基于枚举搜索算法的数独问题求解

本文中，以计算机编程求解数独问题为例，为初学者演示枚举（穷举）搜索算法应用的基本思想和做法。以 MATLAB 作为程序设计语言，展示本例中使用该语言编写程序代码的一点有用技巧，给出代码的参考设计。

1. 一个六阶（6x6）数独问题

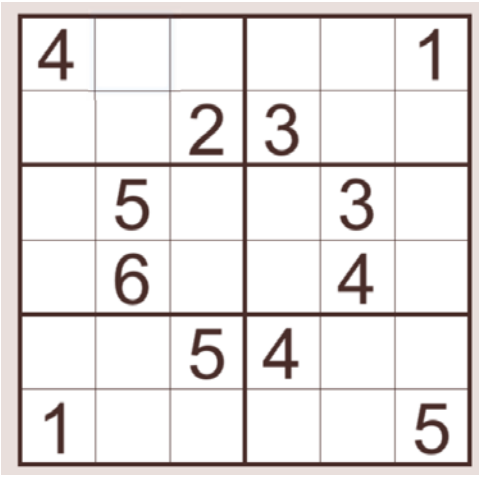

图 1 一个六阶数独

形如图 1 所示的六阶标准数独游戏，在 6x6 的方格阵列中已有若干预填已知数字。图中，每块被粗实线围在一起的六格，称为一宫。玩家需要通过逻辑推理，在空白格内正确填上 1-6 的数字，满足以下规则。

规则 1：1-6 每个数字在每一行都出现且仅出现一次；

规则 2：1-6 每个数字在每一列都出现且仅出现一次；

规则 3：1-6 每个数字在每一宫都出现且仅出现一次。

2. 求解思路

为方便说话，我们用形如(1,4)的表示方式代表第 1 行第 4 列的格子；并对该数独图形的全部 36 格进行顺序编号，(1,1)编为第 1 格，(1,2)编为第 2 格，(1,3)编为第 3 格，...，(2,1)编为第 7 格，以此类推，直到(6,6)编为第 36 格。

	1	2	3	4	5	6
1	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
2	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
3	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
4	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
5	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6

图 2 步骤 1 的标注

使用枚举的思想，我们可以通过以下迭代推理的步骤，求解上述数独问题。

【步骤 1】如图 2，在 6x6 方格阵列的每一格中标注 1、2、3、4、5、6，代表初始时该格可能的备选数字，暂时不考虑已知数字。

【步骤 2】逐个考虑预填已知数字，并相应修改标注。比如，(1,1)已知数字 4，依据数独游戏规则 1、2、3，按以下方法修改标注。

（1）(1,1)标注改为 0、0、0、4、0、0，其中 0 表示原对应位 1、2、3、5、6 不再能作为备选数字（下同）；

（2）第 1 行（同一行）各格内，4 改作 0，不再作为备选数字；

（3）第 1 列（同一列）各格内，4 改作 0，不再作为备选数字；

（4）同一宫其他各格内，4 改作 0，不再作为备选数字。

经修改后，标注如图 3 示。针对图 1 给出的其余已知数字，一一处理，修改标注后，得到图 4。

	1	2	3	4	5	6
1	0 0 0 4 0 0	1 2 3 0 5 6	1 2 3 0 5 6	1 2 3 0 5 6	1 2 3 0 5 6	1 2 3 0 5 6
2	1 2 3 0 5 6	1 2 3 0 5 6	1 2 3 0 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
3	1 2 3 0 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
4	1 2 3 0 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
5	1 2 3 0 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6
6	1 2 3 0 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6

图 3 关于(1,1)已知数字 4 的标注结果

【步骤 3】增添指针标记，在图 4 中每格首个数字有下划线标记，将用于后续枚举过程的操作。令 k=1。

	1	2	3	4	5	6
1	0 0 0 4 0 0	0 0 3 0 0 0	0 0 3 0 0 6	0 2 0 0 5 6	0 2 0 0 5 6	1 0 0 0 0 0
2	0 0 0 0 5 6	1 0 0 0 0 0	0 2 0 0 0 0	0 0 3 0 0 0	0 0 0 0 5 6	0 0 0 4 0 6
3	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 4 0 0	1 2 0 0 0 6	0 0 3 0 0 0	0 2 0 0 0 6
4	0 2 3 0 0 0	0 0 0 0 0 6	1 0 3 0 0 0	1 2 0 0 5 0	0 0 0 4 0 0	0 2 0 0 0 0
5	0 2 3 0 0 6	0 2 3 0 0 0	0 0 0 0 5 0	0 0 0 4 0 0	1 2 0 0 0 6	0 2 3 0 0 6
6	1 0 0 0 0 0	0 2 3 4 0 0	0 0 3 4 0 6	0 2 0 0 0 6	0 2 0 0 0 6	0 0 0 0 5 0

