Project Report

Sudoku

Nupur Garg(MT2014081)

Abstract-

Sudoku is a puzzle game played on a grid that consists of n^2 × n^2 cells each belonging to three groups: one of nine rows, one of nine columns and one of nine boxes.

A Sudoku puzzle is a Sudoku grid that is partially filled, meaning that a set of fixed cells, whose numerals are given (that cannot be chosen/changed by the solver). The objective of the puzzle Solver to fill the Sudoku cells by assigning a numeral between 1 and n^2 to each blank cell in such a way that each numeral is unique in each of its three groups (row, column, box).

DataStructure Used-

Initially one data structure is designed to hold some values for the sudoku matrix.

My structure is-

```
class SudokuStructure

{

    int possible_val[];

      int value;

      int counter;

}
```

possible_val[] is an array of size n^2 which can store possible numbers which can come in the cell.I have stored 0 and 1 in possible_val[],where 0 represents that number can't exists and 1 represent that number exists.!st location represents number 1,2nd location represents number 2 and so on.

Value stores that number which cell contains and counter contain the count of the number of possible values that a cell can contain.For example count=2 reflect that a cell can have 2 possible values.

Algorithm-

Sudoku is solved by using below rules and applying dancing link concept.

Rule 1:

Calculate possible values for each cell which is empty by traversing each row,column and grid.This possible values can be calculated by calling my function- find_p_value(SudokuStructure new_cell[][])-this function also maintains counter for each cell which represents number of possible options(numbers) on each cell.

If a cell can contain only one value than assign that value to that cell.(i.e. If counter is 1 than assign that value to the cell for which bit is 1 in possible_val[] of that particular cell)

Then update possible values for each cell in the same row,column and small grid.

```
Rule1()
{
    traverse each cell
        if(new_cell[i][j].counter==1)
        {
            traverse each possible value of that cell i.e k=0 to n^2
            {
                if(new_cell[i][j].possible_val[k]==1)
                {
                    new_cell[i][j].value=k+1;
                    new_cell[i][j].counter=0;
                    num_empty_cell--;
                    update_p_value(i,j,new_cell);//updates possible
values of corresponding cells in row,column and small grid.
                }
            }
        }
        return num_empty_cell;

}
```

Rule 2:

If there is a number which can go into only one cell, then assign that number to that cell.

For each cell bitwise OR of possible values of all other cells except the cell for which we are checking is done.If that result contain only one 0 entry for any possible value then this number will come in the cell for which we are checking.

This is done for all rows,columns and small grids.

Rule 3:

This is the optional rule 1 that say-If there are only 2 cells in a row/column/box which can take x and y, then x and y can not be assigned to any other cell in the respective row/column/box.

Rule 4:

This is optional rule 2 that say-If there are only 3 cells in a row/column/box which can take x,y and z, then x, y and z can not be assigned to any other cell in the respective row/column/box.

*Rule 1 to 4 are applied in the loop till number of empty cells after applying rule 1 and rule 2 is not same.

```
while(true)
        {
        empty_cell1=sl.rule1(new_cell);
                sl.rule3(new_cell);
                sl.rule4(new_cell);
        empty_cell2=sl.rule2(new_cell);
                if(empty_cell1==empty_cell2)
        {
                break;
        }
        }
```

Dancing links:Sudoku generated after applying above rule is then solved by applying exact cover algorithm using Dancing links.

We generate table which has 4*n^2*n^2 columns and number of rows is dependent on the number of empty cells that is left after applying above rules.

Each columns represents some constraints of filling the sudoku-

There are 4 types of contraints-

1.Cell-each cell can contain only 1 value.

2.Row-Each row has only 1 occurence of the number.

3.Column-Each column has only 1 occurence of the number.

4.Grid-Each small grid has only 1 occurence of the number.

