kuldeep

by Anshul Tickoo

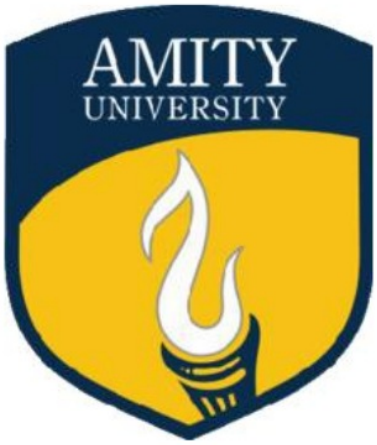
Submission date: 25-May-2020 03:35PM (UTC+0530)
Submission ID: 1331521880
File name: Sudoku_Solver_Report.docx (1.25M)
Word count: 1659
Character count: 7982

TERM PAPER

ON

SUDOKU SOLVER

SUBMITTED TO



AMITY SCHOOL OF ENGINEERING TECHNOLOGY

AMITY UNIVERSITY

GUIDED BY:

MRS ANSHUL TICKOO

SUBMITTED BY:

KULDEEP DWIVEDI

A2305218477

4CSE 7Y

Abstract

Sudoku Solver

Kuldeep Dwivedi, ASET, Amity University

Sudoku is a puzzle game which was built to improve the thinking skill and for better thought processing. This game will improve or nurture our decision making skills and improves the patience level as this game requires a lot of patience and makes you a better person.

It takes time depending on the difficulty level of the puzzle. The objective of this project is to develop a computer aided solver to speed up the solving process. The graphical user interface used in building the game will attract you towards the game and hopefully you won't be impatient and give up easily.

In this report, I have presented the detailed development and implementation of Sudoku Solver game. The game consists of GUI which is implemented using python programming.

The solver is implemented using Recursive Backtracking Algorithm and there are various methods which are used for solving a sudoku game.

The project gives an insight to the different aspects of pygame and a lot about python programming.

Chapter 1

Introduction:

Sudoku is a logical puzzle game built on small assumption which is easy to play and learn. Now- a – days this game is becoming more and more popular all over the world. Earlier also the game was very popular, the puzzles appears in the newspapers, books, magazines, etc. The reason for its popularity was that it improves the neurons connection in our brain. And also the rules are simple and easy and based on numbers instead of letters. It also improves our thinking ability and the thought process.

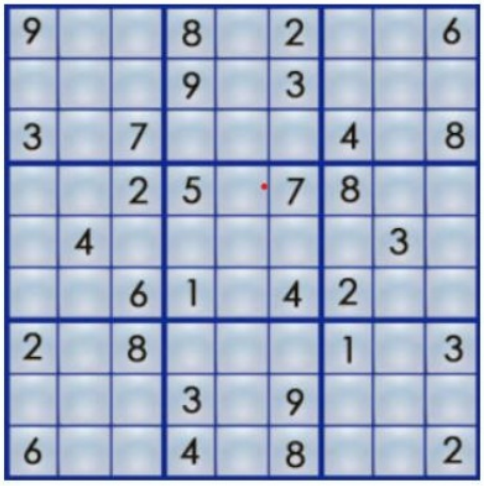


Fig: Sudoku Puzzle

The aim of this project was to make a computer program which aid to solve a puzzle. It will be based on graphical user interface which offers the possibility of solving a sudoku puzzle for the user and the user can check individual moves and finish the required puzzle.

Chapter 2

Problem Statement and Objectives

The aim is to solve the sudoku in as much less time as possible. It can be resolved by comparing various methods or solution required to solve the puzzle. The comparison of various methods or algorithms are based on the factor time complexity.

Actually, there are two major constraints:

1. Time Complexity
2. Memory Used

The quality of optimal sudoku solver is determined by both the constraints Time Complexity and Memory used.

But the main concern is time complexity. Sudoku itself can be solved using Recursive Backtracking in a reasonable amount of time. These constraints vary from algorithms to algorithms.

2.1Objectives:

1. The objective of this project is to solve the sudoku puzzle using Recursive Backtracking algorithm.
2. The graphical user interface is implemented using python programming.
3. Recursive Backtracking algorithm is implemented using python and the purpose of this solver is to check how long it take the computer to calculate a solution to such a problem.

