# 数独求解及生成程序说明文档

## 总流程

1. 简要介绍

本程序共包含两大功能：求解和生成数独。

本程序用结构体Board存储数独，用全局变量vector<Board> g\_ans存储数据的解，用结构体Block存储单个方格的信息。

在计算数独部分：

本程序同时采用搜索法和Dancing Link X算法。

本程序采用Solve函数对输入数独矩阵进行预处理，用Search函数求解数独。用结构体DancingLink来存储Dancing Link X算法所需的全部信息和实现其功能的成员函数，并用SolveDLX函数实现用Dancing Link X算法解数独。

在生成数独部分：

本程序采用GenerateFull函数生成完整的数独，并用GenerateSudoku函数从完整的数独生成有空格的数独题。

1. 总流程介绍

首先是用户输入功能指令：一个整数，1或2。

若为1，本程序将会调用GenerateFull函数生成完整的数独并用结构体Board储存，命名为full，接着调用GenerateSudoku函数将完整的数独full处理成含空的数独题并用结构体Board储存，并命名为question，调用question的成员函数print将其打印到输出端。

若为2，本程序将会创建一个Board变量question，调用它的成员函数read从输入端读取数独。如果输入的数独满足搜索法条件，则调用Solve函数采用搜索法求解数独，否则调用SolveDLX函数用Dancing Link X算法解数独。如果g\_ans为空，表明无解，输出无解信息，如果g\_ans大小为1，则输出唯一解信息并输出一个解，如果g\_ans的大小超过1，输出多解信息并输出两个解。（可能有改动，是按目前GitHub上程序写的）

1. 总流程图

## 主接口1（求解数独）介绍

1. 核心算法原理介绍
   1. 算法总括

本程序同时使用了两种不同的解数独的算法：搜索法和Dancing Link X算法。我们发现，在空格数较少的时候，搜索法的效率更好，而在空格数较多的时候，Dancing Link X算法则更加出色。为了将二者的优势结合起来，我们以空格数为划分，在空格数较少的时候，采用搜索法求解数独，在空格数较多的时候，先采用搜索法填数独，直到所有剩余的空格都有多于两种的可填数字，然后在采用Dancing Link X算法求解数独。下面分别介绍两种算法。

* 1. 搜索法

设计思路：对于所有的空格，尝试每一种数字，向下搜索，如果可以填到最后一个格子，则说明数独有解，则当前的数独填法即为一组可行解。回溯，直到尝试完所有的可能性。然后，我们对于搜索进行了以下优化：

1）改变搜索顺序：在我们自己解数独时，常常会从剩余情况数最少的格子开始填。在该优化中，我们使用了同样的搜索策略。在每次搜索时，分别计算所有的空格子在该状况下有多少个可能的填法，从填法数最少的格子开始搜索。

2）提早截断：由于在题目要求中对于有多个解的情况只用输出两个解，故我们在搜索过程中如果发现了第二组可行解，就不用再继续搜索，可以直接输出答案从而减少时间。

3）对于特殊情况的提早判断：在该算法中，如果在输入时已经有不合法的情况发生，而该算法仍会尝试剩余格子的所有可能填法，导致运行时间变长。因此，我们对给定的数独进行了预处理，如果给的数独已经不合法，则直接输出无解，不再进行搜索，提高运行效率。

* 1. Dancing Link X 算法

Dancing Link X 算法是对精确覆盖问题的一种极其优秀的搜索算法。在介绍该算法之前，先介绍精确覆盖问题，以及如何将求解数独的问题转化为精确覆盖问题。

精确覆盖问题的描述如下：给定一个0/1矩阵，从中选出一些行构成新矩阵使得新矩阵的每一列有且仅有一个1.。

下面介绍如何将求解数独的问题转化为精确覆盖问题。

考虑填写数独时的限制：1，每个格子内只能填写一个数；2，每行内每个数只能出现一次；3，每列内每个数只能出现一次；4，每个宫内每个数只能出现一次。可以将这些限制变成0/1矩阵。矩阵的前81列分别表示在某个格子内填数。对于之后的81列，每九列为一组表示一行，每组中的第i列表示在对应行填了i。对于再之后的81列，每九列为一组表示一列，每组中的第i列表示在对应列填了i。对于最后的81列，每九列为一组表示一宫，每组中的第i列表示在对应宫填了i。对于每个格子所有可能的填法（已经填的格子只有一种填法），按照上述建立一行。对于一组数独的可行解，我们需要选出该矩阵的若干行，并保证每一列都恰有一个1（分别表示在某个格子内有数，在某列、行、宫内有数字1~9）。由此，一个数独就被转化成了一个精确覆盖的问题。

