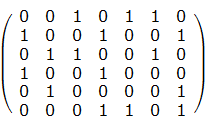
# ****舞蹈链****

**http://www.cnblogs.com/grenet/p/3145800.html**

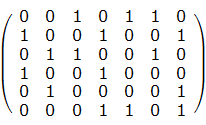
**精确覆盖问题的定义**：给定一个由0-1组成的矩阵，是否能找到一个行的集合，使得集合中每一列都恰好包含一个1

例如：如下的矩阵

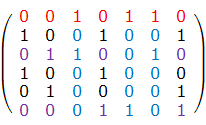
[](http://images.cnitblog.com/blog/93227/201306/20141205-78f0147dae6940b8a2c72d8928fcc806.png)

就包含了这样一个集合（第1、4、5行）

如何利用给定的矩阵求出相应的行的集合呢？我们采用回溯法

矩阵1：[](http://images.cnitblog.com/blog/93227/201306/21074621-b280963e9390402fb0ade0ab7e1695c7.png)

先假定选择第1行，如下所示：

[](http://images.cnitblog.com/blog/93227/201306/20141206-17f0946ed79d46709f39e72eb5eb6ab1.png)

如上图中所示，红色的那行是选中的一行，这一行中有3个1，分别是第3、5、6列。

由于这3列已经包含了1，故，把这三列往下标示，图中的蓝色部分。蓝色部分包含3个1，分别在2行中，把这2行用紫色标示出来

根据定义，同一列的1只能有1个，故紫色的两行，和红色的一行的1相冲突。

那么在接下来的求解中，红色的部分、蓝色的部分、紫色的部分都不能用了，把这些部分都删除，得到一个新的矩阵

矩阵2：[clip_image002[6]](http://images.cnitblog.com/blog/93227/201306/20141207-ec8253ab414b4034a1c85af7c69f2435.png)

行分别对应矩阵1中的第2、4、5行

列分别对应矩阵1中的第1、2、4、7列

于是问题就转换为一个规模小点的精确覆盖问题

在新的矩阵中再选择第1行，如下图所示

[clip_image002[8]](http://images.cnitblog.com/blog/93227/201306/20144122-57be909761f64d1dabf84884bf409d39.png)

还是按照之前的步骤，进行标示。红色、蓝色和紫色的部分又全都删除，导致新的空矩阵产生，而红色的一行中有0（有0就说明这一列没有1覆盖）。说明，第1行选择是错误的

那么回到之前，选择第2行，如下图所示

[clip_image002[10]](http://images.cnitblog.com/blog/93227/201306/20144122-bc9faed752664c0281c2ff34856b2d05.png)

按照之前的步骤，进行标示。把红色、蓝色、紫色部分删除后，得到新的矩阵

矩阵3：[clip_image002[12]](http://images.cnitblog.com/blog/93227/201306/20144123-eddc2778fa1b46319b92b686c6928bbf.png)

行对应矩阵2中的第3行，矩阵1中的第5行

列对应矩阵2中的第2、4列，矩阵1中的第2、7列

由于剩下的矩阵只有1行，且都是1，选择这一行，问题就解决

于是该问题的解就是矩阵1中第1行、矩阵2中的第2行、矩阵3中的第1行。也就是矩阵1中的第1、4、5行

在求解这个问题的过程中，我们第1步选择第1行是正确的，但是不是每个题目第1步选择都是正确的，如果选择第1行无法求解出结果出来，那么就要推倒之前的选择，从选择第2行开始，以此类推

