Sudoku problem (Backtracking)

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

Sudoku

- Given an incomplete Sudoku in the form of matrix mat[][] of order 9*9, the task is to complete the Sudoku.
- A sudoku solution must satisfy all of the following rules:

- Each of the digits 1-9 must occur exactly once in each row.
- Each of the digits 1-9 must occur exactly once in each column.
- Each of the digits 1-9 must occur exactly once in each of the 9, 3x3 sub-boxes of the grid.
- Note: Zeros in the mat[][] indicate blanks, which are to be filled with some number between 1 to 9. You can not replace the element in the cell which is not blank.

Sudoku

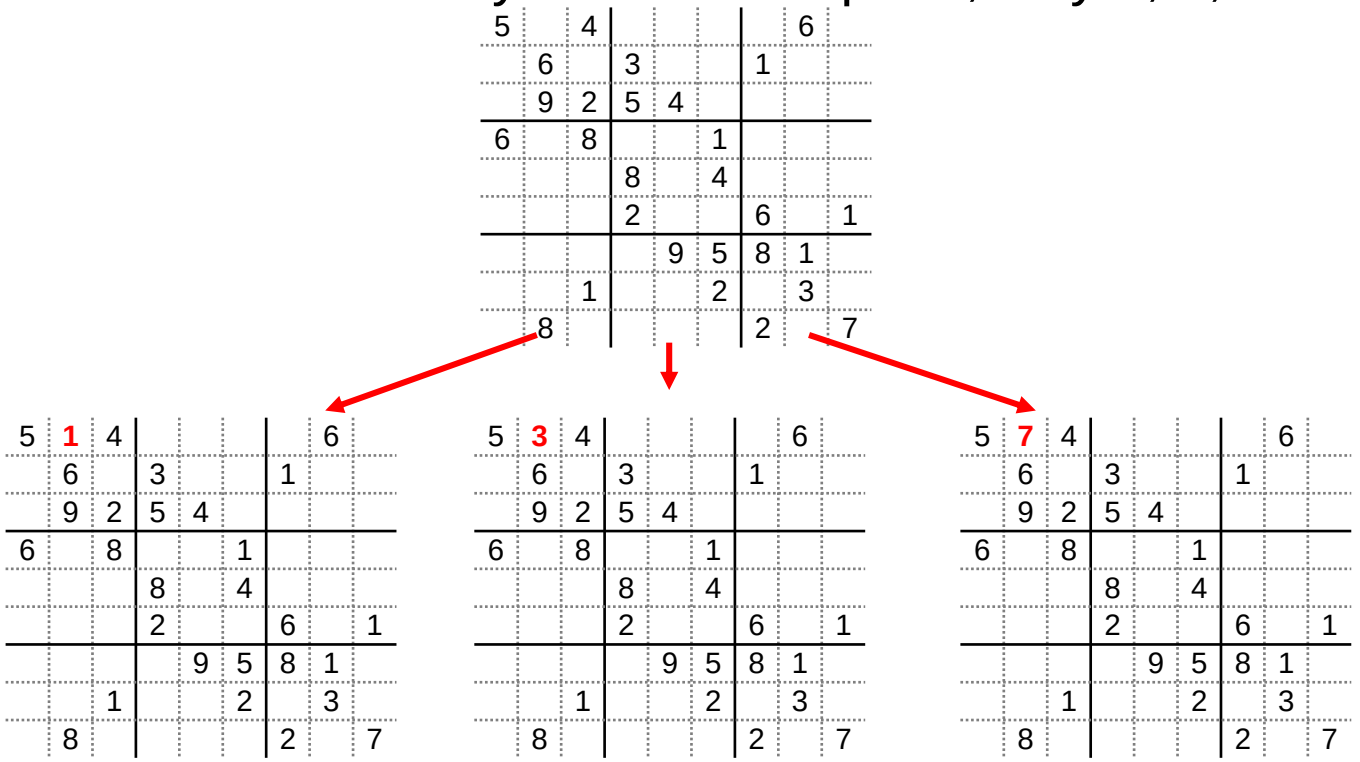
In the first case, consider the game Sudoku:

