**人工智能实验报告**

姓名：孟衍璋 学号：16337183

1. **实验目的**

用brute force、back-tracking、forward-checking、enforce Generalized Arc Consistency求解数独。

1. **实验内容**

写一个python程序，输入一个数独问题，返回它的计算结果。使用几种不同的算法来解决数独问题，返回数独的解决方案。几种算法如下：

• brute force (exhaustive search) method

• back-tracking (Constraint Satisfaction Problem (CSP)

• forward-checking with Minimum Remaining Values (MRV) heuristics

• enforce Generalized Arc Consistency (GAC) algorithm

主程序文件必须被命名为SudokuSolver.py，数独谜题存在puzzle.txt文件中，解决方案存在solution.txt文件中。

程序必须使用命令行输入，样例如下:

Python SudokuSolver puzzle1.txt BF

Python SudokuSolver puzzle2.txt BT

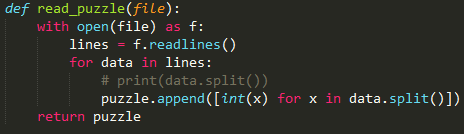
Python SudokuSolver puzzle3.txt FC-MRV

Python SudokuSolver puzzle4.txt GAC

输出需要包含总运行时间，搜索时间，遍历的结点个数。将这些信息存在performance文件中。

1. **实验步骤**
2. **实现brute\_force**

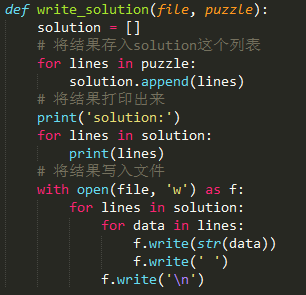
主要由七个函数构成，首先是读取文件的函数：



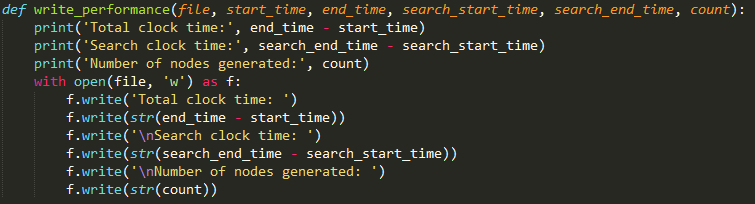
第二个是判断某一位置还能填什么数字的函数：



