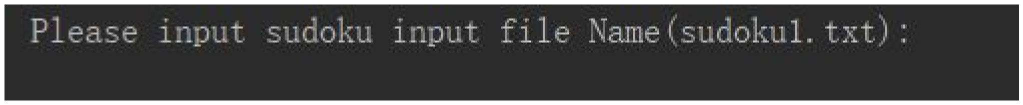
Project Part 2: Sudoku solver

Jiayi Li, Yichen Li

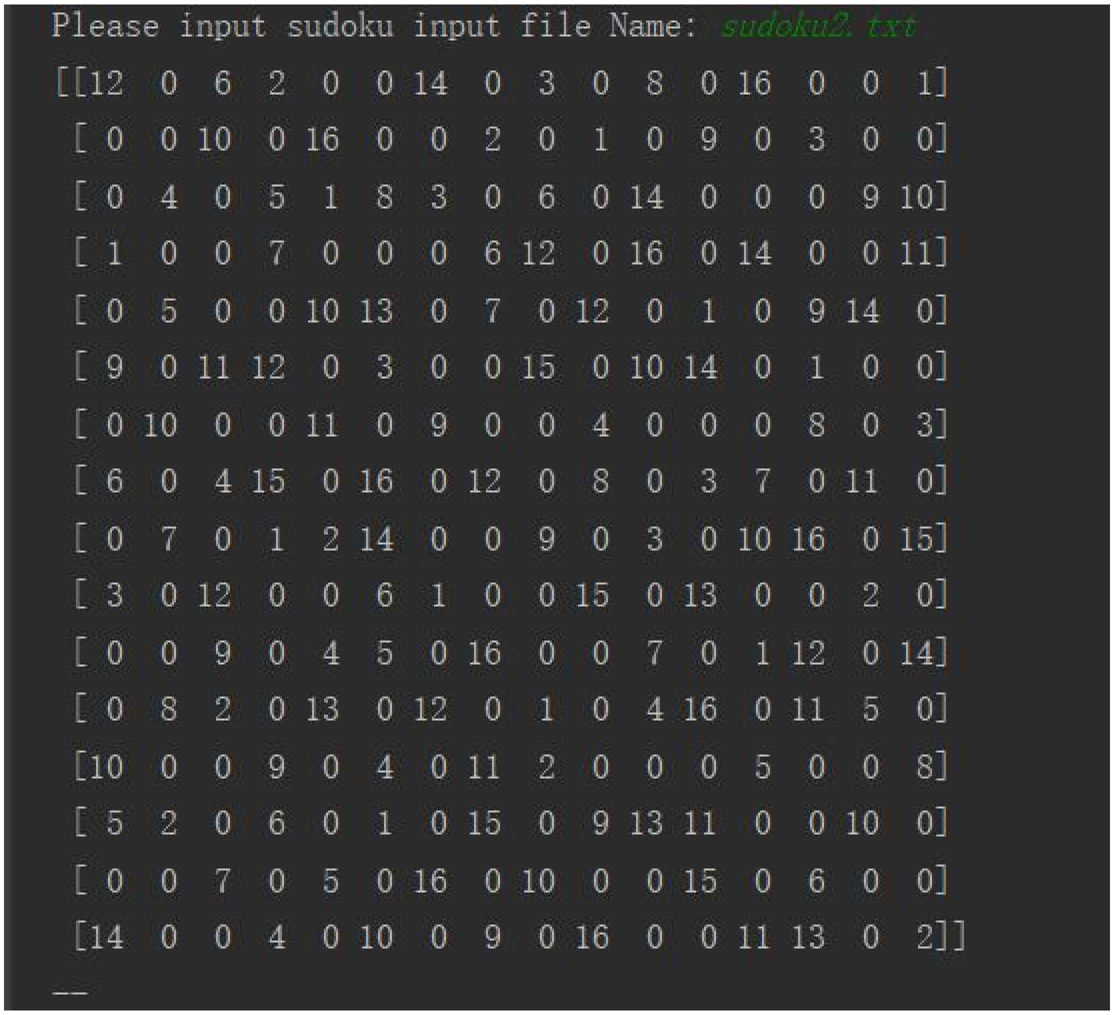
Instruction

Our part 2 is a solution of Sudoku based on our part 1 job.