从上面的求解过程来看，实际上求解过程可以如下表示

1、从矩阵中选择一行

2、根据定义，标示矩阵中其他行的元素

3、删除相关行和列的元素，得到新矩阵

4、如果新矩阵是空矩阵，并且之前的一行都是1，那么求解结束，跳转到6；新矩阵不是空矩阵，继续求解，跳转到1；新矩阵是空矩阵，之前的一行中有0，跳转到5

5、说明之前的选择有误，回溯到之前的一个矩阵，跳转到1；如果没有矩阵可以回溯，说明该问题无解，跳转到7

6、求解结束，把结果输出

7、求解结束，输出无解消息

Dancing Links的核心是基于双向链的方便操作（移除、恢复加入）

我们用例子来说明

假设双向链的三个连续的元素，A1、A2、A3，每个元素有两个分量Left和Right，分别指向左边和右边的元素。由定义可知

A1.Right=A2，A2.Right=A3

A2.Left=A1，A3.Left=A2

在这个双向链中，可以由任一个元素得到其他两个元素，A1.Right.Right=A3，A3.Left.Left=A1等等

现在把A2这个元素从双向链中移除（不是删除）出去，那么执行下面的操作就可以了

A1.Right=A3，A3.Left=A1

那么就直接连接起A1和A3。A2从双向链中移除出去了。但仅仅是从双向链中移除了，A2这个实体还在，并没有删除。只是在双向链中遍历的话，遍历不到A2了。

那么A2这个实体中的两个分量Left和Right指向谁？由于实体还在，而且没有修改A2分量的操作，那么A2的两个分量指向没有发生变化，也就是在移除前的指向。即A2.Left=A1和A2.Right=A3

如果此时发现，需要把A2这个元素重新加入到双向链中的原来的位置，也就是A1和A3的中间。由于A2的两个分量没有发生变化，仍然指向A1和A3。那么只要修改A1的Right分量和A3的Left就行了。也就是下面的操作

A1.Right=A2，A3.Left=A2

**即以列为单位进行删除，如果是普通点单个点则左右的连接不删除，删除上下链接。如果是要删除一整列，则该列的列头删除左右链接，而保留上下链接，且该列其他的点也保存上下链接。这样一来方便恢复。**

仔细想想，上面两个操作（移除和恢复加入）对应了什么？是不是对应了之前的算法过程中的关键的两步？

移除操作对应着缓存数据、恢复加入操作对应着回溯数据。而美妙的是，这两个操作不再占用新的空间，时间上也是极快速的

在很多实际运用中，把双向链的首尾相连，构成循环双向链

Dancing Links用的数据结构是**交叉十字循环双向链**

而Dancing Links中的每个元素不仅是横向循环双向链中的一份子，又是纵向循环双向链的一份子。

因为精确覆盖问题的矩阵往往是稀疏矩阵（矩阵中，0的个数多于1），Dancing Links仅仅记录矩阵中值是1的元素。

Dancing Links中的每个元素有6个分量

分别：Left指向左边的元素、Right指向右边的元素、Up指向上边的元素、Down指向下边的元素、Col指向列标元素、Row指示当前元素所在的行

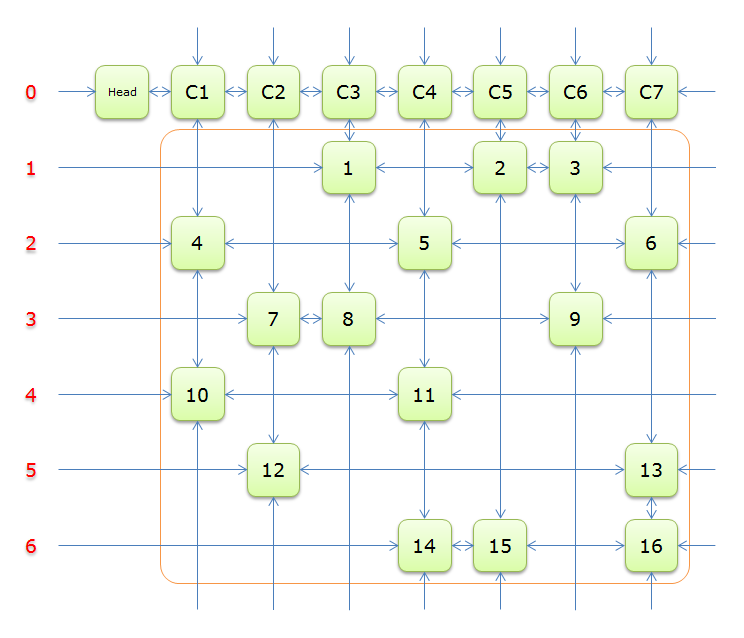
Dancing Links还要准备一些辅助元素（为什么需要这些辅助元素？没有太多的道理，大师认为这能解决问题，实际上是解决了问题）

Ans（）：Ans数组，在求解的过程中保留当前的答案，以供最后输出答案用。

Head元素：求解的辅助元素，在求解的过程中，当判断出Head.Right=Head（也可以是Head.Left=Head）时，求解结束，输出答案。Head元素只有两个分量有用。其余的分量对求解没啥用

