数据结构与算法
作业报告

第四次

姓名

班级

学号

电话

Email

日期

目录

任务 1

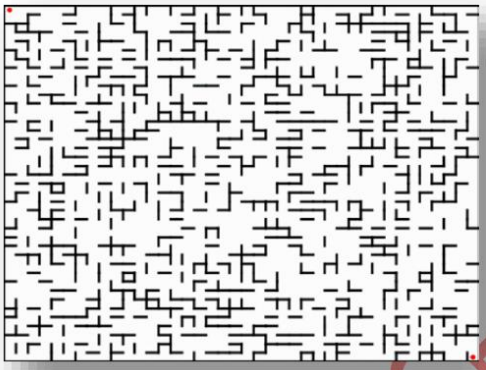
该任务的主要内容是完成用图数据结构对迷宫的表示实现，成功表示之后，可以使用如下两个策略生成迷宫：

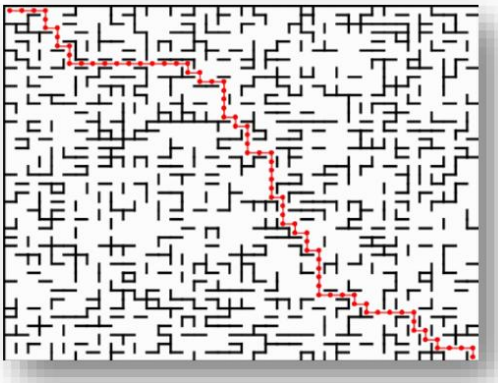
策略 1：随机擦除 70%的图中的边；

策略 2：随机擦除 50%的图中的边；

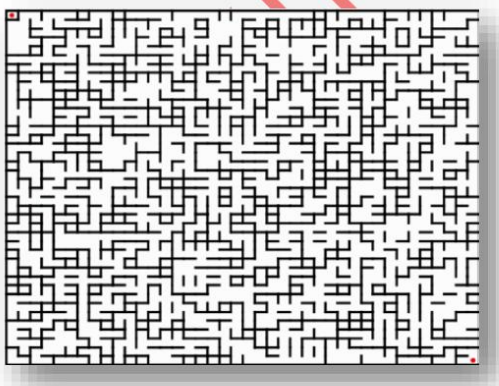
针对不同的策略，使用 Dijkstra 算法检测所生成的迷宫从入口（左上角）到出口（右下角）是否有路径可达。对每一个策略可以执行 100 次，给出成功生成迷宫的概率（成功即代表从入口到出口一定有路径可达）。如下图所示，分别是两种策略下迷宫生成以及检测是否有路径的一个实例示例：

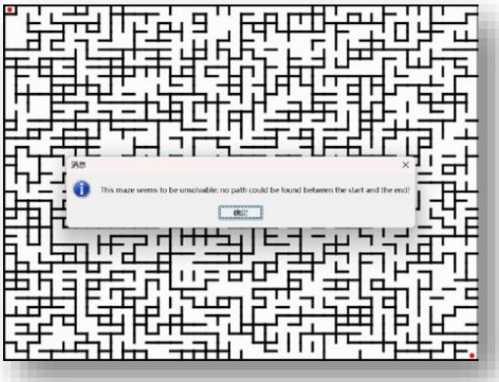
策略 1（迷宫的大小为 40*40）





策略 2（迷宫的大小为 40*40）





设计：

一、编写 MazeGenerator 类：

1.因为涉及用图表示迷宫，故应继承 JPanel 类，通过创建窗口将生成的迷宫表示出来；

2.创建 int 类型变量 CELL_SIZE，用于记录单元格的大小，其初始值可根据需求改变，代码展示部分使用的大小是 20；

3.创建 Color 类对象 WALL_COLOR，用于记录边的颜色，代码展示部分使用的颜色是 Color.BLACK；

4.创建 int 类型二维数组 maze，记录所需创建的迷宫的单元格；

5.创建 int 类型变量 n，其值决定了迷宫的大小；

6.编写有参构造器，接收参数 int[ ][ ] maze，给对象的属性 maze 和 n 赋值，并调用 setPreferredSize（new Dimension（n * CELL_SIZE, n * CELL_SIZE）），用于创建窗口显示迷宫；

