

Automated Sudoku Solver

Marty Otzenberger

EGGN-510

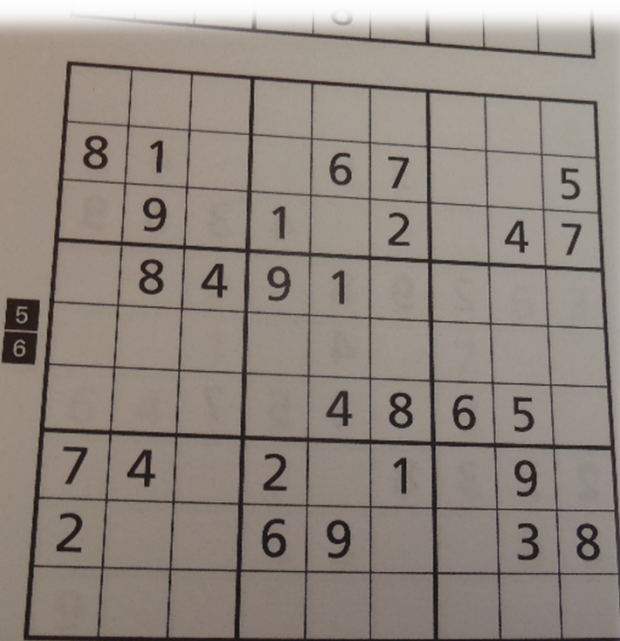
December 4, 2012

Outline

- Goal
- Problem Elements
- Initial Testing
- Test Images
- Approaches
- Final Algorithm
- Results/Statistics
- Conclusions
- Problems/Limits
- Future Work/Improvements
- References
- Questions

Goal

- From an image of a Sudoku puzzle, extract and solve the puzzle, and display the solution over the top of the puzzle.



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

Problem Elements

- Extract the puzzle from the image.
- Identify the numbers in each cell.
 - Correlate the numbers to their respective cell position.
- Solve the puzzle.
- Display the solution over the image.

Initial Testing

5	3			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

- Began by using an ideal digitally fabricated image to reduce complexity.
 - Chose the Wikipedia image for Sudoku.
- Used a Hough Transform to extract the lines in the puzzle, and segmented the image based on the rho values.
 - Parameters very sensitive to capture only one Hough line per puzzle line.
 - Used averages if multiple existed close together

5	3			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

Initial Testing cont.

- Cropped image between each set of rho values, and performed Normalized-Cross-Correlation on each sub-image.
- I saved the max score for a cross-correlation of each sub-image with all 9 template digits, and then used the max of those peak scores to identify the digit.
 - Used the total number of black pixels in a square as a threshold for determining if it was empty.
 - Cropped the template images out of the puzzle.
 - Processed sub-images in order to preserve location information.

Initial Testing cont.

- Solved the puzzle using a MATLAB script I found which recursively solves for possible values for every blank cell in the puzzle[1].
- Used Hough line rho values to re-project the solution onto the image using MATLAB text.

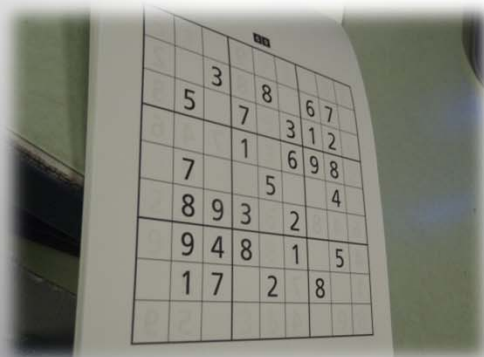
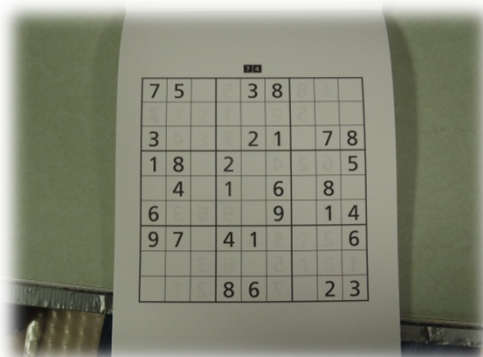
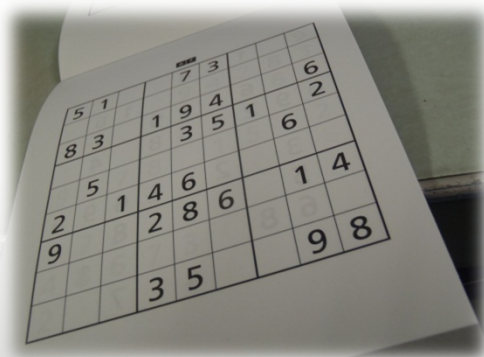
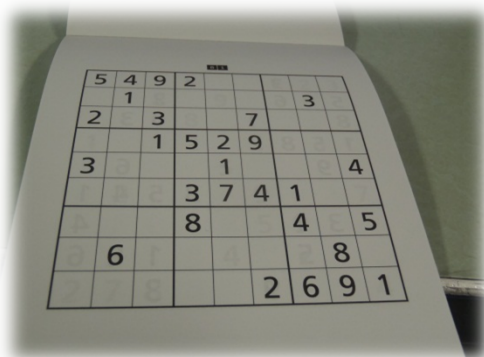
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

