**华南农业大学**

**《算法分析与设计》  
课程实验**

专业年级： 2014级信息与计算科学

学生学号： 201430120308

学生姓名： 何 唯

实验题目： 棋盘覆盖

指导老师： 赵 峰

实验时间： 2017/03/14

1. **实验题目**

• [题目描述]

在一个的棋盘中，有一个特殊方格，要求用L型骨牌覆盖满除特殊方格外的所有其他方格，且骨牌不得重叠.（骨牌可以旋转放置）

• [输入说明]

k值、特殊方格坐标

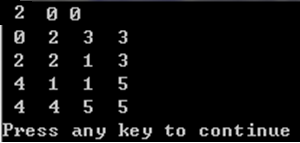
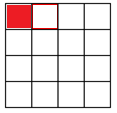
• [输出说明]

骨牌放法.其中用0表示特殊方格，同一张骨牌所占方格用同一个数字表示，不同骨牌用不同数字.

• [实验要求]

1.至少运行3组不同的输入数据；

2.用分治法求解.

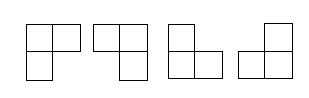
1. **实验实现**
2. 输入、输出
3. 骨牌
4. 棋盘

**• 1．**[编程语言]本实验用Java完成算法设计和程序设计并调试通过。

**• 2．**[解题思想、方法]

本实验题目为简单的棋盘覆盖问题，程序采用Java在MyEclipse下完成。在一个个方格组成的棋盘中，恰有一个方格与其他方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。

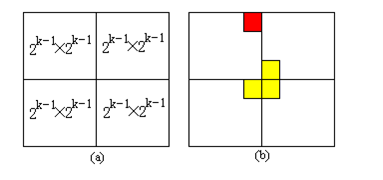
解题思想采用分治法，利用递归的方式，程序中设计一个ChessBoard类，其中有方法chessBoard，该方法参数为一个二维整型数组board表示方法要处理的棋盘,整型参数tr表示棋盘左上角方格的行数，整型参数tc表示棋盘左上角方格的列数，整型参数dr表示特殊方格的所在行数，整型参数dc表示特殊方格所在列数，整型参数k表示棋盘大小的指数,。全局变量tile表示当前L型骨牌序号,L型骨牌可以分为以下四种情况，见下图一。



图一

分治法的思想体现在递归上面，每次递归调用chessBoard函数，把棋盘大小缩小一倍，即k=k-1，进行判断每次先记录下整个大方块的左上角方格的行列坐标，然后再与特殊方格坐标进行比较，就可以知道特殊方格是否在该块中。如果特殊方块在里面，这直接递归下去求即可，如果不在，这根据分割的四个方块的不同位置，把右下角、左下角、右上角或者左上角的方格标记为特殊方块，然后继续递归。直接调用下一次递归；否则，将把子棋盘最靠近父棋盘中心位置的方块设置为特殊方块处理，再以其为特殊方块来进行下一次递归。当子棋盘大小为1时，即k=0的时候，递归停止并返回。

整个函数按左上子棋盘、右上子棋盘、左下子棋盘、右下子棋盘的顺序进行判断，逐次进行判断并进行下一次递归。每次递归利用分治法的思想，把问题规模分解成4个规模小一倍的子问题来处理,如下图二(a)中把的父问题分解为4个子问题,直到最终解决的是大小为的棋盘覆盖问题。



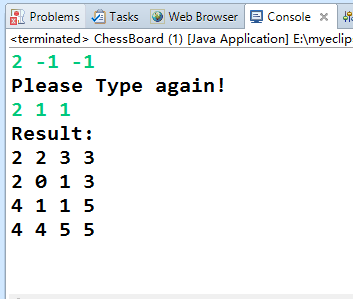
图二

**• 3．**[实验说明，包括输入、输出、结果截图]

本实验程序按照实验要求的输入输出进行设计，并且具有一定健壮性，保证了输入标准的问题，后面会补充改进一系列相关异常处理。下面是三组实验的输入输入以及结果截图：

输入1: **2 1 1**

输出1:

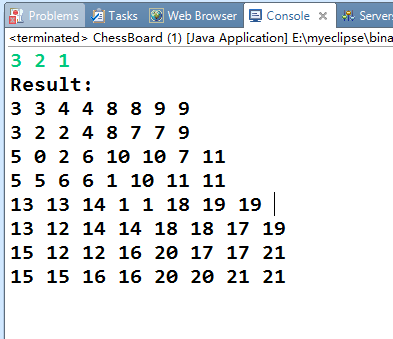


**即：红色表示特殊方格**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

输入2: **3 2 1**

输出2:

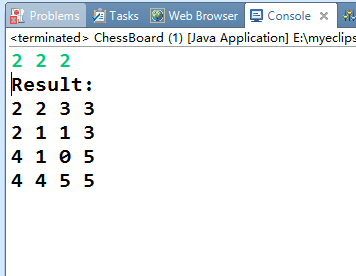


**即：红色表示特殊方格**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

输入3: **2 2 2**

输出3:



**即：红色表示特殊方格**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**• 4．**[算法复杂性计算过程]

设是算法chessBoard覆盖一个棋盘所需时间。从算法的分割策略可知，满足以下递归方程



**｛**



解此递归方程可得****。由于覆盖棋盘所需的L型骨牌个数为，故算法chessBoard是一个在渐进意义下最优的算法。**小结和心得**

[例如：设计中遇到的问题，如何解决，实验心得，对算法改进的设想等]

在本次实验中遇到的问题是，L型骨牌的计数项t的范围问题，假如设计的程序中，没有把棋盘chessBoard设计为类的话，t的计数会比较难处理。比如以下程序段：

**import** java.util.Scanner;

**import** java.lang.Math;

**public** **class** ChessBoard2{

**private** **static** **int** *t*=0;

**private** **static** **void** chessBoard(**int**[][] board,**int** tr,**int** tc,**int** k, **int** dr,**int** dc)

{

//覆盖棋盘算法主体跟附件的程序一致

}

}

**public** **static** **void** main(String[] args){

Scanner scan=**new** Scanner(System.*in*);

**int** k=scan.nextInt();

**int** dr=scan.nextInt();

**int** dc=scan.nextInt();

**int** board[][]=**new** **int**[(**int**) Math.*pow*(2.0, k)][(**int**) Math.*pow*(2.0, k)];

**int** t=0;

*chessBoard*(board, 0, 0, k, dr, dc);

System.*out*.println("Result:");

**for**(**int**[]i:board){

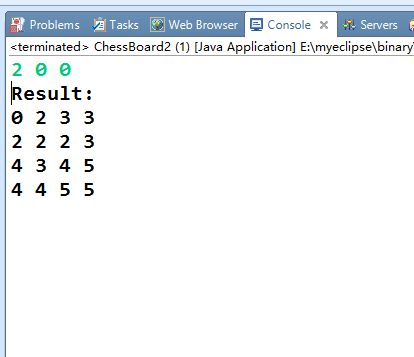
//

}

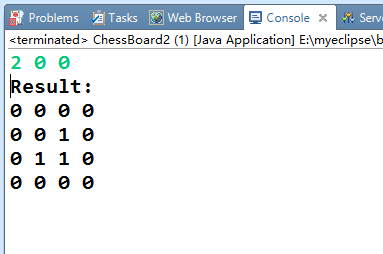
}

}

程序输入2 0 0时，输出如下图：



分析其原因，可以发现在第一步递归就停止的话，就输入2 0 0，输出的结果是这样的:



此结果说明第一步是没有问题的，问题出在t被设置为静态全局变量这里，即当函数递归一次，t就自增1，当递归函数返回到上一层的时候，因为作为静态全局变量，t的值为自增后的值，因此会出现一开始中间骨牌为234的情况。

解决的方法，将采用面向对象的程序设计思想，把chessBoard定义为类，然后将t定义为类的实例变量，这样就可以得出正确的实验结果。

说到实验的心得的话，自己本阶段也正在复习Java的基本语法，此实验正好可以拿来练练手。其次，本实验也让我更加熟悉分治法的基本思想，将一个规模比较大的问题分解为多个规模较小或者规模足以用简单算法解决的问题。最后再将小问题的结果组合成规模较大问题上。

改进算法的想法的话，这里我觉得可以从递归入手，因为在大多数问题中，递归的效率并不是最高的，往往耗费更多的时间空间，因为它是基于栈的数据结构，调用一次函数递归一次的话就往栈里压一次。而改进递归的方法有很多，可以改为迭代来分析下算法复杂度。

举例来说，对于求阶乘的函数来说，其迭代算法的时间复杂度为O(n)：

int fact(n)

{

int i;

int r = 1;

for(i = 1; i < = n; i++)

{

r \*= i;

}

return r;

}

而该算法同样地递归算法时间复杂度也是O(n)

int fact\_r(n)

{

if(n == 0)

return 1;

else

return n \* f(n);