7.重写 paintComponent (Graphics g) 方法，用于绘制迷宫：

（1）调用 super.paintComponent（g）方法，保留父类的默认绘制行为；

（2）调用 for 循环遍历二维数组 maze 中的每一个元素，通过对 Graphics 类对象 g 调用 setColor（WALL_COLOR）方法和 drawLine（）方法给每一个点周围的所有边上色。

8.编写 isConnected（int[ ][ ] maze）方法，使用 Dijkstra 算法检查迷宫中的起点和终点是否连通：

（1）创建 int 类型对象 n 用于记录迷宫每行（或每列）的元素个数，其值为 maze.length；

（2）创建 int 类型对象 start 用于记录迷宫起点的索引，其值为 0；

（3）创建 int 类型对象 end 记录迷宫终点的索引，其值为 n*n-1；

（4）创建 boolean 类数组 visited，数组大小为 n*n，用于检测单元格是否被遍历到；

（5）创建 int 类型数组 dist，数组大小为 n*n；

（6）对 Arrays 调用 fill（dist，Integer.MAX_VALUE）方法，将数组的所有元素初始化为整数类型的最大值；

（7）将 dist[start]初始化为 0；

（8）创建优先队列对象 pq，并对其调用 offer（start）方法，将起点添加到待处理的顶点集合中，并在后续的算法执行中对其进行处理；

（9）调用 while 循环，当 pq 不为空时进行循环：

①设 int 类型临时变量 u，其值为 pq.poll（），用于取出当前距离起点最短的顶点，作为下一步算法要处理的顶点；

②判断 u 是否等于 end，若是则返回 true，此时找到路径，说明起点和终点是连通的；

③把遍历到的单元格的状态改为 true（visited[u] = true）；

④创建 int 类型临时变量 row 和 col，分别记录检索到的迷宫矩阵的单元格的行数和列数；

⑤依次遍历当前点的上、下、左、右相邻的点，更新合适的顶点到起点的最短距离，并将该顶点加入到待处理的顶点集合中；

⑥若上述遍历相邻的点均失败，则判断起点和终点不连通，返回 false。

9.编写 generateMaze（int n，double noEdge）方法用于生成迷宫：

（1）创建 int 类型二维数组 maze，用于记录迷宫的每一个单元格，数组的行数和列数都是 n；

（2）调用两层 for 循环，初始化 maze 的每一个单元格，给每个单元格的四周都创建边；

（3）创建 Random 类对象 rand，用于随机擦除边的操作；

（4）调用两层 for 循环，依次遍历 maze 的每一个单元格，并分别对当前单元格的下边和右边进行是否需要擦除的判定（不对左边和上边进行判定是因为会造成每个边被多次判定是否擦除，导致设定的擦除概率与实际的不相符合）。

（5）上述 for 循环结束后，擦除操作均已完成，此时返回 maze；

10. 编写 showMaze（）方法，用于将实验所需的两个策略的迷宫通过窗口可视化：

（1）设迷宫的大小为 n=40；

（2）策略 1 的相关代码书写：

①设 double 类型变量 noEdge2，其为移除边占所有边的百分比，赋值为 0.7，表示要擦除 70%的边；

②调用 generateMaze（n，noEdge2）方法生成迷宫，并用 int 类型二维数组 maze2 接收；

③创建窗口并显示迷宫，具体操作如下：

```
// 创建窗口并显示迷宫
JFrame frame2 = new JFrame( title: "策略1");
MazeGenerator mazePanel2 = new MazeGenerator(maze2);
frame2.add(mazePanel2);
frame2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame2.pack();
frame2.setLocationRelativeTo(null); // 将窗口居中显示
frame2.setVisible(true);
```

（3）策略 2 的相关代码书写与策略 1 大致相当，只需将 noEdge2 改为 noEdge1，给其赋值为 0.5，生成的迷宫用 maze1 接收即可，因此最后运行代码会出现两个窗口，分别显示策略 1 和策略 2 的迷宫。

