Project Report

on

SUDOKU SOLVER Using Backtracking Algorithm

Design and Analysis of Algorithms

By

L.Keerthana (2010030512)

M.Manideepa (2010030517)

D.Dedeepya (2010030526)

K.Sri teja (2010030530)

under the supervision of

# Ms G Udaya Rani

# Assistant Professor



Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

March, 2022

**Page 1**

**Declaration**

The Project Report entitled “SUDOKU SOLVER using backtracking algorithm” is a record of bonafide work of Ms. Keerthana( 2010030512 ), Ms. Manideepa ( 2010030517 ), Ms. Dedeepya

( 2010030526 ), Ms. Sri teja ( 2010030530 ) submitted in partial fulfillment for the award of B.Tech in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

Keerthana

Manideepa

Dedeepya

Sri teja

**Page 3**

**Certificate**

This is to certify that the Project Report entitled “SUDOKU SOLVER using backtracking algorithm” is being submitted by Keerthana , Manideepa , Dedeepya and Sri teja submitted in partial fulfillment for the award of B.Tech in Computer Science Engineering to the K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

## Signature of the Supervisor

Ms G Udaya Rani

Assistant Professor

## Signature of the HOD Signature of the External Examine

**Page 4**

**ACKNOWLEDGEMENTS**

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

It is great pleasure for us to express gratitude to our honorable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project-based laboratory report.

We express the sincere gratitude to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us and also for his administration towards our academic growth.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for his exemplary guidance, monitoring and constant encouragement throughout the course of the project.

We express sincere gratitude to our Course Coordinator  **Dr.Deepthi Kalavala** for her leadership and constant motivation provided in successful completion of our academic semester.

We even express our sincere thanks to our Course Instructor **Ms G Udaya Rani** for her novel association of ideas, encouragement, appreciation, and intellectual zeal which motivated us to venture this project successfully and also for supporting us throughout this project holding a position of supervisor.

We whole heartedly thank all the teaching and non-teaching staff of our department without whom we won’t have made this project a reality. We would like to extend our sincere thanks especially to our parent, our family members and friends who have supported us to make this project a grand success.

**Page 5**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Abstract | 7 |
| 2 | Introduction  2.1 4x4 SUDOKU  2.2 6X6 SUDOKU  2.3 STANDARD SUDOKU  2.4 JIGSAW SUDOKU | 8 - 12 |
| 3 | Literature Survey | 13 - 14 |
| 4 | Hardware and Software Requirements | 15 |
| 5 | Proposed System | 16-17 |
| 6 | Flow diagram | 18 |
| 7 | Implementation  🡪 Code  🡪 Results | 19 - 31 |
| 8 | Conclusion and Future Work | 32 |
| 9 | References | 33 |

**Page 6**

**CHAPTER - 01**

**ABSTRACT**

* We present the detailed development and implementation of simple sudoku game.
* The sudoku game consists of graphical user interface ,solver and implemented using python
* The solver finds the solution to the puzzles entered by the user.
* This project gives an insight to the different aspects of python programming
* Like all other Backtracking problems, Sudoku can be solved by one by one assigning numbers to empty cells.
* Before assigning a number, check whether it is safe to assign.
* Check that the same number is not present in the current row, current column and current 3X3 subgrid.
* After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not.
* If the assignment doesn’t lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

**Page 7**

**CHAPTER - 02**

**INTRODUCTION**

* The aim of the puzzle is to enter a numerical digit from 1 through 9 in each cell of a 9x9 grid made up of 3×3 subsquares or subgrids.
* starting with various digits given in some cells; each row, column, and subsquares region must contain each of the numbers 1 to 9 exactly once.
* A sudoku solution must satisfy all of the following rules:-

1. Each horizontal row contains each digit exactly once

2. Each vertical column contains each digit exactly once

3. Each subgrid or region contains each digit exactly once

* There are different types of sudoku’s:

4x4 SUDOKU  
6X6 SUDOKU

STANDARD SUDOKU

JIGSAW SUDOKU

**Page 8**

**2.1 4x4 SUDOKU**

* **Small sudoku is propably the smallest and easiest sudoku variant**
* **Grid size :- 4 x 4**
* **Number of inner boxes :- 4**
* **Inner box shape :- Sqaure**
* **Number range :- 1 to 4**

