constraint 函数：用来判断是否满足数独的条件

检查行内是否满足

```
def constraint(i_row, i_col):
    _row = sorted(table[i_row])
    for i in range(len(_row) - 1):
        # 检查i_row行内有没有相同的数字
        if _row[i] != ' ' and _row[i] == _row[i + 1]:
            return False
```

检查列内是否满足

```
_col = sorted([table[i][i_col] for i in range(len(table))])
for i in range(len(_col) - 1):
    if _col[i] != ' ' and _col[i] == _col[i + 1]:
        return False
```

检查九宫格内是否满足

```
if 0 < i_row < 8 and 0 < i_col < 8:
    i = i_row // 3 * 3  # 该3*3的左上角
    j = i_col // 3 * 3  # 该3*3的左上角
    _box = sorted([table[i0][j0] for i0 in range(i, i + 3) for j0 in range(j, j + 3)])
    for i in range(len(_box) - 1):
        if _box[i] != ' ' and _box[i] == _box[i + 1]:
            return False
```

find_next 函数：寻找下一个需要填数字的空格的坐标

先把本行找完

```python
def find_next(_row, _col):
    for j in range(_col + 1, 9):
        if table[_row][j] == " ":
            return _row, j
```

然后是其他行

```python
for i in range(_row + 1, 9):
    for j in range(0, 9):
        if table[i][j] == " ":
            return i, j
```

回溯函数

```python
def back_tracing(_row, _col):
    if _row > 8:
        print("Output:")
        for row in table:
            print(row)
    else:
        for _value in range(1, 10):
            table[_row][_col] = str(_value)
            if constraint(_row, _col):
                back_tracing(*find_next(_row, _col))
        table[_row][_col] = " "
```

思路同上课 ppt 的伪代码，

需要多注意的一点是：最后一行的退出函数时需要讲空格还原：

table[_row][_col]=""

- **子集树回溯算法**

```
Backtrack( int t)   //搜索到树的第t层
{ //由第t层向第t+1层扩展，确定x[t]的值
    if  t>n then output(x);  //叶子结点是可行解
    else
        while( all Xt) do        //Xt为当前扩展结点的所有可能取值集合
        {
            x[t] = Xt中的第i个值；
            if( Constraint(t) and Bound(t) )
                Backtrack( t+1 );
        }
}
```

运行结果

```
main
D:\Work\sudoku\venv\Scripts\python.exe D:/Work/sudoku/main.py
Input:
['1', ' ', ' ', '4', '5', '6', '3', '7', '2']
['7', '3', '6', '8', ' ', ' ', ' ', ' ', '5']
[' ', '2', ' ', '7', ' ', '9', '1', ' ', '8']
['6', '4', ' ', ' ', ' ', '8', '7', '5', ' ']
[' ', '8', '2', ' ', ' ', '5', ' ', ' ', ' ']
['5', ' ', '1', '6', ' ', '3', '8', ' ', ' ']
['3', '5', '7', '2', '9', '1', ' ', '8', '4']
[' ', ' ', ' ', '5', ' ', '7', ' ', ' ', '3']
['2', ' ', ' ', '3', '8', ' ', ' ', ' ', ' ']

Output:
['1', '9', '8', '4', '5', '6', '3', '7', '2']
['7', '3', '6', '8', '1', '2', '9', '4', '5']
['4', '2', '5', '7', '3', '9', '1', '6', '8']
['6', '4', '3', '9', '2', '8', '7', '5', '1']
['9', '8', '2', '1', '7', '5', '4', '3', '6']
['5', '7', '1', '6', '4', '3', '8', '2', '9']
['3', '5', '7', '2', '9', '1', '6', '8', '4']
['8', '1', '4', '5', '6', '7', '2', '9', '3']
['2', '6', '9', '3', '8', '4', '5', '1', '7']

进程已结束,退出代码0
```