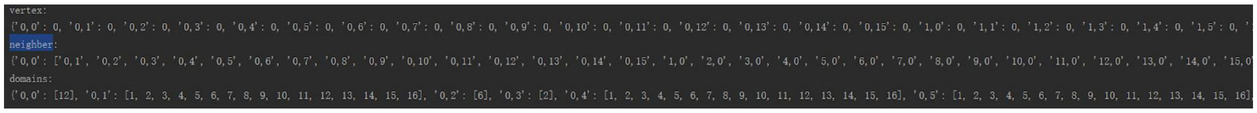
```
Please input sudoku input file Name(sudoku1.txt):
```

Running our program, the first thing is to input the sudoku file that needs to be solved. After inputting the filename, if the file is existing, the program will show you the input sudoku problem as a matrix.



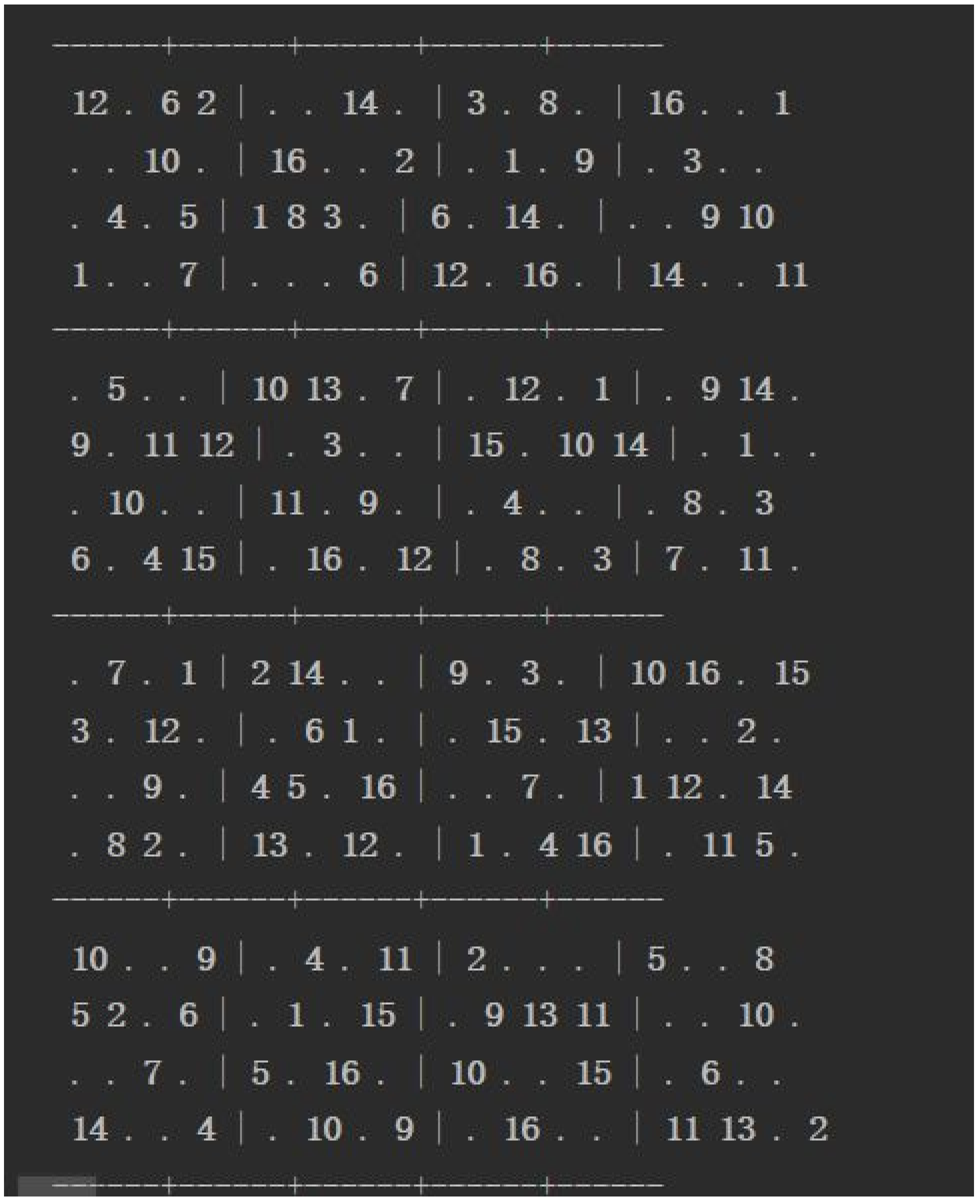
```
Please input sudoku input file Name: sudoku2.txt
[[12  0  6  2  0  0 14  0  3  0  8  0 16  0  0  1]
 [ 0  0 10  0 16  0  0  2  0  1  0  9  0  3  0  0]
 [ 0  4  0  5  1  8  3  0  6  0 14  0  0  0  9 10]
 [ 1  0  0  7  0  0  0  6 12  0 16  0 14  0  0 11]
 [ 0  5  0  0 10 13  0  7  0 12  0  1  0  9 14  0]
 [ 9  0 11 12  0  3  0  0 15  0 10 14  0  1  0  0]
 [ 0 10  0  0 11  0  9  0  0  4  0  0  0  8  0  3]
 [ 6  0  4 15  0 16  0 12  0  8  0  3  7  0 11  0]
 [ 0  7  0  1  2 14  0  0  9  0  3  0 10 16  0 15]
 [ 3  0 12  0  0  6  1  0  0 15  0 13  0  0  2  0]
 [ 0  0  9  0  4  5  0 16  0  0  7  0  1 12  0 14]
 [ 0  8  2  0 13  0 12  0  1  0  4 16  0 11  5  0]
 [10  0  0  9  0  4  0 11  2  0  0  0  5  0  0  8]
 [ 5  2  0  6  0  1  0 15  0  9 13 11  0  0 10  0]
 [ 0  0  7  0  5  0 16  0 10  0  0 15  0  6  0  0]
 [14  0  0  4  0 10  0  9  0 16  0  0 11 13  0  2]]
--
```