- The search space is 9^{53}

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

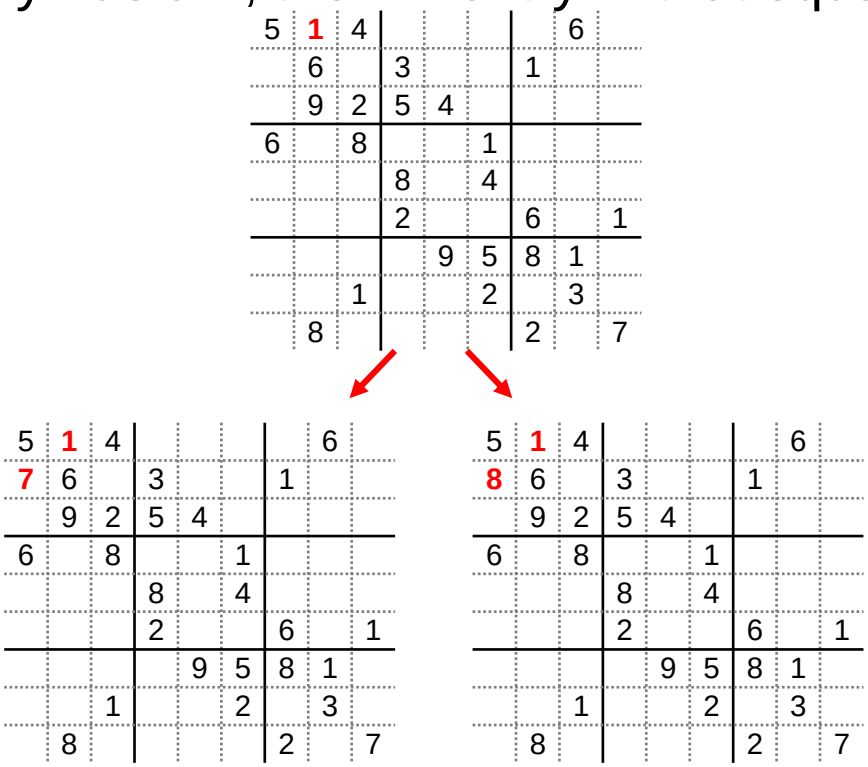
Sudoku

At least for the first entry in the first square, only 1, 3, 7 fit



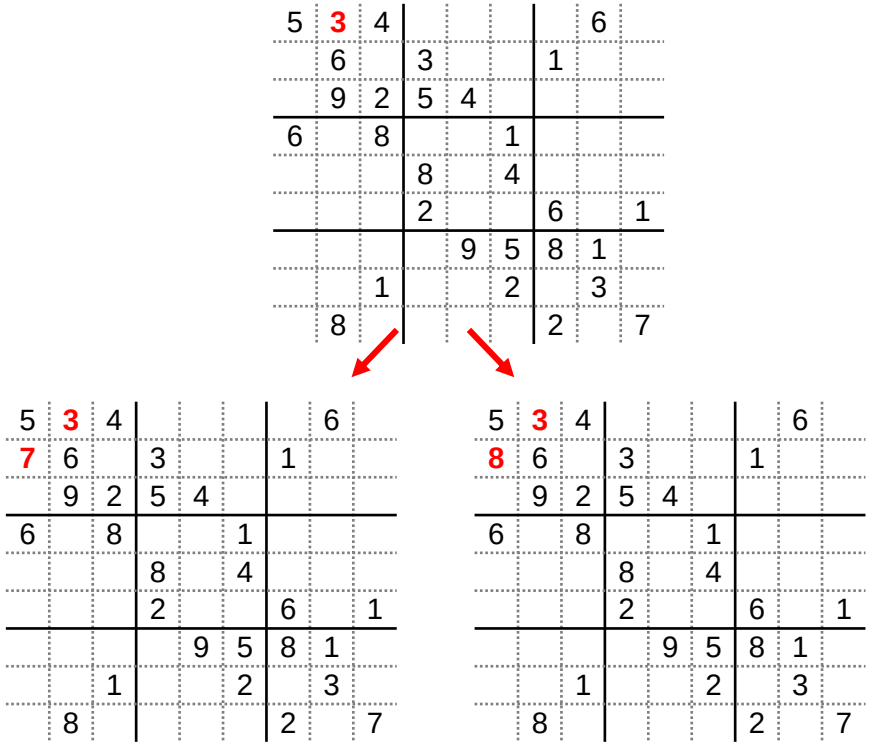
Sudoku

If the first entry has a 1, the 2nd entry in that square could be 7 or 8



Sudoku

If the first entry has a 3, the 2nd entry in that square could be 7 or 8



Sudoku

