SUDOKU

Objective:

- To fill a 9 x 9 sudoku by assigning 1-9 digits to each unassigned cell such that each row, column, and 3 x 3 sub-grid contains exactly one instance of each digits.

Motivation:

In childhood, filling Sudoku which appears on the last page of Eenadu Weekly Magazine was a weekly hobby. Most of the times I successfully filled it, but it used to take nearly half-an-hour and so. Sometimes, I used to make wrong choices at an instance and had to erase the whole grid and restart.

As I have gained the knowledge of programming and algorithms, I thought to implement backtracking algorithm and basic OpenCV to fill the standard 9x9 Sudoku.

Tools used:

- OpenCV
- Visual Studio
- C++ Programming language

Description:

This is a self-project carried out to implement backtracking algorithm and basic OpenCV library to complete the Sudoku.

Input of Sudoku is given as row-wise space separated digits which will be stored into a 2D vector of pairs or Sudoku values can be initialised in the code.

The program uses backtracking and checks if the Sudoku can be filled under given conditions and if successful, displays the filled Sudoku as an image.

Algorithm:

- Check for the first unassigned cell in the grid and if found start filling it up with any of the 1 to 9 digits (whichever comes first) such that the primary condition is not violated and recursively check the assignment leads to solution. If not, fill the cell with next digit.
- If there is no unassigned location, return true (i.e., Sudoku has been filled). If a cell cannot be assigned with any of the digits, then return false.
- Finally display the Sudoku image.

