GeeksforGeeks

A computer science portal for geeks

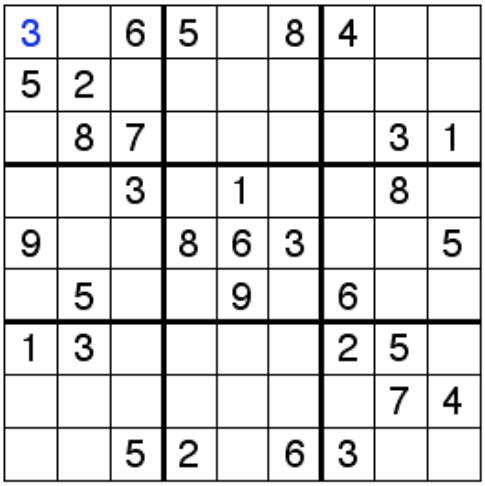
GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Backtracking | Set 7 (Sudoku)

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.



Naive Algorithm
The Naive Algorithm is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found.

Backtracking Algorithm
Like all other Backtracking problems, we can solve Sudoku by one by one assigning numbers to empty cells. Before assigning a number, we check whether it is safe to assign. We basically check that the same number is not present in current row, current column and current 3X3 subgrid. After checking for safety, we 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 we try next number for current empty cell. And if none of number (1 to 9) lead to solution, we return false.

```
  Find row, col of an unassigned cell
  If there is none, return true
  For digits from 1 to 9
    a) If there is no conflict for digit at row,col
        assign digit to row,col and recursively try fill in rest of grid
    b) If recursion successful, return true
    c) Else, remove digit and try another
  If all digits have been tried and nothing worked, return false
```

Following is C++ implementation for Sudoku problem. It prints the completely filled grid as output.

```
// A Backtracking program  in C++ to solve Sudoku problem
#include <stdio.h>

// UNASSIGNED is used for empty cells in sudoku grid
#define UNASSIGNED 0

// N is used for size of Sudoku grid. Size will be NxN
#define N 9

// This function finds an entry in grid that is still unassigned
bool FindUnassignedLocation(int grid[N][N], int &row, int &col);

// Checks whether it will be legal to assign num to the given row,col
bool isSafe(int grid[N][N], int row, int col, int num);

/* Takes a partially filled-in grid and attempts to assign values to
  all unassigned locations in such a way to meet the requirements
  for Sudoku solution (non-duplication across rows, columns, and boxes) */
bool SolveSudoku(int grid[N][N])
```

```
{
    int row, col;

    // If there is no unassigned location, we are done
    if (!FindUnassignedLocation(grid, row, col))
       return true; // success!

    // consider digits 1 to 9
    for (int num = 1; num <= 9; num++)
    {
        // if looks promising
        if (isSafe(grid, row, col, num))
        {
            // make tentative assignment
            grid[row][col] = num;

            // return, if success, yay!
            if (SolveSudoku(grid))
                return true;

            // failure, unmake & try again
            grid[row][col] = UNASSIGNED;
        }
    }
    return false; // this triggers backtracking
}

/* Searches the grid to find an entry that is still unassigned. If
   found, the reference parameters row, col will be set the location
   that is unassigned, and true is returned. If no unassigned entries
   remain, false is returned. */
bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry
   in the specified row matches the given number. */
bool UsedInRow(int grid[N][N], int row, int num)
{
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry
   in the specified column matches the given number. */
bool UsedInCol(int grid[N][N], int col, int num)
{
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Returns a boolean which indicates whether any assigned entry
```

```
   within the specified 3x3 box matches the given number. */
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num)
{
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row+boxStartRow][col+boxStartCol] == num)
                return true;
    return false;
}

/* Returns a boolean which indicates whether it will be legal to assign
   num to the given row,col location. */
bool isSafe(int grid[N][N], int row, int col, int num)
{
    /* Check if 'num' is not already placed in current row,
       current column and current 3x3 box */
    return !UsedInRow(grid, row, num) &&
           !UsedInCol(grid, col, num) &&
           !UsedInBox(grid, row - row%3 , col - col%3, num);
}

/* A utility function to print grid  */
void printGrid(int grid[N][N])
{
    for (int row = 0; row < N; row++)
    {
       for (int col = 0; col < N; col++)
             printf("%2d", grid[row][col]);
        printf("\n");
    }
}

/* Driver Program to test above functions */
int main()
{
    // 0 means unassigned cells
    int grid[N][N] = {{3, 0, 6, 5, 0, 8, 4, 0, 0},
                      {5, 2, 0, 0, 0, 0, 0, 0, 0},
                      {0, 8, 7, 0, 0, 0, 0, 3, 1},
                      {0, 0, 3, 0, 1, 0, 0, 8, 0},
                      {9, 0, 0, 8, 6, 3, 0, 0, 5},
                      {0, 5, 0, 0, 9, 0, 6, 0, 0},
                      {1, 3, 0, 0, 0, 0, 2, 5, 0},
                      {0, 0, 0, 0, 0, 0, 0, 7, 4},
                      {0, 0, 5, 2, 0, 6, 3, 0, 0}};
    if (SolveSudoku(grid) == true)
          printGrid(grid);
    else
         printf("No solution exists");

    return 0;
}
```