然后是打印并写入solution文件的函数：



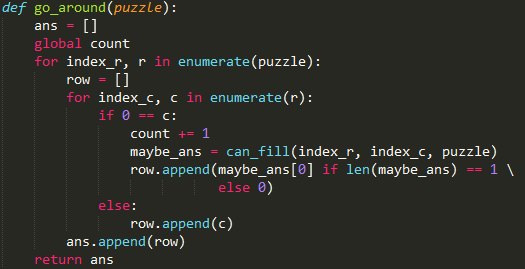
然后是打印并记录performance文件的函数：



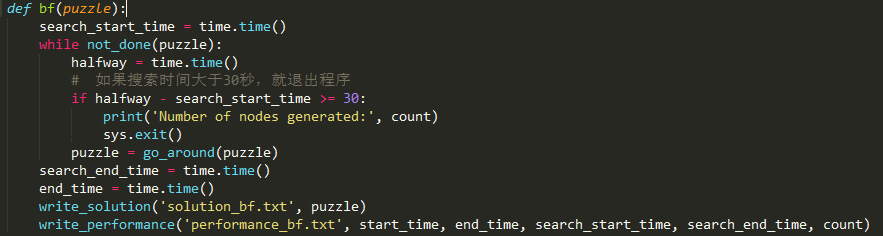
然后是判断数独是否已经填完的函数：



然后是依次填入的函数：



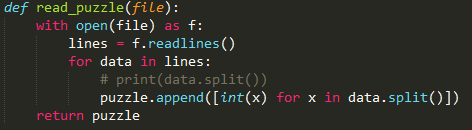
最后则是实现brute\_force的函数：



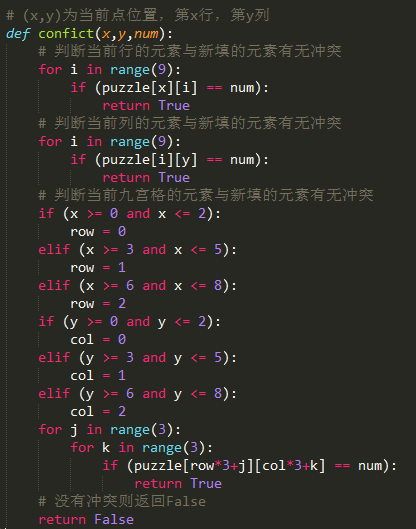
在该函数中，设置了如果搜索时间大于30秒就退出程序的操作。

1. **实现backtracking**

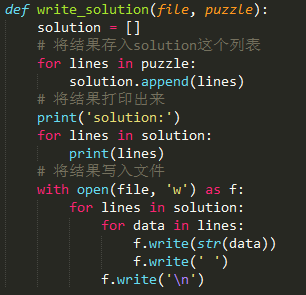
主要由六个函数构成，首先是读取文件的函数：



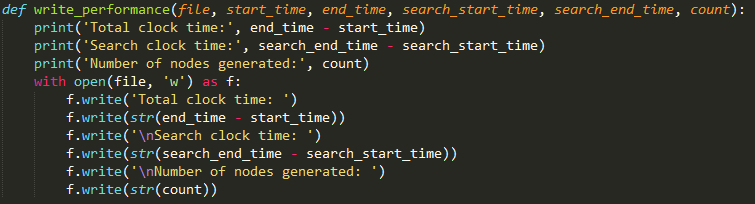
然后是判断冲突的函数，即在数独的空余位置填上一个数字之后，判断是否会与同行、同列、同九宫格的元素冲突：



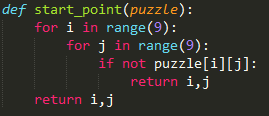
然后是打印并写入solution文件的函数：



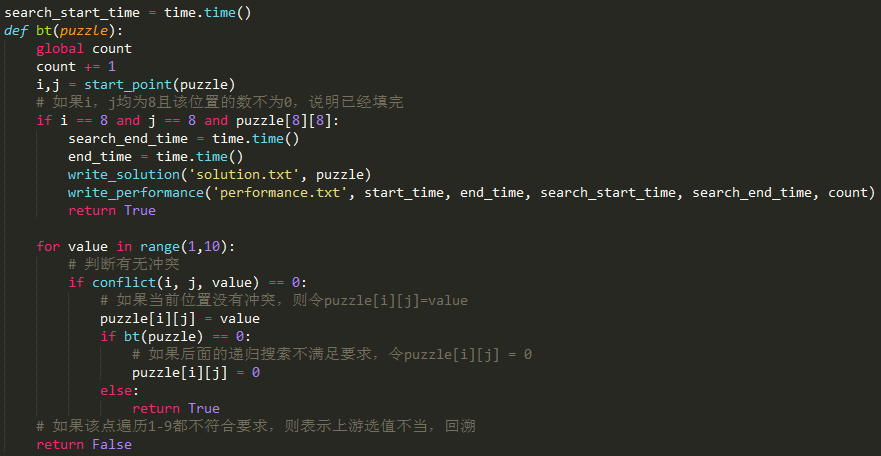
然后是打印并记录performance文件的函数：



然后是start\_point函数，用来指定每次递归访问的位置：



最后是backtracking函数，用递归的方法实现：



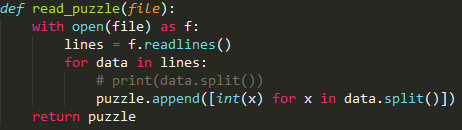
每递归调用一次函数，全局变量count就加1，这个值就是访问的节点数。

函数中先用start\_point函数确定访问的位置，如果已经到了第9行第9列且当前数字不为0，说明已经将数独填完，记录下解决方案和算法表现。

如果是在平常的位置，用之前的冲突函数判断，如果没有冲突，从1-9依次填入数字。如果递归搜索不满足条件，则将刚刚填的值置为0。如果某个点遍历1-9都不符合要求，则表示之前选值不当，回溯。