Also, vertex, neighbor and domains.



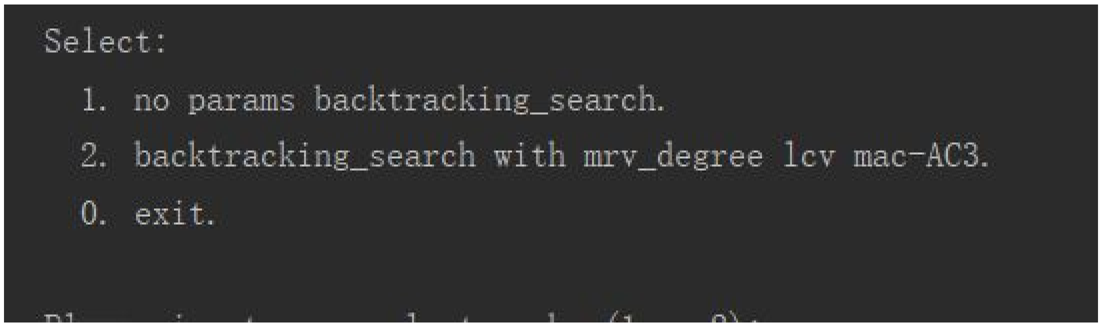
```
vertex:
{'0,0': 0, '0,1': 0, '0,2': 0, '0,3': 0, '0,4': 0, '0,5': 0, '0,6': 0, '0,7': 0, '0,8': 0, '0,9': 0, '0,10': 0, '0,11': 0, '0,12': 0, '0,13': 0, '0,14': 0, '0,15': 0, '1,0': 0, '1,1': 0, '1,2': 0, '1,3': 0, '1,4': 0, '1,5': 0, '
neighber:
{'0,0': ['0,1', '0,2', '0,3', '0,4', '0,5', '0,6', '0,7', '0,8', '0,9', '0,10', '0,11', '0,12', '0,13', '0,14', '0,15', '1,0', '2,0', '3,0', '4,0', '5,0', '6,0', '7,0', '8,0', '9,0', '10,0', '11,0', '12,0', '13,0', '14,0', '15,0
domains:
{'0,0': [12], '0,1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], '0,2': [6], '0,3': [2], '0,4': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], '0,5': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
```

And Initial state.



```
 ------+------+------+------+------
  12 . 6 2 | . . 14 . | 3 . 8 . | 16 . . 1
  . . 10 . | 16 . . 2 | . 1 . 9 | . 3 . .
  . 4 . 5 | 1 8 3 . | 6 . 14 . | . . 9 10
  1 . . 7 | . . . 6 | 12 . 16 . | 14 . . 11
 ------+------+------+------+------
  . 5 . . | 10 13 . 7 | . 12 . 1 | . 9 14 .
  9 . 11 12 | . 3 . . | 15 . 10 14 | . 1 . .
  . 10 . . | 11 . 9 . | . 4 . . | . 8 . 3
  6 . 4 15 | . 16 . 12 | . 8 . 3 | 7 . 11 .
 ------+------+------+------+------
  . 7 . 1 | 2 14 . . | 9 . 3 . | 10 16 . 15
  3 . 12 . | . 6 1 . | . 15 . 13 | . . 2 .
  . . 9 . | 4 5 . 16 | . . 7 . | 1 12 . 14
  . 8 2 . | 13 . 12 . | 1 . 4 16 | . 11 5 .
 ------+------+------+------+------
  10 . . 9 | . 4 . 11 | 2 . . . | 5 . . 8
  5 2 . 6 | . 1 . 15 | . 9 13 11 | . . 10 .
  . . 7 . | 5 . 16 . | 10 . . 15 | . 6 . .
  14 . . 4 | . 10 . 9 | . 16 . . | 11 13 . 2
 ------+------+------+------+------
```

