**盘点数独终盘生成算法**

数独难玩，那设置数独题目容易吗？为了讲解方便，先给数独的九宫格下一个定义，如下图所示，将数独分为9个九宫格，从上到下，从左到右依次编号1-9。将数独81个小格子定义为一个二维数组array[9][9]。如果对于数独的玩法还不了解的，那么本文并不适合你，请先移步[数独百度百科](https://baike.baidu.com/item/%E6%95%B0%E7%8B%AC/74847?fr=aladdin)了解一下数独游戏的规则。

如要构思一个生成数独题目的程序，应该从哪里入手呢？这里有两种方案：方案一，提前设置好数独库，将题量充足的数独题目先 作为数据保存起来，用的时候随机取出数独题目， 留一些空出来即可；方案二，通过算法实时 生成数独题目，再留空出来。作为一名优秀的程序员，肯定要追求难度更大的方案二啦，那么如何设计算法 快速生成有解的 数独题目呢？我在学习了《编程之美》和各位 博主的答案之后，总结成如下四种解决方案。

1、常规回溯方法

2、以宫为单位的矩阵置换法

3、以数字顺序的以宫为单位的回溯法

4、以某一数独为基础的数字替换法

1、常规回溯方法

这种方法是最容易想到的， 但是执行效率可能不是很高的方法。

思路分析：

常规回朔法的最原始方法即从数独的左上角（0,0）处开始，生成随机数，然后依次按照从左往右、从上往下的顺序逐渐生成满足数独规则的最终数独。不过从前人总结的经验和我自己编程过程中遇到的问题来看：这种方法实在是太笨了！笨到你不可能简简单单的通过一个个随机生成的数来生成最终的数独。因此需要采用回朔法：在某一步生成1-9的数字均不能满足数独要求的时候需要退回到上一个状态重新生成其他可能的解。如此直至最终拼成一个完整的数独地图，这一个过程如下图示意：

C++代码：

#include <iostream>

#include <stdlib.h> // srand()

#include <time.h> // time()

using namespace std;

int sudoMap[9][9] = {0}; // initialise map.

int counter = 1; // the number of map already generated.

int maxMapNumber = 1; // how many map you want to generate.

void printMap()

{

for(int i = 0; i < 9; i++)

{

for(int j = 0; j < 9; j++)

{

cout << sudoMap[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

bool check()

{

//check the row

for(int i = 0; i < 9; i++)

{

int tempArr[9] = {0};

for(int j = 0; j < 9; j++)

{

if(sudoMap[i][j] != 0)

{

if(tempArr[sudoMap[i][j] - 1] == 0)

{

tempArr[sudoMap[i][j] - 1] = 1;

}

else

{

return false;

}

}

}

}

//check the col

for(int j = 0; j < 9; j++)

{

int tempArr[9] = {0};

for(int i = 0; i < 9; i++)

{

if(sudoMap[i][j] != 0)

{

if(tempArr[sudoMap[i][j] - 1] == 0)

{

tempArr[sudoMap[i][j] - 1] = 1;

}

else

{

return false;

}

}

}

}

//check the 9-box

for(int m = 0; m < 9; m += 3)

{

for(int n = 0; n < 9; n += 3)

{

int tempArr[9] = {0};

for(int i = 0; i < 3; i++)

{

for(int j = 0; j < 3; j++)

{

if(sudoMap[m + i][n + j] != 0)

{

if(tempArr[sudoMap[m + i][n + j] - 1] == 0)

{

tempArr[sudoMap[m + i][n + j] - 1] = 1;

}

else

{

return false;

}

}

}

}

}

}

return true;

}

bool isMapFull()

{

int flag[9\*9] = {0};

for(int i = 0; i < 9; i++)

{

for(int j = 0; j < 9; j++)

{

if(sudoMap[i][j] != 0)

{

flag[i\*9 + j] = 1;

}

}

}

for(int i = 0; i < 81; i++)

{

if(flag[i] == 0)

{

return false;

}

}

return true;

}

bool checkIsNumberFull(int arr[9])

{

for(int i = 0; i < 9; i++)

{

if(arr[i] == 0)

{

return false;

}

}

return true;

}

int getRandNumber()

{

return rand()%9 + 1;

}

void generateMap(int curIndex)

{

if(isMapFull())

{

if(counter <= maxMapNumber)

{

cout << "SudoMap " << counter++ << endl;

printMap();

return;

}

}

srand((unsigned int)time(0));

for(int i = 1; i < 10; i++)

{

sudoMap[(int)(curIndex / 9)][curIndex % 9] = i;

if(check()) // if curNumber is legal.

{

// cout << "curLevel:" << curIndex << endl;

// printMap();

generateMap(curIndex + 1);

}

}

sudoMap[(int)(curIndex / 9)][curIndex % 9] = 0;

return;

}

void generateLine1()

{

int numberFlag[9] = {0};

int curNumber;

int counter = 0;

srand((unsigned)time(NULL));

while(!checkIsNumberFull(numberFlag))

{

curNumber = getRandNumber();

if(numberFlag[curNumber - 1] == 0)

{

numberFlag[curNumber - 1] = 1;

sudoMap[counter / 9][counter % 9] = curNumber;

counter++;

}

}

}

int main()

{

cout << "enter the number of sudoMap you want to generate: ";

cin >> maxMapNumber;

cout << "generating first line of the map..." << endl;

generateLine1();

cout << "generating the full sudoMap..." << endl;

generateMap(9);

return 0;

}

2、以宫为单位的矩阵置换法

这种方法比较巧妙，采用矩阵转换的方法实现简单，不过缺点在于不能包含所有合法的数独。下面介绍其生成的步骤：

第一步：先在第五宫填好一个满足数独要求的9个数字。

第二步：将第5宫的九宫格内容通过行变换放置到左边和右边的两个九宫格。

 第三步：将第5宫的九宫格内容通过列变换放置到上边和下边的两个九宫格。

第四步：将第4宫的九宫格内容通过列变换放置到上边和下边的两个九宫格，将第6宫的九宫格内容也通过列变换放置到上边和下边的两个九宫格。如此一来便通过变换生成了一个完整的数独地图。

3、以数字顺序的以宫为单位的回溯法

这可以说是方法一的优化版本。

方法一生成数独是按照从左到右、从上到下的顺序一格一格填下来的，不过程序运行的情况告诉我们这种方法太笨了，需要回朔的过程太多了，因此性能十分差劲。有博主提到优化的方总结如下：在生成数独的过程中我们首先填1，将9个宫按照从左到右从上到下的顺序先把9个1的位置确定，之后再按照同样的顺序填2，直至完成所有的数字。遇到不满足数独规则的情况则回朔到上一个情况。

4、以某一数独为基础的数字替换法

这种方法也是比较巧妙，不过存在一个很致命的问题：只是替换了数字，数独的基本结构不变，因此结构是固定的。比如下面这个数独：

我们将第5宫的数字依次标记如下：a(3), b(6), c(5), d(7), e(9), f(8), g(2), h(1), i(4)。通过这样的替换规则可以将上图的数独全部替换为字母a-i代替的字母数独，然后字母a可以用1-9的所有数字代替，b可以用1-9中除了a对应数字之外的8种数字代替，以此类推一共有9！中替代方法。因此通过这种替代法，可以生成9！中模式相同但数字不同的数独地图。