C元素：辅助元素，称列标元素，每列有一个列标元素。本文开始的题目的列标元素分别是C1、C2、C3、C4、C5、C6、C7。每一列的元素的Col分量都指向所在列的列标元素。列标元素的Col分量指向自己（也可以是没有）。在初始化的状态下，Head.Right=C1、C1.Right=C2、……、C7.Right=Head、Head.Left=C7等等。列标元素的分量Row=0，表示是处在第0行。

下图就是根据题目构建好的**交叉十字循环双向链**（构建的过程后面的详述）

[](http://images.cnitblog.com/blog/93227/201306/29144604-86559ddc694840869aebb129c2874afb.png)

就上图解释一下

每个绿色方块是一个元素，其中Head和C1、C2、……、C7是辅助元素。橙色框中的元素是原矩阵中1的元素，给他们标上号（从1到16）

左侧的红色，标示的是行号，辅助元素所在的行是0行，其余元素所在的行从1到6

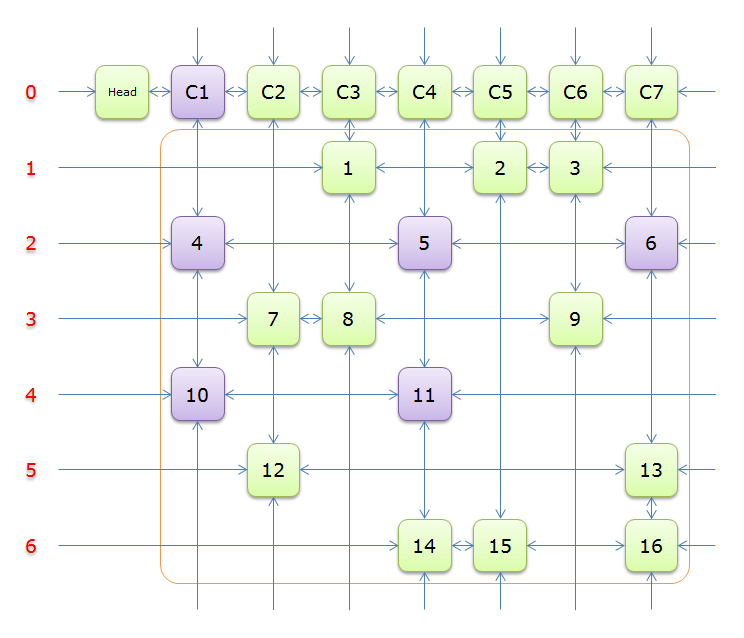
每两个元素之间有一个双向箭头连线，表示双向链中相邻两个元素的关系（水平的是左右关系、垂直的是上下关系）

单向的箭头并不是表示单向关系，而因为是循环双向链，左侧的单向箭头和右侧的单向箭头（上边的和下边的）组成了一个双向箭头，例如元素14左侧的单向箭头和元素16右侧的单项箭头组成一个双向箭头，表示14.Left=16、16.Right=14；同理，元素14下边的单项箭头和元素C4上边的单向箭头组成一个双向箭头，表示14.Down=C4、C4.Up=14