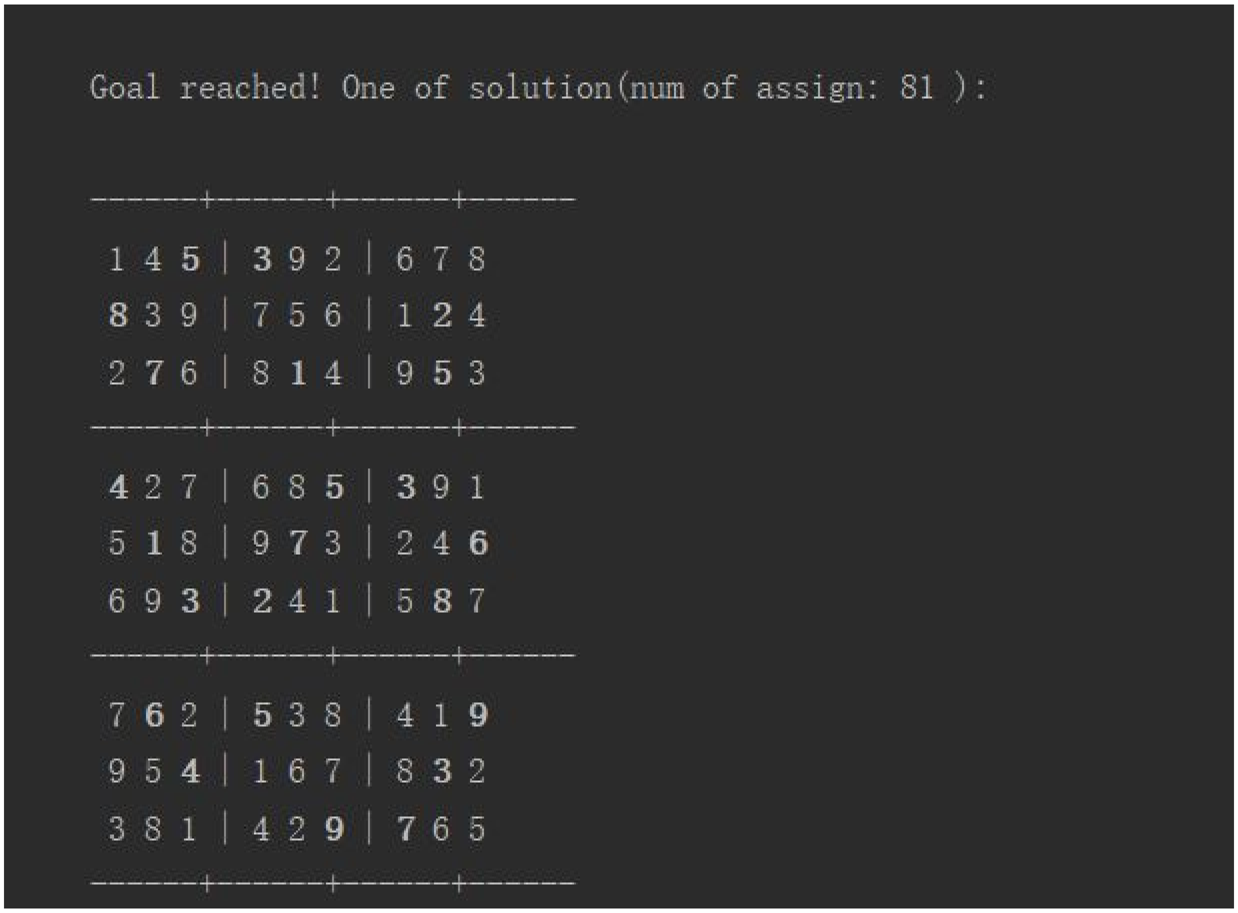
Then the program will do AC-3 to check this problem, after this, you could input number to choose different methods to solve the problem.



```
Select:
  1. no params backtracking_search.
  2. backtracking_search with mrv_degree lcv mac-AC3.
  0. exit.

[illegible]
```

