**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：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 上机学时：4 | | 日期： 2019.4.25 | |
| 课程设计题目：基本数据结构及其应用之残缺棋盘的问题 | | | |
| 软件开发环境：  Microsoft Visual Studio 2019 | | | |
| 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 2^k×2^k个方格的棋盘，其中恰有一个方格残缺。对于任k，恰好存2^k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  第一行输入一个整数k  第二行输入残缺格的坐标  1.4 输出说明  输出一个2^k\*2^k的矩阵表示答案  2.设计  2.1 系统结构设计  使用标准输入输出读入数据和输出答案。  使用分治思想求解问题。  2.2 设计思路  如果棋盘大小为2\*2，则将另外3个不是不是残缺的格子填充即可。  否则将棋盘按照中轴线分割成左上/右上/左下/右下4个小的棋盘，残缺格一定存在于其中一个。假设残缺格在右下的小棋盘中，用一个三格板覆盖左上棋盘的右下角，右上棋盘的左下角，左下棋盘的右上角，并将其视为残缺格递归求解即可。  2.3 数据及数据类(型)定义  使用二维数组表示棋盘。采用递归函数进行求解。  递归函数定义为void dfs(int x, int y, int l, int px, int py)表示求解左上角坐标为(x, y),长度为2^l，残缺格坐标为（px, py）的递归程序。  2.4.算法设计及分析  （各模块算法及类内函数的算法伪码表示）  当l=1时先检测每个方格四周已经用哪些颜色填充过然后确定当前三格板应当用什么颜色填充。    否则就分情况讨论判断残缺格在左上/左下/右上/右下哪个小棋盘中    3. 测试结果    4. 分析与探讨  该方法利用了分而治之的思想，将一个大的棋盘分为4个小的棋盘，4个小棋盘的答案可以组合成大棋盘的答案，且每个小棋盘的求解模式与大棋盘的求解模式类似，因此可以递归求解问题，时间复杂度为O(k)  5. 附录：实现源代码  #include <iostream>  #include <cassert>  using namespace std;  void dfs(int x, int y, int l, int px, int py, int\*\* e) {  if (l == 1) {  bool vis[5] = { false };  int dx[] = { 0, -1, 0, 1 };  int dy[] = { 1, 0, -1, 0 };  for (int i = x; i < x + (1 << l); ++i) {  for (int j = y; j < y + (1 << l); ++j) {  for (int k = 0; k < 4; ++k) {  int tx = i + dx[k];  int ty = j + dy[k];  vis[e[tx][ty]] = true;  }  }  }  int col;  for (col = 1; col <= 4 && vis[col]; ++col);  for (int i = x; i < x + (1 << l); ++i) {  for (int j = y; j < y + (1 << l); ++j) {  if (i == px && j == py) continue;  e[i][j] = col;  }  }  return;  }  int len = 1 << l;  if (px < x + len / 2 && py < y + len / 2) {  e[x + len / 2 - 1][y + len / 2] = 1;  e[x + len / 2][y + len / 2 - 1] = 1;  e[x + len / 2][y + len / 2] = 1;  dfs(x, y, l - 1, px, py, e);  dfs(x, y + len / 2, l - 1, x + len / 2 - 1, y + len / 2, e);  dfs(x + len / 2, y, l - 1, x + len / 2, y + len / 2 - 1, e);  dfs(x + len / 2, y + len / 2, l - 1, x + len / 2, y + len / 2, e);  }  else if (px < x + len / 2 && py >= y + len / 2) {  e[x + len / 2 - 1][y + len / 2 - 1] = 1;  e[x + len / 2][y + len / 2 - 1] = 1;  e[x + len / 2][y + len / 2] = 1;  dfs(x, y, l - 1, x + len / 2 - 1, y + len / 2 - 1, e);  dfs(x, y + len / 2, l - 1, px, py, e);  dfs(x + len / 2, y, l - 1, x + len / 2, y + len / 2 - 1, e);  dfs(x + len / 2, y + len / 2, l - 1, x + len / 2, y + len / 2, e);  }  else if (px >= x + (1 << (l - 1)) && py < y + (1 << (l - 1))) {  e[x + len / 2 - 1][y + len / 2 - 1] = 1;  e[x + len / 2 - 1][y + len / 2] = 1;  e[x + len / 2][y + len / 2] = 1;  dfs(x, y, l - 1, x + len / 2 - 1, y + len / 2 - 1, e);  dfs(x, y + len / 2, l - 1, x + len / 2 - 1, y + len / 2, e);  dfs(x + len / 2, y, l - 1, px, py, e);  dfs(x + len / 2, y + len / 2, l - 1, x + len / 2, y + len / 2, e);  }  else {  e[x + len / 2 - 1][y + len / 2 - 1] = 1;  e[x + len / 2 - 1][y + len / 2] = 1;  e[x + len / 2][y + len / 2 - 1] = 1;  dfs(x, y, l - 1, x + len / 2 - 1, y + len / 2 - 1, e);  dfs(x, y + len / 2, l - 1, x + len / 2 - 1, y + len / 2, e);  dfs(x + len / 2, y, l - 1, x + len / 2, y + len / 2 - 1, e);  dfs(x + len / 2, y + len / 2, l - 1, px, py, e);  }  }  int main() {  int k, len;  int px, py;  cin >> k;  cin >> px >> py;  len = 1 << k;  assert(px > 0 && px <= len && py > 0 && py <= len);  int\*\* e = new int\* [len + 2];  for (int i = 0; i <= len + 1; ++i) e[i] = new int[len + 2];  for (int i = 0; i <= len + 1; ++i)  for (int j = 0; j <= len + 1; ++j)  e[i][j] = 0;  dfs(1, 1, k, px, py, e);  for (int i = 1; i <= len; ++i) {  for (int j = 1; j <= len; ++j)  cout << e[i][j] << ' ';  cout << endl;  }  for (int i = 0; i <= len + 1; ++i) delete[] e[i];  delete[] e;  return 0;  } | | | |
|  | | | |