

实验二打印

▼ DP多段图最短路径问题

Java |

```
1  import java.util.Scanner;
2
3  public class test5 {
4
5      private static final int INF = Integer.MAX_VALUE;
6
7      public static void main(String[] args) {
8          Scanner scanner = new Scanner(System.in);
9
10         int[][] node = createGraph(scanner);
11         int n = node.length;
12
13         int[] cost = new int[n];
14         int[] path = new int[n];
15
16         computeShortestPath(node, cost, path);
17
18         printResults(n, cost, path);
19
20         scanner.close();
21     }
22
23     private static int[][] createGraph(Scanner scanner) {
24         System.out.println("请输入顶点和边的个数: ");
25         int point = scanner.nextInt();
26         int edge = scanner.nextInt();
27
28         int[][] node = new int[point][point];
29
30         // 初始化边的权值
31         for (int i = 0; i < point; i++) {
32             for (int j = 0; j < point; j++) {
33                 node[i][j] = INF;
34             }
35         }
36
37         // 输入每条边的信息
38         for (int k = 0; k < edge; k++) {
```

The screenshot shows an IDE window with the file `test23.java` open. The '运行' (Run) tab is active, showing the execution of a Java program. The program prompts for the number of vertices and edges, and then for 8 edges with their respective vertex IDs and weights. The inputs are shown in green text.

```
"C:\Program Files\Java\jdk1.8.0_2
请输入顶点和边的个数:
7 12
请输入第 1 条边的两个顶点和权值:
0 1 4
请输入第 2 条边的两个顶点和权值:
0 3 8
请输入第 3 条边的两个顶点和权值:
0 2 5
请输入第 4 条边的两个顶点和权值:
1 3 6
请输入第 5 条边的两个顶点和权值:
2 3 5
请输入第 6 条边的两个顶点和权值:
1 4 6
请输入第 7 条边的两个顶点和权值:
3 4 8
请输入第 8 条边的两个顶点和权值:
2 5 7
```