NOTE: There are four major constraints necessary for every cell :

1. There should be a single value in every cell.
2. The values from 1 to 9 should exist only once in a row.
3. The values from 1 to 9 should exist only once in a column.
4. The values from 1 to 9 should exist only once in each square.

Chapter 3

The Sudoku Solver

3.1 Summary:

Sudoku solver is a puzzle game which is implemented using Recursive Backtracking algorithm in python in form of graphical user interface. The timer is implemented in the GUI board using the time module in python.

Depending on the branching factor the time efficiency also differs. As the branching factor increases it causes time inefficiency. But the algorithm used in this solver is very easy to implement and requires less line of code.

Also the algorithm used in this solver is independent of the programming language. All the effects used is making this solver is derived from Pygame module.

3.2 Advantages:

1. Solving time of a puzzle is not related with the difficulty level of the puzzle.
2. There will be solution for every puzzle unless it is invalid.
3. Backtracking algorithms takes less time as compared to Dancing link algorithm.
4. It provides step by step solution which is easy to understand.
5. Debugging is quite easy because in every step it has its own logical sequence.
6. It does not depend on any programming language.
7. The LOC(line of code) is less, so the chances of error in the program also reduces.

3.3 Disadvantages:

1. According to me, the only con I feel that this game requires huge level of patience.

2. It might be slow as compared to human solving.

3. It is of polynomial time complexity.

4. It requires recursion which is not good because space of stack is limited and can be used easily by recursion.

5.It is non-deterministic in nature.

6.Branching factor reduces the time efficiency.

7. It has various functions which are little bit expensive.

8. It is quite time consuming game.

9. Sense of incompletion in our mind while playing the game.

10. It may lead to anxiety.

Chapter 4

Methodology used for Sudoku Solver

4.1 Recursive Backtracking:

The strategy used in solving the sudoku is Backtracking. The procedure involves simply reverting back to the previous step as soon as we determine the solution of current step.

To find the solution for the grid , we simply find a blank location in the grid. The according to the rules of sudoku simply fill the grid from the number 1 to 9. We then try to recursively solve the grid with the new number. If there are no more grid left then it indicates that the puzzle is completely solved. On the other hand if we fail to find a solution for a grid , we backtrack and try a different number for a location and try a different solution.