Implementation:

```
#include<opencv2/objdetect.hpp>
#include<opencv2/highgui.hpp>
#include<opencv2/imgproc.hpp>
#include<opencv2/imgproc.hpp>
#include<iostream>
#include<vector>
#include<utility>
#define vvs vector<vector<pair<short, bool>>>
#define N 9

using namespace std;
using namespace cv;

void buildSudoku(Mat&, Rect, Scalar);

vvs takeinput(void);

bool SolveSudoku(vvs&);

void fillSudoku(Mat&, Rect, const vvs);

void printSudoku(const vvs);

int main()
{
	unsigned w = 720, h = 720, indent = 60; /*Indent is the indentation space*/
	Scalar bgcolor(255, 255, 255), linecolor(0, 0, 0);
	Mat sudoku(w, h, CV_8UC3, bgcolor);

	Point2f stpt(indent, indent), endpt(w-indent, h - indent);

	/*Object storing the border rectangle of Sudoku*/
	Rect border_rect(stpt, endpt);

	buildSudoku(sudoku, border_rect, linecolor);

	/*Calling the function to take input*/
	//vvs vec = takeinput();

	/*Declaring and initializing the Sudoku*/
	vvs vec = {
			{{2, 1}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {1, 1}, {0, 0}, {0, 0}, {7, 1}},
			{{0, 0}, {0, 0}, {0, 0}, {8, 1}, {0, 0}, {9, 1}, {4, 1}, {2, 1}, {0, 0}},
			{{0, 0}, {0, 0}, {3, 1}, {6, 1}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}},
			{{0, 0}, {7, 1}, {2, 1}, {0, 0}, {3, 1}, {0, 0}, {0, 0}, {0, 0}, {0, 0}},
			{{9, 1}, {3, 1}, {0, 0}, {0, 0}, {0, 0}, {2, 1}, {1, 1}, {5, 1}, {0, 0}},
			{{8, 1}, {6, 1}, {0, 0}, {0, 0}, {9, 1}, {0, 0}, {7, 1}, {0, 0}, {0, 0}},
			{{5, 1}, {0, 0}, {0, 0}, {9, 1}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}},
			{{0, 0}, {1, 1}, {0, 0}, {2, 1}, {0, 0}, {3, 1}, {6, 1}, {0, 0}, {8, 1}},
			{{0, 0}, {8, 1}, {0, 0}, {0, 0}, {6, 1}, {7, 1}, {2, 1}, {0, 0}, {0, 0}},
		};
	if(!SolveSudoku(vec))
	{
		cout << endl << "The solution for this Sudoku doesn't exist" << endl << endl;
		return 0;
	}

	printSudoku(vec);

	fillSudoku(sudoku, border_rect, vec);

	imshow("Sudoku", sudoku);
	waitKey(0);

	return 0;
}
```

```
/*Fills the Sudoku with the digits i.e., fills the boxes of Sudoku with the initial and filled
values*/
void fillSudoku(Mat& img, Rect border,const vvs vec)
{
	/*Width and height of each box of Sudoku*/
	unsigned horzgap = border.width / 9;
	unsigned vertgap = border.height / 9;

	/*Color of digits*/
	Scalar initcolor(0, 0, 0);  //Color of initial given digits --> black
	Scalar rescolor(255, 0, 0); //Color of found digits --> blue

	/*Adjusting the coordinates of bottom-left corner of text box*/
	int xcoord = border.x+20;
	int ycoord = border.y+vertgap-13;

	for (unsigned i = 0; i < 9; i++)
	{
		for (unsigned j = 0; j < 9; j++)
		{
			/*putText function puts the digit in every box*/
			if (vec[i][j].second == 0)  putText(img, to_string(vec[i][j].first),
Point(xcoord + j * horzgap, ycoord + i * vertgap), FONT_HERSHEY_PLAIN, 3, initcolor, 3);
			else
			{
				putText(img, to_string(vec[i][j].first), Point(xcoord + j * horzgap,
ycoord + i * vertgap), FONT_HERSHEY_PLAIN, 3, rescolor, 3);
			}
		}
	}
	return;
}

/*Draws the layout of the Sudoku*/
void buildSudoku(Mat& imgsrc, Rect border, Scalar color)
{
	Point titlept(border.x+(border.width/3), border.y-20);
	double titlescale = getFontScaleFromHeight(FONT_HERSHEY_DUPLEX, 40, 5);
	Scalar titlecolor(127, 0, 255);

	/*Putting the Title on image*/
	putText(imgsrc, "SUDOKU", titlept, FONT_HERSHEY_DUPLEX, titlescale, titlecolor, 5);
	rectangle(imgsrc, border, color, 3);
	double wid = border.width, height = border.height;

	double slopex = wid / N;
	double slopey = height / N;

	for (short i = 1; i <= (N-1); i++)
	{
		Point vertsrc(border.x + (slopex * i), border.y);
		Point vertdest(border.x + (slopex * i), border.y+height);

		Point horsrc(border.x , border.y + (slopey*i));
		Point hordest(border.x + wid, border.y + (slopey * i));

		if (i % 3 == 0)
		{
			line(imgsrc, horsrc, hordest, color, 2);
			line(imgsrc, vertsrc, vertdest, color, 2);
		}
		else
		{
			line(imgsrc, horsrc, hordest, color, 1.5);
			line(imgsrc, vertsrc, vertdest, color, 1.5);
		}
	}

	return;
```

```
}

/*Prints instructions on console*/
void instructions(void)
{
	cout << endl << "---INSTRUCTIONS---"<<endl
		<<"Enter the initial contents of sudoku row-wise" << endl
		<<"For every row, enter the digits as space seperated"<<endl
		<< "If there is a blank, enter it as '0' (without quotes)" << endl<<endl;
	return;
}

/*Takes input into a 2D vector and returns it*/
vvs takeinput(void)
{
	/*Declaring a 2D vector of pairs*/
	vvs arr(N, vector<pair<short, bool>>(N, {0, 0}));

	/*Taking input from the user*/
	instructions();
	for (short i = 0; i < N; i++)
	{
		cout << "Enter the row " << (i + 1) << " content: ";
		for (short j = 0; j < N; j++)
		{
			cin >> arr[i][j].first;
			if(arr[i][j].first!=0)
				arr[i][j].second = 1;	/*This digit has been given as input*/
		}
	}

	arr.shrink_to_fit(); //destroys any extra space allotted
	return arr;
}

/* Returns whether any assigned entry in the specified row matches the given number */
bool UsedInRow(vvs vec, short row, short num)
{
	for (short col = 0; col < N; col++)
		if (vec[row][col].first == num)
			return true;
	return false;
}

/* Returns whether any assigned entry in the specified column matches the given number */
bool UsedInCol(vvs vec, short col, short num)
{
	for (short row = 0; row < N; row++)
		if (vec[row][col].first == num)
			return true;
	return false;
}

/* Returns whether any assigned entry within the specified 3x3 box matches the given number. */
bool UsedInBox(vvs vec, short boxStartRow, short boxStartCol, short num)
{
	for (short row = 0; row < 3; row++)
		for (short col = 0; col < 3; col++)
			if (vec[row + boxStartRow][col + boxStartCol].first == num)
				return true;
	return false;
}

/* Returns whether it will be allowed to assign num to the given row,col location */
bool isSafe(vvs vec, short row, short col, short num)
{
	return !UsedInRow(vec, row, num) && !UsedInCol(vec, col, num) &&
		!UsedInBox(vec, row - row % 3, col - col % 3, num);
```

```
}

/*Searches to find whether any unassigned location exists in Sudoku*/
bool FindUnassignedLocation(vvs &arr, short& i, short& j)
{
	for (i = 0; i < N; i++)
		for (j = 0; j < N; j++)
			if (arr[i][j].first == 0)
				return true;
	return false;
}

/*Solves the Sudoku*/
bool SolveSudoku(vvs &arr)
{
	short row, col;
	if (!FindUnassignedLocation(arr, row, col))
		return true;
	for (short num = 1; num <= N; num++)
	{
		if (isSafe(arr, row, col, num))
		{
			arr[row][col].first = num;
			if (SolveSudoku(arr))
				return true;
			arr[row][col].first = 0;
		}
	}
	return false;
}

/*Prints the sudoku on console*/
void printSudoku(const vvs sudoku)
{
	cout << endl << "Printing the sudoku" << endl;
	for (short row = 0; row < N; row++)
	{
		for (short col = 0; col < N; col++)
			cout << sudoku[row][col].first << "  ";
		cout << endl;
	}
	cout << endl;
	return;
}
```

Example:

SUDOKU

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

SUDOKU

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

Complexity:

Time Complexity – $O\left(N^{N^2}\right)$

- Number of unassigned cells will be $O(N^2)$ and every unassigned cell has to be checked with worst case O(N) digits to fill it up with.

Space Complexity – $O\left(N^2\right)$

- For 2D vector of pairs

Here N = 9

Future Scope:

- To train a model which will recognize digits and empty spaces from image of Sudoku so that input will be given as image and output will be filled image.
- The time complexity of the solution is exponential. Its better to search for polynomial complexity solution at least.

Reference: GeeksforGeeks

Conclusion:

A good project to learn about Backtracking and OpenCV.

By

Rekha Lokesh

19EC10052

E&ECE Department

IIT Kharagpur