**Calendar

Description automatically generated**

**Page 9**

**2.2 6X6 SUDOKU**

TCP/IP protocol suit addresses four levels

of addressing schemes. Each level represents

different addressing techniques and sources

TCP/IP protocol suit addresses four levels

of addressing schemes. Each level represents

different addressing techniques and sources

TCP/IP protocol suit addresses four levels

of addressing schemes. Each level represents

different addressing techniques and source

* Mini sudoku is a relatively easy variant with non-square inner boxes.
* Grid size :-6 x 6
* Number of Inner boxes :- 6
* Inner box shape :- 3 x 2 rectangle
* Number range :- 1 to 6

Table

Description automatically generated

**Page 10**

2.3 STANDARD SUDOKU

* Grid size :- 9 x 9
* Number of inner boxes :- 9
* Inner box shape :- 3 x 3 Sqaure
* Number range :- 1 to 9

Text

Description automatically generated with medium confidence

**Page 11**

**2.4 JIGSAW SUDOKU**

* **The only difference to the standard sudoku is the shape of the inner blocks**
* **Grid size :- 9 x 9**
* **Number of inner boxes :- 9**
* **Inner box shape :- 3 x 3 Sqaure**
* **Number range :- 1 to 9**

**A picture containing text, crossword puzzle

Description automatically generated**

**Page 12**

**CHAPTER - 03**

**LITERATURE SURVEY**

**EXISTING SOLUTIONS:-**

* Sudoku puzzles are NP-complete problems and as such, their solution can be found by performing an exhaustive search.

**1) Pen and Paper Algorithm** :- a solution based on some strategies used by humans when solving the puzzle, therefore, it is called pencil-and-paper algorithm.

* As there are puzzles with different types of difficulty, the easy and medium
* puzzles can be solved using some simple techniques such as unique missing method, naked singles

**2) Brute Force Search/BackTracking/Depth For Search** :-

* Backtracking checks every possible outcome that can occur on the board and is terminated when correct output occurs in one of these outcomes.
* Backtråcking is also known as "Complete Brute Force Search" OR "Depth For Search Traversal Approach in Grid"

**3)** **Crooks Algorithm** :-

* Crooks algorithm, which is an imitation of pencil and paper algorithm, is a simple algorithm which we use to solve the sudoku on our own.
* This algorithm sounds simple, but it has its limitations. Sometimes, we get incomplete answer to the sudoku because of multiple values that can occur on a particular empty space.

**COMPARISION:-**

* According to the data, crooks algorithm is much faster than backtracking algorithm as it eliminates the impossible outcomes and then solves the algorithm, as compared to backtracking which generates all possible sudoku matrices which takes more time.
* On the other hand, complete version of Crooks algorithm is that, solve the puzzle, and if it is incomplete, complete it with backtracking.

**Page 13**

Chart, line chart

Description automatically generated

**WHICH IS BEST ALGORITHM:-**

* Crook's on its own is an incomplete algorithm, but it imitates solving by pencil and paper like us. It Might Get Stuck Somewhere due to Own Wrong Assumption
* Backtracking checks all possible combinations but it provides correct solution. Although, it Takes More Time, But is Able to Provide Correct Answer
* Best Algorithm => Backtracking Approach/ DFS Approach

**Page 14**

**CHAPTER - 04**

**Hardware and Software Requirements :-**

SOFTWARE REQUIREMENTS :

* python(pycharm)
* windows operating system (can’t use mac coz have compatibility issues )

HARDWARE REQUIREMENTS :

* mouse or trackpad

**Page 15**

**CHAPTER – 05**

**PROPOSED SYSTEM**

5.1 BACKTRACKING ALGORITHM :-

* In simple terms, we first start with the first empty space on the sudoku board. We add the first element that can come in that space, let's say 1.
* Next, we move on to the second empty space, and add the first element, 1, to it.
* If this is valid, it will go to next space, or else the element will become 2. This repeats for every element that can be placed in that space.
* This process is repeated for every empty space and if the sudoku is not solved, it goes to the first empty space and places next possible element, that is, 2, in our case.
* This whole process is repeated until the sudoku is solved.