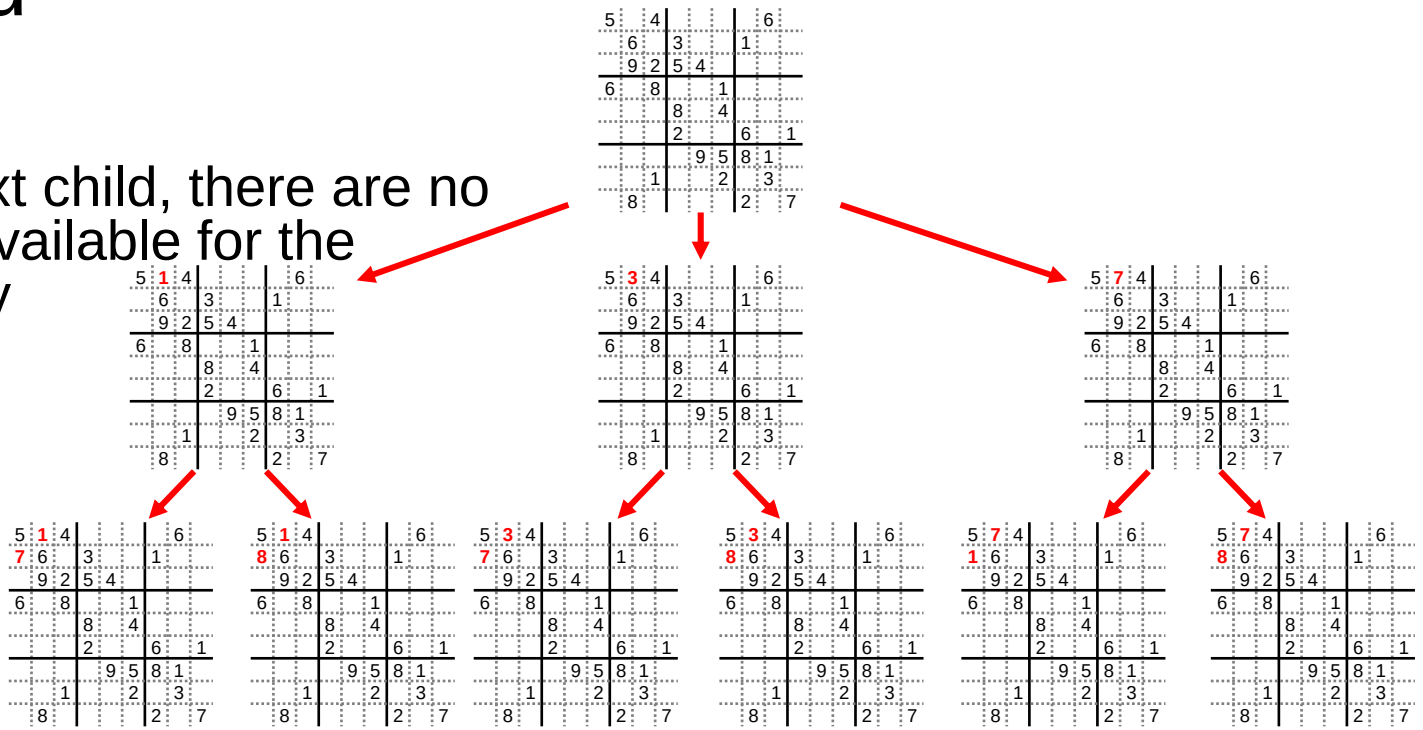
If the first entry has a 7, the 2nd entry in that square could be 8

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

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

Sudoku

In the next child, there are no options available for the next entry



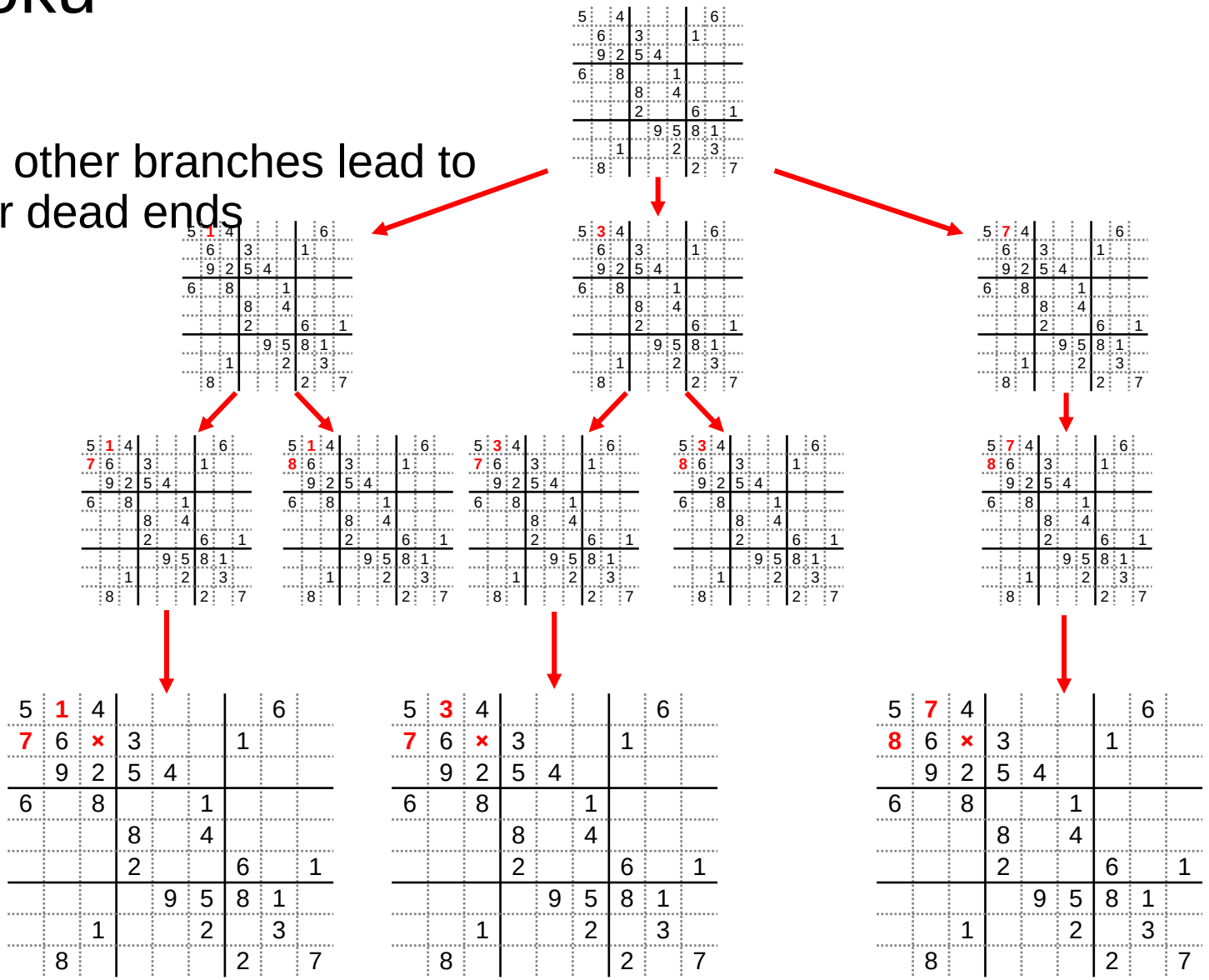
5	1	4					6	
7	6	×	3			1		
	9	2	5	4				
6		8			1			
			8		4			
			2			6		1
				9	5	8	1	
		1			2		3	
	8					2		7