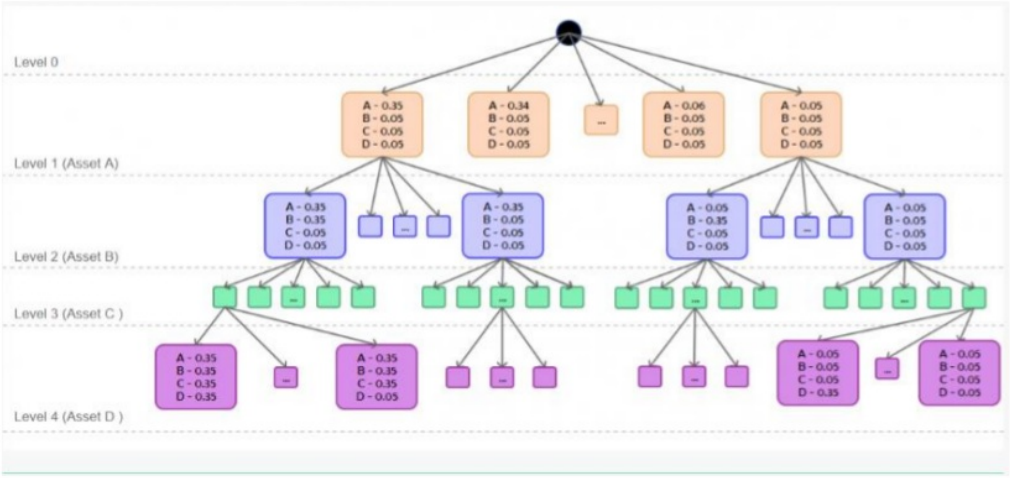


Fig: Recursive Backtracking

4.1.1 Algorithm:

Step 1: find the empty spaces in the grid

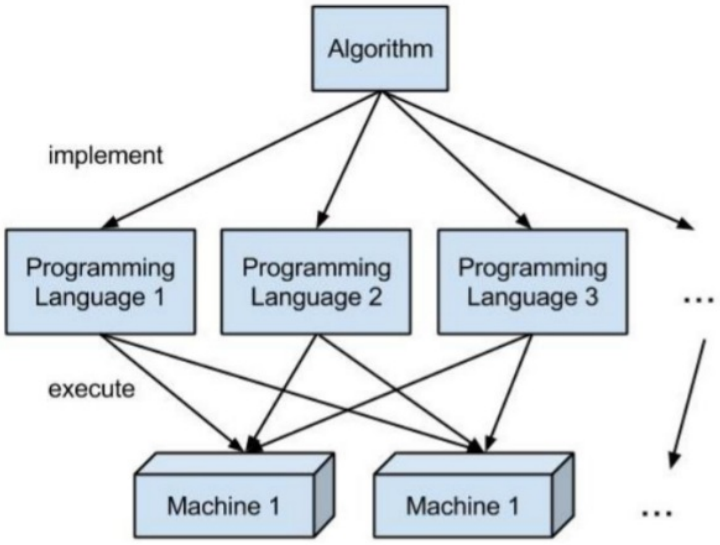
Step2: Then attempt to fill the digits from 1 to 9 in to the vacant spaces.

Step 3: Check whether the digit is valid in the vacant block present in the board.

Step 4: a) If the digit in the current grid is valid, recursively attempt to fill the remaining grids in the board using steps 1 to 3.

b) If it is not valid, reset the grid you just filled and go back to the previous step.

Step 5: Once the board is filled by the definition of algorithm we have to find a solution.



Code implementation of Recursive Backtracking:

```
# 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):
    # Check row
    for i in range(len(bo[0])):
        if bo[pos[0]][i] == num and pos[1] != i:
            return False

    # 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
```

➔ Result of Implementation of Recursive Backtracking:

```
C:\Users\User\PycharmProjects\pravtic
7 8 0  | 4 0 0  | 1 2 0
6 0 0  | 0 7 5  | 0 0 9
0 0 0  | 6 0 1  | 0 7 8
- - - - - - - - - - - -
0 0 7  | 0 4 0  | 2 6 0
0 0 1  | 0 5 0  | 9 3 0
9 0 4  | 0 6 0  | 0 0 5
- - - - - - - - - - - -
0 7 0  | 3 0 0  | 0 1 2
1 2 0  | 0 0 7  | 4 0 0
0 4 9  | 2 0 6  | 0 0 7
-------------------------
```

Fig: input for a sudoku without GUI

```
7 8 5  | 4 3 9  | 1 2 6
6 1 2  | 8 7 5  | 3 4 9
4 9 3  | 6 2 1  | 5 7 8
- - - - - - - - - - - -
8 5 7  | 9 4 3  | 2 6 1
2 6 1  | 7 5 8  | 9 3 4
9 3 4  | 1 6 2  | 7 8 5
- - - - - - - - - - - -
5 7 8  | 3 9 4  | 6 1 2
1 2 6  | 5 8 7  | 4 9 3
3 4 9  | 2 1 6  | 8 5 7

Process finished with exit code 0
```