图 4 关于所有已知数字标注后的结果

【步骤 4】针对第 k 格寻找和测试一个合适备选数字，称为一轮（搜索）处理，可细分为 3 个小步骤。

【步骤 4.1】针对第 k 格，检查指针（下划线）所在位置的数值。若该数值可作为备选（不为 0），则至步骤 4.3，尝试修改并检查标注；若该数值为 0，则至步骤 4.2。

【步骤 4.2】尝试把指针标记后移一位，回到步骤 4.1，但如果不再有下一位（当前已只在第 6 数项，本格 6 个标注都已尝试）则表示需要回退一格搜索，执行步骤 5。

【步骤 4.3】保存当前全图标注状态。尝试修改标注，将第 k 格中指针所指数字之外备选数字清 0；在同行、同列、同宫其他各格内，该数字不再作为备选数字（对应标注清 0）。检查修改后的全图标注局面，若发现某格标注全 0，说明此路不通，放弃上述对标注的修改，前往步骤 4.2 继续；否则，将本次对标注的修改记录为“处理第 k 格时的修改”，前往步骤 6。

【步骤 5】（回退一格）将第 k 格指针复原指向首个数字（无论其是否为 0），令 k=k-1，如果 k 变为 0 说明本题无解，终止算法，否则继续。将全图标注恢复到“处理第 k 格时的修改”之前的状态，将第 k 格指针后移一位，转往步骤 4.1，但如果不再有下一位，则表示需要再次回退，重复执行本步骤 5。

【步骤 6】（前进一格）令 k=k+1，前往步骤 4.1，但如果已经有 k=37，说明求解已完成，停止算法，此时全图每一格都只有一个有效备选数字标注，可作为答案。

	1	2	3	4	5	6
1	0 0 0 4 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0
2	0 0 0 0 5 0	1 0 0 0 0 0	0 2 0 0 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 0 0 4 0 0
3	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 4 0 0	1 0 0 0 0 6	0 0 3 0 0 0	0 0 0 0 0 6
4	0 0 3 0 0 0	0 0 0 0 0 6	1 0 3 0 0 0	1 0 0 0 5 0	0 0 0 4 0 0	0 2 0 0 0 0
5	0 0 3 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	0 0 0 4 0 0	1 2 0 0 0 6	0 2 3 0 0 6
6	1 0 0 0 0 0	0 2 0 4 0 0	0 0 3 4 0 6	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0

图 5 例 1 第 14 格处理后第 15 格处理前

	1	2	3	4	5	6
1	0 0 0 4 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0
2	0 0 0 0 5 0	1 0 0 0 0 0	0 2 0 0 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 0 0 4 0 0
3	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0	0 0 0 0 0 6	0 0 3 0 0 0	0 0 0 0 0 6
4	0 0 3 0 0 0	0 0 0 0 0 6	0 0 3 0 0 0	1 0 0 0 5 0	0 0 0 4 0 0	0 2 0 0 0 0
5	0 0 3 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	0 0 0 4 0 0	1 2 0 0 0 6	0 2 3 0 0 6
6	1 0 0 0 0 0	0 2 0 4 0 0	0 0 3 4 0 6	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0

图 6 例 1 第 15 格尝试取 1 进入第 16 格

为更加清楚地说明算法的操作，以下举例。

【例 1】前进一格的操作。

图 5 是第 14 格即(3,2)已处理后开始处理第 15 格时的标注状态，保存好此时的全图标注情况。为方便说明，加深了(3,3)的底色。此时，第 15 格指针位于首个数字 1，所以选择它作为尝试。(3,3)

中除 1 之外，全部清 0；同行、同列、同宫的其他格，1 不再作为备选，(4,3)、(3,4)因此受到实质性影响。全图标注状态如图 6，经检查未发现有任何一格标注变为全 0 的情况。因此，前进一格处理第 16 格。