Output:

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

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

References:
http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Backtracking

Writing code in comment? Please use **ideone.com** and share the link here.

imshashank • a month ago
A simple implementation in JAVA:

http://ideone.com/bnlW8f
• Reply • Share ›

the_c0der • 3 months ago
what is the time complexity of this solution ?
1 • Reply • Share ›

Mohamed Bilal • 5 months ago
The above backtracking algorithm can solve only SIMPLE sudoku puzzles in reasonable time because of following two reasons.

1) It finds a random unassigned array which is not efficient. We have squares which are most constrained and solving this first would prune our search tree a lot

2) It tests again all possible values for a chosen square which would exploits the search tree. Many times choosing a value would make some future unassigned squares can not hold any acceptable values. Finding this information prior and avoiding such values will help us prune the search tree a lot.

Here are the two improvements I propose to prune the search tree.

1) When choosing an unassigned square, always find the most constrained square in the grid

2) When choosing possible values for a square, look ahead if such values can cause future open squares to be invalid. If so remove those values from possible candidates for that square.

I have coded the solution with above improvements in python. Please check the code here at http://ideone.com/mCzsC0

1 ^ | v • Reply • Share ›

helper • 6 months ago

we read this backtracking in our Artificial intelligence course. thanks a lot for sharing so many examples.....

^ | v • Reply • Share ›

aa1992 • 7 months ago

can anyone suggest which i the best book for learning algorithms specially puzzles?

^ | v • Reply • Share ›

Sanky • 8 months ago

I think there is a flaw with the isSafe() function. If row or col is passed as 3,6 or 9. Then the value passed to UsedinBox() will be 3,6 or 9 respectively and therefore the function will check cells 3,4, and 5 instead of 1,2,3.

1 ^ | v • Reply • Share ›

goldconker ➚ Sanky • 6 months ago

It's 0 indexed, that is the correct behavior.

^ | v • Reply • Share ›

Bala Poornima • 8 months ago

Hi,

I have implemented the same program in Java. But i am not getting the correct output. Its giving solution not exits.

Can anyone tell what is wrong in my implementation ?

Can anyone tell what is wrong in my implementation ?

```
package com.practice.backtracking;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class SudokuProblem {

/**

* @param args
```

see more

4 • Reply • Share ›

shashi ➚ Bala Poornima • 4 months ago

solve it like this

```
/**
*
*/
package com.kant.microsoftPrep;

/**
* @author shashi
*
*/
public class SudokuProblem {

/**
* @param args
*/
public static void main(String[] args) {
int grid[][] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
{ 5, 2, 0, 0, 0, 0, 0, 0, 0 } { 0, 8, 7, 0, 0, 0, 0, 3, 1 }
```

see more

• Reply • Share ›

<HoldOnLife!#> • 10 months ago

what is the value assigned to int row, col; ?? in solve sudoku method () , row and col are passed in findunassigned location() too !