接下来，利用图来解释Dancing Links是如何求解精确覆盖问题

1、首先判断Head.Right=Head？若是，求解结束，输出解；若不是，求解还没结束，到步骤2（也可以判断Head.Left=Head？）

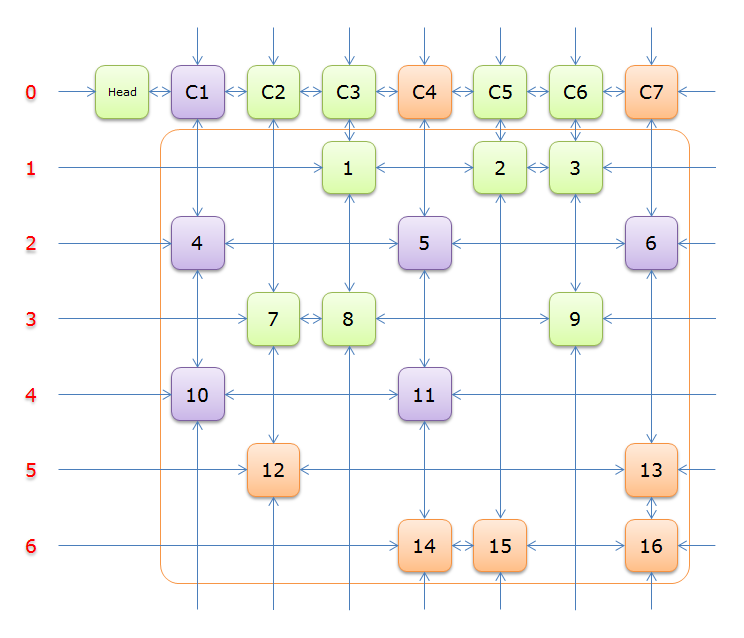
2、获取Head.Right元素，即元素C1，并**标示元素C1**（**标示元素C1**，指的是标示C1、和C1所在列的所有元素、以及该元素所在行的元素，并从双向链中移除这些元素）。如下图中的紫色部分。

[](http://images.cnitblog.com/blog/93227/201306/29191205-b1a6267f14f448918334eceda468139e.png)

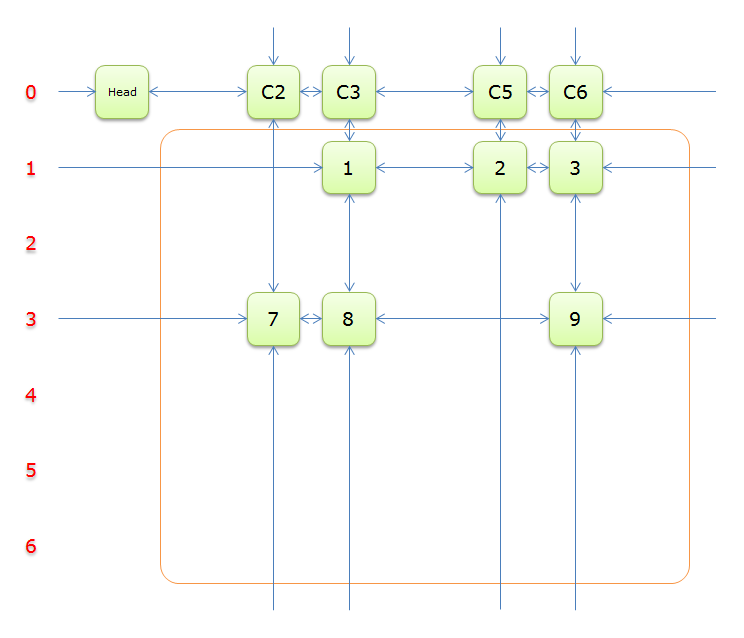
如上图可知，行2和行4中的一个必是答案的一部分（其他行中没有元素能覆盖列C1），先假设选择的是行2

3、选择行2（在答案栈中压入2），标示该行中的其他元素（元素5和元素6）所在的列首元素，即**标示元素C4**和**标示元素C7**，下图中的橙色部分。

注意的是，即使元素5在步骤2中就从双向链中移除，但是元素5的Col分量还是指向元素C4的，这里体现了双向链的强大作用。

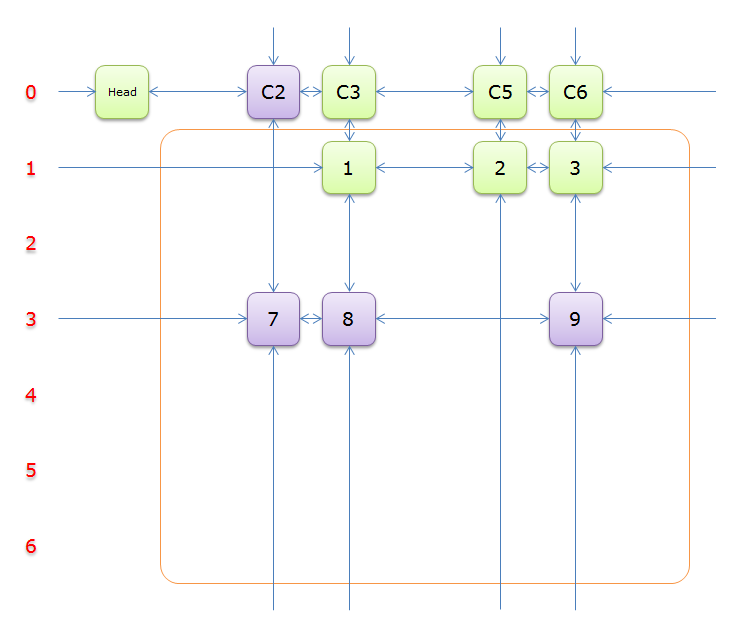
[](http://images.cnitblog.com/blog/93227/201306/29191217-3d619d4ed8014755be9be60067822d5b.png)