[1] G.M. Boynton, "MATLAB Sudoku Solver," MATLAB Central File Exchange, 2005,
Accessed: 12/2/2012,
<http://www.mathworks.com/matlabcentral/fileexchange/8083-matlab-sudoku-solver>

Test Images

- Want to expand code to handle real images of puzzles taken from a variety of angles.
- Took a series of 25 test images of 25 different puzzles to test with.
 - All images taken from the same Sudoku book to keep digits in a common font.
- Intentionally took some extreme images to test the robustness of my algorithm.

Test Images cont.

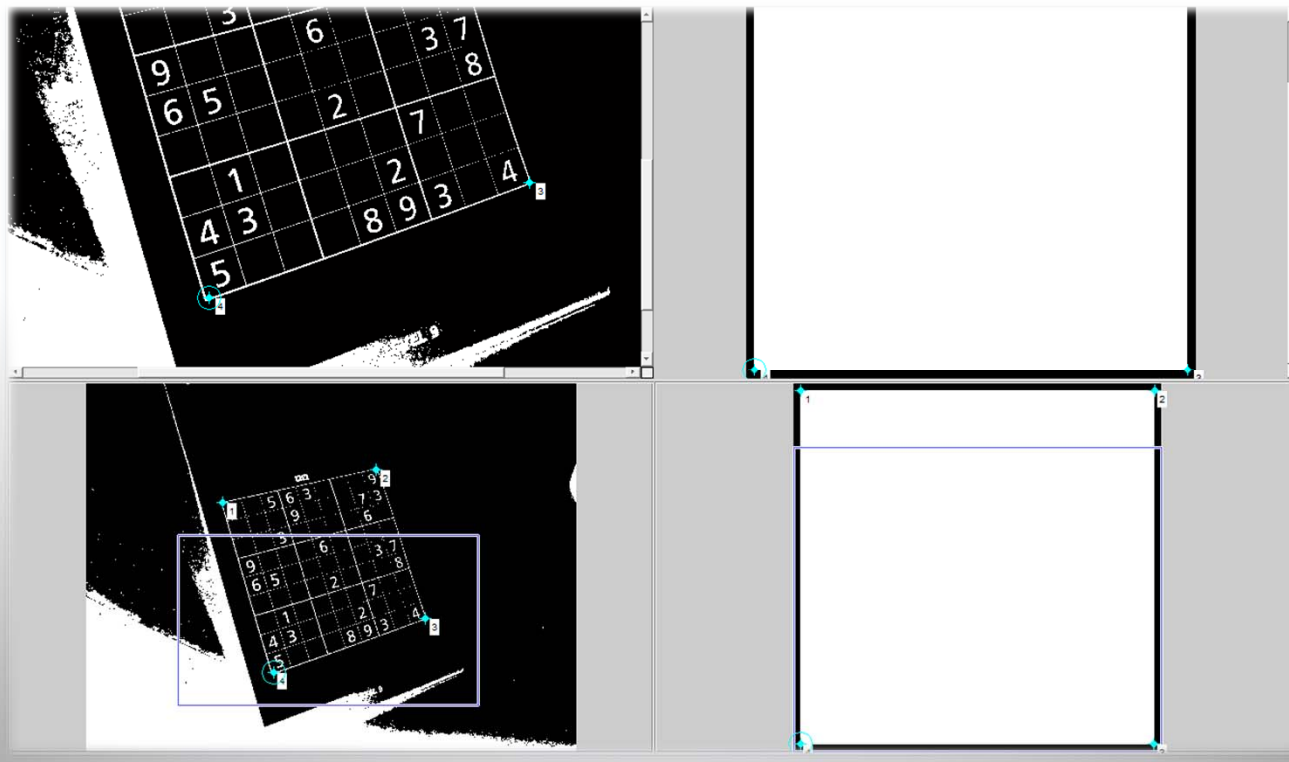


Approaches

- Initially tried to again use Hough Transforms, but had difficulty.
 - Curvature in the lines caused finding a single line difficult without human intervention on every image.
 - Even extraction of the puzzle boundary was difficult using Hough.
- Found connected components helpful to extract puzzle region by looking for objects of the right size.
 - Still had difficulty finding lines.
 - Could have caused problems if the images were at different zoom levels.

Final Algorithm