Any candidate solution built from this partial is infeasible

– We can ignore this branch

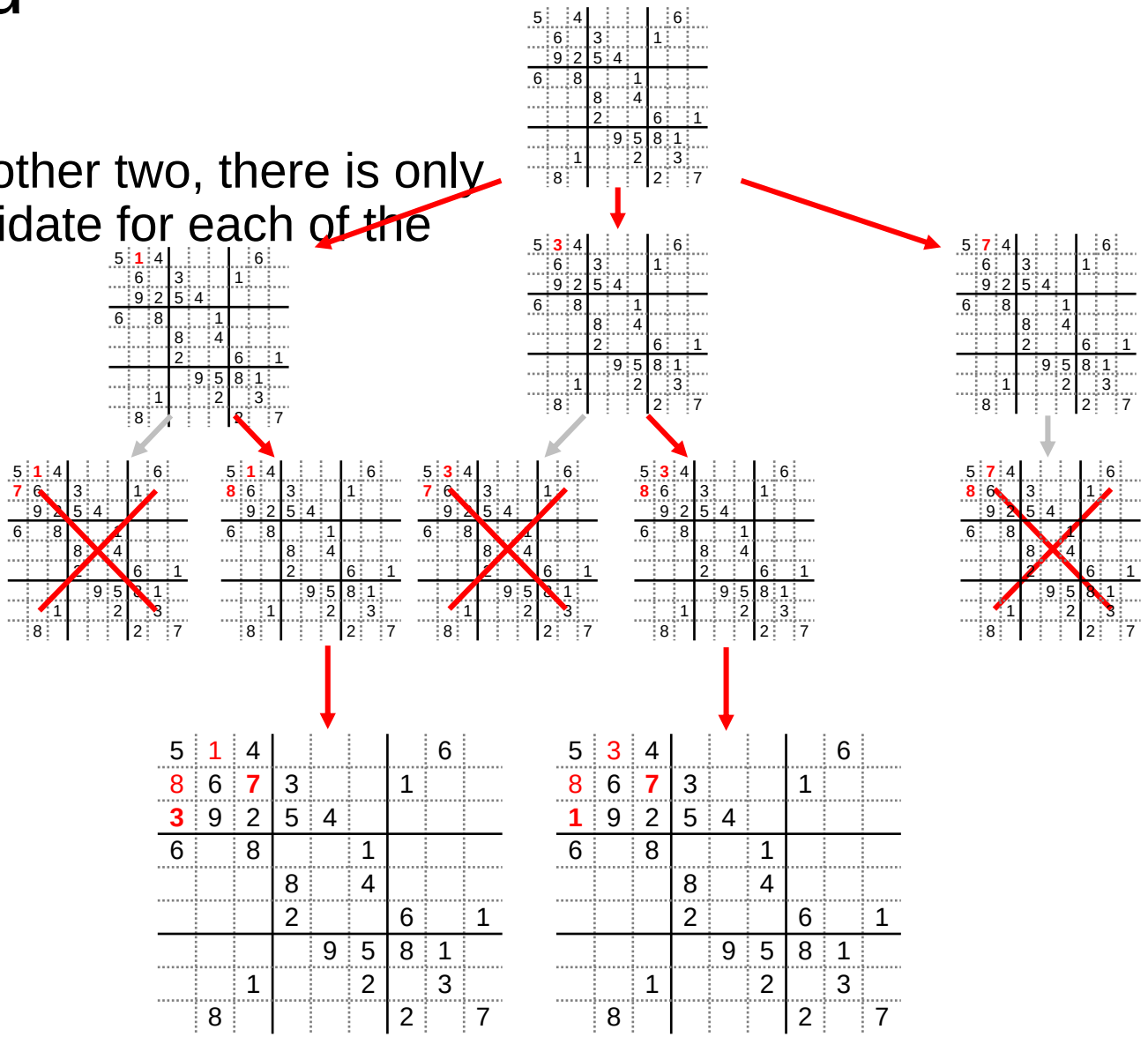
Sudoku

Three other branches lead to similar dead ends



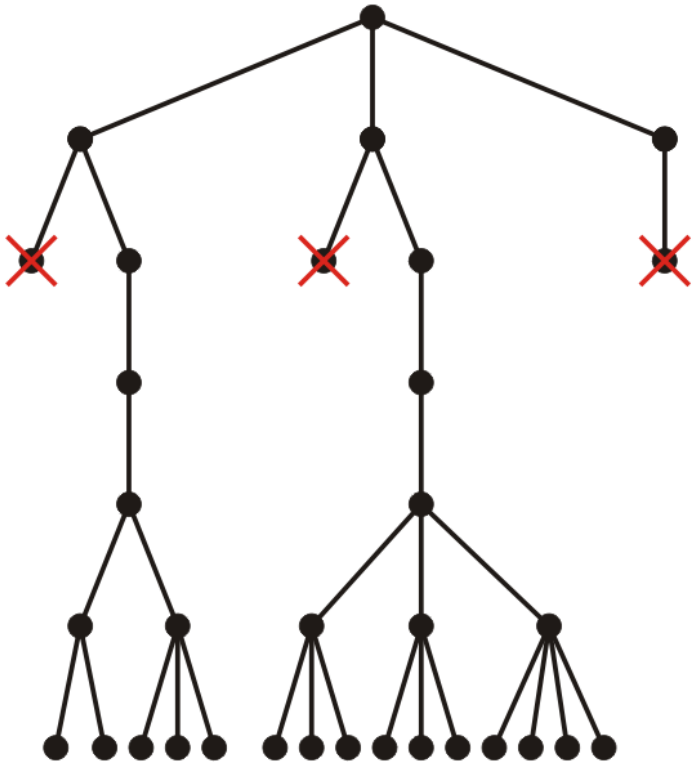
Sudoku

With the other two, there is only one candidate for each of the last two entries



Sudoku

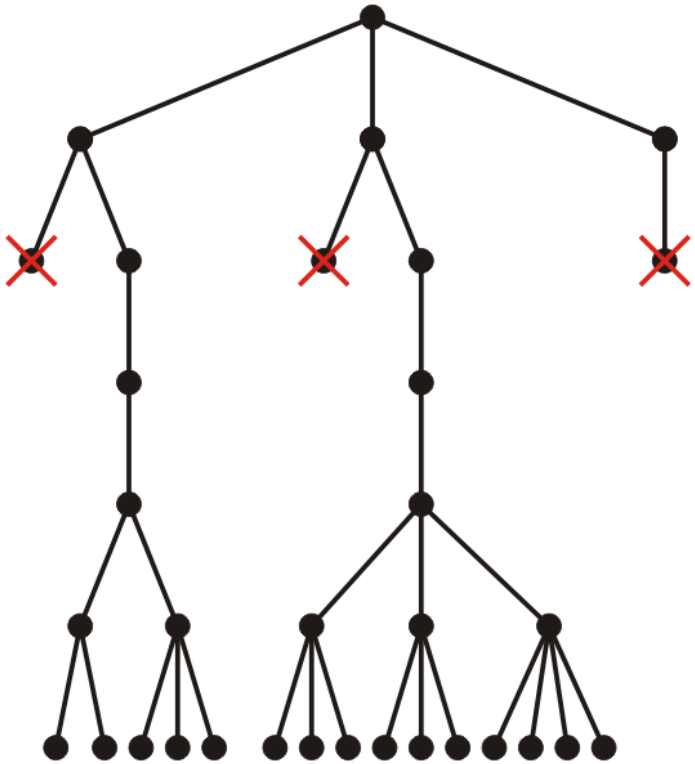
It may seem that this is a reasonably straight-forward method; however, the decision tree continues to branch quick once we start filling the second square



Sudoku

A binary tree of this height would have around $2^{54} - 1$ nodes

- Fortunately, as we get deeper into the tree, more get cut



Implementation

- The idea is to use backtracking and recursively generate all possible configurations of numbers from 1 to 9 to fill the empty cells of matrix mat[][]. To do so, for every unassigned cell, fill the cell with a number from 1 to 9 one by one. After filling the unassigned cell check if the matrix is safe or not. If safe, move to the next cell else backtrack for other cases.
- To check if it is safe to place value num in the cell mat[i][j], iterate through all the columns of row i, rows of column j and the 3*3 matrix containing cell (i, j) and check if they already has value num, if so return false, else return true.

