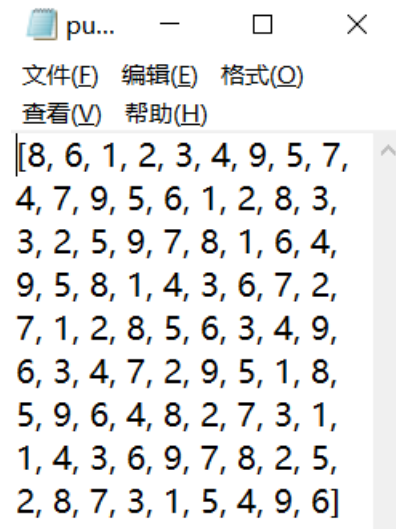


## Assignment 2: Sudoku Solver

17343154 张嵌竹

Correctness of the solver:

Puzzle1:



```
pu... - □ ×
文件(E) 编辑(E) 格式(O)
查看(V) 帮助(H)
[8, 6, 1, 2, 3, 4, 9, 5, 7,
4, 7, 9, 5, 6, 1, 2, 8, 3,
3, 2, 5, 9, 7, 8, 1, 6, 4,
9, 5, 8, 1, 4, 3, 6, 7, 2,
7, 1, 2, 8, 5, 6, 3, 4, 9,
6, 3, 4, 7, 2, 9, 5, 1, 8,
5, 9, 6, 4, 8, 2, 7, 3, 1,
1, 4, 3, 6, 9, 7, 8, 2, 5,
2, 8, 7, 3, 1, 5, 4, 9, 6]
```

### Algorithm results& analysis

BF:

由于纯粹的暴力搜索的程序运行时间实在是太长了，于是对暴力搜索进行了改进，采用回溯+暴力搜索的方式，并使用一维数组。先无顺序地从 1-9 中选出数字填写，然后检查其是否对现有矩阵造成冲突，若有冲突则回溯到上一个节点并选用下一个数字填写。

```
while cellIndex < len(puzzle):
    count += 1
    if isValid(spuzzle, cellIndex, symIndex):
        spuzzle[cellIndex] = symbols[symIndex]
        cellIndex += 1
        while cellIndex < len(puzzle) - 1 and puzzle[cellIndex] != 0:
            cellIndex += 1
        symIndex = 0
    else:
        symIndex += 1
    if symIndex >= limit:
        spuzzle[cellIndex] = 0
        cellIndex -= 1
        while puzzle[cellIndex] != 0:
            cellIndex -= 1
        symIndex = symbols.index(spuzzle[cellIndex]) + 1
```

BT:

第二种算法是先将每个空白节点的所有可用数字挑选出来再进行实验和递归，这种算法明显地减少了遍历的节点个数

```

listr = [puzzle[j] for j in range(len(puzzle)) if j // limit == space.row]
listc = [puzzle[j] for j in range(len(puzzle)) if j % limit == space.col]
listb = [puzzle[j] for j in range(len(puzzle)) if (j // limit) // boxsize * boxsize * limit + (
    j % limit) // boxsize * boxsize == space.box]

constraint = set(symbols) - set(listr) - set(listc) - set(listb)
space.set_constraints(constraint)

```

### FC-MRV:

第三种算法在第二种的基础上将所有空白节点按照可选数字的个数进行排序, 首先填写可选数字最少的空白节点

```

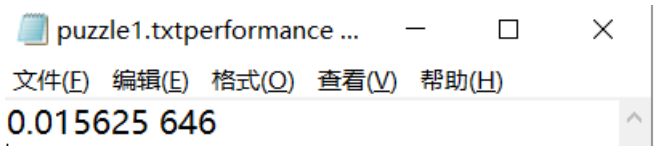
def __lt__(self, other):
    if len(self.constraints) < len(other.constraints):
        return True

```

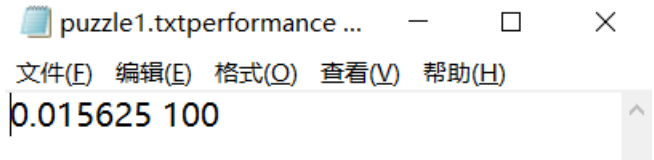
### Performance analysis on the time/space complexity :

#### Puzzle1:

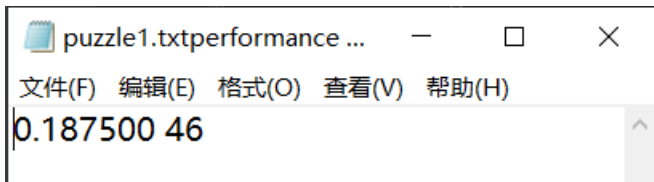
##### 1. Brute Force:



##### 2. Back-tracking (CSP):



##### 3. Forward-checking with MRV heuristics:



Table

	BF	BT	MRV
Puzzle1	0.015625/646	0.015625/100	0.187500/46
Puzzle2	0.000000/905	0.015625/258	0.140625/42
Puzzle3	0.062500/14632		0.328125/57
Puzzle4	0.031250/4650	0.031250/332	0.250000/62
Puzzle5	0.015625/2576	0.171875/1487	0.359375/67

在暴力搜索中, 复杂度为  $O(n^m)$  其中  $n$  是每个方格的可能性数(即经典数独中的 9),  $m$  是空白的空格数, 采用 Back-tracking(CSP)可以有效减少每个空格访问的节点数量, 这与  $m$  有关, 使用 MRV 方式由于对空白节点进行排序, 重复实验将会发现其节点访问量近似  $O(n)$  因为每个空白节点确定数字后都将减少后面的空白节点的访问数量

同时, 由于采用递归的方式, 第二三种算法的时间会超过第一种循环时间, 加上 MRV 我在设计的时候每次递归都 sort 一下, 更加增加了时间复杂度。