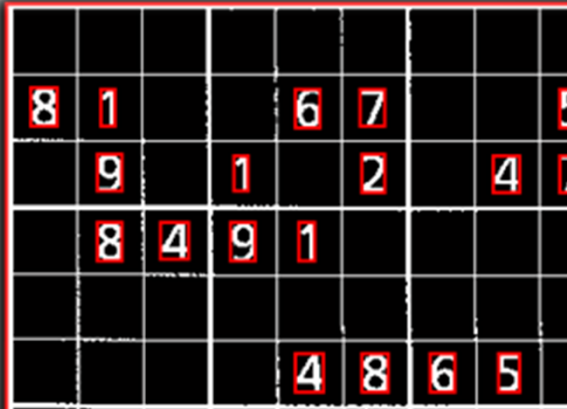
- Use MATLAB `cpselect` tool to manually identify the corners of the puzzle, and project them onto a template image of a square.

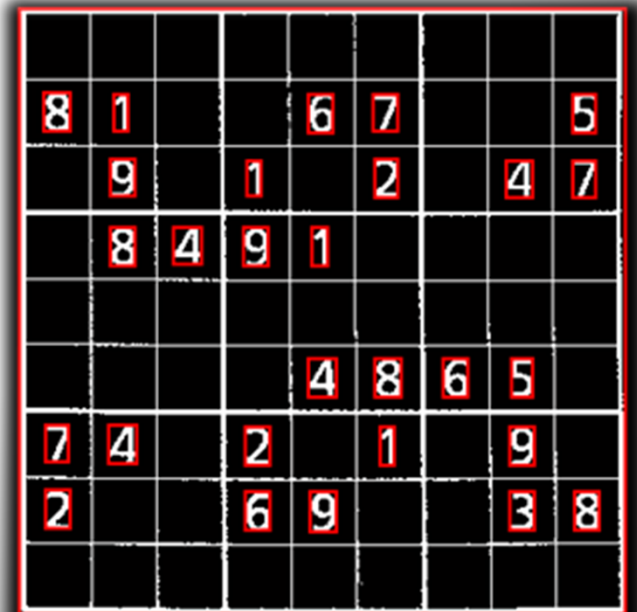


Final Algorithm cont.

- Used a projective transform which preserves quadrilaterals through scaling, rotation, and translation.
 - This made the puzzle the same size and shape in every image.
 - Also reduced the size of the images to speed up processing.
- Next used `regionprops` to find connected components and extract the puzzle by looking at the component's width, and height.
 - Absolute pixel changes every time because MATLAB does not crop the image when transforming.
- Used the bounding box of the puzzle to crop the image.

Final Algorithm cont.

- After cropping the image to extract the puzzle, I again looked at the connected components to extract the numbers.
 - I then cropped out the region around the centroid of each number to ensure it was larger than my template image and used the same normalized-cross-correlation strategy as before.
- 