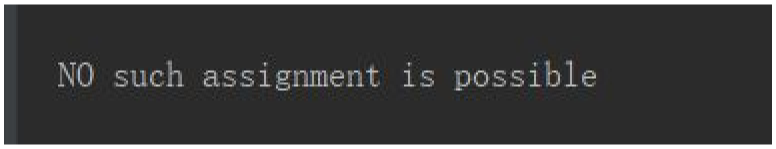
After this, the program will tell you the result, if this problem has a solution.



```
Goal reached! One of solution(num of assign: 81 ):

------+------+------+------
 1 4 5 | 3 9 2 | 6 7 8
 8 3 9 | 7 5 6 | 1 2 4
 2 7 6 | 8 1 4 | 9 5 3
------+------+------+------
 4 2 7 | 6 8 5 | 3 9 1
 5 1 8 | 9 7 3 | 2 4 6
 6 9 3 | 2 4 1 | 5 8 7
------+------+------+------
 7 6 2 | 5 3 8 | 4 1 9
 9 5 4 | 1 6 7 | 8 3 2
 3 8 1 | 4 2 9 | 7 6 5
------+------+------+------
```

If the input problem doesn’t have a solution, the program will tell you “No such assignment is possible”.



```
NO such assignment is possible
```

The detailed description of the application

Our problem could not only solve the 9*9 sudoku problem but also the n*n sudoku problem(n must be a number squared). The vertex of our program is the x-axis and y-axis in the sudoku map. The constraint of our program follows the rules of sudoku problem.

Each row, column, and √n * √n grids can contain each number (in 9*9 sudoku problem, 1 to 9, in n*n problem, 1 to n) exactly once.
The domain is different based on the different problem, in n*n sudoku problem, the domain is 1 to n.

Adaption of project 1

The sudoku extends the class of the CSP with parameters variables, values, neighbors and constraints. The constraints has been changed in the instance of sudoku part, and compared with part 1, the deadline constraint is canceled. The detailed adaption of algorithm and heuristic functions in part 1 is described in the following paragraph.

AC3

AC3 algorithm is firstly used before backtracking search to reduce the domains of each variable. For any two variables that have mutual constraints (any two squares in the same row, or the same column, or in the same large square in Sudoku), the AC3 is used to reduce the domain of each grid. Reduce subsequent space and time consumption. In the first round of assignment, the known squares in the title are filled in. At this time, using AC3 algorithm to judge the arc consistency can greatly reduce the domain of some squares. For some simpler (more known values) Sudoku, you can get the solution directly without continuing the subsequent search. AC3 is also used in backtracking, maintaining the arc consistency every time a value was assigned.

```
Function AC-3(csp) returns false if an inconsistency is found and true otherwise
    Inputs: csp
    Local variables: queue(initially all the arcs in constraints)

    While queue is not empty do
    (Xi,Xj)=queue.pop()
    If Revise(csp,Xi,Xj) then
        If size of Di=0 then return false
        For each Xk in Xi.Neighbors-{Xj} do
            Add(Xk,Xi)to queue
Return true

Function Revise(csp,Xi,Xj) returns true iff revise the domain of Xi
    Revise=false
    For each x in Di do
```

```
        If no value y in Dj allows(x,y) to satisfy the constraint between Xi and Xj
then
        Delete x from Di
        Revised=true
    Return revised
```

Backtracking

Backtracking is used to search assignments for each empty grid. In the implementation of Sudoku, the first round of assignment is to fill the know squares, so there is no need to do backtracking for the assignment. For the problem that cannot be fully solved by AC3, backtracking is used to find the assignment for the rest squares. In this approach, minimum-remaining value heuristic is used to choose the variable. Least constraint value heuristic is used to choose the value that will be assigned to the variable. And for each assignment, AC3 is used to inference the availability.

```
Function Backtracking_search(csp) returns a solution or failure
    Function Backtrack(assignment):
        If assignment is complete return assignment
        Var=select_unassigned_variable(assignment, csp)
        (using MRV)
        For values in Order_domain_values(var,assignment,csp)
        (using LCV)
            If value is consistent with assignment then
                Csp.assign(var, value, assignment)
                Remove value from csp.current_domain
                If inference !=failure(using AC3 as inference)
                Result=Backtrack(assignment)
                If result is not none:
                        Return result
        Return none
```

Minimum remaining value heuristic

In Sudoku, after AC3, the domains of some variables are reduced. Then assign value to variable that has least minimum remaining values first. The heuristic is used in backtracking to choose the variable that first to be assigned.