}

但是关键的是，递归不断调用自己，增大了空间开销，其效率是明显更差的。

另外，也可以从算法本身入手，考虑能否不用分治法，改为用动态规划、备忘录等等其他算法。

**源程序清单**

/\*\*

\* **@Project**：Algorithm

\* **@File**：ChessBoard

\* **@Author**：何唯

\* **@Date**：2017年3月27日 下午11:13:46

\* **@Description**： 棋盘覆盖问题

\*/

**package** Algorism;

**import** java.util.Scanner;

**import** java.lang.Math;

**public** **class** ChessBoard{

**int** tile=1;//全局变量

/\*\*

\*

\* **@Description**: **TODO**

\* **@author** ：何唯

\* **@date** ： 2017年3月27日 下午11:23:52

\* **@return** void

\* **@param** **@param** board

\* **@param** **@param** tr

\* **@param** **@param** tc

\* **@param** **@param** k

\* **@param** **@param** dr

\* **@param** **@param** dc

\* **@file** ChessBoard.java

\* **@throws**

\*/

**public** **void** chessBoard(**int**[][] board,**int** tr,**int** tc,**int** k, **int** dr,**int** dc)

{

**if**(Math.*pow*(2.0, k)==1)**return**;

**int** t=tile++;

//以单元格[(int) Math.pow(2.0, k-1)-1,(int) Math.pow(2.0, k-1)-1]为基准做判断

//覆盖左上棋盘

**if**(dr<=tr+(**int**) Math.*pow*(2.0, k-1)-1&&dc<=tc+(**int**) Math.*pow*(2.0, k-1)-1)//特殊方格在左上棋盘

{

chessBoard(board,tr,tc,k-1,dr,dc);

}**else**{

board[tr+(**int**) Math.*pow*(2.0, k-1)-1][tc+(**int**) Math.*pow*(2.0, k-1)-1]=t;

chessBoard(board,tr,tc,k-1,tr+(**int**) Math.*pow*(2.0, k-1)-1,tc+(**int**) Math.*pow*(2.0, k-1)-1);

}

//覆盖右上棋盘

**if**(dr<=tr+(**int**) Math.*pow*(2.0, k-1)-1&&dc>tc+(**int**) Math.*pow*(2.0, k-1)-1)//特殊方格在右上棋盘

{

chessBoard(board,tr,tc+(**int**) Math.*pow*(2.0, k-1),k-1,dr,dc);

}**else**{

board[tr+(**int**) Math.*pow*(2.0, k-1)-1][tc+(**int**) Math.*pow*(2.0, k-1)]=t;

chessBoard(board,tr,tc+(**int**) Math.*pow*(2.0, k-1),k-1,tr+(**int**) Math.*pow*(2.0, k-1)-1,tc+(**int**) Math.*pow*(2.0, k-1));

}

//覆盖左下棋盘

**if**(dr>tr+(**int**) Math.*pow*(2.0, k-1)-1&&dc<=tc+(**int**) Math.*pow*(2.0, k-1)-1)//特殊方格在左下棋盘

{

chessBoard(board,tr+(**int**) Math.*pow*(2.0, k-1),tc,k-1,dr,dc);

}**else**{

board[tr+(**int**) Math.*pow*(2.0, k-1)][tc+(**int**) Math.*pow*(2.0, k-1)-1]=t;

chessBoard(board,tr+(**int**) Math.*pow*(2.0, k-1),tc,k-1,tr+(**int**) Math.*pow*(2.0, k-1),tc+(**int**) Math.*pow*(2.0, k-1)-1);

}

//覆盖右下棋盘

**if**(dr>tr+(**int**) Math.*pow*(2.0, k-1)-1&&dc>tc+(**int**) Math.*pow*(2.0, k-1)-1)//特殊方格在右下棋盘

{

chessBoard(board,tr+(**int**) Math.*pow*(2.0, k-1),tc+(**int**) Math.*pow*(2.0, k-1),k-1,dr,dc);

}**else**{

board[tr+(**int**) Math.*pow*(2.0, k-1)][tc+(**int**) Math.*pow*(2.0, k-1)]=t;

chessBoard(board,tr+(**int**) Math.*pow*(2.0, k-1),tc+(**int**) Math.*pow*(2.0, k-1),k-1,tr+(**int**) Math.*pow*(2.0, k-1),tc+(**int**) Math.*pow*(2.0, k-1));

}

}

/\*\*

\*

\* **@Description**: **TODO**

\* **@author** ：何唯

\* **@date** ： 2017年3月27日 下午11:24:05

\* **@return** void

\* **@param** **@param** args

\* **@file** ChessBoard.java

\* **@throws**

\*/

**public** **static** **void** main(String[] args){

Scanner scan=**new** Scanner(System.*in*);

**int** k=scan.nextInt();

**int** dr=scan.nextInt();

**int** dc=scan.nextInt();

**int** board[][]=**new** **int**[(**int**) Math.*pow*(2.0, k)][(**int**) Math.*pow*(2.0, k)];

**int** t=0;

**while**(dr>(**int**) Math.*pow*(2.0, k)-1||dr<0||dc<0||dc>(**int**) Math.*pow*(2.0, k))

{

System.*out*.println("Please Type again!");

k=scan.nextInt();

dr=scan.nextInt();

dc=scan.nextInt();

}

//创建棋盘对象

ChessBoard cb=**new** ChessBoard();

cb.chessBoard(board, 0, 0, k, dr, dc);

//打印输出

System.*out*.println("Result:");

**for**(**int**[]i:board){

**for**(**int** j:i){

System.*out*.print(j+" ");

}

System.*out*.println();

}

}}

**五、评语及评分**

|  |
| --- |
| **评语及评分**  评阅人签名： |