Fig: solution by recursive backtracking without GUI

Chapter 5

Other algorithms for Sudoku Solver

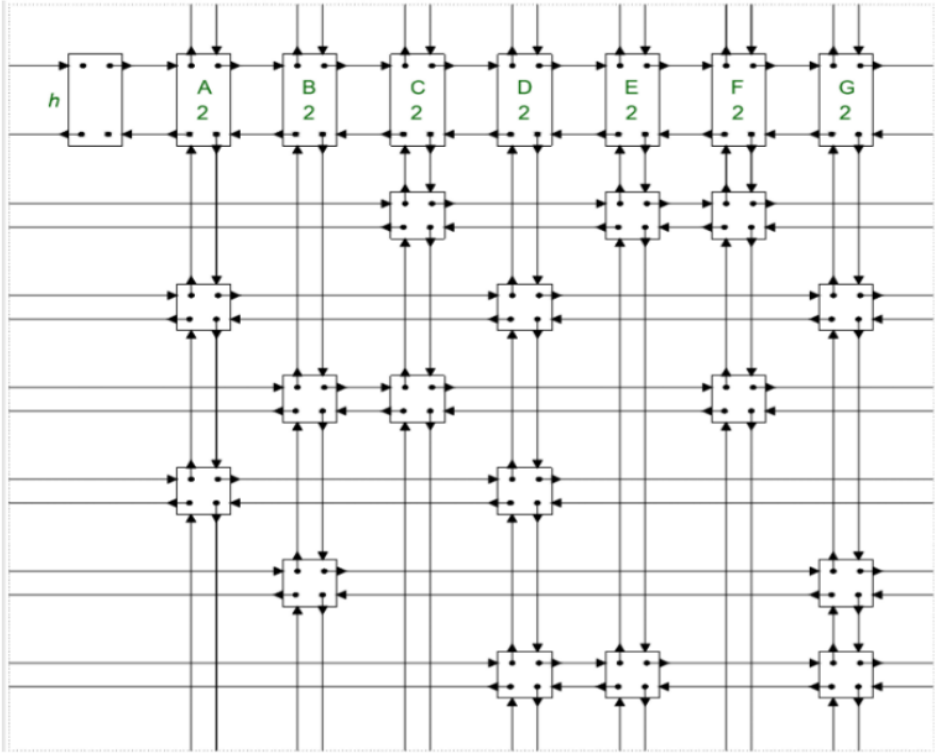
5.1 Dancing Link Algorithm:

Dancing link algorithm was proposed by Knuth which contain a set of rules to solve genuine cowl problem. It is also known as Algorithm X.

Algorithm X is recursive, depth-first search and backtracking algorithm. It is non-deterministic in nature means for a single input it may have multiple output.

It is very simple to build a complicated data structure, every requirement is fulfilled by python.

It is also readable and very flexible.



5.2 Pseudo code for Algorithm X:

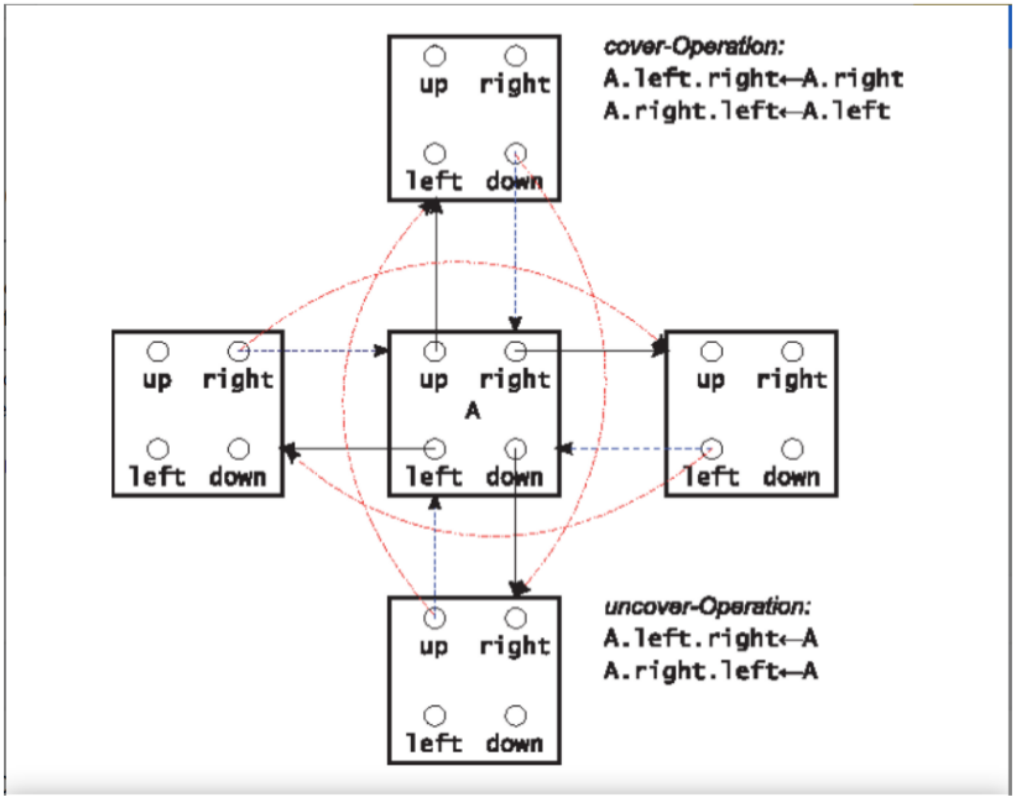
1. If the matrix *A* has no columns, the current partial solution is a valid solution; terminate successfully.
2. Otherwise choose a column *c*
3. Choose a row *r* such that $A_{r,c} = 1$
4. Include row *r* in the partial solution.
5. For each column *j* such that $A_{r,j} = 1$,

   for each row *i* such that $A_{i,j} = 1$,

   delete row *i* from matrix *A*.

   delete column *j* from matrix *A*.