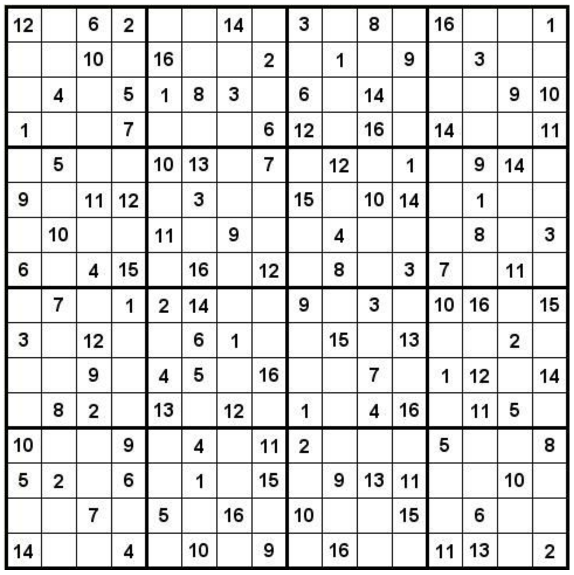
```
Function  mrv_degree(assignment, csp):
    Dictionary[csp.variable]=variable.current_domain
    Order=Sort Dictionary by the quantities of items
```

```
While quantities of variables:
        Variable=the first variable in order
If  there are two variables has the same remain value
        Variable= the length is the largest
        Return variable
```

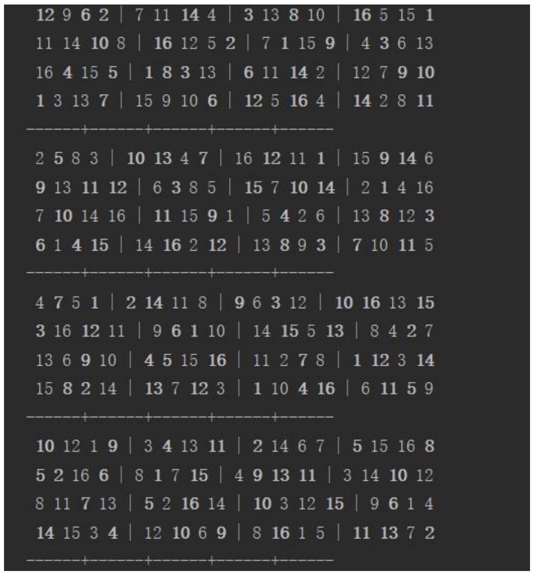
Result

We test many problems, includes 9*9 sudoku, 16 * 16 sudoku and 25* 25 sudoku.
Our program passed all 9*9 and 16*16 sudoku. When solving 25*25 sudoku, our program will spend a long time, and because of space is not enough, so we can't get the correct answer on our computer.
Notice that in p4, our program could directly reach the goal by only doing AC3, it will save a lot of time.
When solving 9*9 and 16*16 sudoku problem, our program performs very well, however, if solving more complicated sudoku problem, such as 25*25, our program will have some efficiency problem and memory out of space problem so may not get the correct answer in our computer.

p1.

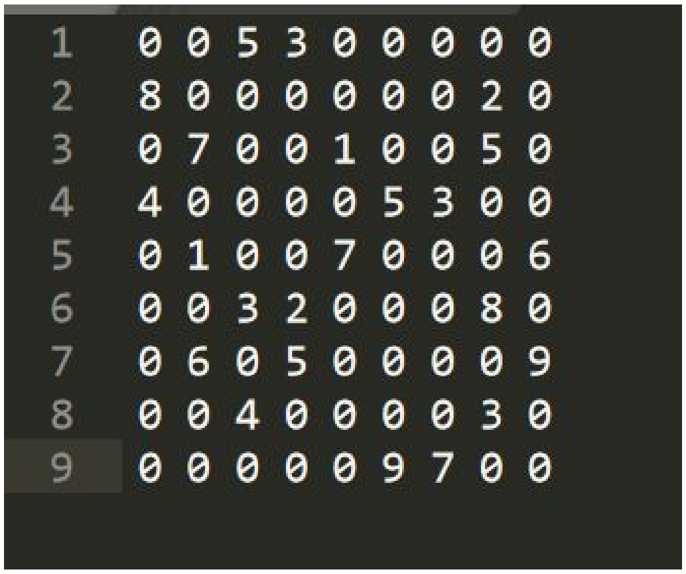


12		6	2			14		3		8		16			1
		10		16			2		1		9		3		
	4		5	1	8	3		6		14				9	10
1			7				6	12		16		14			11
	5			10	13		7		12		1		9	14	
9		11	12		3			15		10	14		1		
	10			11		9			4				8		3
6		4	15		16		12		8		3	7		11	
	7		1	2	14			9		3		10	16		15
3		12			6	1			15		13			2	
		9		4	5		16			7		1	12		14
	8	2		13		12		1		4	16		11	5	
10			9		4		11	2				5			8
5	2		6		1		15		9	13	11			10	
		7		5		16		10			15		6		
14			4		10		9		16			11	13		2