把上图中的紫色部分和橙色部分移除的话，剩下的绿色部分就如下图所示

[](http://images.cnitblog.com/blog/93227/201306/29200540-37cc75d6ef9b408d8718ff401e85d32b.png)

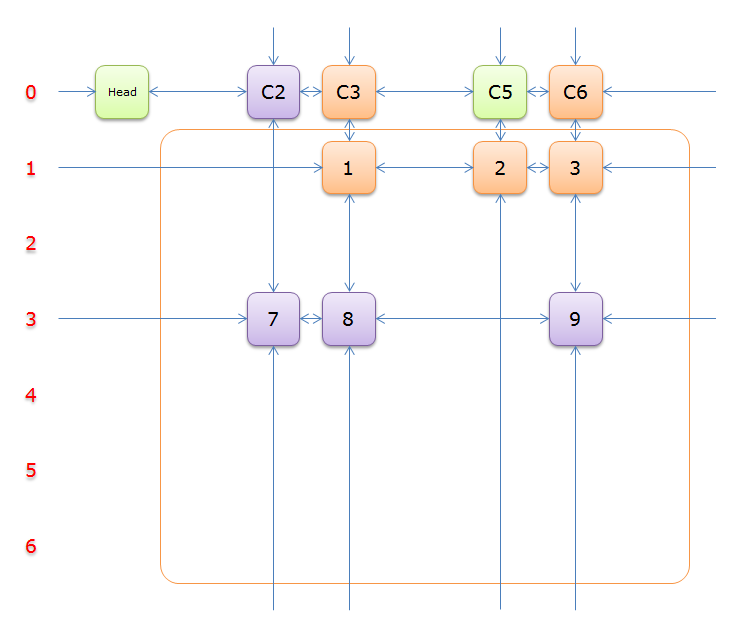
一下子空了好多，是不是转换为一个少了很多元素的精确覆盖问题？，利用递归的思想，很快就能写出求解的过程来。我们继续完成求解过程

4、获取Head.Right元素，即元素C2，并**标示元素C2**。如下图中的紫色部分。

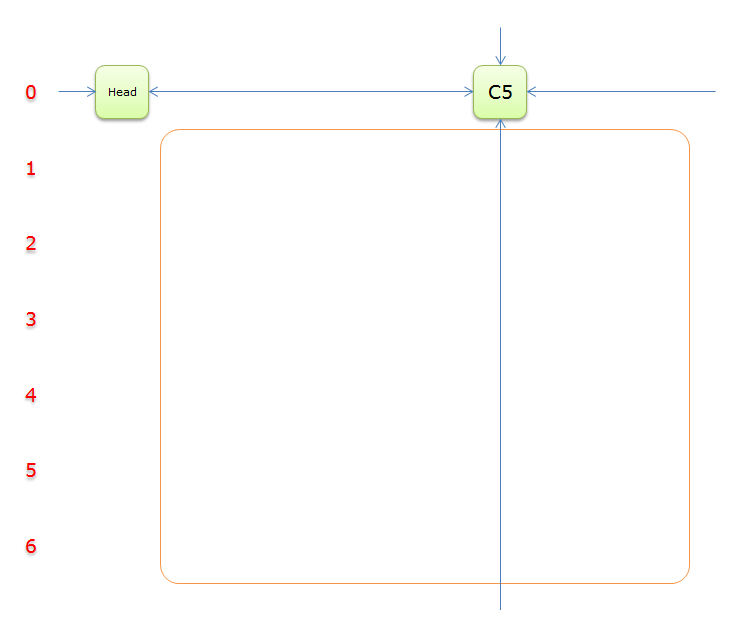
[](http://images.cnitblog.com/blog/93227/201306/29203052-0745493325a444a08462a942cb3e75d0.png)