请输入第 8 条边的两个顶点和权值：

2 5 7

请输入第 9 条边的两个顶点和权值：

3 6 9

请输入第 10 条边的两个顶点和权值：

4 6 5

请输入第 11 条边的两个顶点和权值：

5 6 4

请输入第 12 条边的两个顶点和权值：

3 5 9

最短路径长度：

从起点到顶点 0 的最短路径长度为：0

从起点到顶点 1 的最短路径长度为：4

从起点到顶点 2 的最短路径长度为：5

从起点到顶点 3 的最短路径长度为：8

从起点到顶点 4 的最短路径长度为：10

从起点到顶点 5 的最短路径长度为：12

从起点到顶点 6 的最短路径长度为：15

```
3 5 9
最短路径长度:
从起点到顶点 0 的最短路径长度为: 0
从起点到顶点 1 的最短路径长度为: 4
从起点到顶点 2 的最短路径长度为: 5
从起点到顶点 3 的最短路径长度为: 8
从起点到顶点 4 的最短路径长度为: 10
从起点到顶点 5 的最短路径长度为: 12
从起点到顶点 6 的最短路径长度为: 15
最短路径:
从起点到顶点 0 的路径为: 0
从起点到顶点 1 的路径为: 0 1
从起点到顶点 2 的路径为: 0 2
从起点到顶点 3 的路径为: 0 3
从起点到顶点 4 的路径为: 0 1 4
从起点到顶点 5 的路径为: 0 2 5
从起点到顶点 6 的路径为: 0 1 4 6

进程已结束, 退出代码为 0
```

```

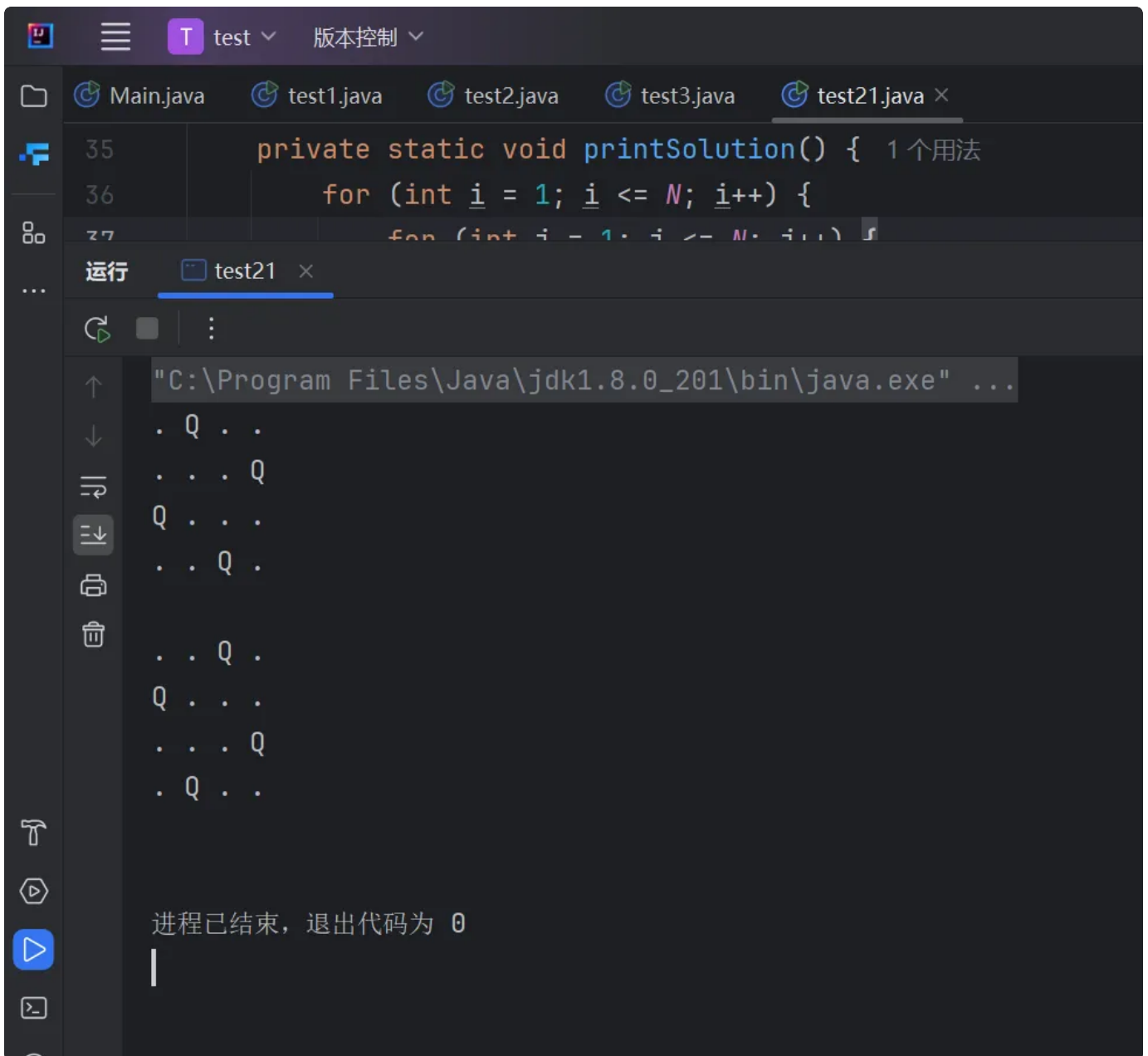
1 public class QueenSolver {
2
3     private static final int BOARD_SIZE = 4; // 定义棋盘大小
4     private static int[] positions = new int[BOARD_SIZE + 1]; // 记录每行皇
        后的位置，索引从1开始
5
6     public static void main(String[] args) {
7         findQueenPositions(1); // 从第1行开始尝试放置皇后
8     }
9
10    /**
11     * 检查在指定行和列位置放置皇后是否安全。
12     *
13     * @param currentRow 当前行号
14     * @param column 列号
15     * @return 如果位置安全返回true，否则返回false
16     */
17    private static boolean checkSafety(int currentRow, int column) {
18        for (int prevRow = 1; prevRow < currentRow; prevRow++) {
19            // 检查列冲突和对角线冲突
20            if (positions[prevRow] == column || Math.abs(positions[prevRow
        ] - column) == Math.abs(prevRow - currentRow)) {
21                return false;
22            }
23        }
24        return true;
25    }
26
27    /**
28     * 使用回溯法解决N皇后问题。
29     *
30     * @param currentRow 当前正在处理的行号
31     */
32    private static void findQueenPositions(int currentRow) {
33        for (int column = 1; column <= BOARD_SIZE; column++) {
34            if (checkSafety(currentRow, column)) { // 如果当前位置安全
35                positions[currentRow] = column; // 在当前行放置皇后
36                if (currentRow == BOARD_SIZE) { // 如果所有皇后都已经放置
37                    displaySolution(); // 输出当前解
38                } else {
39                    findQueenPositions(currentRow + 1); // 尝试放置下一行的皇
        后
40                }
41            }
42        }
    }
}

