

高级语言程序设计

合成十实验报告

作 者 姓 名： 王灏廷

学 号： 1953609

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

二〇二一年六月

1. **题目及基本要求描述**

**1.1.题目综述**

1.要求命令行方式找出可合并项并标识（不使用递归方式）。

2.要求使用递归方式，找出可合并项并标识。

3.命令行方式下完成一次合成

（包括查找相邻项、合并相邻项、计算得分、下落消除0、在0位置产生新数据）。

4.命令行方式完整版

（合成到预期目标值后，给出提示信息，但不结束，可继续进行游戏）。

5.cmd 伪图形窗口显示内部数组的内容（数字色块间无分隔线）

键盘输入行列数，随机产生数组，将数组的内容在 cmd 窗口中用伪图形显示出来。

6.cmd 伪图形窗口显示内部数组的内容（数字色块间有分隔线）。

7.cmd 伪图形窗口下的“当前选择”色块的选择

用箭头键实现“当前选择”色块的选择，按回车确认

越过边界后采用环绕的方式。

8.cmd 伪图形窗口下完成一次合成

箭头键选择色块，回车键选定合成位置，并将所有可合成的色块标注出来

选定后再次按箭头键则取消本次选定，重新选择，按回车则进行一次合成

逐步展现 合并相邻项、下落消除0、0位置产生新数据、计算得分的操作。

9.cmd 伪图形窗口完整版

达到合成目标后游戏不结束，合成目标+1后游戏继续

如果某次合并后无法找到可合并位置，则提示游戏结束

cmd窗口的上下各有一个状态栏，显示得分、目标、操作提示等。

**1.2.要求与限制**

1.随机数按规则产生。

初始最大值设置为3，,之后由合并后最大值决定随机数产生的概率。

2.判断规则

若输入坐标位置无相邻相同数，要提示出错并重新输入

如果整个数组无相邻位置值相等，并提示游戏结束。

如果达到合成目标，则提示完成，游戏不结束，继续向更高目标前进。

3.分数累加规则

本次新增得分=消除值\*消除个数\*3

4.整个程序，不允许使用任何形式的全局变量/数组/指针，

允许使用全局的宏定义或常变量。

**2.整体设计思路**

**2.1.合成十问题本身的求解思路**

合成十问题本质上是求二维数组中一个值全部相等的相邻值。可以用到图算法中的BFS或是DFS算法。这里我采用的是BFS，分为以下几步

1.访问所选中的顶点v；

2.依次从v的未被访问的邻接点出发，对图进行宽度优先遍历；直至图中和v有路径相通的同一层顶点都被访问；

3.若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行宽度优先遍历，直到图中所有顶点均被访问过为止。

然后可以分别使用递归和非递归两种方法实现这个算法，对应功能2与功能1，所得结果全部存在findmoto数组中。

**2.2.抽象化合成十及其移动的思路**

合成后需要进行的操作有包括查找相邻项、合并相邻项、计算得分、下落消除0和在0位置产生新数据等。通过设置二维数组moto[8][10]来存储游戏功能的图，值即为其显示值；设置findmoto[8][10]来存储查找相邻值功能的图，相等则置1，不等则置0。通过这两个数组的操作，加以随机数产生函数，即可完成全部功能。

**2.3.程序整体实现思路**

Main函数用一个循环和switch()函数来做到各小题函数的调用。而与上次不同的是，我将函数按功能分为了几类：逻辑功能函数、数组函数、进行命令行输出的函数和进行cmd伪图形化输出的函数。并分别将其放在不同的cpp文件中。在各个小题中我使用这些函数并使用一些基础的循环、条件语句来完成所给的要求。各个部分将在下一部分进行具体介绍，而那些基础的循环、条件语句则略去。

整体的设计思路是处理输入的行列，通过改变数组的函数来改变数组。之后再通过进行命令行输出的函数和进行cmd伪图形化输出的函数来将这种变化可视化，从而合成十游戏的任务。

整个程序分为五个部分。

头文件用于存放所有cpp文件中的函数声明，方便cpp之间能够互相访问。

Main文件用于初始化屏幕、调用菜单函数并返回选项以及根据选项调用initial/initial\_graph函数。

Tools文件存放数组输出和图形化实现共用的底层逻辑函数，如找结果、判0与随机数函数等。

Base文件存放数组输出的各类函数和数组化实现的初始化initial函数，用以实现1-4功能。

Console文件存放图形化实现的各类函数与图形化实现的初始化initial\_graph函数，用以实现5-9功能。

**3.主要功能的实现**

**3.1.递归和非递归方法解决找到坐标相邻的所有相同值**

**3.1.1 非递归部分**

这个部分可能是我整个程序花费时间最长的部分。

1.首先定义了一个Node结构体，用来存放符合条件点的横纵坐标和访问情况，被访问过置1，未访问过置0。然后将目标点作为结构体数组的第一个，访问情况置为1.