1. **实现forward\_checking**

主要由六个函数构成，第一个是读取数独的函数：



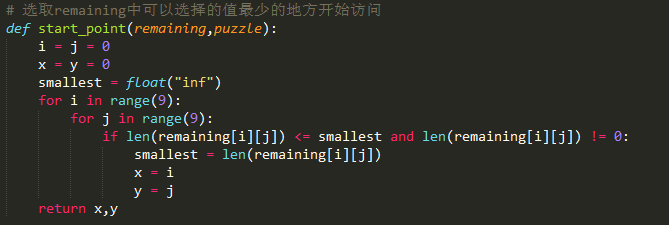
第二个是判断某一个位置还能填什么数字的函数：



它访问数独中某一个元素，然后统计与它同行、同列、同九宫格中都出现过哪些元素，将剩下的元素存为一个列表。

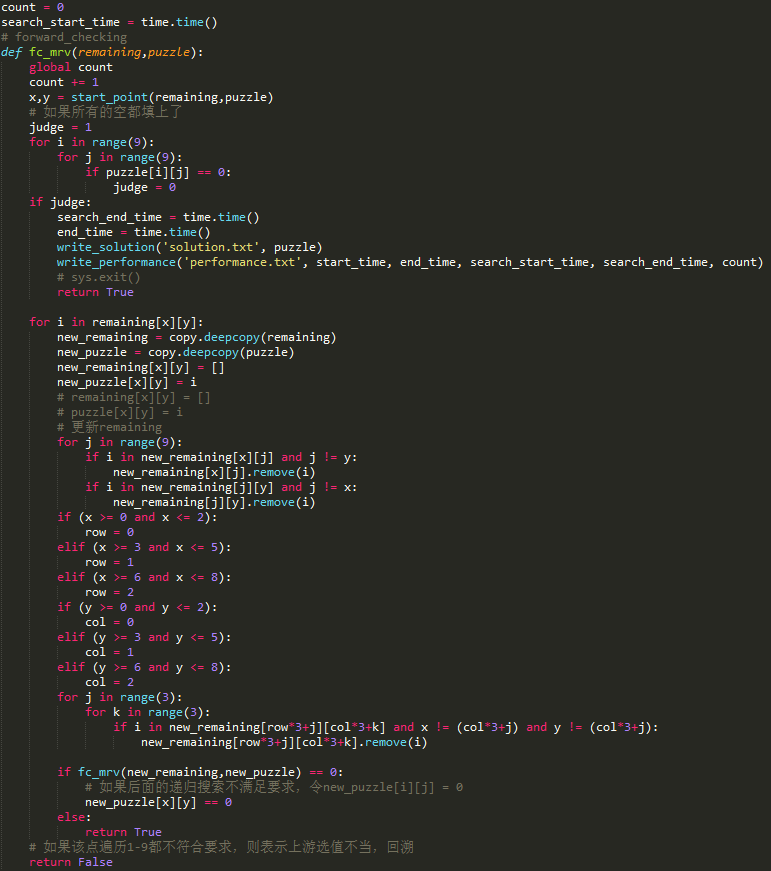
第三第四个是写solution与performance的函数，同backtracking一样。

第五个是决定该访问哪一个节点的函数：



该函数与backtracking的有些不同，FC-MRV是先选择约束条件最多的位置，也即剩余能填的数字最少的位置。然后返回这个位置的坐标。

最后是forward\_checking的实现：

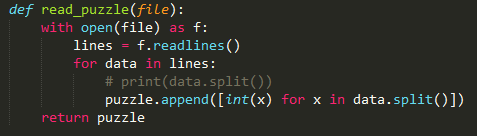


与backtracking不一样的地方在于，选取访问节点的方式不同，而且不需要判断冲突，利用can\_fill函数得出的remaining列表，就能得到每个位置能填的数字。然后每填一个数字，对remaining进行一次更新。