4 • Reply • Share ›

adasd <HoldOnLife!#> • 7 months ago
Many compilers assign a 0 as the default value ..
so in the first call to solveSudoku() the values are
row == and col = 0
• Reply • Share ›

pkzz • 10 months ago
why have we used row - row%3 and col - col%3
• Reply • Share ›

Rahul Shah pkzz • 10 months ago
So as to get the upper left corner of the current box
• Reply • Share ›

Sanky Rahul Shah • 8 months ago
I think there is a flaw with the isSafe() function. If row or col is passed as 3,6 or 9. Then the value passed to UsedinBox() will be 3,6 or 9 respectively and therefore the function will check cells 3,4, and 5 instead of 1,2,3. Right?
1 • Reply • Share ›

Karthikeyan Kumar Sanky • 3 months ago
You are right. I too had the same doubt and corrected them in my version of the program written in C#.
http://ideone.com/i0il82
• Reply • Share ›

vishal sinha • 10 months ago
could u plz explain the backtracking part.
• Reply • Share ›

Aveek Biswas • 10 months ago
Java code using same algo. http://ideone.com/dQnXY2
Used a wrapper class to make up for the reference variables.
• Reply • Share ›

anil • a year ago
During backtracking aren't we modifying the original input too ??
• Reply • Share ›

AlienOnEarth anil • a year ago
Check FindUnassignedLocation Method. It finds unassigned location and then fills data to it. If it returns false, that means everything went well. So returning true. I hope this clears your doubt

clears your doubt.

• Reply • Share ›

anil ➚ anil • a year ago

ok got it, we are actually not overwriting the original input positions at all ..

1 • Reply • Share ›

Argha • a year ago

//A simple c++ version

```
#include <iostream>

#include <conio.h>

#include <fstream>

#define UNASSIGNED 0

#define MAX 9

using namespace std;

class Sudoku

{

private:

int A[MAX][MAX];
```

see more

• Reply • Share ›

KK • a year ago

An improvement to FindUnassignedLocation as proposed by Skiena in The Algorithm Design Manual. This might not make any different in easy puzzles as mentioned in this page. But for the puzzle at the end of this comment, it took 20 seconds on my PC without this optimization and 1 second with it.

```
int SuitableVals(int grid[N][N], int row, int col)
{
int total = 0;
for (int i = 1; i < 10; i++)
{
if (isSafe(grid, row, col, i))
total++;
}
return total;
```

```
}

bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
```

see more

• Reply • Share ›

Sanjay Agarwal ➔ KK • a year ago

Good optimization. :)

• Reply • Share ›

Shailedra • a year ago

@admin this algo. does not give solution to many sudoku puzzle...like

{{0, 0, 6, 0, 0, 0, 0, 0, 4},
{0, 0, 0, 8, 6, 0, 7, 3, 0},
{0, 4, 0, 3, 5, 4, 0, 0, 2},
{1, 7, 0, 4, 0, 0, 6, 0, 0},
{0, 9, 0, 0, 0, 0, 0, 8, 0},
{0, 0, 8, 0, 0, 6, 0, 1, 7},
{2, 0, 0, 0, 8, 1, 0, 4, 0},
{0, 6, 7, 0, 4, 3, 0, 0, 0},
{8, 0, 0, 0, 0, 0, 3, 0, 0}} and many more test cases ...
i tried many test cases...
i think it should include to find the min. number to get the optimal solution....

• Reply • Share ›

affiszerv ➔ Shailedra • a year ago

Your example has two 4s on row 3, that's why it gives no solution.