6. Repeat this algorithm recursively on the reduced matrix *A*.

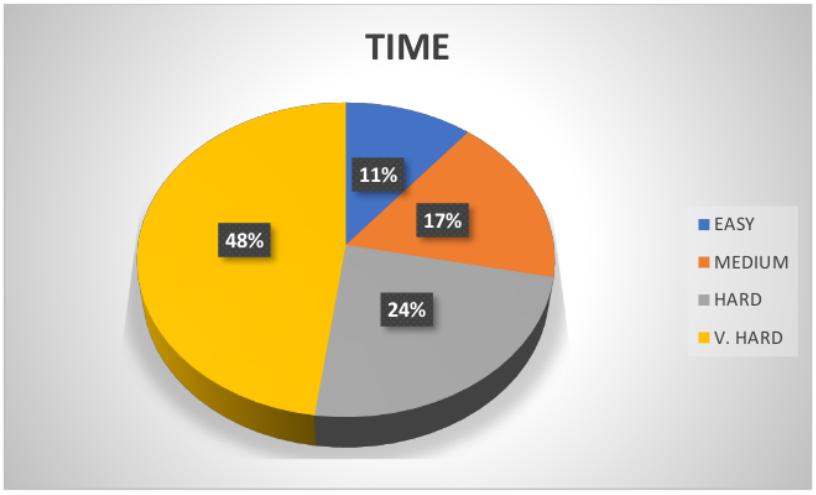


5.3Comparison between Recursive Backtracking and Dancing Links:

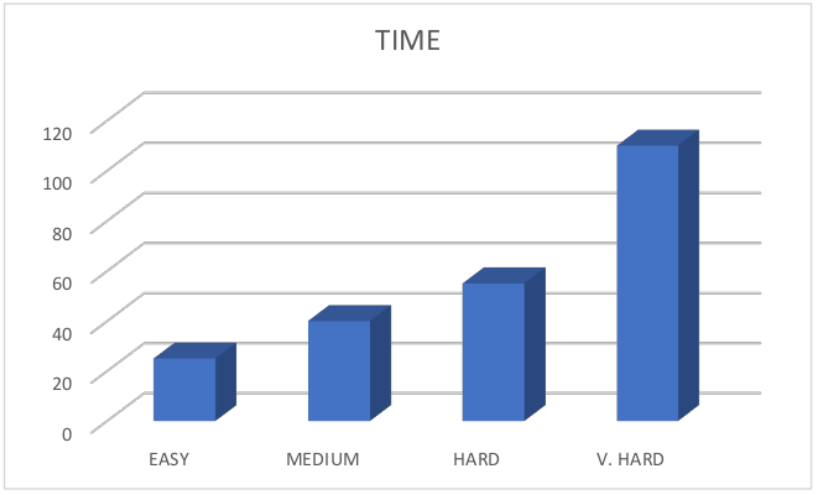
On the basis of time complexity the graph is plotted between the time and the level of the game.

1. Backtracking:

a)

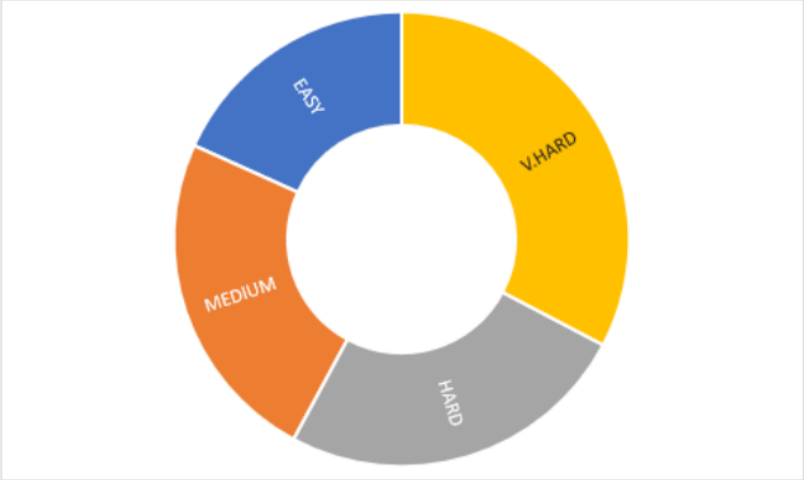


b)