Implementation

```
// Function to check if it is safe to place num at mat[row][col]
bool isSafe(vector<vector<int>> &mat, int row, int col, int num) {

    // Check if num exist in the row or col
    for (int x = 0; x <= 8; x++){
        if (mat[row][x] == num)
            return false;
        if (mat[x][col] == num)
            return false;
    }

    // Check if num exist in the 3x3 sub-matrix
    int startRow = row - (row % 3), startCol = col - (col % 3);

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (mat[i + startRow][j + startCol] == num)
                return false;

    return true;
}
```

Implementation

```
// Function to solve the Sudoku problem
bool solveSudokuRec(vector<vector<int>> &mat, int row, int col) {
    int n = mat.size();

    // base case: Reached nth column of last row
    if (row == n - 1 && col == n)
        return true;

    // If last column of the row go to next row
    if (col == n) {
        row++;
        col = 0;
    }

  // If cell is already occupied then move forward
    if (mat[row][col] != 0)
        return solveSudokuRec(mat, row, col + 1);
```

Implementation

```
for (int num = 1; num <= n; num++) {

// If it is safe to place num at current position
     if (isSafe(mat, row, col, num)) {
           mat[row][col] = num;
           if (solveSudokuRec(mat, row, col + 1))
                return true;
            mat[row][col] = 0;
        }
    }

         return false;
}
```