5.2 BACKTRACKING ALGORITHM

find an empty cell with coordinates (row, col):

if no cell is found then

return true;

end

for digits x from 1 to 9 do

if we can place x at position (row,col) and the puzzle remains valid then recursively continue to fill the rest of the puzzle:

if recursion succeeds i.e. returns true then

return true

end

else

free cell and try with the next digit;

end

end

end

if none of the digits lead to a solution then

return false

end

**Page 16**

**5.3** BACKTRACKING –ANALYSIS

* The time complexity for solving the sudoku is O(n^m), where m = nxn. Where n is the number of squares in one side of the sudoku square. In classical sudoku, n = 9. Thus, time complexity for classical sudoku is 0(9^81).
* The recurrence relation for backtracking algorithm for classical sudoku can be written as T(m) = 9\*T(m-1) + O(1)
* This is because, we have to check 9 values in a particular empty space.

5.4 Data Structures needed

3D array or matrix

* Sudoku is a 2D grid game; so this data structure comes in mind intuitively.
* Mostly the operation involves reading the value of cell and writing value to the cell. So, these operations must be as fast as possible.
* With the use of 2D array, these operations will be O(1).
* So this is the best data structure which we can use.
* Assuming that you will use backtracking, you might check what are the possible values for a cell.
* To keep track of all values, you will store them.
* So it will be way easier to store them in Z-direction for each cell.
* Hence we need the third dimension.

**Page 17**

**CHAPTER - 06**

**FLOW DAIGRAM**

**Diagram

Description automatically generated**

**Page 18**

**CHAPTER – 07**

**IMPLEMENTATION**

**7.1 CODE**

SUDOKU.py

board = [

[7 ,6 ,0, 0, 0, 9, 0, 1, 0],

[8, 0, 0, 0, 0, 0, 7,2, 4],

[0, 0, 0, 7, 1, 4, 0, 0, 9],

[4, 0, 6, 0, 0, 0, 3, 0, 2],

[0, 7, 0, 4, 5, 6, 0, 0, 0],

[0, 5, 1, 0, 0, 0, 0, 7, 6],

[1, 0, 0, 2, 9, 0, 0, 4, 0],

[2, 0, 7, 6, 3, 0, 1 ,0, 0],

[0, 9, 8, 0, 0, 5, 2, 0, 7],

]

print("====SUDOKU SOLVER====")

def solve(bo):

find = find\_empty(bo)

if not find:

return True

else:

row, col = find

for i in range(1,10):

if valid(bo, i, (row, col)):

bo[row][col] = i

if solve(bo):

return True

bo[row][col] = 0

return False

def valid(bo, num, pos):

# Check row

for i in range(len(bo[0])):

if bo[pos[0]][i] == num and pos[1] != i:

return False

**Page 19**

# Check column

for i in range(len(bo)):

if bo[i][pos[1]] == num and pos[0] != i:

return False

# Check box

box\_x = pos[1] // 3

box\_y = pos[0] // 3

for i in range(box\_y\*3, box\_y\*3 + 3):

for j in range(box\_x \* 3, box\_x\*3 + 3):

if bo[i][j] == num and (i,j) != pos:

return False

return True

def print\_board(bo):

for i in range(len(bo)):

if i % 3 == 0 and i != 0:

print("- - - - - - - - - - - - - ")

for j in range(len(bo[0])):

if j % 3 == 0 and j != 0:

print(" | ", end="")

if j == 8:

print(bo[i][j])

else:

print(str(bo[i][j]) + " ", end="")

def find\_empty(bo):

for i in range(len(bo)):

for j in range(len(bo[0])):

if bo[i][j] == 0:

return (i, j) # row, col

return None

print\_board(board)

solve(board)

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("====Solution for the SUDOKU SOLVER====")

print\_board(board)

page 20

**GUI.py:**  
# GUI.py

import pygame

from solver import solve, valid

import time

pygame.font.init()

class Grid:

# To change the starting board change this

board = [

[7, 6, 0, 0, 0, 9, 0, 1, 0],

[8, 0, 0, 0, 0, 0, 7, 2, 4],

[0, 0, 0, 7, 1, 4, 0, 0, 9],

[4, 0, 6, 0, 0, 0, 3, 0, 2],

[0, 7, 0, 4, 5, 6, 0, 0, 0],

[0, 5, 1, 0, 0, 0, 0, 7, 6],

[1, 0, 0, 2, 9, 0, 0, 4, 0],

[2, 0, 7, 6, 3, 0, 1, 0, 0],

[0, 9, 8, 0, 0, 5, 2, 0, 7],

]

def \_\_init\_\_(self, rows, cols, width, height):