2.先通过一个遍历，将所有与目标值相等的值筛选出来，存进结构体数组中。

3.对这个结构体数组进行遍历，设置一个final数组存放最终结果结构体的下标。

若相邻且访问情况为0，则将结果送进final数组，表明这个值可以被消去，访问情况置1。

对一个点的四个方向遍历结束后，通过final数组访问下一个目标点的下标，作为下一个起始点，直至整个Node结构体数组被访问完或没有找到新的相邻点。

4.一次循环，将所有final数组中对应的点在findmoto数组中置1，表示可以与目标点值相同，可以被同时消去。

**3.1.2 递归部分**

这个就轻松很多了，很快搞定。

1.x/y小于0或大于行/列上限，则直接返回，表明已经到底了。

2.分别判断所选点四个方向是否有相等且未被访问过的点

若有，则选中这个点作为下一次递归的起始点；若无，遍历后返回。

通过findmoto作标记，为1是访问过，为0是没被访问过。

**3.2.随机值部分**

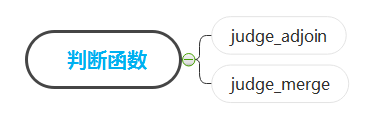
1.首先在程序开始时，使用srand函数初始化随机数种子，保证出现的随机数不是固定的。

2.设置一个找最大值的函数，通过遍历图来返回图中的最大值，用于随机数函数调用。

3.根据返回的最大值（初始为3），分别对应不同的情况。各种情况概率的最大公约数是5%，

所以可以n%20之后完成操作，分别对应即可。

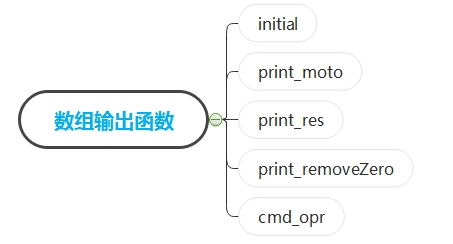
**3.3.判断函数**



1.遍历目标点，相邻有相等值返回1，否则返回0.

2.遍历整个图，可以进行合并返回1，否则返回0.

**3.4.数组输出函数**



1.初始化选择。首先规范输入，然后根据不同选择进入1-4的不同功能。

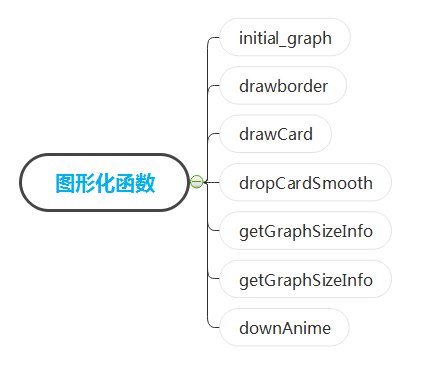
2.打印初始数组。

3.打印消除后的结果。

4.打印除去0之后的结果。

5.用于功能3/4的函数，根据选择可以完成循环消去并计分。

**3.5.图形化函数**



这一部分不用一一介绍，根据名字即可理解

（主要的实现过程非常麻烦，调整坐标很繁琐，但没有什么技术层面的问题

**4.调试过程碰到的问题**

**4.1.越界问题**

这个问题老生常谈，但是在写的函数非常多的时候很难察觉到，虽然知道数组下标加减之前要判断是否越界，但是写的代码一多真的很容易忘记。

举几个例子

判断相邻的函数。在达到边界条件时应该判断+1/-1之后是否越界，不然同样会越界。

递归函数的返回条件。Row其实是行数+1，column是列数+1，所以如果判断是到row/column的话，事实上已经越界了，但是发现bug之后才想到这一点，耗费了很多时间。

**4.2.循环后Findmoto初始化问题**

Findmoto数组是用来存放一次合并中消去值情况的数组，在完成一次合并之后每次都需要初始化全部置0。我起初直接写了Findmoto=0，然后moto出问题的时候才发现只改了首地址的值。

之后写了一个循环，把每一个值都改成0才能完成初始化，问题解决。

**4.3 gotoxy函数输出问题**

我最开始的各类print函数中使用了cct\_gotoxy()函数来分别输出框架和内部的数组。但是后来我发现gotoxy()函数只能到达当前窗口的坐标，而不能产生滚轮而自动向下。这就导致当我输出到最下面的时候再使用gotoxy函数就会出现预期之外的结果。最后我将函数改写成了单纯使用cout来一行行输出，解决了这个问题。