	1	2	3	4	5	6
1	0 0 0 4 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0
2	0 0 0 0 5 0	1 0 0 0 0 0	0 2 0 0 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 0 0 4 0 0
3	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0	0 0 0 0 0 6	0 0 3 0 0 0	0 0 0 0 0 0
4	0 0 3 0 0 0	0 0 0 0 0 6	0 0 3 0 0 0	1 0 0 0 5 0	0 0 0 4 0 0	0 2 0 0 0 0
5	0 0 3 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	0 0 0 4 0 0	1 2 0 0 0 6	0 2 3 0 0 6
6	1 0 0 0 0 0	0 2 0 4 0 0	0 0 3 4 0 6	0 0 0 0 0 0	0 2 0 0 0 0	0 0 0 0 5 0

图 7 例 2 若第 16 格取 6 发现行不通

	1	2	3	4	5	6
1	0 0 0 4 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 0 0 0
2	0 0 0 0 5 0	1 0 0 0 0 0	0 2 0 0 0 0	0 0 3 0 0 0	0 0 0 0 0 6	0 0 0 4 0 0
3	0 2 0 0 0 0	0 0 0 0 5 0	1 0 0 4 0 0	1 0 0 0 0 6	0 0 3 0 0 0	0 0 0 0 0 6
4	0 0 3 0 0 0	0 0 0 0 0 6	1 0 3 0 0 0	1 0 0 0 5 0	0 0 0 4 0 0	0 2 0 0 0 0
5	0 0 3 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0	0 0 0 4 0 0	1 2 0 0 0 6	0 2 3 0 0 6
6	1 0 0 0 0 0	0 2 0 4 0 0	0 0 3 4 0 6	0 0 0 0 0 6	0 2 0 0 0 0	0 0 0 0 5 0

图 8 例 2 从第 16 格退回到第 15 格的标注状态

【例 2】回退一格的操作。

图 6 是第 15 格即(3,3)经处理后开始处理第 16 格时的标注状态，保存好此时的全图标注情况。此时，第 16 格指针位于首个数字 0，无法作为备选，经多次后移指针搜索，确定数字 6 作为尝试。(3,4)中除 6 之外，已全部为 0；同行、同列、同宫的其他格，6 不再作为备选，(3,6)、(6,4)因此受到实质性影响。全图标注状态如图 7，经检查发现(3,6)、(6,4)标注出现全 0 的情况，说明此路不通，需回退一格继续搜索。将全图标注还原成第 15 格处理前状态，如图 8，请注意看，它与图 5 的差异仅在于第 15 格指针已后移到第 2 位的 0。

3. 代码的数据结构设计

根据上述对算法的描述，设计相应的数据结构以便开展程序设计。表 1 展示了主要的变量设置及说明。为更好地解释部分变量的作用，举例说明。

表 1 主要的变量设置和说明

变量名	作用	下标说明
cur_mark(6,6,6)	三维数组。记录当前全图标注状态。	每个元素对应数独图上一个标注数字。三个维度的下标分别对应行坐标、列坐标、数项。
ptrs(36)	一维数组。记录每格中指针所对位置。	下标代表格子的编号。
diff_mark(6,6,6,36)	四维数组。记录搜索过程中关键步骤的标注变化，在发生回退搜索操作时需要这些记录来恢复标注场景。	第四维下标，对应记录发生时正在开始处理的格子编号。前三维下标分别对应行坐标、列坐标、数项。
groups(6,2,6)	规定哪些格组成一宫。	第三维下标表示第几宫。前二维形成格子的行列坐标列表。
cell	表示当前正在处理的格子的编号。	--

【例 3】关于 cur_mark，比如针对图 4 中(6,3)的标注状况，相应有

cur_mark(6,3,1) = 0
cur_mark(6,3,2) = 0
cur_mark(6,3,3) = 1
cur_mark(6,3,4) = 1
cur_mark(6,3,5) = 0
cur_mark(6,3,6) = 1

