Data Structures and Algorithms Project on Sudoku Solver (for n>=3 and n<=10)

By,

Ankit Arora

MT2014007

Index

1. Techniques used to solve the Sudoku:

The given sudoku can be of varying size ranging from N=3 to N=10. Also, the difficulty levels could be varying. So, many techniques are used to tackle as many puzzles as possible:

1. Crosshatching: A simple technique which can be used alone to solve easy puzzles.
2. Rule 1: Rule 1 corresponds to single candidate square rule which is used to fill a single candidate in a cell, if that candidate is the only possibility for that cell.
3. Rule 2: Rule 2 corresponds to single square candidate square rule which is used to traverse the sudoku areawise(box, row and column) and fill a cell which has the count for that candidate as 1.
4. Pencilling in: It is required to prepare a candidate list for each cell of the sudoku.
5. Rule 3: Rule 3 correponds to pairing rule. It targets to reduce the candidate list of the cells of sudoku.
6. Rule 4: Rule 4 corresponds to generic grouping rule(triples, quadruples, etc.). It targets to reduce the candidate list of the cells of sudoku.

2. Implementation of Crosshatching:

Crosshatching is “finding squares for numbers”.

Before crosshatching, I found out the missing numbers for each box. A 2 dimensional array missing_num which is the size of entire sudoku is initialized for each box. Each row of this array correspond to each box of the sudoku. The (column index+1) corresponds to numbers to be filled in each box. If a number is there in the box, initialize that particular location of missing_num to 1, otherwise to 0.

Crosshatching works in boxes. Traverse boxwise the entire sudoku puzzle and also keep track of the corresponding location in the missing_num array. With the known candidate list of each box (via missing_num array), for each candidate traverse the boxes and look out the entire row and entire column at each location of every box (if there is a filled sudoku cell, move on to the next location in the box). If that candidate fits that location, increase a counter. Do this for the entire box and if at the end, count is 1, then assign that candidate to that cell, otherwise move on.

Eg:

0 1 2 0	Here, consider box 1:	Candidate list: 3, 4
2 0 3 0	0 1	
0 0 0 0	2 0	
1 2 0 0		

After checking for 3 in the entire 1st box, it was found out that candidate count=1 for 3 in the (0,0) location. So, assign 3 to the 1st cell. Similarly traverse the box for 4. The candidate 4 is thus assigned to cell (1,1). In the same way the puzzle can be solved for other boxes.

3. Implementation of Pencilling in.

The solving techniques needed for more difficult puzzles all depend on having an accurate list of the possible numbers (called 'candidates') for each empty square.

It's called 'pencilling-in' because on a printed puzzle many people use a pencil, so that they can rub numbers out later (an essential part of the solving process).

Here's the box with all its missing numbers pencilled into their possible squares:

4	0(1->5->9)	0
0(5->9)	8	3
7	2	6

This list must be complete and accurate, otherwise you risk creating another chain of errors. That's why it's essential to crosshatch every missing number for every box before starting the next stage of solving.

Whenever you fill in a square, remove the number you've used from all candidate lists in the same row, column and box.

In my code, pencilling in technique is being done in the find_candidates function. An array of candidate list (linked list) is formed for each cell. Corresponding to this array, a count_candidate array is created which contains the number of candidates in the list for each cell.

Pencilling in is used for all the further rules that follow.

4. Implementation of Rule-1 (Single Candidate Squares)

This part of the solving process is where you switch approach and start finding numbers for squares instead of squares for numbers. You do this by checking your pencilled-in candidate lists for a series of rules (or 'candidate patterns'), and acting on them.

When a square has just one candidate, that number goes into the square.

Gven a box below, with candidate list of each cell:

Eg:

1	0(6->9)	0(4->9)
2	7	0(4)
0(5->8)	3	0(5)

For the cell (1,2) the candidate list is having only 4. So, 4 goes into that cell and then 4 is removed from the candidate lists of entire area that this cell covers (i.e. the corresponding box, row and column). Similarly, it can be done for (2,2) cell.