11. 编写 TestMaze（）方法，对上述两种策略生成的迷宫分别进行 100 次测试，记录成功生成迷宫的次数：

（1）设迷宫大小为 n = 40；

（2）将两个策略的擦除概率打包为 double 类型数组 noEdges（={50,70}）；

（3）设测试次数 numTests = 100;

（4）调用 for 循环，分别对策略 1 和策略 2 使用的擦除概率进行测试：

①设成功生成迷宫的个数为 connectedCount，初始值为 0;

②调用 for 循环，循环 numTests 次，每次通过调用方法 generateMaze(n, noEdge/100)生成迷宫并赋值给 maze，再调用 isConnected(maze)判断迷宫是否连通，若是则 connectedCount 的值加一;

（5）将测试结果以一定的格式输出到控制台。

12. 编写主函数：

（1）调用 showMaze()方法创建两个策略生成迷宫的窗口;

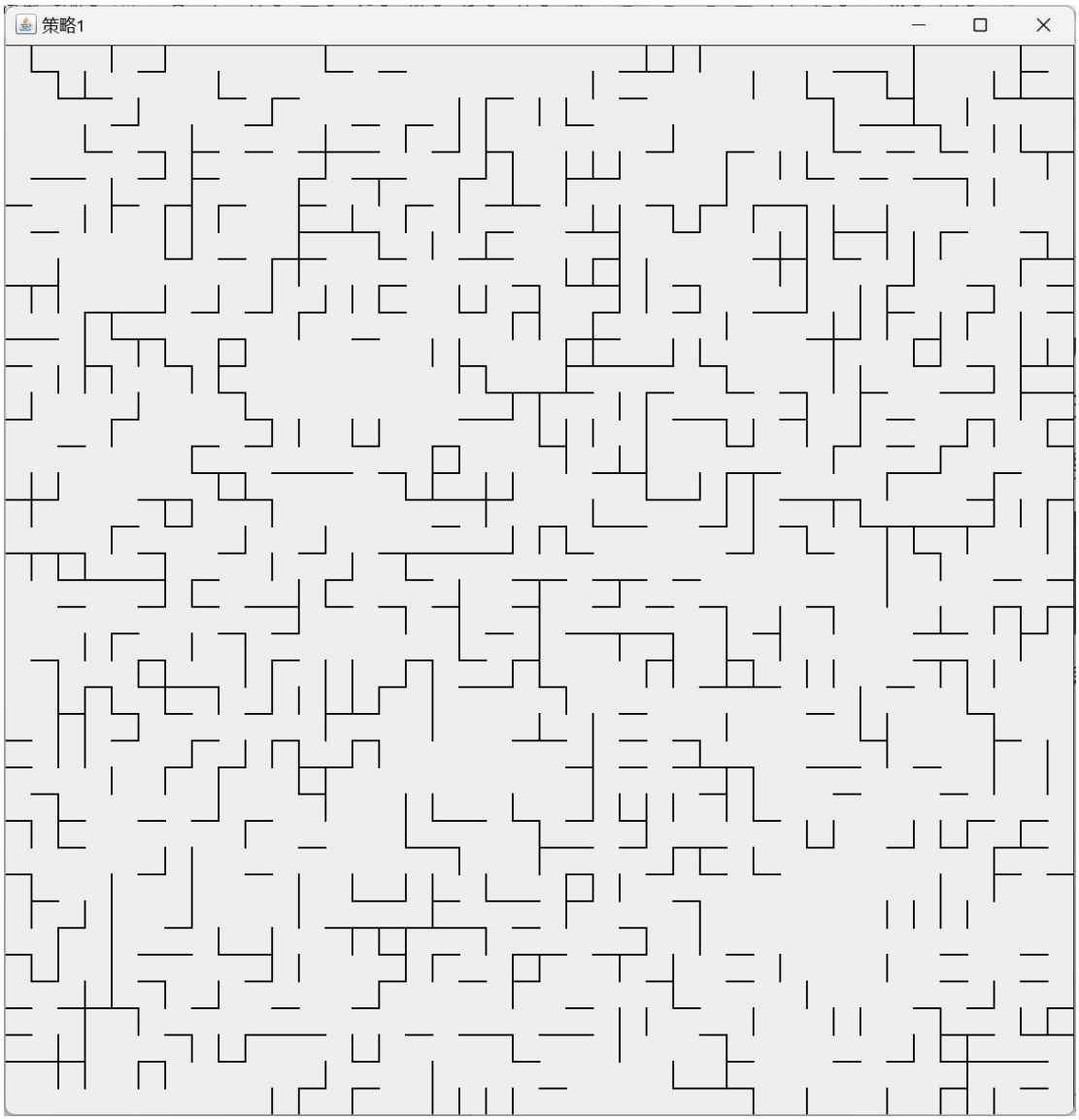
（2）调用 TestMaze（）方法输出两个策略成功生成迷宫的概率。

运行结果展示：

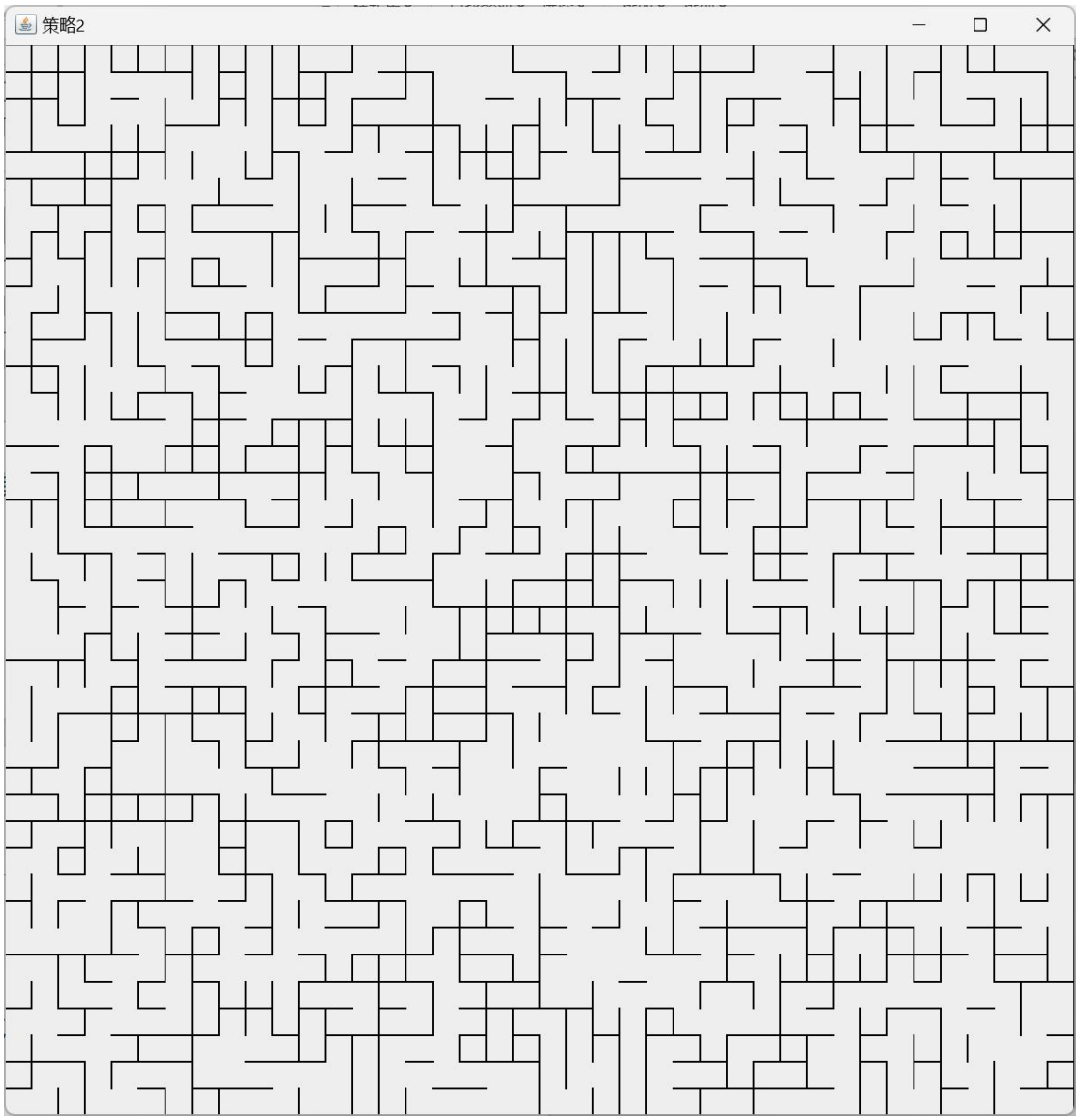
经过多次测试，擦除 70%的边后成功生成迷宫的概率大致处于 50%~80%之间，而擦除 50%的边后成功生成迷宫的概率一般小于 5%，下面是一次测试的结果：

```
随机擦除50%的边时，成功生成迷宫的概率为：3
随机擦除70%的边时，成功生成迷宫的概率为：62
```