【例 4】关于 groups，比如针对第一宫由(1,1)、(1,2)、(1,3)、(2,1)、(2,2)、(2,3)六格组成，相应有

groups(1,1,1) = 1	groups(1,2,1) = 1
groups(2,1,1) = 1	groups(2,2,1) = 2
groups(3,1,1) = 1	groups(3,2,1) = 3
groups(4,1,1) = 2	groups(4,2,1) = 1
groups(5,1,1) = 2	groups(5,2,1) = 2
groups(6,1,1) = 2	groups(6,2,1) = 3

4. 程序代码的简要说明

基于以上算法步骤设计和数据结构设计，作为程序设计示例，本文给出 MATLAB 程序源代码，共包含 7 个程序文件，列表 2 说明。

表 2 程序文件及其说明

文件名	说明
refresh_mark	M 函数。当指定格子选定数项后，依据数独游戏规则 1、2、3 对全图标注进行修改，并返回修改结果。
check_mark	M 函数。检查全图标注，如果有任意一格 6 项标注全 0，返回结果 0；否则返回结果 1。
init_data	M 文件。主要变量的初始化赋值。
oneround	M 文件。一轮处理。按照既定算法步骤，对当前格进行处理，根据处理情况决定前进一格还是回退一格。
print_result	M 文件。屏幕打印经求解得到的有效答案。
go	M 文件。顶层程序。

print_mark	M 文件。一段独立程序，作为调试工具。以较接近图 3 的形式屏幕打印给出当前的全图标注，仅用于程序编写过程中的检查调试。

除了 print_mark 之外，其余 6 个程序文件间逻辑层次和调用关系如图 9 所示，图 10 是程序整体流程。

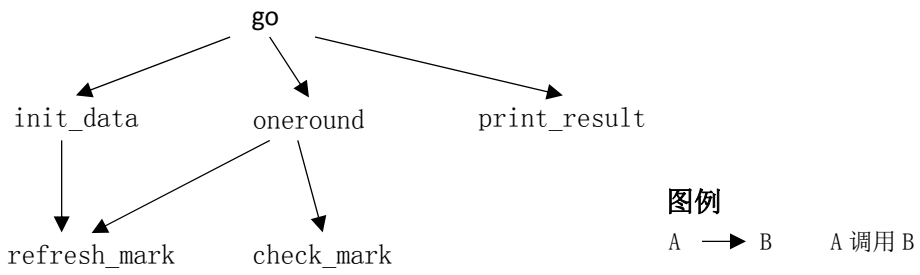

图 9 程序文件的逻辑层次和调用关系

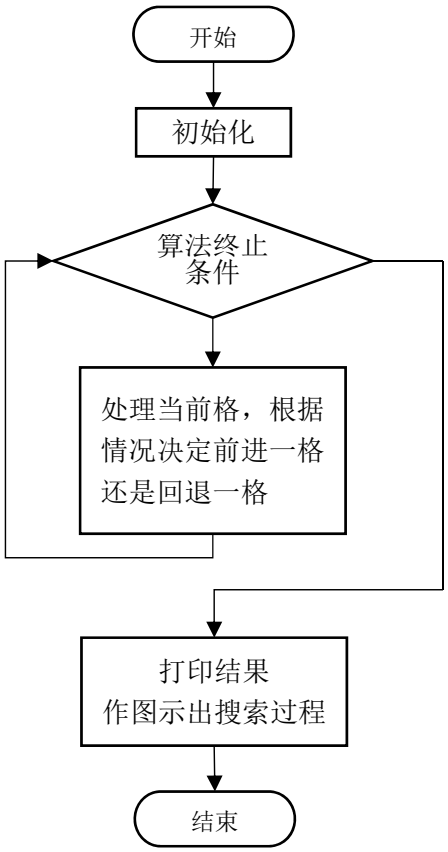

图 10 程序流程

5. 书写简洁代码的几点技巧

灵活运用 MATLAB 的指令和标准函数，可以有效简化代码的书写，以下结合示例程序举例。

- **灵活使用通配符**

MATLAB 中，对数组操作时可以用通配符。比如示例程序中，M 函数 refresh_mark 内将(x,y)格中

所有数项标注置 0，使用了带通配符的指令写作

```
next_mark(x,y,:) = 0;
```

如果使用一般程序语言常规风格可能比较繁复，比如要使用循环语句写作

```
for i=1:6
   next_mark(x,y,i)=0;
end
```

类似，当(x,y)选定数字 ptr，则同一行和同一列的数项 ptr 标注要清零，可以简洁地写作

```
next_mark(x,:,ptr) =0;
next_mark(:,y,ptr) =0;
```