Implementation

```
int main() {
    vector<vector<int>> mat = {
        {3, 0, 6, 5, 0, 8, 4, 0, 0},
        {5, 2, 0, 0, 0, 0, 0, 0, 0},
        {0, 8, 7, 0, 0, 0, 0, 3, 1},
        {0, 0, 3, 0, 1, 0, 0, 8, 0},
        {9, 0, 0, 8, 6, 3, 0, 0, 5},
        {0, 5, 0, 0, 9, 0, 6, 0, 0},
        {1, 3, 0, 0, 0, 0, 2, 5, 0},
        {0, 0, 0, 0, 0, 0, 0, 7, 4},
        {0, 0, 5, 2, 0, 6, 3, 0, 0}};

        solveSudokuRec(mat, 0, 0);

     for (int i = 0; i < mat.size(); i++) {
        for (int j = 0; j < mat.size(); j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

Implementation

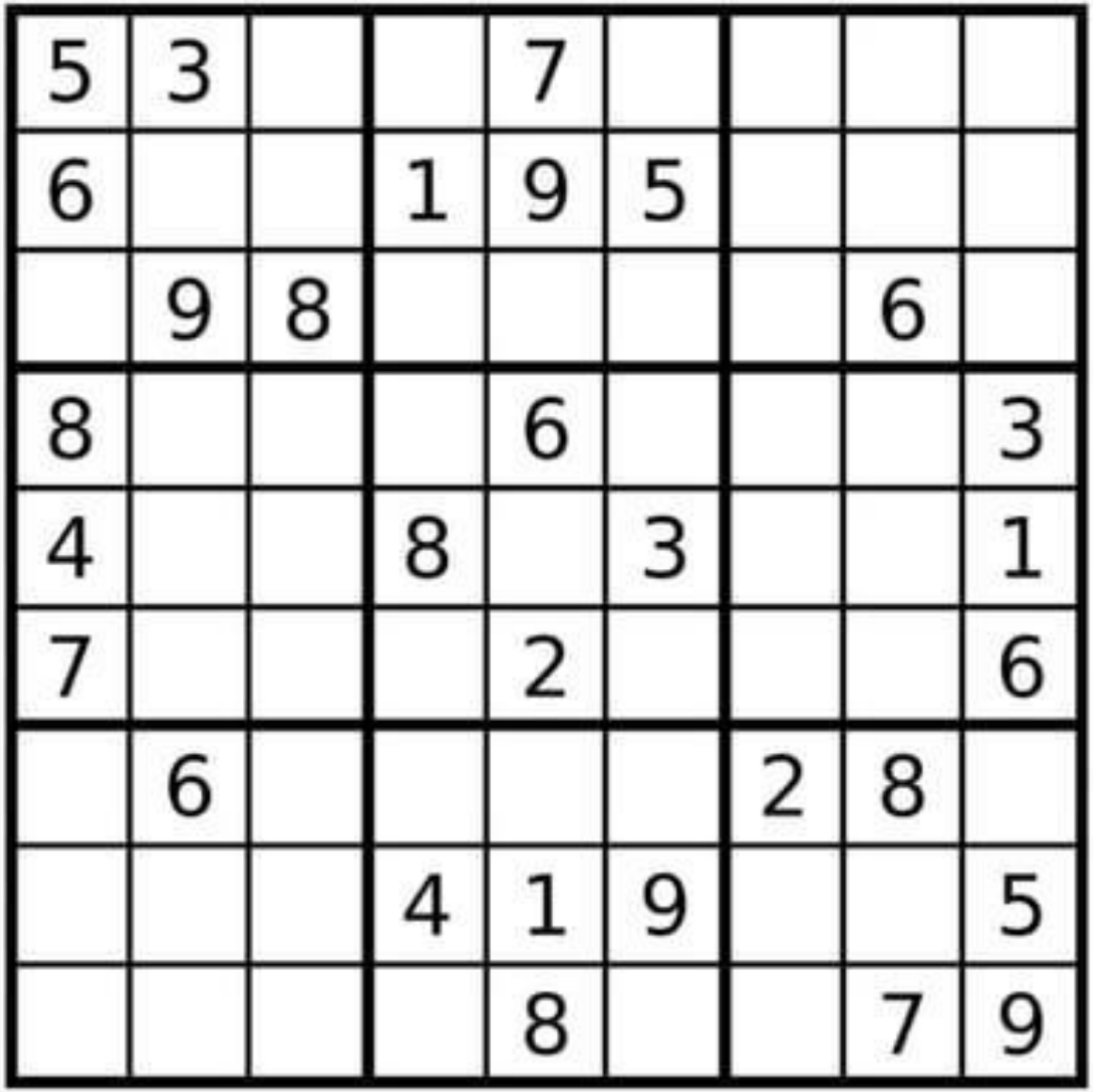
In this case, the traversal:

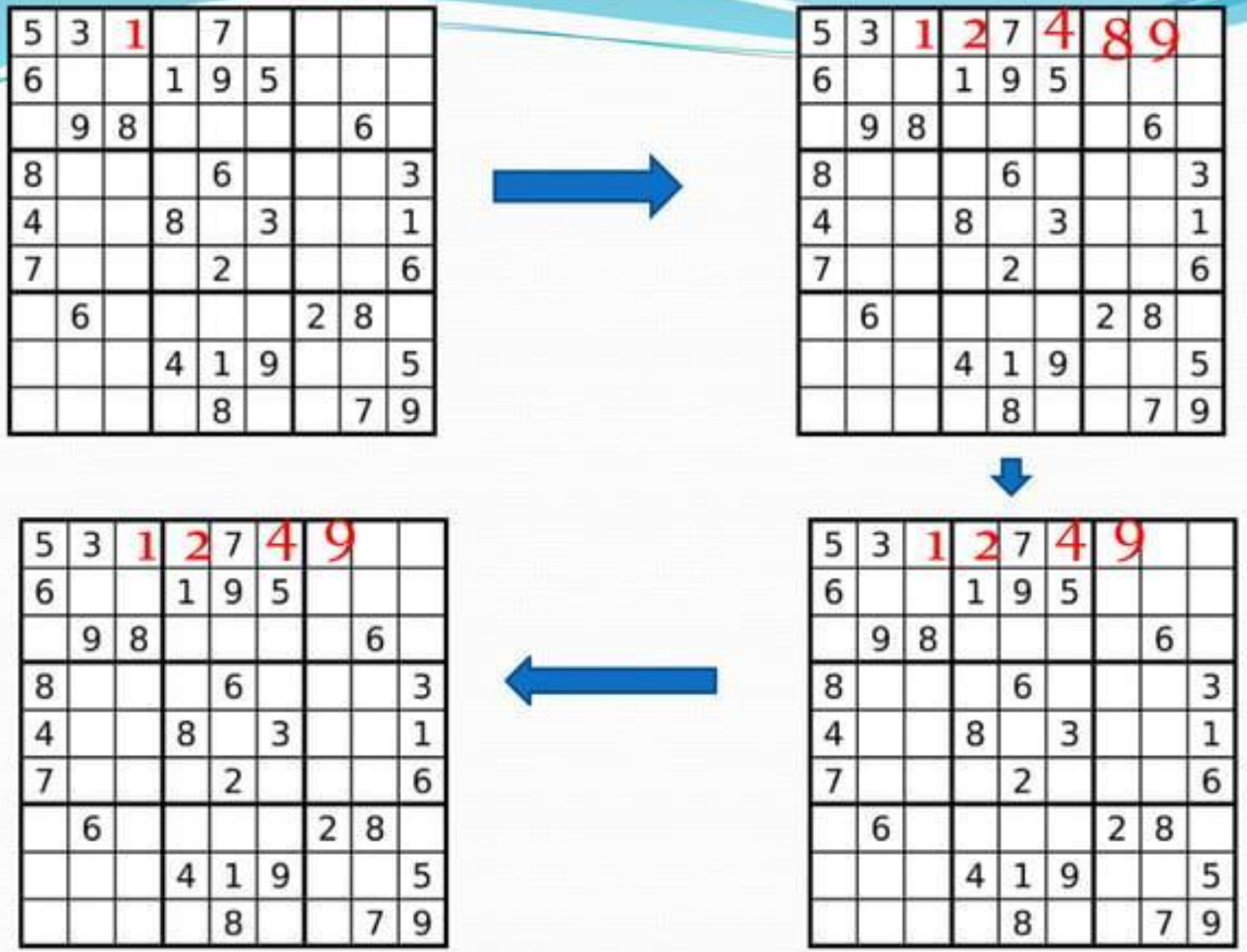
- Recursively calls backtrack 874 times
  - The last one determines that there are no unoccupied entries