**5.心得体会**

**5.1.在完成本次作业中的心得体会与经验教训**

**5.1.1.心得体会**

1)反复出现的常量应该用常变量或者宏定义的方式写在头文件中，便于维护。

2)变量命名要准确易于理解，最好使用下划线命名法或驼峰命名法。

本次作业我尝试了这两种命名法都应用一遍。Cosole中使用驼峰命名法，其他cpp中使用下划线命名法，让我对这两种命名方法的理解更进了一步。

3)在源程序中要善于写注释，并且对不同的函数和不同的定义做分类以及说明，减少维护

线程序时浪费的时间。在同一函数中也要注意用空行来区分不同代码块，便于区分。

4)要善于构建临时测试，并学会打断点，边写代码边做测试，这样可以大大节省后续调试的时间，同时也不会因为程序过于庞大而找不到错误在哪。

5)充分思考后再开始写代码，增加代码精巧程度，减少无效代码和垃圾代码。

**5.1.2 函数分类**

这次大作业中我将函数按照功能分为了五类，并放在了不同的cpp文件中。这样做能让我更加方便的找到各个函数的位置，并且能增加我对于程序的理解

**5.1.3 通过改变参数来使函数完成不同的功能**

在这次大作业中我利用改变参数来使函数完成不同的功能，合并了一些函数和更改了以前的公共函数。在我看来，这样做的最大好处就是能够使函数的数量减少，看起来也更加简洁明了。

**5.2分为若干小题的方式是否对你的设计起到了一定的提示作用？**

有一定的提示作用，可以让我按照小题的顺序去一步步实现功能，将一个很大的程序分解成一个个很小的功能，再一点一点去实现，我觉得这个指引很有帮助。

小意见：我觉得找到坐标相邻的所有相同值的话，非递归方法是要比递归方法难很多的。

可以将1/2小题调换位置，方便由简单到困难实现。

**5.3.1与汉诺塔相比，你在函数的分解上是否做到了更合理？**

总体上来讲，我认为我实现了。

首先汉诺塔的所有功能都是杂糅在solution一个cpp中的，需要调用函数的时候会非常麻烦，很容易混淆。而合成十则将功能拆分为了tools/base/console三个cpp，分别实现底层逻辑/数组输出/图形化实现，我觉得这个模块化对函数的分解是有很大帮助的。

其次汉诺塔公用的递归函数其实是非常拥挤的，15行内要实现8个不同功能的递归，只能体外再写一个分支函数，在划分上会显得特别杂糅。而本次底层函数没有了这种限制，就可以实现具体功能具体函数实现，我觉得这种分解相较于上一次有了很大的提高。

**5.3.2介绍一下你的几个重点函数的分类、方法及使用情况（特别是涉及到用参数区分差异的部分）**

分类上文中已经详细阐述，这种规整的模块化其实很令人赏心悦目。

方法：实现相似功能的函数可以放在一起，调用的时候方便对比和查找。

实现同一组功能的函数可以放在一个文件，如数组输出/图形化实现就可以划分不同的函数。

使用情况：参数区分差异主要是为了减少重复的代码冗余，用相同的代码实现尽量多的功能。

这里以print\_res为例。

调用参数设置标识符tag，若tag为1，则表示是第一次调用该函数，需要额外显示findmoto数组的情况，所以将打印findmoto数组，为1打印\*,为0打印0。

若tag为0，则表示是消去后调用该函数，就不需要打印findmoto数组的情况了，只用打印moto数组情况即可。

**5.4以本次作业为例，谈谈编写一个相对复杂的程序的心得体会**

相对复杂函数的主要特点就是大。代码量大，函数多，很容易出现遗忘的问题。

这个时候平时小作业所嗤之以鼻的写注释、规范命名、模块函数分类就显得非常重要。

不写注释，过几天就忘了函数/参数每个是干嘛的了，非常不利于维护。

不规范命名，比如全部命名为abcd1234，甚至不用过几天，只要这个函数写得稍微长一点，轻则忘了参数是干嘛的，重则相互混淆，造成很难排除的bug。函数更是如此，命名如果没有意义，后续调用都不知道是什么功能，整个程序很容易崩掉。

模块函数分类。这次的作业，应该是函数写的最多的一次作业，有很多很多功能需要实现，如果不将其分类，找起来真的是眼花缭乱，很容易看困了，或者上下翻半天才知道是干嘛的，非常浪费时间不说，心态还容易受影响。

所以我觉得应该：

1.做好函数分类：

将函数按照功能分为类，放在不同的cpp文件中。便于寻找与理解。（详见5.1.2）

2.养成好的函数与变量命名规则：

易于理解函数的含义，减少错误引用。（详见5.1.1）

**6.附件：源程序**

**6.1.所有找到相邻的相等数字，非递归方式实现**

void find\_res(int moto[][10], int row, int column, int findmoto[][10], int x, int y, int res)//所有找到相邻的相等数字，非递归方式实现