- **灵活使用标准函数**

假定三维数组 mark(6,6,6)用于记录全图标注状态。要检查是否存在任一格子 6 项标注全为 0 的情况，初学者通常会使用这样风格的写法

```
all_zeros=0;
for i=1:6
    for j=1:6
       c=0;
       for k=1:6
          if mark(i,j,k)==0
             c=c+1;
          end
       end
       if c==6
           all_zeros=1;
       end
    end
end
```

运行结果若 all_zeros 为 1，则说明至少有一格 6 项标注全 0。但其实可以使用类似这样的逻辑判别

```
min(min(sum(mark,3))) == 0
```

若为真，说明至少有一格 6 项标注全 0。其中 sum 是数组求和函数，min 是数组最小值函数，两者的详细用法说明请参考 MATLAB 帮助文档。

6. 求解结果

图 1 的 6 阶数独问题，经上述枚举搜索算法求解，可得到图 11 展示的答案。图 12 是该例求解中迭代搜索的过程示意，可见共经历了 38 轮迭代处理，其间只发生过一次回退。

4	3	6	2	5	1
5	1	2	3	6	4
2	5	4	1	3	6
3	6	1	5	4	2
6	2	5	4	1	3
1	4	3	6	2	5

图 11 答案

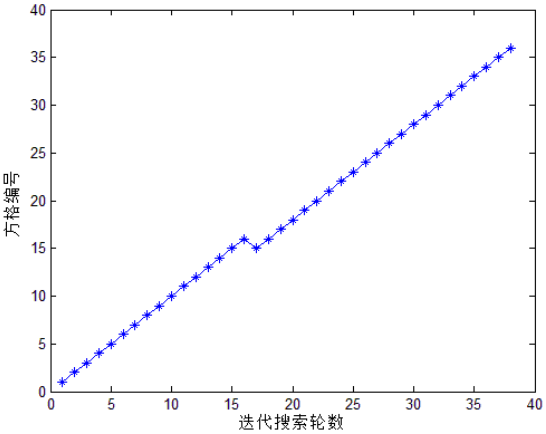

图 12 迭代搜索的过程

7. 练习题

以上讨论了一个 6 阶数独问题的枚举法搜索求解算法及其 MATLAB 编程实现。请考虑

（1） 如何将该算法扩展为可以求解更高阶数独问题，比如形如图 13 和 14 的数独问题？

（2） 如何优化算法，提高求解效率？

请开展编程实验，用实验结果验证你的设计。

		7				2		
				3				
2			6	9	5			7
		5				7		
	9	4		8		5	2	
		8				3		
4			9	1	7			6
				5				
		3				1		

图 13 9 阶数独问题 1

		7		2				
			3			4		
9				1			2	
	1				2			
7		8				3		9
			4				7	
	5			7				8
		3			6			
				3		1		

图 14 9 阶数独问题 2

8. 附录：示例程序源代码

➢ **M 文件 go.m**

```
init_data;
tic;

cell=1;
while (cell<=Order*Order && cell>0)
    oneround;
end

if cell==Order*Order+1
    fprintf('\nIt took %6.2f s.\n',toc);
```

```
    fprintf('The answer is:\n');
    print_result;
elseif cell==0;
    fprintf('\nThe puzzle has no answer!\n');
end

plot([1:cell_record_ptr-1],cell_record(1:cell_record_ptr-1),'-*');
```