如图，列C2只有元素7覆盖，故答案只能选择行3

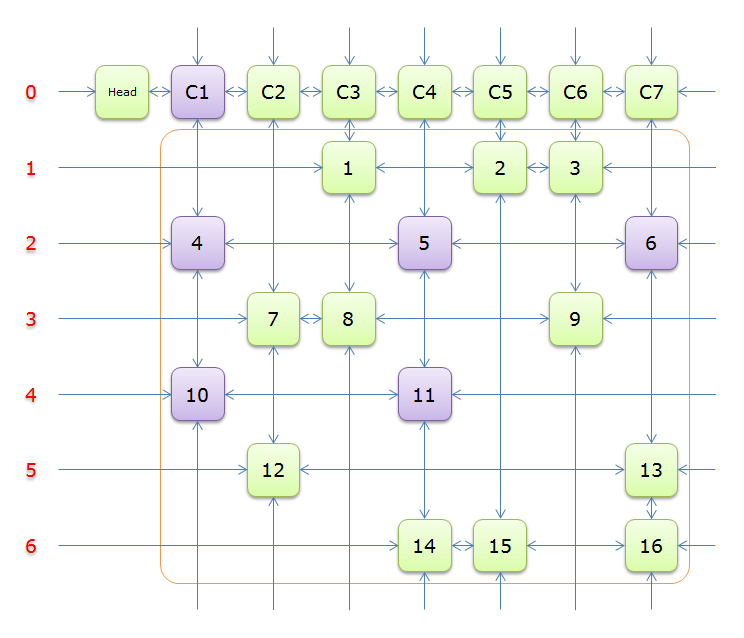
5、选择行3（在答案栈中压入3），标示该行中的其他元素（元素8和元素9）所在的列首元素，即**标示元素C3**和**标示元素C6**，下图中的橙色部分。

[](http://images.cnitblog.com/blog/93227/201306/29203128-9cc10a73ea3d4fa4a8afa47c54ee913f.png)

把上图中的紫色部分和橙色部分移除的话，剩下的绿色部分就如下图所示

[](http://images.cnitblog.com/blog/93227/201306/29203132-a0192b4014e64ee2b0b0b87229759e07.png)

6、获取Head.Right元素，即元素C5，元素C5中的垂直双向链中没有其他元素，也就是没有元素覆盖列C5。说明当前求解失败。要回溯到之前的分叉选择步骤（步骤2）。那要**回标列首元素**（把列首元素、所在列的元素，以及对应行其余的元素。并恢复这些元素到双向链中），**回标列首元素**的顺序是**标示元素**的顺序的反过来。从前文可知，顺序是**回标列首C6**、**回标列首C3**、**回标列首C2**、**回标列首C7**、**回标列首C4**。表面上看起来比较复杂，实际上利用递归，是一件很简单的事。并把答案栈恢复到步骤2（清空的状态）的时候。又回到下图所示

[](http://images.cnitblog.com/blog/93227/201306/29203137-cf812112bef841778df91d379b13e146.png)

7、由于之前选择行2导致无解，因此这次选择行4（再无解就整个问题就无解了）。选择行4（在答案栈中压入4），标示该行中的其他元素（元素11）所在的列首元素，即**标示元素C4**，下图中的橙色部分。再继续递归求解。

本文介绍该算法的实际运用，利用舞蹈链（Dancing Links）算法求解数独

在前文中可知，舞蹈链（Dancing Links）算法在求解精确覆盖问题时效率惊人。

那利用舞蹈链（Dancing Links）算法求解数独问题，实际上就是下面一个流程

1、把数独问题转换为精确覆盖问题

2、设计出数据矩阵

3、用舞蹈链（Dancing Links）算法求解该精确覆盖问题

4、把该精确覆盖问题的解转换为数独的解

首先看看数独问题（9\*9的方格）的规则

1、每个格子只能填一个数字

2、每行每个数字只能填一遍

3、每列每个数字只能填一遍

4、每宫每个数字只能填一遍（宫的概念，参看“[算法实践——数独的基本解法](http://www.cnblogs.com/grenet/p/3138654.html)”）