Some squares will be single-candidate from the start of the puzzle. Most, however, will start with multiple candidates and gradually reduce down to single-candidate status.

This will happen as you remove numbers that you've placed in other squares in the same row, column and box, and as you apply the last two rules described below.

5. Implementation of Rule 2 (Single-Square Candidates)

When a candidate number appears just once in an area (row, column or box), that number goes into the square.

Look at the mid-left box again:

1	0(6->9)	0(4->9)
2	7	0(4)
0(5->8)	3	0(5)

The number 6 only appears in one square's candidate list within this box (top-middle). This must, therefore, be the right place for the 6.

Now 6 can be removed from the candidate list of cells in this row and column.

Here in the example, the rule is applied boxwise. The rule is also implemented in the code rowwise and columnwise.

The remaining two rules let you remove numbers from candidate lists, reducing them down towards meeting one of the first two rules.

Logic: For an area (consider boxwise), traverse the box given above and create an integer temp array indexed from 0 to N*N (initialize to 0) and corresponding to candidates in the given cells of the box, increase the count at that index(eg: for candidate 6, temp[5]++). After filling this array for a box, traverse this temp array and for a candidate with count=1, traverse the box and fill the candidate at that location(In above case: 6). After filling of this box, remove the correponding candidate from corresponding candidate lists of each cell in that row and column (corresponding to the candidate 6).

6. Implementation of Rule 3 - Pairing

When two squares in the same area (row, column or box) have identical two-number candidate lists, you can remove both numbers from other candidate lists in that area.

Here's the second row of the puzzle:

5	8	3	1	0(6->7)	4	0(6->9)	0(6->7)	2

Two of the squares have the same candidate list - 67. This means that between them, they will use up the 6 and 7 for this row.

That means that the other square can't possibly contain a 6. We can remove the 6 from its candidate list, leaving just 9 - square solved!

The squares in a pair must have exactly two candidates. If one of the above squares had been 679, it couldn't have been part of a pair.

Logic: Initialize an array pairing check with all 0s. Check for candidate count=2 in an area(row, box and column in separate functions) and make pairing_check=1 for cell under consideration. Again traverse the corresponding area and find another cell with candidate count=2. Make pairing check=1 for that other cell if the previous cell and the other cell are forming a pair. Now, remove the corresponding candidates from the corresponding area except the ones marked as 1 in pairing check.

7. Implementation of Rule 4 – Generic Grouping including triples,quadruples,etc.

Case for a triple:

Three squares in an area (row, column or box) form a triple when:

- None of them has more than three candidates.
- Their candidate lists are all full or sub sets of the same three-candidate list (explained below!).

Similarly, for generic grouping, above 2 points apply(replace 3 by N).

You can remove numbers that appear in the triple from other candidate lists in the same area.

Here's a row of a puzzle:

1	6	0(4->9)	0(2->3)	0(2->3)	7	0(2->3->4)	5	0(4->8)

Note the three squares in the middle, with candidates of 23, 23 and 234. These form a triple.

234 is the full, three-candidate list, and 23 is a subset of it (i.e. all its numbers appear in the full list). Because there are three squares, and none of them have any candidate numbers outside of those in the three-candidate list, they must use up the three candidate numbers (2, 3 and 4) between them.

This lets us remove the 4 from the other two candidate lists in this row, solving their squares.

It's worth looking hard for subset triples. In this example, the 23 lists make an obvious pair (see above), but it's the triple that instantly solves the two outside squares (once you've dispensed with them, you can treat the 23s as a pair again, and use them to solve the 234!).

Note - the squares in a pair or triple don't have to appear next to each other, or in any particular order. In the example above, the triple could have occurred in, say, the first, third and fifth empty squares of the row, with the 234 in the middle.

The triple rule can be true even if none of the squares have three candidates. Take these three candidate lists:

1->3   1->6   3->6

All three lists are subsets of the list 136. Between them, these three squares will use up the 1, 3 and 6 for the area they're in.