下面是策略 1（擦除 70%边）的某一次结果：



下面是策略 2（擦除 50%边）的某一次结果:



总结与收获：

一．总结

1. 通过构建图数据结构相关的代码，实现了对一定大小的迷宫的创建；

2. 经过多次实验，发现擦除的边越多，迷宫的起点和终点相连接的概率越高。

3. 通过本次实验，成功实现了根据不同的擦除概率生成迷宫，并使用 Dijkstra 算法检查起点和终点的连通性。

二．收获

1. 回忆并加深了图数据结构相关的知识体系及代码书写；

2. 掌握了迷宫生成算法的原理和实现方法；

3. 巩固加深了 Dijkstra 算法相关的知识体系；

4. 灵活运用优先队列、数组等相关辅助知识，利于我们多方面思考题目并成功解题；

5. 提高了对 Java Swing 图形界面编程的理解和应用能力。

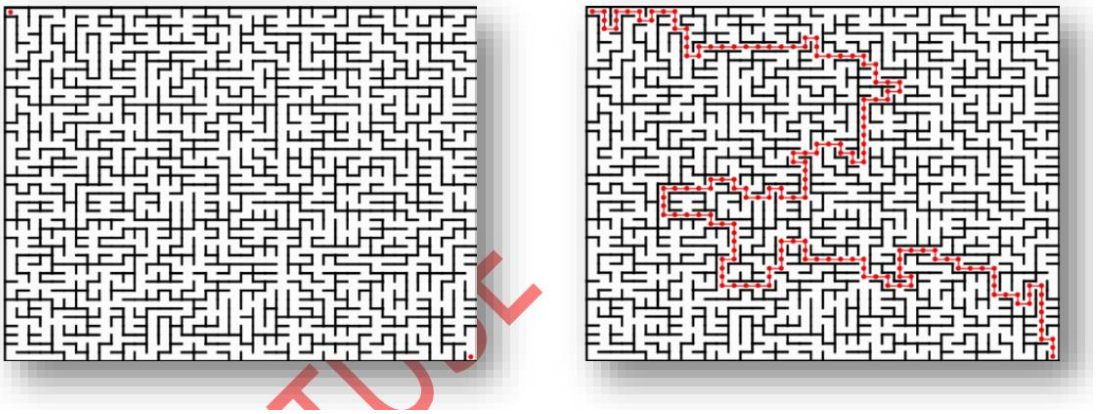
任务 2

为了确保每次迷宫的生成都是成功的，将不再采用任务 1 中的随机擦除边的生成方式，而是采用 Kruskal 最小支撑树的算法来实现迷宫生成。具体的执行步骤如下：

1. 为任务 1 中用来表示迷宫的图中的每一条边都随机生成一个权值（此时的图是一个带权图，所以在表示时可能会和任务 1 的表示有出入，请注意这个细节）；
2. 利用 Kruskal 算法对步骤 1 中的图生成最小支撑树 T；
3. 将 T 中的每一条边相对应在迷宫中的边擦除掉，此时迷宫就生成了，这样生成的迷宫一定是成功的。