- Checks if a placement is valid 7658 times

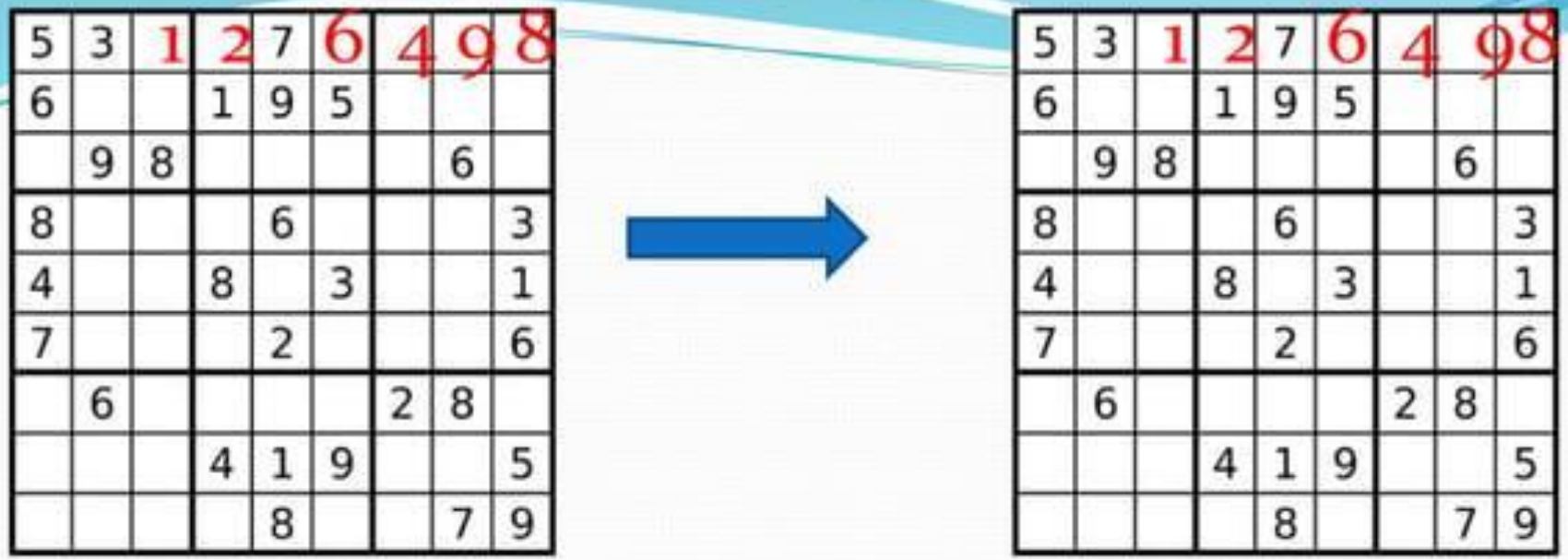
5	3	4	1	7	8	9	6	2
8	6	7	3	2	9	1	4	5
1	9	2	5	4	6	3	7	8
6	7	8	9	3	1	5	2	4
2	1	5	8	6	4	7	9	3
3	4	9	2	5	7	6	8	1
4	2	3	7	9	5	8	1	6
7	5	1	6	8	2	4	3	9
9	8	6	4	1	3	2	5	7

Example:

Solve the given Sudoku problem.




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



MOVING THROUGH THE NEXT LINES AND FOLLOWING THE SAME PROCEDURE WE WILL BE ABLE TO SOLVE THIS PROBLEM WITH BACKTRACKING ALGORITHM.

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

Algorithm Performance

- **Time complexity:** $O(9^{(n*n)})$, For every unassigned index, there are 9 possible options and for each index, we are checking other columns, rows and boxes.
  Auxiliary Space: O(1)