So, there are 2 rules implemented: rule 4a(takes care of 1st kind of grouping above) and rule 4b(takes care of the latter grouping as seen above).

Rule 4a: Example-169,16,19

A mark array is initialized to 0. Then, we look out for cells containing candidate count=M(Here 3). Mark 1 for “cell under consideration”. Mark 2 for “grouping found” (keeping track by the count==M-1where M=size of group. Here, M=3 for triples). Mark -1 for “temporarirly grouping under consideration”. Mark all -1 to 2 and specific cell under consideration to 2, if grouping is found. If at last, count does not become M-1 (for rest of the M-1 cells except the cell under consideration), turn all -1 to 0. Now remove from that corresponding area the candidates of the grouping except the cells marked 2. Same procedure is done for row, column and box.

Rule 4b: Example-16,69,19

A mark array is initialized to 0. Then, we look out for cells containing candidate count>=2 and <M(Here 3). Traverse the entire area and mark 1 for "cell under consideration". For the cell under consideration, look out for cells containing candidate count>=2 and candidate count<M(Here 3) and mark=0, mark the cell as -N. For the cell marked as -N, look out for cells containing candidate count>=2 and candidate count<M(Here 3) and mark=0, mark cell as -(N-1). Increase possible count and then check for possible count to be =M-2. If it is found, grouping is there, mark the cell under consideration as 2 and all the negative marked cells as 2 except the cell marked -(N+1). Now, remove from the corresponding area, the candidates except from the cells marked as 2. If grouping is not found, mark the cell marked as -N to -(N+1) and all the other negative cells to 0. Now the cell marked as -(N+1) will not be considered for grouping with cell marked as 1. A possible candidate array is used to compare the entire candidate list with the corresponding cell candidate list.

Similar logic is used for all the areas including box, row and column.

Solving Checklist:

Here's a quick checklist of the solving plan for tough puzzles.

1. **Crosshatch the entire puzzle box-by-box.**
2. **Pencilling-in complete candidate lists.**
3. **Scan the puzzle** for the following rules:

- **Single-candidate squares** - solve immediately
- **Single-square candidates** within an area (row/column/box) - solve immediately.
- **Pairs** within an area - remove the pair squares' candidates from other lists within that area.
- **Generalized grouping including triples, quadruples, etc.** within an area - remove the grouped candidates from others lists within that area.

4. **Whenever you solve a square,** immediately check and update all candidate lists in the same row, column and box.

5. **Whenever you've updated a candidate list,** check to see if one of the rules now applies (e.g. you've created a triple). We can put the entire thing in a loop.