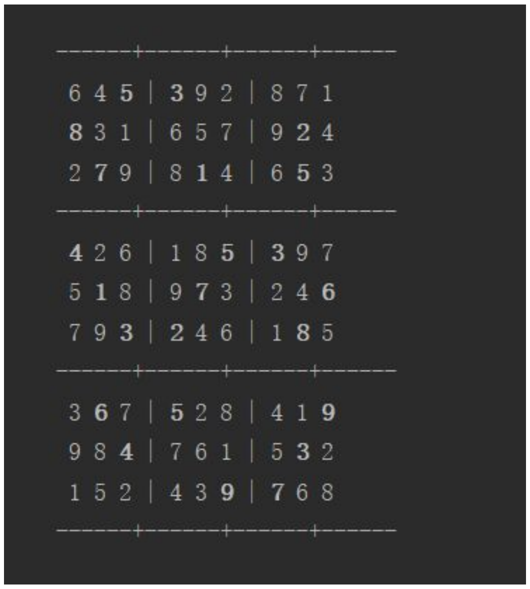


```
 12 9 6 2 | 7 11 14 4 | 3 13 8 10 | 16 5 15 1
 11 14 10 8 | 16 12 5 2 | 7 1 15 9 | 4 3 6 13
 16 4 15 5 | 1 8 3 13 | 6 11 14 2 | 12 7 9 10
 1 3 13 7 | 15 9 10 6 | 12 5 16 4 | 14 2 8 11
------+------+------+------+------
 2 5 8 3 | 10 13 4 7 | 16 12 11 1 | 15 9 14 6
 9 13 11 12 | 6 3 8 5 | 15 7 10 14 | 2 1 4 16
 7 10 14 16 | 11 15 9 1 | 5 4 2 6 | 13 8 12 3
 6 1 4 15 | 14 16 2 12 | 13 8 9 3 | 7 10 11 5
------+------+------+------+------
 4 7 5 1 | 2 14 11 8 | 9 6 3 12 | 10 16 13 15
 3 16 12 11 | 9 6 1 10 | 14 15 5 13 | 8 4 2 7
 13 6 9 10 | 4 5 15 16 | 11 2 7 8 | 1 12 3 14
 15 8 2 14 | 13 7 12 3 | 1 10 4 16 | 6 11 5 9
------+------+------+------+------
 10 12 1 9 | 3 4 13 11 | 2 14 6 7 | 5 15 16 8
 5 2 16 6 | 8 1 7 15 | 4 9 13 11 | 3 14 10 12
 8 11 7 13 | 5 2 16 14 | 10 3 12 15 | 9 6 1 4
 14 15 3 4 | 12 10 6 9 | 8 16 1 5 | 11 13 7 2
------+------+------+------+------
```

p2.

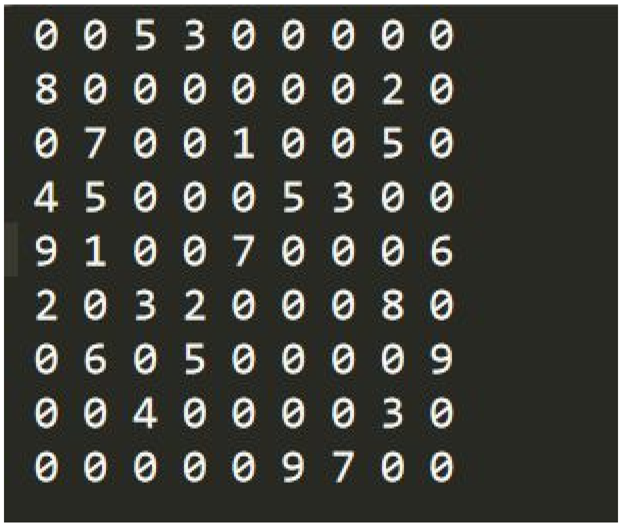


```
0 0 5 3 0 0 0 0 0
8 0 0 0 0 0 0 2 0
0 7 0 0 1 0 0 5 0
4 0 0 0 0 5 3 0 0
0 1 0 0 7 0 0 0 6
0 0 3 2 0 0 0 8 0
0 6 0 5 0 0 0 0 9
0 0 4 0 0 0 0 3 0
0 0 0 0 0 9 7 0 0
```