如下图所示就是用此策略生成的迷宫的一个实例的示例。

Kruskal 策略（迷宫的大小为 40*40）



设计：

一、编写 Edge 类，给单元格之间的边进行定义：

1.实现 Comparable<Edge>接口；

2.创建 int 类型成员变量 from、to，作为边的两个顶点；

3.创建 double 类型成员变量 weight，作为边的权重；

4.编写有参构造器，接收参数(int from, int to, double weight)，给对应的成员变量赋值；

5.编写 compareTo（Edge other）方法：返回 Double.compare(this.weight, other.weight)。

二、编写 UnionFind 类并查集类：

1.创建 int[ ]类型成员对象 parent，用于存储每个节点的父节点信息，用于表示节点之间的关系；

2.编写有参构造器，接收参数(int n)，n 表示结点的数量，给 parent 的长度设为 n，调用 for 循环初始化父节点数组；

3.编写 find（int x）方法，用于查找节点 x 的根节点，即该节点所在集合的代表节点：调用 if 语句判断 parent[x]不等于 x 时，令 parent[x]=find（parent[x]），递归查找根节点，最后将根节点返回；

4.编写 union（int x，int y）方法，用于合并两个节点所在的集合，将它们的根节点设为相同的值：令 parent[find（x）]=find（y）。

三、编写 MazeGenerator 类：

1.对于成员 CELL_SIZE、WALL_COLOR、maze、n 的定义均与题 1MazeGenerator 类的定义相同；

2.有参构造器的编写照搬题 1MazeGenerator 类的有参构造器；

3.对于重写的 paintComponent 方法，直接采用题 1 重写的该方法即可；

4.编写 generateMaze（int n）方法，用于生成使用 Kruskal 算法计算所得的迷宫：

（1）创建迷宫数组 maze，大小为 n 行 n 列；

（2）调用两层 for 循环，变量 maze 的每一个单元格，为单元格四周加上边；

（3）创建带权图并随机分配权值：

①创建 List<Edge>类对象 edges 和 Random 类对象 rand；

②调用两层 for 循环，遍历 maze 的每一个单元格，并调用两次 if 语句判断该单元格的右边和下边是否为边界，如果不是则给边附上随机的权重值。

（4）使用 Kruskal 算法找到最小生成树：

①调用 Collections.sort(edges)方法，给边按权值进行排序；

②创建并初始化并查集 uf，节点数量为 n*n，表示每个单元格作为一个节点；

③调用 for 循环遍历 edges 中的每一条边：

·调用 if 语句判断边的起点和终点是否在同一个连通分量中，若非则对并查集 uf 调用 union（edge.from，edge.to）方法将边的起点和终点合并到同一个连通分量中；