那现在就是利用这个规则把数独问题转换为精确覆盖问题

可是，直观上面的规则，发现比较难以转换为精确覆盖问题。因此，把上面的表述换个说法

1、每个格子只能填一个数字

2、每行1-9的这9个数字都得填一遍（也就意味着每个数字只能填一遍）

3、每列1-9的这9个数字都得填一遍

4、每宫1-9的这9个数字都得填一遍

这样理解的话，数独问题转换为精确覆盖问题就相对简单多了。关键就是如何构造精确覆盖问题中的矩阵

我们把矩阵的每个列都定义成一个约束条件。

第1列定义成：（1，1）填了一个数字

第2列定义成：（1，2）填了一个数字

……

第9列定义成：（1，9）填了一个数字

第10列定义成：（2，1）填了一个数字

……

第18列定义成：（2，9）填了一个数字

……

第81列定义成：（9，9）填了一个数字

至此，用第1-81列完成了**约束条件1：每个格子只能填一个数字**

第N列（1≤N≤81）定义成：（X，Y）填了一个数字。

N、X、Y之间的关系是：X=INT（（N-1）/9）+1；Y=（（N-1） Mod 9）+1；N=（X-1）×9+Y

第82列定义成：在第1行填了数字1

第83列定义成：在第1行填了数字2

……

第90列定义成：在第1行填了数字9

第91列定义成：在第2行填了数字1

……

第99列定义成：在第2行填了数字9

……

第162列定义成：在第9行填了数字9

至此，用第82-162列（共81列）完成了**约束条件2：每行1-9的这9个数字都得填一遍**

第N列（82≤N≤162）定义成：在第X行填了数字Y。

N、X、Y之间的关系是：X=INT（（N-81-1）/9）+1；Y=（（N-81-1） Mod 9）+1；N=（X-1）×9+Y+81

第163列定义成：在第1列填了数字1

第164列定义成：在第1列填了数字2

……

第171列定义成：在第1列填了数字9

第172列定义成：在第2列填了数字1

……

第180列定义成：在第2列填了数字9

……

第243列定义成：在第9列填了数字9

至此，用第163-243列（共81列）完成了**约束条件3：每列1-9的这9个数字都得填一遍**

第N列（163≤N≤243）定义成：在第X列填了数字Y。

N、X、Y之间的关系是：X=INT（（N-162-1）/9）+1；Y=（（N-162-1） Mod 9）+1；N=（X-1）×9+Y+162

第244列定义成：在第1宫填了数字1

第245列定义成：在第1宫填了数字2

……

第252列定义成：在第1宫填了数字9

第253列定义成：在第2宫填了数字1

……

第261列定义成：在第2宫填了数字9

……

第324列定义成：在第9宫填了数字9

至此，用第244-324列（共81列）完成了**约束条件4：每宫1-9的这9个数字都得填一遍**

第N列（244≤N≤324）定义成：在第X宫填了数字Y。

N、X、Y之间的关系是：X=INT（（N-243-1）/9）+1；Y=（（N-243-1） Mod 9）+1；N=（X-1）×9+Y+243

至此，用了324列完成了数独的四个约束条件，矩阵的列定义完成

那接下来，就是把数独转换为矩阵

数独问题中，每个格子分两种情况。有数字的格子、没数字的格子。

**有数字的格子**

以例子来说明，在（4，2）中填的是7

把（4，2）中填的是7，解释成4个约束条件

1、在（4，2）中填了一个数字。

2、在第4行填了数字7

3、在第2列填了数字7

4、在第4宫填了数字7（坐标（X，Y）到宫N的公式为：N=INT（（X-1）/3）×3+INT（（Y-1）/3）+1）

那么这4个条件，分别转换成矩阵对应的列为

1、在（4，2）中填了一个数字。对应的列N=（**4**-1）×9+**2**=29

2、在第4行填了数字7。对应的列N=（**4**-1）×9+**7**+81=115

3、在第2列填了数字7。对应的列N=（**2**-1）×9+**7**+162=178

4、在第4宫填了数字7。对应的列N=（**4**-1）×9+**7**+243=277

于是，（4，2）中填的是7，转成矩阵的一行就是，第29、115、178、277列是1，其余列是0。把这1行插入到矩阵中去。