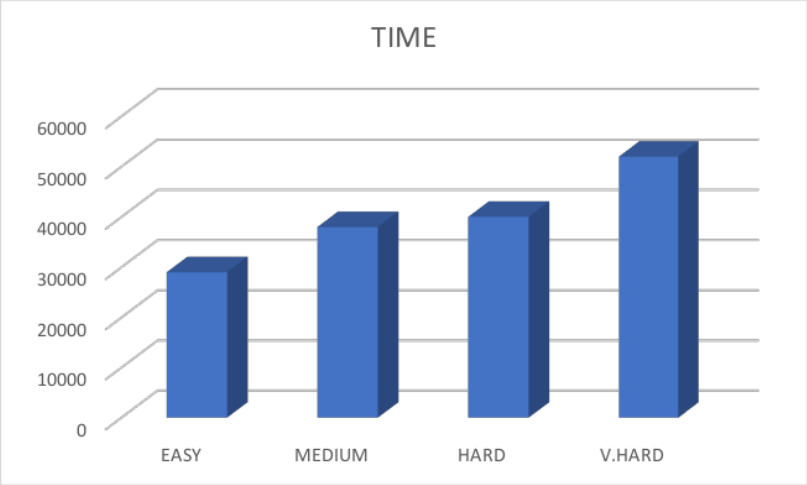


1. Dancing Links:

a)



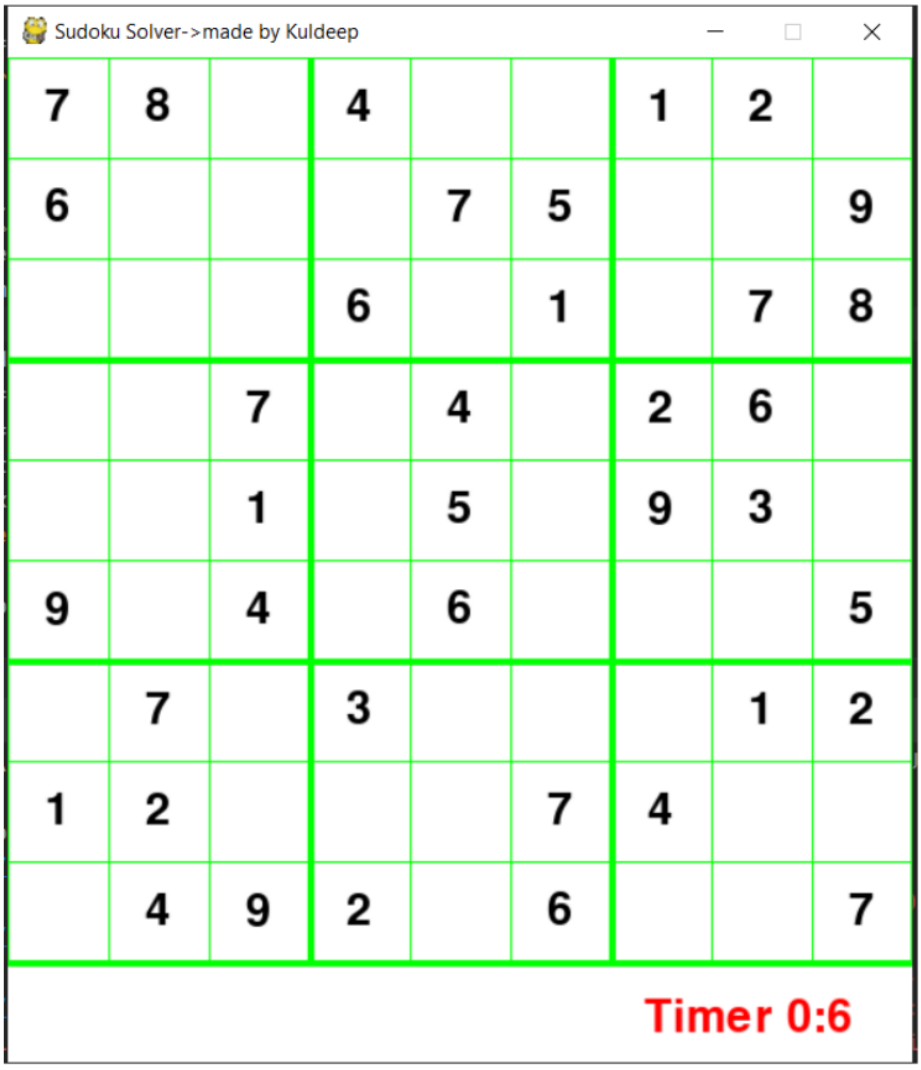
b)



