**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：201700140056 | 姓名：李港 | | 班级：17.4 |
| 上机学时：4 | | 日期：2020.04.11 | |
| 课程设计题目：残缺棋盘问题 | | | |
| 软件环境：VS2019 | | | |
| 报告内容：  **一、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**   1. 输入棋盘大小和残缺方格的位置。 2. 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。 3. 棋盘是平面图，要求使用最少的颜色覆盖着色。   **1.3 输入说明**   1. 输入设计：   采用终端输入或文件进行输入，仅需输入一个数字和一个坐标。   1. 输入数据样例：     **1.4 输出说明**  输出界面设计  采用色彩+数字的形式进行输出，残缺部分为0号。  需要注意，彩色输出在不同平台上有不同实现。需要具体情况具体分析。  输出数据样例    **二、分析与设计**  **2.1 问题分析**  需要设计两个大部分，解题函数与输出函数。  **2.2 主程序设计**  主程序进行输入，计算，输出一系列过程。  **2.3 设计思路**   1. 对于解题函数，采用四路递归设计。 2. 对于每一层递归，分成四个子象限。 3. 对于每个子象限，先进行残缺判断，若不残缺则向靠近棋盘中部的一个格子填充当前三格板编号，然后调用递归；若残缺，则直接递归。   **2.4 数据及数据类(型)定义**    int total\_num = 1; //表示三格板编号，最终等于三格板总数量  int Board[2048+10][2048+10]; //表示棋盘，每格数值为三格板编号  **2.5.算法设计及分析**  **递归设计：**   1. 对于解题函数，采用四路递归设计。 2. 对于每一层递归，分成四个子象限。 3. 对于每个子象限，先进行残缺判断，若不残缺则向靠近棋盘中部的一个格子填充当前三格板编号，然后调用递归；若残缺，则直接递归。   **三、测试**    可见，程序运行正常。  **四、分析与探讨**   1. 本实验较为简单，代码简洁，逻辑清晰。 2. 这次实验中我学习了终端的彩色输出方式，收货非常大。   **五、附录：实现源代码**  #pragma warning(disable:4996)  #include <cstdio>  #include<iostream>  #include <cmath>  #include <iomanip>  //#define OS\_TYPE\_WINDOWS\_CC  #include"ColorfulConsoleIO.h"  void ChessBoard(int tr, int tc, int dr, int dc, int size);  void OutputBoard(int size);  int total\_num = 1; //表示三格板编号，最终等于三格板总数量  int Board[2048+10][2048+10]; //表示棋盘，每格数值为三格板编号  int main(){  int n, a, b;  std::cout << R"( \_\_\_\_\_ \_ \_\_\_\_ \_  / \_\_\_\_| | | \_ \ | |  | | | |\_\_ \_\_\_ \_\_\_ \_\_\_| |\_) | \_\_\_ \_\_ \_ \_ \_\_ \_\_| |  | | | '\_ \ / \_ \/ \_\_/ \_\_| \_ < / \_ \ / \_` | '\_\_/ \_` |  | |\_\_\_\_| | | | \_\_/\\_\_ \\_\_ \ |\_) | (\_) | (\_| | | | (\_| |  \\_\_\_\_\_|\_| |\_|\\_\_\_||\_\_\_/\_\_\_/\_\_\_\_/ \\_\_\_/ \\_\_,\_|\_| \\_\_,\_|  )";  std::cout << "请输入递归次数：";  scanf("%d", &n);//输入2的幂次  int sum;  sum = pow(2, n);  std::cout << "请输入两个数字代表残缺点：";  scanf("%d %d", &a, &b);  Board[sum - a - 1][b - 1] = 0;      std::cout << "开始递归计算。\n";  ChessBoard(0, 0, a - 1, b - 1, sum);  std::cout << "递归计算完成。\n";  OutputBoard(sum);  std::cout << "输出完成。";  return 0;  }  //残缺棋盘的递归程序  void ChessBoard(int tr, int tc, int dr, int dc, int size){  if (size == 1) return;  int num = total\_num++; //所使用的三格板的编号  int sub\_size = size / 2;//象限大小    //对于左上象限  if (dr < tr + sub\_size && dc < tc + sub\_size) {//残缺方格位于本象限  ChessBoard(tr, tc, dr, dc, sub\_size);  }else{//本象限中没有残缺方格, 把三格板num号放在右下角，这样四个象限均有一个残缺  Board[tr + sub\_size - 1][tc + sub\_size - 1] = num;  ChessBoard(tr, tc, tr + sub\_size - 1, tc + sub\_size - 1, sub\_size);  }    //右上象限  if (dr < tr + sub\_size && dc >= tc + sub\_size) {//残缺方格位于本象限  ChessBoard(tr, tc + sub\_size, dr, dc, sub\_size);  }else { //本象限中没有残缺方格, 把三格板t放在左下角  Board[tr + sub\_size - 1][tc + sub\_size] = num;  ChessBoard(tr, tc + sub\_size, tr + sub\_size - 1, tc + sub\_size, sub\_size);  }    //左下象限  if (dr >= tr + sub\_size && dc < tc + sub\_size) {  ChessBoard(tr + sub\_size, tc, dr, dc, sub\_size);  }else{  Board[tr + sub\_size][tc + sub\_size - 1] = num;  ChessBoard(tr + sub\_size, tc, tr + sub\_size, tc + sub\_size - 1, sub\_size);  }    //右下象限  if (dr >= tr + sub\_size && dc >= tc + sub\_size) {  ChessBoard(tr + sub\_size, tc + sub\_size, dr, dc, sub\_size);  }else{  Board[tr + sub\_size][tc + sub\_size] = num;  ChessBoard(tr + sub\_size, tc + sub\_size, tr + sub\_size, tc + sub\_size, sub\_size);  }  }  //彩色显示输出  void OutputBoard(int size){  for (int i = 0; i < size; i++){  for (int j = 0; j < size; j++) {  std::wcout << (ConsoleBackgroundColor)(Board[i][j]%10) <<(ConsoleColor)((Board[i][j]+1) % 10)<<std::setw(2) << Board[i][j];  }  std::wcout << ConsoleBackgroundColor::None;  std::wcout << ConsoleColor::None;  printf("\n");  }  std::wcout << ConsoleBackgroundColor::None;  std::wcout << ConsoleColor::None;  } | | | |