    }

    public int insert(byte[] array) {
        Node node = this.root;
        for (byte el : array) {
```

```
            if (node.next[el] == null) {
                node.next[el] = new Node();
            }
            node = node.next[el];
        }
        node.count++;
        return node.count;
    }
};
```

Test.java

```
import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        int n = 4;
        BoolMatrix[] matrices = new BoolMatrix[n];
        for (int i = 0; i < n; ++i) {
            matrices[i] = new BoolMatrix(3, 4);
        }

        System.out.println("До сортировки:");
        for (BoolMatrix matrix : matrices) {
            System.out.println(matrix);
        }

        Arrays.sort(matrices);

        System.out.println("Отсортированные:");
        for (BoolMatrix matrix : matrices) {
            System.out.println(matrix);
        }
    }
};
```

Пример вывода

```
До сортировки:
Одинаковых строк/столбцов: 2
```

```
1 1 1 1
1 1 0 1
1 0 0 0

Одинаковых строк/столбцов: 2
1 1 1 1
1 0 1 0
0 0 0 1

Одинаковых строк/столбцов: 4
0 1 1 0
0 1 1 0
1 1 1 0

Одинаковых строк/столбцов: 0
0 0 0 1
0 1 1 0
1 0 1 1

Отсортированные:
Одинаковых строк/столбцов: 0
0 0 0 1
0 1 1 0
1 0 1 1

Одинаковых строк/столбцов: 2
1 1 1 1
1 1 0 1
1 0 0 0

Одинаковых строк/столбцов: 2
1 1 1 1
1 0 1 0
0 0 0 1

Одинаковых строк/столбцов: 4
0 1 1 0
0 1 1 0
1 1 1 0
```