Chapter : 6

Result and Implementation:

In this project I have used pygame module in python progrramming for making the gui og sudoku solver.



Output:

```
C:\Users\User\PycharmProjects\HelloKullu\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pydev\pydevconsole.py"

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\User\\PycharmProjects\\HelloKullu', 'C:/Users/User/PycharmProjects/HelloKullu'])

Python Console
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
C:/Users/User/PycharmProjects/HelloKullu/hello.py:60: DeprecationWarning: an integer is required (got type float).  Implicit conversion to integers using __int__ is deprecate
  pygame.draw.line(win, (0,0,0), (0, i*gap), (self.width, i*gap), thick)
C:/Users/User/PycharmProjects/HelloKullu/hello.py:61: DeprecationWarning: an integer is required (got type float).  Implicit conversion to integers using __int__ is deprecate
  pygame.draw.line(win, (0, 0, 0), (i * gap, 0), (i * gap, self.height), thick)
C:/Users/User/PycharmProjects/HelloKullu/hello.py:128: DeprecationWarning: an integer is required (got type float).  Implicit conversion to integers using __int__ is deprecate
  win.blit(text, (x + (gap/2 - text.get_width()/2), y + (gap/2 - text.get_height()/2)))
C:/Users/User/PycharmProjects/HelloKullu/hello.py:131: DeprecationWarning: an integer is required (got type float).  Implicit conversion to integers using __int__ is deprecate
  pygame.draw.rect(win, (255,0,0), (x,y, gap ,gap), 3)
C:/Users/User/PycharmProjects/HelloKullu/hello.py:125: DeprecationWarning: an integer is required (got type float).  Implicit conversion to integers using __int__ is deprecate
  win.blit(text, (x+5, y+5))
Success
Success
Success
Success
Success
Success
Success
Success
```

```
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Success
Wrong
Success
Success
Success
Success
Success
Success
Success
Game over
```

Chapter 7

Conclusion:

All we conclude from the research is that our solver can solve puzzles with small amount of perspective.

On the basis various comarison it is clear that Dancing links algorithm is slower than the other algorithms used for solving sudoku. But as we increase the level of difficulty the dancing link performs better.It is around 100 times faster.

Coming to Recursive Backtracking algorithm , it is best for among the both either compared to Constrain algorithm or any Rule based algorithms.

It is found that rule based algorithm is somewhat slower than Recursive Backtracking on counting the initiation and memory allocation. Eventhough we can improve the memory management based on some rules but it was considered unlikely. As Recursive Backtracking is very easy to implement we consider finding a solution based rule algorithm unnecessary.

From the above discussion, it is clear that Recursive Backtracking is easy to implement and a better algorithm for computationally solving Sudoku.

Chapter 8

Future Scope:

This project has potential, but it can be extended. There are are some improvements such as it is only built for desktop. This project should be made for android and for ios also.

And it also requires improvement for the '0' problem. This is somewhat serious problem that should have addressed already and it can be removed by using of some 'if' statements or the grid must remain empty instead of zero.

By analysing the result it is clear that time complexity factor still depends on the difficulty level of the puzzle. As the difficulty level of the puzzle increases the time complexity also increases.

So in future, we need the better version of algorithms which can reduce the factor of time complexity.

kuldeep

ORIGINALITY REPORT

7%	6%	4%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.geeksforgeeks.org Internet Source	6%
2	Submitted to University of Liverpool Student Paper	1%

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

kuldeep