·创建四个临时变量 row1、col1、row2、col2，分别记录当前遍历到的边的起点的行数和列数及当前边的终点的行数和列数；

·调用 if 语句判断边的起点和终点在迷宫中的相对位置，然后移除相应的边。

（5）移除完所有边后返回 maze。

5.编写主函数：

（1）设迷宫的行数和列数的大小为 n=40；

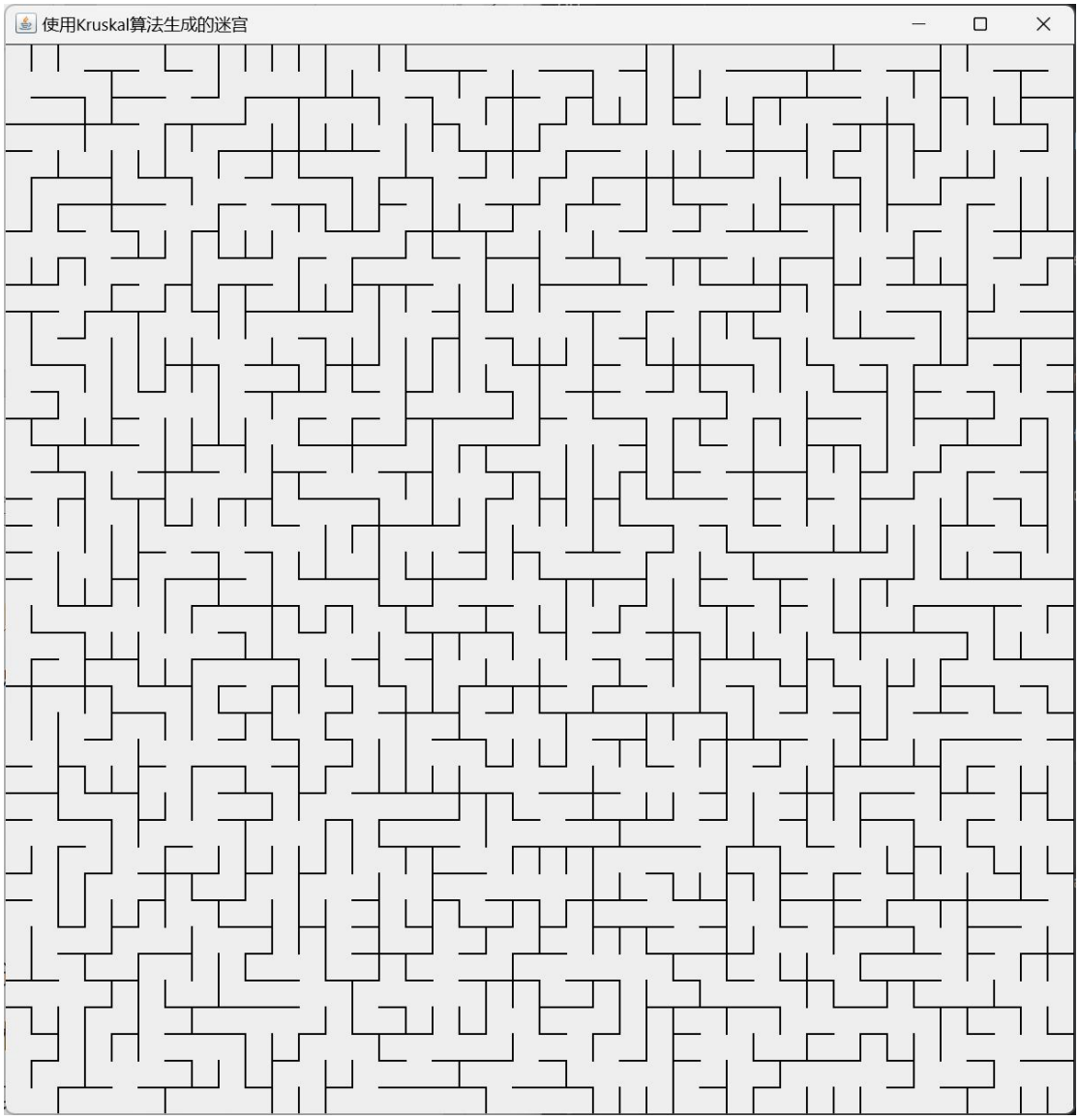
（2）调用 generateMaze（n）生成 40*40 大小的迷宫，并用临时变量 maze 接收；

（3）编写代码将迷宫通过窗口输出，具体操作如下：

```
// 创建窗口并显示迷宫
JFrame frame = new JFrame( title: "使用Kruskal算法生成的迷宫");
MazeGenerator mazePanel = new MazeGenerator(maze);
frame.add(mazePanel);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setLocationRelativeTo(null); // 将窗口居中显示
frame.setVisible(true);
```

运行结果展示：

下列为给边赋予随机权重，通过 Kruskal 算法找到最小生成树，擦除最小生成树上的边后生成的起点和终点一定连通的迷宫：



总结与收获:

一．总结

通过对该部分任务的探究与思考，再一次巩固加深了图数据结构相关的理论知识的印象，且回忆并使用了 Kruskal 算法的相关内容，对于窗口的创建也是更加熟练，美中不足的是进行了多次对最小生成树路径的显示的代码编写，但最后还是无法完善该部分代码，导致最终的运行结果虽然一定能生成合理的迷宫，但起点和终点的连通并不能直观的表现出来。