{

static int find = 1;

struct node

{

int x;

int y;

int tag = 0;

};

node a[80];

a[0].x = x;

a[0].y = y;

a[0].tag = 1;

int final[80] = { 0 };

int sum = 1;

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

if (moto[i][j] == res)

{

a[find].x = i;

a[find].y = j;

find++;

}

}

}

for (int i = 0; i < sum; i++)

{

for (int j = 1; j < find; j++)

{

if (a[final[i]].x + 1 == a[j].x && a[final[i]].y == a[j].y && a[j].tag == 0)

{

final[sum] = j;

sum++;

a[j].tag = 1;

}

else if (a[final[i]].x - 1 == a[j].x && a[final[i]].y == a[j].y && a[j].tag == 0)

{

final[sum] = j;

sum++;

a[j].tag = 1;

}

else if (a[final[i]].x == a[j].x && a[final[i]].y + 1 == a[j].y && a[j].tag == 0)

{

final[sum] = j;

sum++;

a[j].tag = 1;

}

else if (a[final[i]].x == a[j].x && a[final[i]].y - 1 == a[j].y && a[j].tag == 0)

{

final[sum] = j;

sum++;

a[j].tag = 1;

}

}

}

for (int i = 0; i < sum; i++)

{

findmoto[a[final[i]].x][a[final[i]].y] = 1;

}

}

**6.2.所有找到相邻的相等数字，递归方式实现**

void find\_res\_rec(int moto[][10], int row, int column, int findmoto[][10], int x, int y, int res)

{

if (x < 0 || y < 0 || x >= row || y >= column)

return;

static int num = 1;

if (x > 0)

if (moto[x - 1][y] == res && !findmoto[x - 1][y])

{

num++;

findmoto[x - 1][y] = 1;

find\_res\_rec(moto, row, column, findmoto, x - 1, y, res);

}

if (y > 0)

if (moto[x][y - 1] == res && !findmoto[x][y - 1])

{

num++;

findmoto[x][y - 1] = 1;

find\_res\_rec(moto, row, column, findmoto, x, y - 1, res);

}

if (x < row - 1)

if (moto[x + 1][y] == res && !findmoto[x + 1][y])

{

num++;

findmoto[x + 1][y] = 1;

find\_res\_rec(moto, row, column, findmoto, x + 1, y, res);

}

if (y < column - 1)

if (moto[x][y + 1] == res && !findmoto[x][y + 1])

{

num++;

findmoto[x][y + 1] = 1;

find\_res\_rec(moto, row, column, findmoto, x, y + 1, res);

}

}

**6.3.随机数函数**

int random(int max)//随机值函数

{

int n;

switch (max)

{

case 3:

n = (rand() % 3) + 1;;

return n;

case 4:

n = rand() % 10 + 1;

if (n == 10)

return 4;

else

return n % 3 + 1;

case 5:

n = rand() % 20 + 1;

if (n == 1 || n == 2)

return 5;

else if (n == 3 || n == 4 || n == 5)

return 4;

else

return n % 3 + 1;

case 6:

n = rand() % 20 + 1;

if (n == 1)

return 6;

else if (n == 3 || n == 4 || n == 5)

return 5;

else

return n % 4 + 1;

default:

n = rand() % 20 + 1;

if (n == 1)

return max;

else if (n == 2)

return max - 1;

else if (n == 3 || n == 4)

return max - 2;

else

return rand() % (max - 3) + 1;

}

}

**6.3.判断相邻函数**

int judge\_merge(int moto[][10], int row, int column)

{

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

if (i >= 1)

{

if (moto[i][j] == moto[i - 1][j])

continue;

}

if (j >= 1)

{

if (moto[i][j] == moto[i][j - 1])

continue;

}

if (i < row - 1)

{

if (moto[i][j] == moto[i + 1][j])

continue;

}

if (j < column - 1)

{

if (moto[i][j] == moto[i][j + 1])

continue;

}

else

return 0;

}

}

return 1;

}

int judge\_adjoin(int moto[][10], int row, int column, int x, int y)

{

if (x >= 1)

{

if (moto[x][y] == moto[x - 1][y])

return 1;

}

if (y >= 1)

{

if (moto[x][y] == moto[x][y - 1])

return 1;

}

if (x < row - 1)

{

if (moto[x][y] == moto[x + 1][y])

return 1;

}

if (y < column - 1)

{

if (moto[x][y] == moto[x][y + 1])

return 1;

}

return 0;

}