其余操作便和backtracking差不了太多。

1. **实现generalized arc consistency**

主要由六个函数构成，第一个是读取数独的函数：



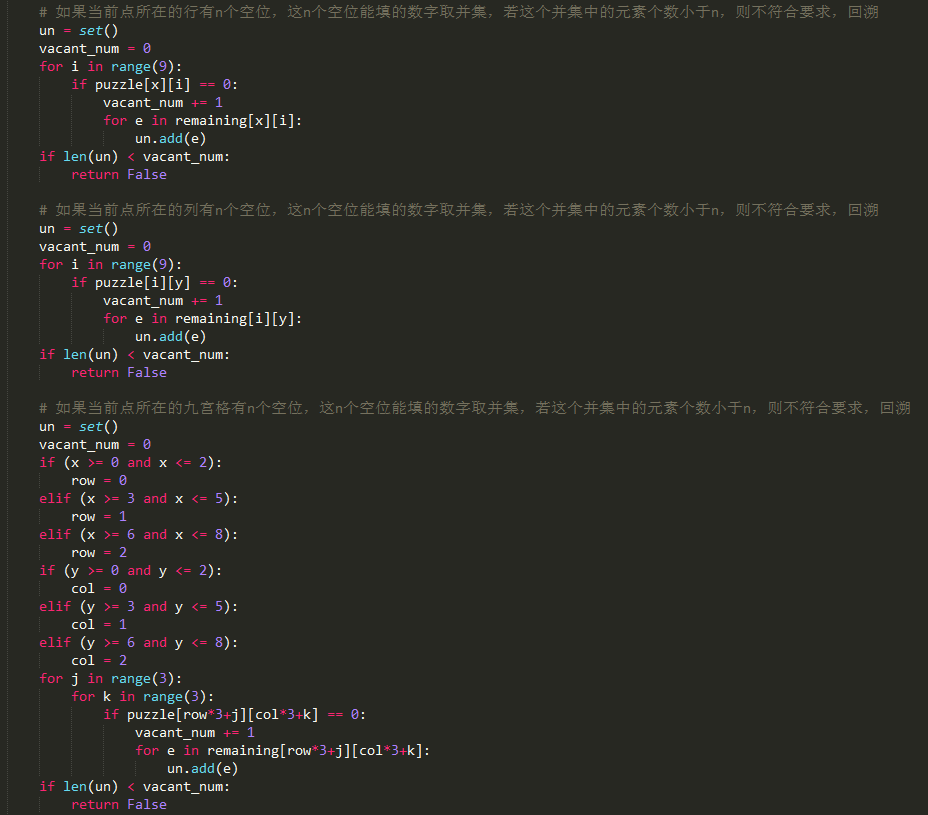
第二个是判断某个位置还能填什么数字的函数，用于生成remaining列表：



第三个和第四个是第三第四个是写solution与performance的函数，同之前一样。

第五个是start\_point函数，和forward\_checking的一样，选取remaining中可以选择的值最少的地方开始访问。

第六个便是gac函数，其余部分都和FC一样，不同的是加了这样一个约束条件：



上述约束条件实际上是指，对于当前访问的位置，计算与其同行或者同列、或者同九宫格剩余的空位能填的数字，取一个并集，如果这个并集中的数字个数小于空位的个数，则之前填的方案一定是错误的，需要回溯。