1 • Reply • Share ›

charamander • a year ago

I took the array b[][] to keep track of the data initially filled in the question. In this problem I gave it manually to suit the input(written at last)....b[][] contains 100 at the place already given in the quesiton and so cant be modified while solving the sudoku. I am not getting any output. I am thankful to anyone who can pull me out of this.....

whats wrong with my fill_sudoku function????

```
# include <stdlib.h>

# include <stdio.h>

#define N 9
```

int b[N][N]={{100, 0, 100, 100, 0, 100, 100, 0, 0},

{100, 100, 0, 0, 0, 0, 0, 0, 0},

{0, 100, 100, 0, 0, 0, 0, 100, 100},

{0, 0, 100, 0, 100, 0, 0, 100, 0},

see more

Reply • Share ›

chandeepsignh • 2 years ago

I copied the above code in Java but it produces the same result as the default input not the output shown above.
Where am I making mistake? Please suggest.

Reply • Share ›

LinuxWorld • 2 years ago

/* this solving the sudo ku using hashing // bashed on the // advice list and the backtracking */

#include
#include
#include
int sudoku[10][10] = { {0,0,0,0,0,0,0,0,0,0},
{0,0,8,0,0,0,9,0,7,2},
{0,6,0,0,0,0,0,0,0,4},
{0,0,5,0,0,6,0,9,0,1},
{0,2,0,0,0,0,5,0,0,0},
{0,3,7,6,1,8,4,2,9,5},
{0,0,0,0,3,0,0,0,0,6},
{0,9,0,1,0,2,0,0,4,0},
{0,5,0,0,0,0,0,0,0,9},
{0,8,4,0,9,0,0,0,2,0}}; // very hard

int row[10][10] = {0} ; // this matrix used for the hashing the row and the digit
int col[10][10] = {0} ; // this is for the hashing the column

see more

3 Reply • Share ›

Gengis Khan • 2 years ago

Never mind. I got it.

Reply • Share ›

Gengis Khan • 2 years ago

There's a bug. How are the row and col variables getting initialized? They aren't even declared.

1 ^ | v • Reply • Share ›

Harshit Gupta ➚ Gengis Khan • 2 years ago

They are initialised by FindUnassignedLocation() ...
it took address of both.

2 ^ | v • Reply • Share ›

Hitesh • 2 years ago

```
/*
    Title                   # SUDOKU solver in C language
    Author                  # Hitesh Dholaria
    Email                   # hitesh.dholaria@gmail.com

    Input                   # Following is a sample SUDOKU problem stored in the file n
                                # Here, 0 represents empty place
                                # So, our overall goal is to replace such 0-places
                                # There are total of 9 rows, 9 columns and 9 blocks

                                008000050
                                370009010
                                000140007
                                050020600
                                001908300
                                003060020
                                700084000
                                080200063
```

see more

3 ^ | v • Reply • Share ›

minhaz • 2 years ago

Don't get it, how the row and column being updated in SolveSudko function?

^ | v • Reply • Share ›

rohit ➚ minhaz • 2 years ago

Can you please tell me how are Rows and Column getting updated

```
/* Paste your code here (You may delete these lines if not writing code) */
```

^ | v • Reply • Share ›

vignesh m ➚ rohit • 2 years ago

we are doing pass by reference hence values are updated , check signature below

```
bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
```

1 • Reply • Share ›

minhaz minhaz • 2 years ago

Oops got it.

• Reply • Share ›

proxhotdog • 2 years ago

I have ported this code in ruby and it said stack level too deep (SystemStackError)

• Reply • Share ›

abc • 3 years ago

why row-row%3

• Reply • Share ›

abc abc • 3 years ago

Sorry understood!

1 • Reply • Share ›

mani abc • 2 years ago

I don't understand. can u please explain ?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 • Reply • Share ›

mahesh mani • 2 years ago

Incase the UNASSIGNED number is 5th row we need to check if the num fits in the 3*3 block.

5 - 5%3 = 5 - 2 = 3 (starting row of that 3*3 block)

take for example we are updating in 8th row

8 - 8%3 = 8 - 2 = 6 (starting row of that 3*3 block). Beautiful thinking .

• Reply • Share ›

Venki • 3 years ago

An interesting question would be "How can we assure that the given input is feasible?". The situation faced by Sudoku game designers while drafting Easy, Medium and Hard puzzles.

Any thoughts on how can we ensure minimum input which can lead to a solution? Or sufficiency of input to lead a valid solution.