8. Testcase

Input: for N=6

```
31 4 0 0 34 16 20 22 26 0 12 27 3 0 32 0 13 14 17 28 0 6 0 21 11 30 0 19 33 15 9 18 0 0 10 24
0 15 10 12 24 25 0 0 36 0 1 35 8 0 0 30 2 23 16 7 11 0 0 32 4 6 0 31 0 0 19 3 20 29 34 0
27 0 14 0 0 19 0 0 0 0 0 25 28 17 0 0 0 0 0 0 0 29 18 7 0 0 0 0 0 13 0 0 22 0 30
13 2 0 0 17 0 0 10 0 18 19 0 24 0 0 0 0 11 30 0 0 0 0 8 0 16 32 0 29 0 0 7 0 0 33 36
23 21 0 0 0 0 0 0 7 0 0 30 0 0 15 0 0 18 13 0 0 24 0 0 34 0 0 1 0 0 0 0 0 0 2 6
1 0 36 11 29 0 0 13 0 0 8 34 7 0 6 27 0 0 0 0 2 9 0 20 10 17 0 0 3 0 0 26 12 14 0 31
6 0 0 19 0 0 13 0 22 0 30 0 35 16 0 4 0 0 0 0 26 0 15 3 0 29 0 25 0 1 0 0 24 0 0 2
2 3 0 35 10 0 0 33 6 7 5 4 0 27 30 0 12 0 0 13 0 21 8 0 17 23 28 20 11 0 0 29 22 0 26 25
0 16 12 20 0 21 9 0 0 23 29 15 0 0 13 24 31 0 0 18 17 25 0 0 19 36 10 0 0 8 14 0 30 3 4 0
0 14 11 34 23 36 10 25 21 0 28 31 0 5 0 0 29 0 0 35 0 0 9 0 27 7 0 30 24 13 15 19 6 1 8 0
7 29 13 0 25 33 8 24 20 0 0 0 32 10 0 21 0 0 0 0 12 0 28 23 0 0 0 3 2 26 11 36 0 5 27 9
0 22 24 0 0 0 18 19 0 0 2 0 0 36 0 11 0 8 29 0 7 0 32 0 0 15 0 0 21 33 0 0 0 28 20 0
9 6 0 21 11 27 29 0 30 10 0 2 16 0 0 7 0 25 26 0 33 0 0 24 20 0 31 28 0 4 23 35 13 0 19 8
29 0 2 0 0 20 31 32 27 0 16 0 21 34 0 23 6 28 15 3 18 0 13 25 0 19 0 9 8 30 26 0 0 11 0 14
17 1 0 24 16 0 0 28 19 22 14 0 0 0 4 3 11 0 0 8 21 29 0 0 0 25 26 27 34 0 0 32 7 0 9 20
12 0 0 0 0 34 24 18 0 20 36 0 22 0 9 19 0 0 0 0 5 31 0 2 0 13 15 0 35 7 25 0 0 0 0 28
0 0 0 13 5 3 0 0 33 0 7 11 27 2 20 15 8 24 14 9 28 32 1 30 23 18 0 16 0 0 34 6 10 0 0 0
0 7 18 0 0 28 23 6 13 9 0 5 36 26 0 0 0 30 20 0 0 0 27 17 29 0 11 24 22 10 1 0 0 2 16 0