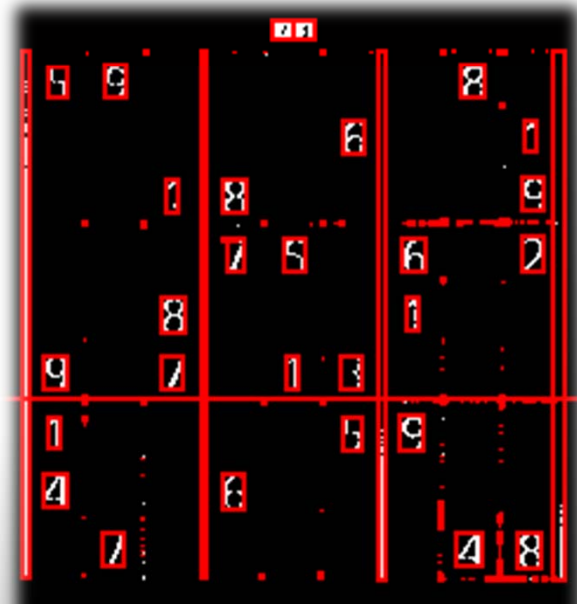
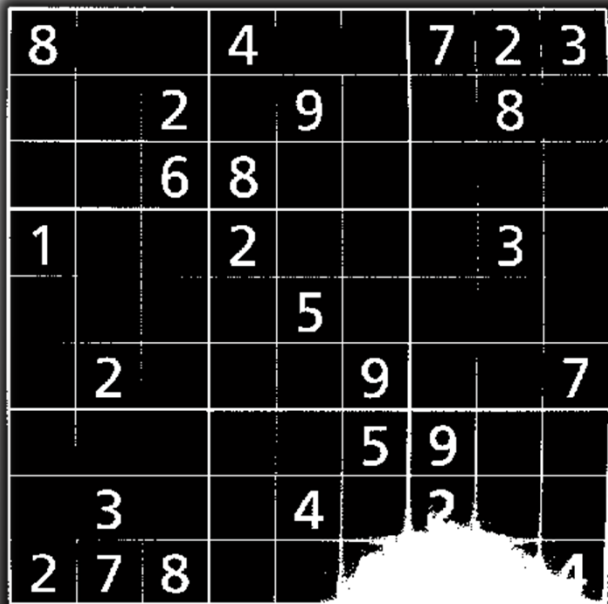
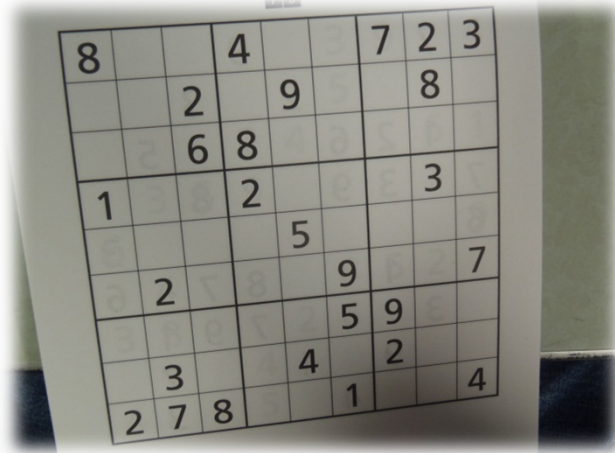
Final Algorithm cont.

- Used the absolute pixel value of the digit centroid to identify its row and column within the puzzle.
 - Possible because the projective transform made the puzzle the same size in every image.
- From here I used the same solver to solve the puzzle, and used absolute pixel locations to display the solution on top of the puzzle.