1 ▲ | ▼ • Reply • Share ›

Aashish ➚ Venki • 3 years ago

@Venki,
Bit vector can be used to ensure whether an input will
lead to solution.
9 bits can be used to represent any of the digits from 1 to 9[we are talking about 9x9 Sudoku,right!].
To check whether a Cell is pre-filled & thus can't be used for user input, it can be easily checked by ANDing the 9 bit vector with 511[2^9-1].If the result of AND operation is zero[none of the bit of the bit vector is set], the cell is empty.

The below structure can be handy.

```
 struct mask
{
        int bit:9;
};

struct mask m[9][9];
```

A 2D bit vector of size NxN[9 here] is needed to represent each cell.
The memory has thus reduced from 9x9x4 = 324 bytes[if bit vectors are not used] to 9x9x9/8 = 92 bytes.

Let me know if we can do better.

▲ | ▼ • Reply • Share ›

Venki ➚ Aashish • 3 years ago

Whether to represent the matrix as bit vector or boolean array, or int/char array, how does it matter? We will conserve space, but all are same.

My question is given a partially filled grid, how can we ensure that the user can successfully arrive at valid solution?

Putting in other words, generate all possible grid permutations. Now discard the invalid grids. From the remaining grids, take any grid. Gradually remove random elements one after another from this grid. How many such random deletions can be possible? Say, we left with N elements in the grid after 81-N deletions. Can we ensure that the user will be able to fill these deleted elements? What if we delete one more element, and left with N-1 elements on the grid. Now is it possible to fill it in?

There must be a criteria to design the puzzle. I am looking for more thoughts.

• Reply • Share ›

asdad ➚ Venki • 7 months ago

Even I had the same thought while solving the question :
fora a 9*9 sudoku
min # givens = 17

This wiki link has a brief explaination
http://en.wikipedia.org/wiki/M...

Refer Topic: Minimum number of givens

• Reply • Share ›

Diksha • 3 years ago

```
 #include <stdio.h>
#include <stdlib.h>
#define SIZE 9

int isValid(int (*sudoku)[SIZE],int row,int col,int weight)
{
        int i,j,start_row,start_col;

        for(i=0;i<SIZE;i++)
                if(sudoku[row][i]==weight)
                        return 0;

        for(i=0;i<SIZE;i++)
                if(sudoku[i][col]==weight)
                        return 0;

        start_row=(row/3)*3;
        start_col=(col/3)*3;
```

see more

1 • Reply • Share ›

jinzhi chen ➚ Diksha • a year ago

very nice. thanks

• Reply • Share ›

Subin ➚ Diksha • 2 years ago

Nice Code. Its working fine for me.
only you have to handle col variable from going into array index out of bounds error

which can be handled with one simple if case....

```
/* Paste your code here (You may delete these lines if not writing code) */
```

Reply • Share ›

varun • 3 years ago

What should be the complexity for this algo..

Let's say total empty columns are n then I think complexity should be 9^n because for each empty box maximum tries should be 9 (1...9).

What others think about complexity???

```
/* Paste your code here (You may delete these lines if not writing code) */
```

Reply • Share ›

Diksha ➦ varun • 3 years ago

Are you sure the time complexity is 9^n.
AFAIT, each empty slot can be filled in 9 ways.
So, the time complexity should be (no. of empty slots)^9.

Let me know if i am missing anything.

Reply • Share ›

Load more comments

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Recent Comments

- Ashish Aggarwal

  Try Data Structures and Algorithms Made Easy -...

Algorithms · 17 minutes ago

- Vlad

  Thanks. Very interesting lectures.

  Expected Number of Trials until Success · 1 hour ago

- cfh

  My implementation which prints the index of the...

  Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- Gaurav pruthi

  forgot to see that part ;)

  Bloomberg Interview | Set 1 (Phone Interview) · 1 hour ago

- saeid aslami

  thanks

  Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm) · 1 hour ago

- Cracker

  Implementation:...

  Implement Stack using Queues · 2 hours ago