- **M 文件 init_data.m**

```
%数独游戏阶数
Order=6;

%mark 表格，下标含义：行坐标，列坐标，数项
cur_mark=ones(Order,Order,Order);

%记录每一步的 mark 表格变化
%%第 4 维下标代表第几格（从 1 到 Order*Order 对应从左到右，逐行向下）
diff_mark=zeros(Order,Order,Order,Order*Order);

%数项选择指针
%%下标表示第几格，数值代表下一轮将对应的数项
ptrs=ones(1,Order*Order);

%数格成组（哪几个组成一宫）定义
%%第 3 维下标表示第几组（宫）
%%数值含义：行坐标，列坐标
groups=zeros(Order,2,6);
groups(:,:,1)=[1 1; 1 2; 1 3; 2 1; 2 2; 2 3];
groups(:,:,2)=[1 4; 1 5; 1 6; 2 4; 2 5; 2 6];
groups(:,:,3)=[3 1; 3 2; 3 3; 4 1; 4 2; 4 3];
groups(:,:,4)=[3 4; 3 5; 3 6; 4 4; 4 5; 4 6];
groups(:,:,5)=[5 1; 5 2; 5 3; 6 1; 6 2; 6 3];
groups(:,:,6)=[5 4; 5 5; 5 6; 6 4; 6 5; 6 6];

%预先已填的数字
%%数值含义：行坐标，列坐标，数项
init_digit=[1 1 4; 1 6 1; 2 3 2; 2 4 3; 3 2 5; 3 5 3;
    4 2 6; 4 5 4; 5 3 5; 5 4 4; 6 1 1; 6 6 5];
for i=1:size(init_digit,1)
    cur_mark=refresh_mark(groups,cur_mark,init_digit(i,1),init_digit(i,2),init_digit(i,3));
end

%记录搜索过程,预开足够多个记录单元
cell_record=zeros(1,1000);
```

```
cell_record_ptr=1;
```

- **M 文件 oneround.m**

```
fprintf('cell=%d\n',cell);
cell_record(cell_record_ptr)=cell;
cell_record_ptr=cell_record_ptr+1;

xcell=ceil(cell/Order); ycell=mod(cell-1,Order)+1;

goto_nextcell=0;
while((goto_nextcell==0) && (ptrs(cell)<=Order))
    if cur_mark(xcell,ycell,ptrs(cell))==0
        ptrs(cell)=ptrs(cell)+1;
    else
        next_mark=refresh_mark(groups,cur_mark,xcell,ycell,ptrs(cell));
        if check_mark(next_mark)==1
            diff_mark(:,:,:,cell)=next_mark-cur_mark;
            cur_mark=next_mark;
            cell=cell+1;
            goto_nextcell=1;
        else
            ptrs(cell)=ptrs(cell)+1;
        end
    end
end

if goto_nextcell==0
    ptrs(cell)=1;
    cell=cell-1;
    if cell~=0
        cur_mark=cur_mark-diff_mark(:,:,:,cell);
        ptrs(cell)=ptrs(cell)+1;
    end
end
```

- **M 文件 print_result.m**

```
for i=1:6
    for j=1:6
        fprintf([num2str(reshape(cur_mark(i,j,:),1,[])*[1:6]','%1d'),'    ']);
    end
    fprintf('\n');
end
```

- **M 函数 refresh_mark.m**

```
function next_mark = refresh_mark( groups,cur_mark,x,y,ptr )
%MARKING 输入参数含义：组的定义，cur_mark，行坐标，列坐标，数项
%     当行、列坐标指定的格子内填写数项 ptr 后，对 mark 表格进行更新
Order=size(cur_mark,1);
next_mark =cur_mark;

next_mark(x,y,:)=0;       %同一格各项 mark 清零
next_mark(x,:,ptr)=0;    %同一列同项 mark 清零
next_mark(:,y,ptr)=0;    %同一列同项 mark 清零

%同一组的同项 mark 清零
for g=1:size(groups,3)
      found=0;
      for i=1:Order
            if groups(i,:,g)==[x,y]
                  found=1;
            end
      end
      if found==1
            for i=1:Order
                  next_mark(groups(i,1,g),groups(i,2,g),ptr)=0;
            end
      end
end

next_mark(x,y,ptr)=1;
end
```

- **M 函数 check_mark.m**

```
function output = check_mark( mark )
% 输入参数含义：待检查的 mark 表格
%     当发现某一格子的所有数项的 mark 均为零，说明此路行不通，输出返回值零

if min(min(sum(mark,3)))==0
      output=0;   %某一格 mark 全零，已无可选项，提示此路不通
else
      output=1;   %提示未发现有行不通的迹象
end

end
```

- **M 文件 print_mark.m**

```
for i=1:6
      for j=1:6
            fprintf([num2str(reshape(cur_mark(i,j,1:3),1,[]).*[1 2 3],'%1d'),'      ']);
```

```
        end
        fprintf('\n');
        for j=1:6
            fprintf([num2str(reshape(cur_mark(i,j,4:6),1,[]).*[4 5 6],'%1d'),'      ']);
        end
        fprintf('\n');
        fprintf('\n');
end
```