0 20 19 0 0 23 35 2 31 25 0 33 18 11 0 0 0 26 36 0 0 0 12 6 22 0 34 7 17 32 5 0 0 9 3 0
0 0 0 14 12 11 0 0 10 0 23 7 20 8 34 36 28 5 9 32 19 17 21 35 18 26 0 15 0 0 24 2 4 0 0 0
36 0 0 0 0 18 14 30 0 21 11 0 9 0 24 16 0 0 0 0 22 3 0 7 0 20 25 0 28 5 10 0 0 0 0 19
5 24 0 26 7 0 0 16 8 34 32 0 0 0 19 2 35 0 0 1 31 13 0 0 0 11 3 4 30 0 0 20 27 0 28 21
30 0 21 0 0 2 19 29 28 0 26 0 31 3 0 25 14 7 10 20 23 0 33 15 0 9 0 8 12 27 22 0 0 18 0 32
3 28 0 22 13 8 5 0 15 36 0 9 12 0 0 32 0 17 18 0 4 0 0 34 14 0 19 6 0 16 7 25 31 0 35 29
0 27 4 0 0 0 7 8 0 0 18 0 0 33 0 12 0 35 34 0 15 0 19 0 0 32 0 0 9 14 0 0 0 6 5 0
35 30 23 0 33 14 12 15 5 0 0 0 2 4 0 1 0 0 0 0 27 0 17 31 0 0 0 21 6 34 29 16 0 7 13 26
0 25 31 2 6 7 27 20 34 0 35 14 0 30 0 0 3 0 0 10 0 0 24 0 1 12 0 33 4 11 17 9 19 15 23 0
0 34 5 1 0 12 30 0 0 33 17 24 0 0 28 26 21 0 0 6 3 14 0 0 15 27 8 0 0 31 4 0 32 10 25 0
11 13 0 18 15 0 0 21 1 32 4 3 0 31 27 0 7 0 0 12 0 23 36 0 25 10 20 2 16 0 0 33 14 0 30 34
28 0 0 32 0 0 2 0 16 0 9 0 15 14 0 17 0 0 0 0 30 0 11 4 0 5 0 26 0 19 0 0 1 0 0 27
34 0 3 33 2 0 0 5 0 0 31 10 14 0 26 9 0 0 0 0 6 30 0 19 32 4 0 0 25 0 0 27 29 35 0 15
25 9 0 0 0 0 0 0 23 0 0 32 0 0 11 0 0 1 3 0 0 26 0 0 2 0 0 10 0 0 0 0 0 0 7 33
10 35 0 0 21 0 0 12 0 15 13 0 25 0 0 0 0 27 24 0 0 0 0 33 0 22 9 0 26 0 0 28 0 0 31 23
24 0 17 0 0 32 0 0 0 0 0 29 6 21 0 0 0 0 0 0 0 31 12 28 0 0 0 0 0 30 0 0 13 0 10
0 12 1 36 28 29 0 0 4 0 22 17 23 0 0 18 10 31 32 15 25 0 0 11 21 24 0 13 0 0 6 5 9 34 14 0
19 23 0 0 30 6 34 35 2 0 27 21 29 0 16 0 15 36 7 17 0 4 0 9 8 33 0 5 1 12 20 11 0 0 18 22
```