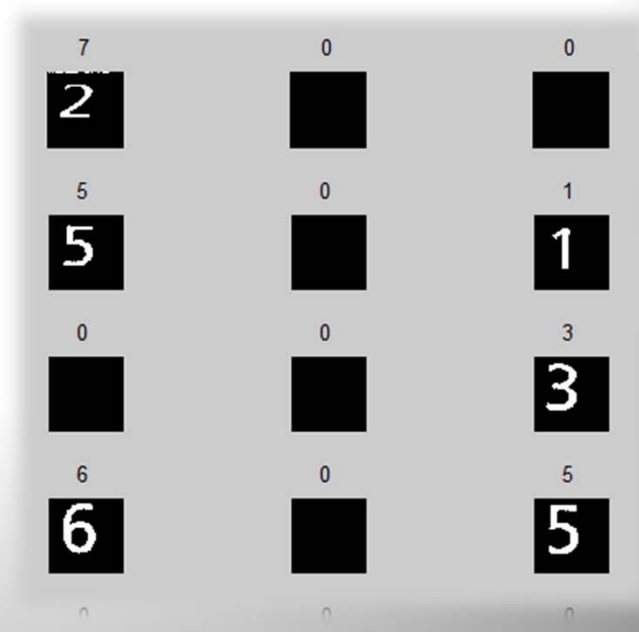
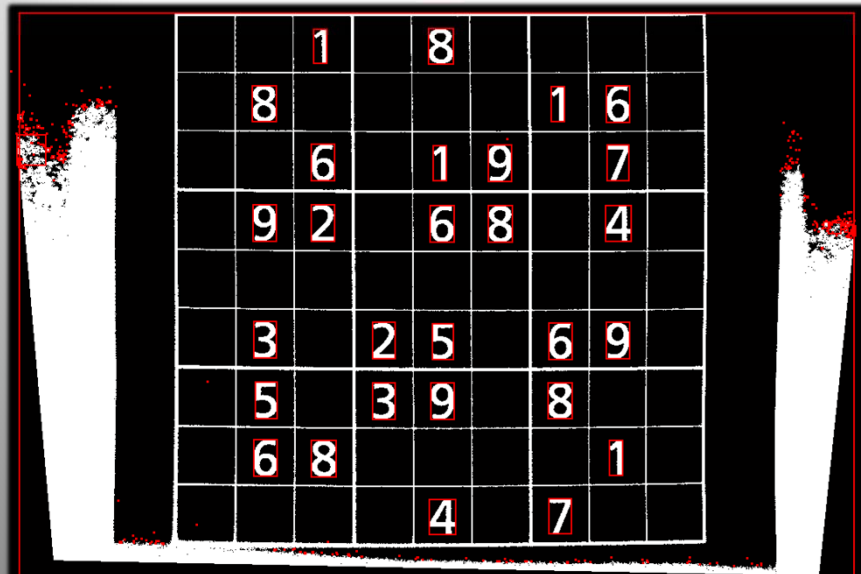
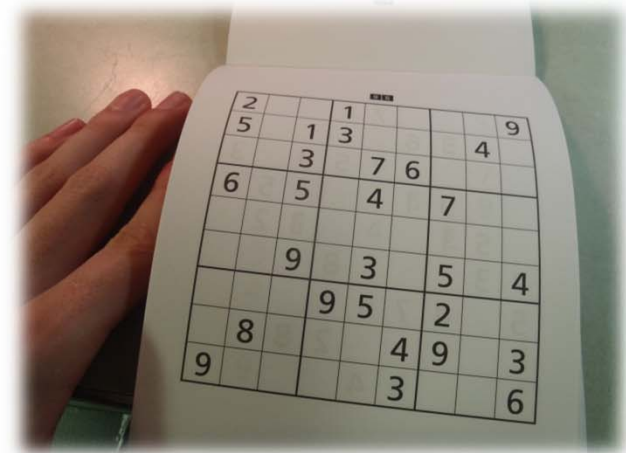
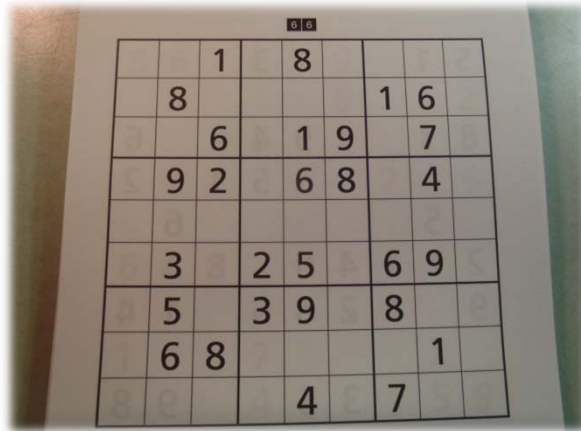
Results/Statistics

- 18 of my 25 test images processed successful.
 - 3 failed due to shadows on the puzzle.
 - 2 failed due to discontinuous borders on the puzzle preventing `regionprops` from finding it.
 - 1 failed due to noise connected to the border of the puzzle.
 - 1 failed due to distortion leading to incorrect numeric classification.
- This is a 72% success rate.

Results/Statistics cont.



Results/Statistics cont.



Conclusions

- Pleasantly surprised by the performance of the algorithm.
- Handled off angle images of the puzzles very well.
- Most issues due noise/thresholding.

Problems/Limits

- Current algorithm requires user interaction.
- Difficulty handling shadows on the puzzle.
- Can only process puzzles using the same font set.
- Cannot handle distortion in the puzzle.
 - This will probably be difficult to fix, particularly without changing away from cross-correlation.

Future Work/Improvements

- Automate corner detection to find projection.
 - Should be able to make process fully automated.
- Look at better thresholding algorithm.
 - Try to eliminate some of the shadow and noise issues.
- Use mean centroid value of digits to align solutions.
 - Sometimes the solutions don't line up well with the grid because of distortion in the projected image.
- Project the solution onto original image.
 - Re-project the solutions back to the orientation of the original image and show them on the original.
- Implement camera calibration.
 - Could help improve cross-correlation reliability.

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

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

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

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

Questions?

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

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