self.rows = rows

self.cols = cols

self.cubes = [[Cube(self.board[i][j], i, j, width, height) for j in range(cols)] for i in range(rows)]

self.width = width

self.height = height

self.model = None

self.selected = None

def update\_model(self):

self.model = [[self.cubes[i][j].value for j in range(self.cols)] for i in range(self.rows)]

def place(self, val):

row, col = self.selected

if self.cubes[row][col].value == 0:

self.cubes[row][col].set(val)

self.update\_model()

page 21

if valid(self.model, val, (row,col)) and solve(self.model):

return True

else:

self.cubes[row][col].set(0)

self.cubes[row][col].set\_temp(0)

self.update\_model()

return False

def sketch(self, val):

row, col = self.selected

self.cubes[row][col].set\_temp(val)

def draw(self, win):

# Draw Grid Lines

gap = self.width / 9

for i in range(self.rows+1):

if i % 3 == 0 and i != 0:

thick = 4

else:

thick = 1

pygame.draw.line(win, (0,0,0), (0, i\*gap), (self.width, i\*gap), thick)

pygame.draw.line(win, (0, 0, 0), (i \* gap, 0), (i \* gap, self.height), thick)

# Draw Cubes

for i in range(self.rows):

for j in range(self.cols):

self.cubes[i][j].draw(win)

def select(self, row, col):

# Reset all other

for i in range(self.rows):

for j in range(self.cols):

self.cubes[i][j].selected = False

self.cubes[row][col].selected = True

self.selected = (row, col)

def clear(self):

row, col = self.selected

page 22

if self.cubes[row][col].value == 0:

self.cubes[row][col].set\_temp(0)

def click(self, pos):

"""

:param: pos

:return: (row, col)

"""

if pos[0] < self.width and pos[1] < self.height:

gap = self.width / 9

x = pos[0] // gap

y = pos[1] // gap

return (int(y),int(x))

else:

return None

def is\_finished(self):

for i in range(self.rows):

for j in range(self.cols):

if self.cubes[i][j].value == 0:

return False

return True

class Cube:

rows = 9

cols = 9

def \_\_init\_\_(self, value, row, col, width ,height):

self.value = value

self.temp = 0

self.row = row

self.col = col

self.width = width

self.height = height

self.selected = False

page 23

def draw(self, win):

fnt = pygame.font.SysFont("comicsans", 40)

gap = self.width / 9

x = self.col \* gap

y = self.row \* gap

if self.temp != 0 and self.value == 0:

text = fnt.render(str(self.temp), 1, (128,128,128))

win.blit(text, (x+5, y+5))

elif not(self.value == 0):

text = fnt.render(str(self.value), 1, (0, 0, 0))

win.blit(text, (x + (gap/2 - text.get\_width()/2), y + (gap/2 - text.get\_height()/2)))

if self.selected:

pygame.draw.rect(win, (255,0,0), (x,y, gap ,gap), 3)

def set(self, val):

self.value = val

def set\_temp(self, val):

self.temp = val

def redraw\_window(win, board, time, strikes):

win.fill((255,255,255))

# Draw time

fnt = pygame.font.SysFont("comicsans", 40)

text = fnt.render("Time: " + format\_time(time), 1, (0,0,0))

win.blit(text, (540 - 160, 560))

# Draw Strikes

text = fnt.render("X " \* strikes, 1, (255, 0, 0))

win.blit(text, (20, 560))

# Draw grid and board

board.draw(win)

def format\_time(secs):

sec = secs%60

page 24

minute = secs//60

hour = minute//60

mat = " " + str(minute) + ":" + str(sec)

return mat

def main():

win = pygame.display.set\_mode((540,600))

pygame.display.set\_caption("Sudoku")

board = Grid(9, 9, 540, 540)

key = None

run = True

start = time.time()

strikes = 0

while run:

play\_time = round(time.time() - start)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_1:

key = 1

if event.key == pygame.K\_2:

key = 2

if event.key == pygame.K\_3:

key = 3

if event.key == pygame.K\_4:

key = 4

if event.key == pygame.K\_5:

key = 5

if event.key == pygame.K\_6:

key = 6

if event.key == pygame.K\_7:

key = 7

if event.key == pygame.K\_8:

key = 8

page 25

if event.key == pygame.K\_9:

key = 9

if event.key == pygame.K\_DELETE:

board.clear()

key = None

if event.key == pygame.K\_RETURN:

i, j = board.selected

if board.cubes[i][j].temp != 0:

if board.place(board.cubes[i][j].temp):

print("Success")

else:

print("Wrong")

strikes += 1

key = None

if board.is\_finished():

print("Game over")

run = False

if event.type == pygame.MOUSEBUTTONDOWN:

pos = pygame.mouse.get\_pos()

clicked = board.click(pos)

if clicked:

board.select(clicked[0], clicked[1])

key = None

if board.selected and key != None:

board.sketch(key)

redraw\_window(win, board, play\_time, strikes)

pygame.display.update()

main()

pygame.quit()