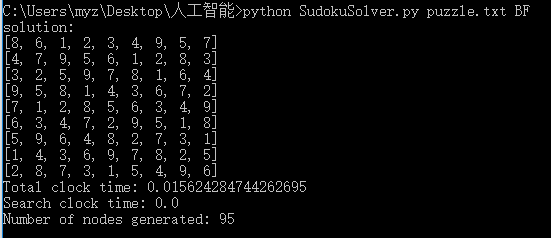
1. **模块化**

因为要实现命令行输入，所以将之前的几种算法都分别存为单独的文件，在同一个目录下新建一个\_\_init\_\_.py文件，表示这个目录是一个包。

利用sys.argv参数，将命令行的输入传入程序中，再调用各个模块的函数实现数独的解答。

1. **实验结果**
2. **brute\_force**

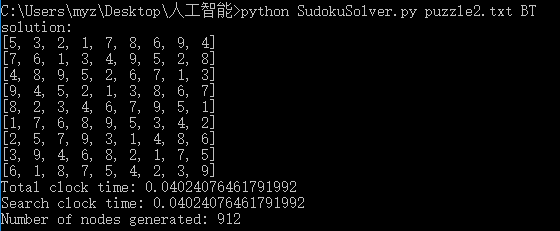
在数独简单的情况下，可以很快算出答案：



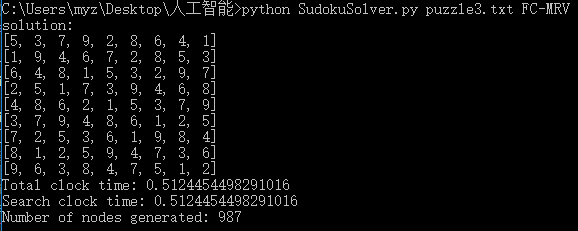
在数独复杂的时候，就需要很长时间，我在程序里设置了如果30秒算不出来就退出的操作。



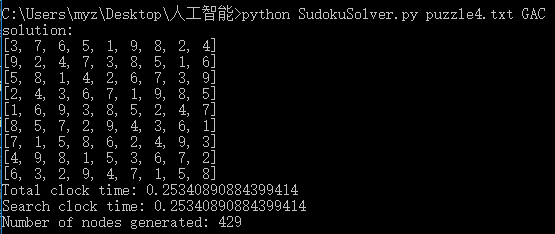
1. **backtracking**



1. **forward\_checking\_MRV**



1. **generalized\_arc\_consistency**



1. **实验心得**

这次实验是用程序解数独，感觉比较有趣，实验要求中需要用四种方式求解，第一种是暴力搜索，第二种是回溯法，第三种是带最小剩余值的前向检查，第四种是广义弧一致性。首先需要理解这些方法的不同之处，和它们分别是怎样实现的。

第一种暴力搜索就是穷举，没有什么技术含量。第二种回溯法，就是依次遍历每个空位，每个空位都依次填入1-9的数字，如果判断出有冲突，就依次加1，如果1-9填入了之后都有冲突，说明之前一位填得有问题，需要回到之前一位，将其加1。

第三种方法是带最小剩余值的前向检查，与回溯法不同的地方在于，不需要判断冲突，在进行递归之前，就将每个空格中除去冲突能填的数字列出来，然后访问到每个位置的时候，就依次填入这些数字，每填入一个数字之后，需要更新这个列表。带最小剩余值的意思则是，在决定访问什么位置的时候，先访问那些可填值列表中最少的位置，这样先满足了约束最多的一项，后面的约束也就会更少。

第四种方法是广义弧一致性，开始没有太理解，后来问了同学老师之后，在带最小值的前向检查的基础上，又加了一个约束条件。如果对于当前访问的位置，计算与其同行或者同列、或者同九宫格剩余的空位能填的数字，取一个并集，如果这个并集中的数字个数小于空位的个数，则之前填的方案一定是错误的，需要回溯。

最后还要考虑在控制台输出的问题，题目要求中需要按照类似Python SudokuSolver puzzle1.txt BF这样的形式来。经过查询之后知道控制台的输入可以用sys.argv来获取，第一个参数下标是0，第二个参数下标是1，依次类推，然后再将之前写的四种算法存为四个文件，组织成模块的形式，便能在主程序中调用。

总而言之，这次实验十分有趣，而且让我学习到了很多东西，感觉收获很多。