精确覆盖问题的一个简单算法（X算法）如下：对于当前的0/1矩阵A，如果A为空，则我们已经找到一组可行解；否则，看A的第一列，找到值为1的行，选取该行，并将该行内值为1的对应列删除，同时删除所有（包括自己）的矛盾行（即在这些行中这些列的至少一列的值为1）。对得到的新矩阵A‘再次重复上述算法。

而Dancing Link X 算法是使用双向链表删除元素的简便性，以及在删除元素时只需改变指针，不需要真正删除元素的性质对上述算法进行的优化。

1. 实现流程
2. 流程图示
3. 子接口介绍
   1. 结构体DancingLinkX介绍
      1. 功能描述

用于存储Dancing Link X算法解数独所需的信息，并且提供函数执行该算法需要的功能。

* + 1. 变量简单描述
    2. 成员函数介绍
  1. Search函数介绍
     1. 功能描述

用于求出数独的解。如果求所有解标志为真，会求出所有数独的解，否则最多求出两个解。

* + 1. 接口描述

输入为一个存储数独信息的Board结构以及求所有解的bool标志。输出为数独的解，存储在g\_ans数组中。

* + 1. 实现原理
  1. Solve函数介绍
     1. 功能描述
     2. 接口描述

输入为一个存储数独信息的Board结构以及求所有解的bool标志。若求所有解的标志为真，则返回解的数量，否则返回解的情况（唯一解、无解或多解）。

* + 1. 实现原理
  1. SolveDLX函数介绍
     1. 功能描述

使用Dancing Link X算法求解数独。

* + 1. 接口描述

输入为一个存储数独信息的Board结构以及求所有解的bool标志。若求所有解的标志为真，则返回解的数量，否则返回解的情况（唯一解、无解或多解）。

* + 1. 实现原理
  1. 的

## 主接口2（生成数独）介绍

1. 核心算法原理介绍

一种简单的算法是：随机在当前（一开始为空）的数独中选取一个格子填入一个随机数字，并通过解数独的算法检查新的数独的解的个数，如果无解，则重新选取格子和数字；如有唯一解，则这个数独就是一个合法的数独题；如有多解，则对新的数独重复以上算法，直到只有唯一解。

但是这种算法有一个弊端：在解很少的情况下，可能很难随机到合法的格子和数。为了解决这个问题，我们使用了相反的方向来生成数独。

首先，我们会生成一个完整的数独棋盘（即每个格子均有数的数独）。方法如下：每次填充一行，随机一个1~9的全排列，分别尝试该排列的每一个轮换与已经填充的行是否矛盾，如果该全排列的所有轮换均不可行，则从第一行重新开始生成。

然后，我们从这个已经生成的完整数独中随机选取一个格子删除数，并判断新的数独是否只有唯一解，如果有多解，则尝试删除其他格子中的数字。如果尝试完所有的格子均无法继续删除，则停止删除。这样我们就获得了一个有较少数字的数独（经测试，生成的数独一般只有23~25个格子中有数，符合题目要求）。

1. 实现流程
2. 流程图示
3. 子接口介绍
   1. Rand函数介绍
      1. 功能描述

用于随机产生一个随机的正整数。为了避免

* + 1. 接口描述
    2. 实现原理
  1. GenerateFull函数介绍
     1. 功能描述

用于产生一个合法的填满数的数独。

* + 1. 接口描述

没有输入。输出是一个存储填满的数独的Board变量。

* + 1. 实现原理
  1. GenerateSudoku函数介绍
     1. 功能描述

用于产生一个少于40个数字的数独，且该数独的解的数量超过一个给定的值。

* + 1. 接口描述

输入是一个存储填满的数独的Board变量以及一个int整型变量solCnt。若solCnt的值为-1，输出一个存储有唯一解的数独题的Board变量，若solCnt的值不为-1，则输出一个存储有解的数量大于等于solCnt的数独题的Board变量。

* + 1. 实现原理

## 其他模块介绍

1. 结构体Board介绍
   1. 功能描述
   2. 变量描述
   3. 成员函数介绍
2. 结构体Block介绍
   1. 功能描述

用于存储单个数独方格的信息。

* 1. 变量描述

含有三个int整型变量x，y，value，分别用于存储方格的行坐标，列坐标和方格内的数值。初始化三者皆为0。

* 1. 成员函数介绍

含有一个构造函数，用于为Block型变量赋值。

1. 宏定义介绍
   1. 用于求解数独的宏

无解标志为NO\_SOLUTION，定义其值为0，唯一解标志为ONE\_SOLUTION，定义其值为1，多解标志为MANY\_SOLUTION，定义其值为-1。使用标志而非数值可以增强程序的可读性。

* 1. 用于Dancing Link X算法解数独的宏

将HEAD定义为0，表明链表头，将COLUMNS定义为324，表明列的数量。

1. 输入模式介绍
   1. Mode 1
   2. Mode 2
   3. Mode 3
   4. Mode 4