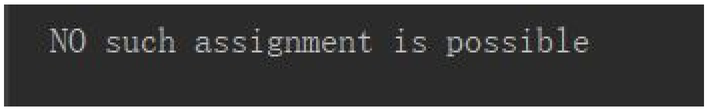


```
------+-------+-------+------
 6 4 5 | 3 9 2 | 8 7 1
 8 3 1 | 6 5 7 | 9 2 4
 2 7 9 | 8 1 4 | 6 5 3
------+-------+-------+------
 4 2 6 | 1 8 5 | 3 9 7
 5 1 8 | 9 7 3 | 2 4 6
 7 9 3 | 2 4 6 | 1 8 5
------+-------+-------+------
 3 6 7 | 5 2 8 | 4 1 9
 9 8 4 | 7 6 1 | 5 3 2
 1 5 2 | 4 3 9 | 7 6 8
------+-------+-------+------
```

p3.

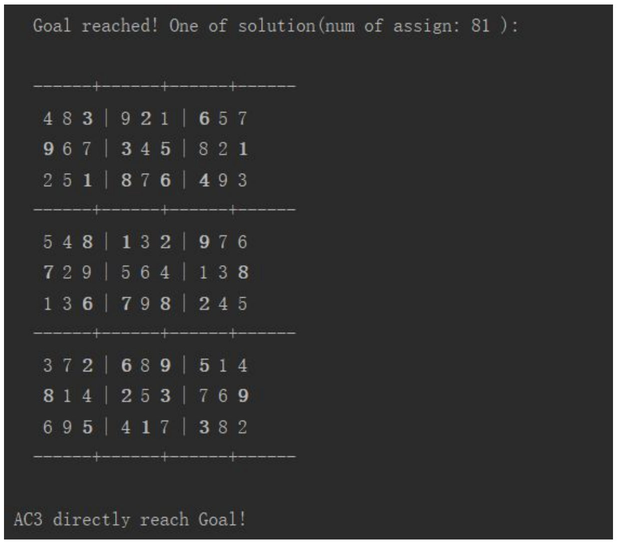


```
0 0 5 3 0 0 0 0 0
8 0 0 0 0 0 0 2 0
0 7 0 0 1 0 0 5 0
4 5 0 0 0 5 3 0 0
9 1 0 0 7 0 0 0 6
2 0 3 2 0 0 0 8 0
0 6 0 5 0 0 0 0 9
0 0 4 0 0 0 0 3 0
0 0 0 0 0 9 7 0 0
```



```
NO such assignment is possible
```

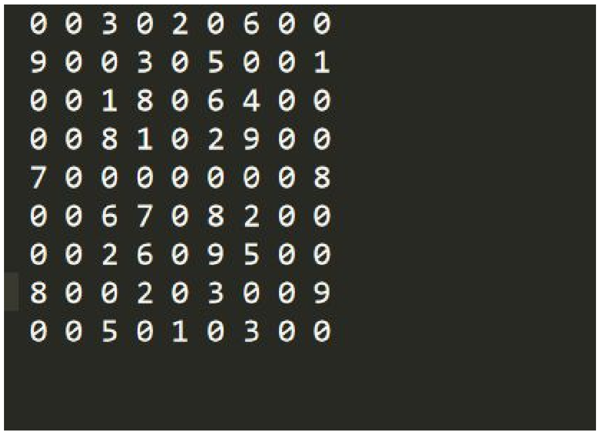
P4



```
  Goal reached! One of solution(num of assign: 81 ):


  ------+------+------+------
   4 8 3 | 9 2 1 | 6 5 7
   9 6 7 | 3 4 5 | 8 2 1
   2 5 1 | 8 7 6 | 4 9 3
  ------+------+------+------
   5 4 8 | 1 3 2 | 9 7 6
   7 2 9 | 5 6 4 | 1 3 8
   1 3 6 | 7 9 8 | 2 4 5
  ------+------+------+------
   3 7 2 | 6 8 9 | 5 1 4
   8 1 4 | 2 5 3 | 7 6 9
   6 9 5 | 4 1 7 | 3 8 2
  ------+------+------+------


AC3 directly reach Goal!
```



```
0 0 3 0 2 0 6 0 0
9 0 0 3 0 5 0 0 1
0 0 1 8 0 6 4 0 0
0 0 8 1 0 2 9 0 0
7 0 0 0 0 0 0 0 8
0 0 6 7 0 8 2 0 0
0 0 2 6 0 9 5 0 0
8 0 0 2 0 3 0 0 9
0 0 5 0 1 0 3 0 0
```