**山东大学计算机科学与技术学院  
《数据结构与算法》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201700301147 | 姓名：杜瀛川 | | 班级： 17.4 |
| 上机学时：4 | | 日期： 2019.3.27 | |
| 课程设计题目：残缺棋盘 | | | |
| 软件开发环境：Qt Creater | | | |
| 项目最终效果图      报告内容：  1.需求描述  1.1 问题描述  残缺棋盘(defective chessboard)：是一个有2k×2k个方格的棋盘，其中恰有一个方格残缺。对于任意k，恰好存在22k种不同的残缺棋盘。在残缺棋盘中，要求用三格板(triominoes)覆盖 残缺棋盘。在覆盖中，任意两个三格板不能重叠，任意一个三格板不能覆盖残缺方格，但三格板必须覆盖其他所有方格.  1.2 基本要求  输入棋盘大小和残缺方格的位置，输出覆盖后的棋盘，输出棋盘时要着色，共享同一边界的覆盖应着不同的颜色。棋盘是平面图，因此最多只需4种颜色，为覆盖着色，要求设计贪婪着色启发式方法，以尽量使用较少的颜色。  1.3 输入说明  1.3.1 初始化  点击左上角菜单栏中的初始化按钮，会弹出输入对话框，用户在对话框中输入k（棋盘大小位2的k次方），row(残缺所在行号），col(残缺所在列号)。点击初始化按钮，如图1所示。      图1  1.3.2 重置  用户点击左上角菜单栏中的重置按钮，应用程序自动完成重启工作。用户可再次输入k row col等值，实现再次初始化。  1.4 输出说明  1.4.1 颜色数量  本程序输出所用颜色数量的最小值，由平面4色定理可知，平面图中最多的填充涂色为4种。输出对话框如图2所示。    图2  1.4.2 残缺棋盘图形化  本程序在中心组件qgraficview中输出一个图形化的残缺棋盘填色方案，所用颜色数量为最小值。如图3所示。      图3  2.设计  2.1 系统结构设计      2.2 设计思路  残缺棋盘问题解决使用分治算法思想。残缺棋盘是一个有2k∗2k的方格棋盘，其中恰有一个方格残缺，k = 0,1,2时各种可能的残缺棋盘，其中残缺棋盘用阴影表示。对于任意k,恰好存在2的2k次方种不同的残缺棋盘。      要求用三格板覆盖残缺棋盘，在此覆盖中，两个三格板不能重叠，三格板不能覆盖残缺方格，但必须覆盖其他所有方格。首先进行棋盘划分。      覆盖2k∗2k残缺棋盘的问题转化为4个2k-1∗2k-1覆盖棋盘问题。递归得使用这种分割技术。当棋盘的大小减为1 \* 1时，递归过程终止。此时的1 \* 1棋盘2包含一个方格且为残缺方格，无需覆盖三格板。  2.3 数据及数据类(型)定义  首先定义棋盘类，其成员变量和成员函数如图所示。其中title变量为当前使用的三格板编号，size为残缺棋盘的边长，defectRow defectColumn分别是残缺位置的行号和列号。board为残缺棋盘储存数组，connectMap为各三格板的邻接矩阵，color colorLess颜色序号数组，colorMax为当前所用颜色的数量。    2.4.算法设计及分析  2.4.1 chessboard类构造函数，通过用户输入的3个量，初始化成员变量，并对棋盘存储矩阵board进行空间申请和初始化。    2.4.2 棋盘处理分治递归函数  首先判断棋盘size = 1，是则递归结束，否则size 减半，分为4个小棋盘，其中，以残缺所在位置分为4种情况，分别是左上、左下、右上、右下，将3个非残缺棋盘的小棋盘转化为残缺棋盘。为此，将一个三格板放置在由这3个小棋盘形成的角上。再次进行递归。      遍历残缺棋盘（board），获得三格板临接矩阵（connectMap），为涂色做准备。    设计基于深度优先搜索的涂色算法，并进行减枝优化，一旦使用的颜色比当前最优多，立即放弃搜索进行回溯。根据邻接矩阵对三格板进行涂色，确保相邻三格板具有不想同的颜色，使用贪婪算法的思想，每一步尽量使用较少的颜色，最终实现找到所需颜色最少的涂色方案。    在getColorMap函数里调用上述两个函数，获得邻接矩阵和涂色方案。    重载chessboard析构函数，释放申请的空间。    由于残缺棋盘的大小为指数增长，故动态申请内存空间。为防止因残缺棋盘规模过大，运算所需时间过长，导致主线程无响应，故以上运算过程均在子线程中进行，之后再根据所得的涂色方案在主线程中进行绘图。子线程run函数如下。    主线程中的绘图函数，使用QT 中的 QGraficView QGraficSence QGraficItem等类进行绘图，其中残缺使用白色表示（不涂色），其余涂色为黑、红、蓝 绿四色。    由于残缺棋盘的建立需要用户输入数据，为防治用户输入错误导致程序崩溃，故在输入函数中加强程序的健壮性。    gui模块中，考虑到生成的残缺棋盘可能超过view的范围，故新建类InteractiveView  继承自QGraphicsView，实现view的缩放、拖动等功能，有利于残缺棋盘的图形化显示，提高用户体验。    3.测试结果  3.1 k = 1 ,row = 1 col = 0      3.2 k = 2,row = 0,col = 1      3.3 k = 3 row = 5 col = 2      3.4 k = 4 row = 15 col = 7      3.5 k = 5 row = 10 col = 20      3.6 k = 6 row = 30 col = 55      3.7 k = 7 row = 100 col = 12    3.8 k = 8 row = 50 col = 255        4 . 分析与探讨  4.1 时间复杂度分析  令f(k)为函数tialBoard覆盖一个2k \* 2k残缺棋盘所需要的时间。当k = 0时，size = 1,所需时间为一个常数d。当k > 0时，需要4次递归函数调用，所需时间为4t(k - 1)。除去这些时间外，if条件测试和3个非残缺方格的覆盖也需要时间。令常数c表示这些额外时间。可以得到t(k)的递归表达式如下：  T(k) = d ,k = 0  T(k) = 4 t( k -1 ) k > 0  用替代方法来计算这个表达式，因为放置每一个三格板至少需要用时o(1)，所以不可能有一个比分而治之算法的渐进性能更好的算法。  首先判断棋盘size = 1，是则递归结束，否则size 减半，分为4个小棋盘，其中，以残缺所在位置分为4种情况，分别是左上、左下、右上、右下，将3个非残缺棋盘的小棋盘转化为残缺棋盘。为此，将一个三格板放置在由这3个小棋盘形成的角上。再次进行递归。  由于残缺棋盘的大小为指数增长，故动态申请内存空间。为防止因残缺棋盘规模过大，运算所需时间过长，导致主线程无响应，故以上运算过程均在子线程中进行，之后再根据所得的涂色方案在主线程中进行绘图。由于残缺棋盘的建立需要用户输入数据，为防治用户输入错误导致程序崩溃，故在输入函数中加强程序的健壮性。   1. 附录：实现源代码   因项目文件和源码过多，故将源码和可运行程序打包附在报告之后。项目图如图所示。 | | | |
|  | | | |