**没数字的格子**

还是举例说明，在（5，8）中没有数字

把（5，8）中没有数字转换成

（5，8）中填的是1，转成矩阵的一行就是，第44、118、226、289列是1，其余列是0。

（5，8）中填的是2，转成矩阵的一行就是，第44、119、227、290列是1，其余列是0。

（5，8）中填的是3，转成矩阵的一行就是，第44、120、228、291列是1，其余列是0。

（5，8）中填的是4，转成矩阵的一行就是，第44、121、229、292列是1，其余列是0。

（5，8）中填的是5，转成矩阵的一行就是，第44、122、230、293列是1，其余列是0。

（5，8）中填的是6，转成矩阵的一行就是，第44、123、231、294列是1，其余列是0。

（5，8）中填的是7，转成矩阵的一行就是，第44、124、232、295列是1，其余列是0。

（5，8）中填的是8，转成矩阵的一行就是，第44、125、233、296列是1，其余列是0。

（5，8）中填的是9，转成矩阵的一行就是，第44、126、234、297列是1，其余列是0。

把这9行插入到矩阵中。由于这9行的第44列都是1（不会有其他行的44列会是1），也就是说这9行中必只有1行（有且只有1行）选中（精确覆盖问题的定义，每列只能有1个1），是最后解的一部分。这就保证了最后解在（5，8）中只有1个数字。

这样，从数独的格子依次转换成行（1行或者9行）插入到矩阵中。完成了数独问题到精确覆盖问题的转换。

优化

1可以发现，在Dance（K）函数调用的时候，是直接调用\_Head.Right来获得未求解列。由于精确覆盖问题是要求每个列都要覆盖到，因此，在算法中调用未求解列的先后顺序那就不是最重要了。假如，现在有两个未求解列C1和C2，C1列有8个元素，C2列有4个元素。最坏的情况，从C1列求解，需要调用8次Dance（K+1），而从C2列求解，需要调用4次Dance（K+1）。感觉上从C2列求解比从C1列求解效率要高些。因此，在Dance（K）函数调用的时候，先找寻列元素最少的未求解列，再依次求解，可能效率会高点。我们把这个称之为**改进的舞蹈链（Improve Dancing Links）算法**给每个列首元素（除却Head元素）添加一个Count分量，表示这个列首所在列的其他元素的个数。

因此，在原算法的基础上，把C1=Head.Right改成获得Count分量最少的列首元素。

2在求解精确覆盖问题中，返回的答案实际上是行的集合，集合的一个特性是无序性。也就意味着，如果答案是唯一的话，改变行在矩阵中的顺序，不影响最后答案的输出，无论这行换到什么位置，最后的答案始终包含着这行（如果答案不是唯一的，也没啥太大的影响）。也就是说，行的顺序不影响最终答案的求解。

我们就从这个方向入手

在构造矩阵的时候，先遍历数独的格子，先把有数字的格子转换为行，插入到矩阵中。很显然，这些行一定会被选中（想想看么，原问题中（4，2）填的是7，如果该行没选中，结果出现了（4，2）填的是9，那不是一件很搞笑的事么）。

由于是精确覆盖问题，每列只能有1个1，而上面的插入的几行一定会被选中。那么，在接下来插入的行如果和上面的行相冲的话（两个行有相同的列有1），那么，后插入的行是个无效的行（肯定不会被选中）。这些无效的行插入到矩阵中，虽然不会影响最终的结果，但是肯定影响求解的效率（空间和时间都有所损耗），而这样的无效行其实有不少。

我们要采用特殊的手法，来避免这些无效的行插入到矩阵中。分两步走

1、先遍历数独的格子，把那些有数字的格子转换为行，插入到矩阵中。在插入的同时，把包含1的列的列首元素的Count分量设置为-1（起到后面判别的作用）。

由于这些行一定能被选中，是答案的一部分，那么把这些行的行号置入到答案列表中，并把这些列的列首元素从水平双向链中移除（手动移除比调用RemoveCol方法快）

2、在遍历没有数字的格子，转换为若干行（1个格子9行）插入到矩阵中。在插入到矩阵的时候，判断包含1的列的列首元素的Count分量。如果是-1，说明新插入的行和第1步中的某些行相冲，是个无效行，没有必要插入到矩阵中；如果不是-1，说明是个有效行，插入到矩阵中。

经过这个优化，能大大减少矩阵的规模（列不变，行减少了不少），我们称之为**数独的舞蹈链（Sudoku Dancing Links）算法**。