Output:

```
31 4 7 8 34 16 20 22 26 2 12 27 3 1 32 29 13 14 17 28 36 6 25 21 11 30 5 19 33 15 9 18 35 23 10 24
18 15 10 12 24 25 21 17 36 28 1 35 8 22 33 30 2 23 16 7 11 27 26 32 4 6 13 31 14 9 19 3 20 29 34 5
27 5 14 6 20 19 3 9 32 16 15 25 28 17 31 10 26 21 12 4 34 33 29 18 7 35 2 36 23 24 13 8 11 22 1 30
13 2 22 28 17 35 6 10 14 18 19 23 24 9 5 34 20 11 30 31 1 15 3 8 26 16 32 12 29 25 27 7 21 4 33 36
23 21 32 3 9 26 11 31 7 29 33 30 4 12 15 35 36 18 13 19 10 24 5 14 34 8 27 1 20 22 16 17 28 25 2 6
1 33 36 11 29 30 4 13 24 5 8 34 7 25 6 27 16 19 23 22 2 9 35 20 10 17 21 18 3 28 32 26 12 14 15 31
6 8 28 19 18 17 13 27 22 14 30 36 35 16 7 4 23 20 5 11 26 34 15 3 9 29 12 25 31 1 33 10 24 21 32 2
2 3 9 35 10 1 32 33 6 7 5 4 34 27 30 14 12 15 19 13 24 21 8 36 17 23 28 20 11 18 31 29 22 16 26 25
26 16 12 20 27 21 9 11 35 23 29 15 1 28 13 24 31 2 33 18 17 25 6 22 19 36 10 32 5 8 14 34 30 3 4 7
32 14 11 34 23 36 10 25 21 12 28 31 26 5 18 33 29 3 4 35 20 2 9 16 27 7 22 30 24 13 15 19 6 1 8 17
7 29 13 15 25 33 8 24 20 17 34 16 32 10 22 21 19 6 31 30 12 1 28 23 35 14 4 3 2 26 11 36 18 5 27 9
4 22 24 30 31 5 18 19 3 1 2 26 17 36 25 11 9 8 29 14 7 10 32 27 16 15 6 34 21 33 35 12 23 28 20 13
9 6 15 21 11 27 29 34 30 10 3 2 16 18 14 7 5 25 26 36 33 12 22 24 20 1 31 28 32 4 23 35 13 17 19 8
29 10 2 4 36 20 31 32 27 35 16 1 21 34 17 23 6 28 15 3 18 7 13 25 12 19 33 9 8 30 26 24 5 11 22 14
17 1 30 24 16 31 15 28 19 22 14 12 33 35 4 3 11 13 6 8 21 29 23 10 5 25 26 27 34 2 18 32 7 36 9 20
12 26 33 23 32 34 24 18 17 20 36 8 22 29 9 19 1 10 11 16 5 31 4 2 6 13 15 14 35 7 25 30 3 27 21 28
22 19 35 13 5 3 25 4 33 26 7 11 27 2 20 15 8 24 14 9 28 32 1 30 23 18 17 16 36 21 34 6 10 31 29 12
14 7 18 25 8 28 23 6 13 9 21 5 36 26 12 31 32 30 20 34 35 19 27 17 29 3 11 24 22 10 1 15 33 2 16 4
15 20 19 10 4 23 35 2 31 25 24 33 18 11 29 13 30 26 36 27 16 28 12 6 22 21 34 7 17 32 5 14 8 9 3 1
33 31 25 14 12 11 22 3 10 27 23 7 20 8 34 36 28 5 9 32 19 17 21 35 18 26 1 15 13 29 24 2 4 30 6 16
36 32 34 29 1 18 14 30 12 21 11 13 9 6 24 16 27 4 8 26 22 3 2 7 31 20 25 35 28 5 10 23 15 33 17 19
5 24 6 26 7 9 17 16 8 34 32 18 10 15 19 2 35 22 25 1 31 13 14 29 33 11 3 4 30 23 36 20 27 12 28 21
30 17 21 16 35 2 19 29 28 4 26 6 31 3 1 25 14 7 10 20 23 5 33 15 24 9 36 8 12 27 22 13 34 18 11 32
3 28 27 22 13 8 5 1 15 36 20 9 12 23 21 32 33 17 18 24 4 11 30 34 14 2 19 6 10 16 7 25 31 26 35 29
20 27 4 17 26 10 7 8 25 31 18 28 13 33 23 12 24 35 34 29 15 16 19 1 36 32 30 22 9 14 3 21 2 6 5 11
35 30 23 9 33 14 12 15 5 11 10 19 2 4 8 1 18 32 22 25 27 20 17 31 3 28 24 21 6 34 29 16 36 7 13 26
21 25 31 2 6 7 27 20 34 13 35 14 5 30 36 22 3 16 28 10 32 8 24 26 1 12 29 33 4 11 17 9 19 15 23 18
16 34 5 1 19 12 30 36 29 33 17 24 11 20 28 26 21 9 2 6 3 14 7 13 15 27 8 23 18 31 4 22 32 10 25 35
11 13 8 18 15 22 26 21 1 32 4 3 19 31 27 6 7 29 35 12 9 23 36 5 25 10 20 2 16 17 28 33 14 24 30 34
28 36 29 32 3 24 2 23 16 6 9 22 15 14 10 17 25 34 21 33 30 18 11 4 13 5 35 26 7 19 8 31 1 20 12 27
34 11 3 33 2 13 28 5 18 8 31 10 14 7 26 9 22 12 1 23 6 30 20 19 32 4 16 17 25 36 21 27 29 35 24 15
25 9 20 27 22 15 36 14 23 24 6 32 30 13 11 5 17 1 3 21 29 26 34 28 2 31 18 10 19 35 12 4 16 8 7 33
10 35 16 7 21 4 1 12 11 15 13 20 25 32 3 8 34 27 24 5 14 36 18 33 30 22 9 29 26 6 2 28 17 19 31 23
24 18 17 5 14 32 16 7 9 19 25 29 6 21 35 20 4 33 27 2 8 22 31 12 28 34 23 11 15 3 30 1 26 13 36 10
8 12 1 36 28 29 33 26 4 30 22 17 23 19 2 18 10 31 32 15 25 35 16 11 21 24 7 13 27 20 6 5 9 34 14 3
19 23 26 31 30 6 34 35 2 3 27 21 29 24 16 28 15 36 7 17 13 4 10 9 8 33 14 5 1 12 20 11 25 32 18 22
```