**Page 26**

**Solver.py:**

def solve(bo):

find = find\_empty(bo)

if not find:

return True

else:

row, col = find

for i in range(1,10):

if valid(bo, i, (row, col)):

bo[row][col] = i

if solve(bo):

return True

bo[row][col] = 0

return False

def valid(bo, num, pos):

for i in range(len(bo[0])):

if bo[pos[0]][i] == num and pos[1] != i:

return False

for i in range(len(bo)):

if bo[i][pos[1]] == num and pos[0] != i:

return False

box\_x = pos[1] // 3

box\_y = pos[0] // 3

for i in range(box\_y\*3, box\_y\*3 + 3):

for j in range(box\_x \* 3, box\_x\*3 + 3):

if bo[i][j] == num and (i,j) != pos:

return False

return True

def print\_board(bo):

for i in range(len(bo)):

if i % 3 == 0 and i != 0:

print("- - - - - - - - - - - - - ")

for j in range(len(bo[0])):

if j % 3 == 0 and j != 0:

print(" | ", end="")

if j == 8:

print(bo[i][j])

else:

print(str(bo[i][j]) + " ", end="")

def find\_empty(bo):

for i in range(len(bo)):

for j in range(len(bo[0])):

if bo[i][j] == 0:

return (i, j) # row, col

return None

page 27

**RESULTS**

8.1 sudoku solver

**Text

Description automatically generated**

**Page 28**

8.2 sudoku using GUI:

Graphical user interface, text, application, table

Description automatically generated

**Page 29**

8.2.1 sudoku(success case):

A screenshot of a computer

Description automatically generated with medium confidence

8.2.2 sudoku(failure case):

Calendar

Description automatically generated

**Page 30**

**8.2.3 sudoku(Game Over)**

A screenshot of a computer

Description automatically generated with medium confidence

**Page 31**

**CHAPTER – 08**

**CONCLUSION AND FUTURE WORK**

CONCLUSION

* The objectives of the proposed project is to increase the thinking capability.
* The game having all the records which you perform in playing.
* you can make your own sudoku at any step you can go back to one step as well as you can see the solution of it.
* It is manually a very difficult job to perform and its need a lot of recalling , reminding and mathematical calculation.
* The game of “ SUDOKU“ helps to increase mental thinking , vision , etc.

FUTURE SCOPE

* Nowadays, there are extensive studies regarding the mathematics of Sudoku as well as many different algorithms to solve the puzzles.
* Some algorithms are designed to solve the puzzles as quick as possible while some others are designed to solve them as efficiently in terms of computational power and memory.
* However, finding a suitable algorithm to solve any particular Sudoku game proved to be difficult.
* The combination of a simple, yet effective algorithm with a graphical user interface allowed us to generate games, solve them and verify the given solutions in a simple and quick way.

**Page 32**

**CHAPTER – 09**

**REFERENCE LINKS**

* [**https://www.ijmter.com/papers/volume-4/issue-12/solving-sudoku-using-backtracking-algorithm.pdf**](https://www.ijmter.com/papers/volume-4/issue-12/solving-sudoku-using-backtracking-algorithm.pdf)
* [**https://dev.to/christinamcmahon/use-backtracking-algorithm-to-solve-sudoku-270**](https://dev.to/christinamcmahon/use-backtracking-algorithm-to-solve-sudoku-270)
* [**https://www.researchgate.net/publication/303553939\_Recursive\_Backtracking\_for\_Solving\_99\_Sudoku\_Puzzle**](https://www.researchgate.net/publication/303553939_Recursive_Backtracking_for_Solving_99_Sudoku_Puzzle)
* [**https://medium.com/swlh/simple-sudoku-with-backtracking-bb4813ddabb1**](https://medium.com/swlh/simple-sudoku-with-backtracking-bb4813ddabb1)

**Page 33**