Now Algorithm X is applied-

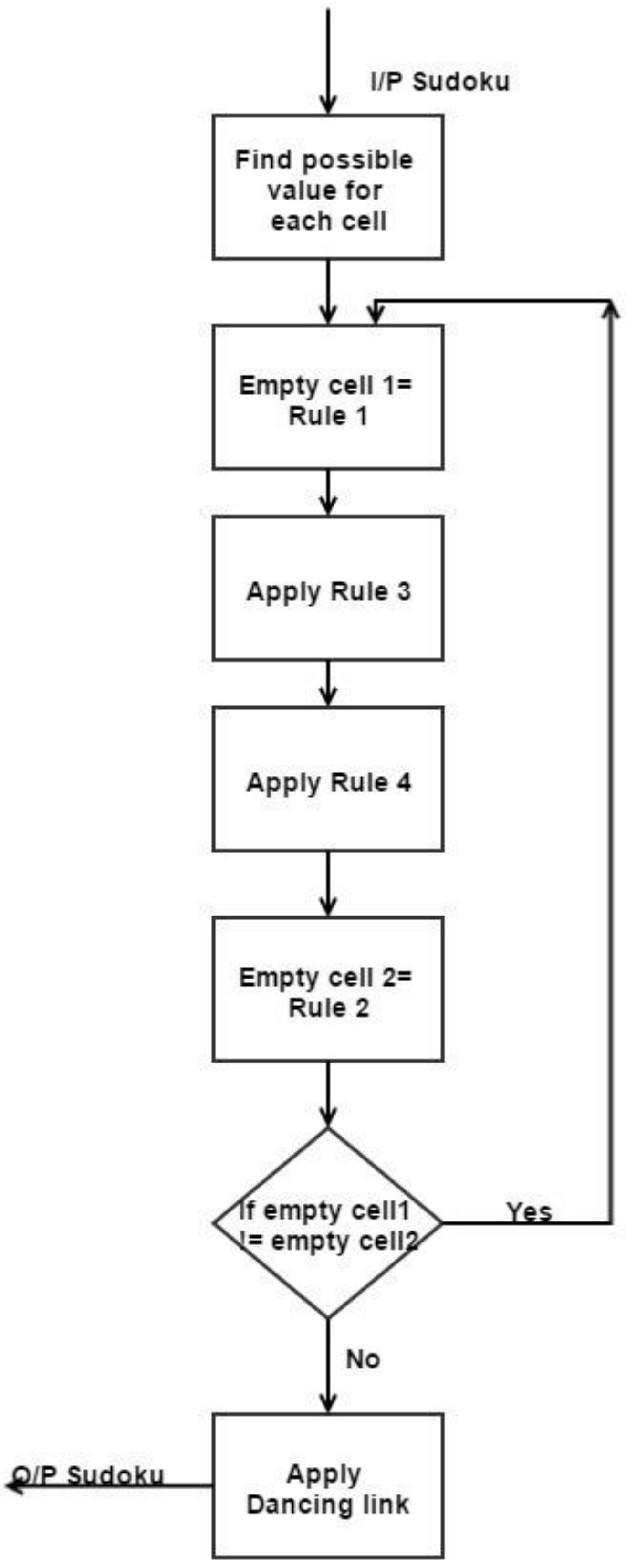
```
sudokufill(DancingLinkTable dt)
{
    if(dt.start.right==dt.start)
        return true;

    DancingLinkNode dn=FindMinColheader(dt);
     cover(dn.col,dt);
```

```
   for(DancingLinkNode i=dn.down; i!=dn; i=i.down)
   {
        Final.push(i.sn);
        for(DancingLinkNode j=i.right; j!=i; j=j.right)
        {
      cover(j.col,dt);
        }

        if(sudokufill(dt))
        {
            return true;
        }
        for(DancingLinkNode j=i.left; j!=i; j=j.left)
        {
      uncover(j.col,dt);
        }
        Final.pop();
      }
      uncover(dn.col,dt);
      return false;
}
```

Algorithm flowchart:



Algorithm discussion:

Algorithm is discussed with Setu patani, Padalia Rajan Kantilal but implemented by my own.