```

```

43     }
44
45     /**
46     * 输出当前找到的皇后布局方案。
47     */
48     private static void displaySolution() {
49         for (int row = 1; row <= BOARD_SIZE; row++) {
50             for (int column = 1; column <= BOARD_SIZE; column++) {
51                 // 如果当前位置有皇后，则输出'Q'，否则输出'.'
52                 if (positions[row] == column) {
53                     System.out.print("Q ");
54                 } else {
55                     System.out.print(". ");
56                 }
57             }
58             System.out.println(); // 换行，准备输出下一行
59         }
60         System.out.println(); // 输出完一个解后，空一行以便区分不同的解
61     }
62 }

```



```
1  import java.util.Scanner;
2
3  public class test23 {
4
5      private static final int MAXN = 1005;
6      private static int[][] G = new int[MAXN][MAXN]; // 用于存储图中的边
7      private static int[] color = new int[MAXN]; // 用于存储每个节点的颜色
8      private static int n, m; // n表示图中节点的数量, m
表示可供选择的颜色数目
9
10     public static boolean ok(int u, int c) {
11         for (int i = 1; i <= n; i++) {
12             if (G[u][i] == 1 && color[i] == c) {
13                 return false;
14             }
15         }
16         return true;
17     }
18
19     public static boolean dfs(int u) {
20         if (u > n) {
21             return true;
22         }
23         for (int i = 1; i <= m; i++) {
24             if (ok(u, i)) {
25                 color[u] = i;
26                 if (dfs(u + 1)) {
27                     return true;
28                 }
29                 color[u] = 0; // 回溯
30             }
31         }
32         return false;
33     }
34
35     public static void main(String[] args) {
36         Scanner scanner = new Scanner(System.in);
37
38         System.out.println("请输入顶点数和可用颜色数: ");
39         n = scanner.nextInt();
40         m = scanner.nextInt();
41
42         System.out.println("请输入边数: ");
43         int e = scanner.nextInt();
44     }
```



```

45      System.out.println("请输入相连接的顶点: "); // 顶点从1开始
46      for (int i = 0; i < e; i++) {
47          int u = scanner.nextInt();
48          int v = scanner.nextInt();
49          G[u][v] = G[v][u] = 1;
50      }
51
52      if (dfs(1)) {
53          for (int i = 1; i <= n; i++) {
54              System.out.println("结点 " + i + " 的颜色为: " + color[i]);
55          }
56      } else {
57          System.out.println("No solution");
58      }
59
60      scanner.close();
61  }
62  }

```

The screenshot shows an IDE with a dark theme. At the top, there's a toolbar with icons for file operations, a search icon, and a 'test' button. Below the toolbar, a tab bar shows several open files: test1.java, test2.java, test3.java, test21.java, and test23.java. The 'test23' tab is active. Below the tab bar, there's a '运行' (Run) button and a status bar showing 'test21' and 'test23'. The main editor area displays the output of a Java program. The program prompts for the number of vertices and available colors, the number of edges, and the connections between vertices. It then outputs the color assigned to each vertex. The output is as follows:

```
"C:\Program Files\Java\jdk1.8.0_201\bin\ja
请输入顶点数和可用颜色数:
5 3
请输入边数:
7
请输入相连接的顶点:
1 2
1 3
2 3
2 4
2 5
4 5
3 5
结点 1 的颜色为: 1
结点 2 的颜色为: 2
结点 3 的颜色为: 3
结点 4 的颜色为: 3
结点 5 的颜色为: 1

进程已结束, 退出代码为 0
```

The status bar at the bottom shows the current file is 'test' in the 'src' directory, and the file 'test23' is open. The time is 3:22, and the encoding is CRLF.