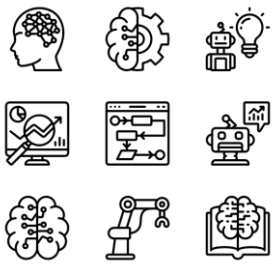
Computer Science for Practicing Engineers

Một số bài toán giải bằng phương pháp quay lui



TS. Huỳnh Bá Diệu
Email: dieuhb@gmail.com
Phone: 0914146868

1

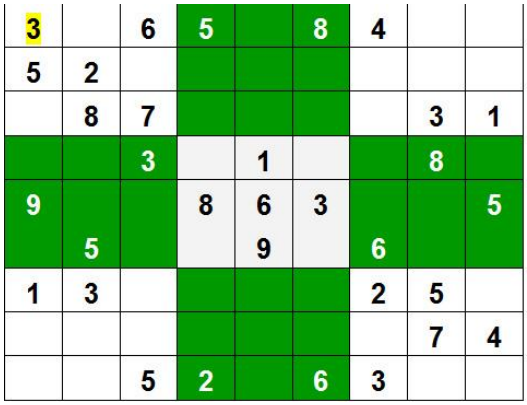
Backtracking Problems

1. Sudoku
2. Rat in a Maze
3. Count number of ways to reach destination in a Maze
4. ***Count number of ways to reach destination in a maze 2***

2

Backtracking - 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.

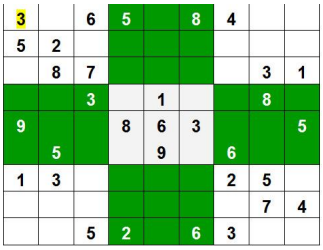


3

Backtracking - Sudoku

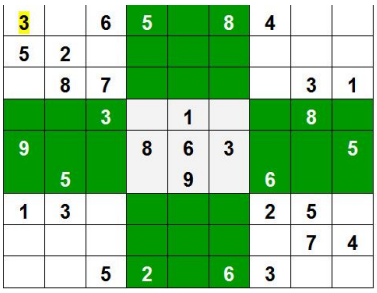
Naive Algorithm

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.



4

Backtracking **Algorithm** - Sudoku

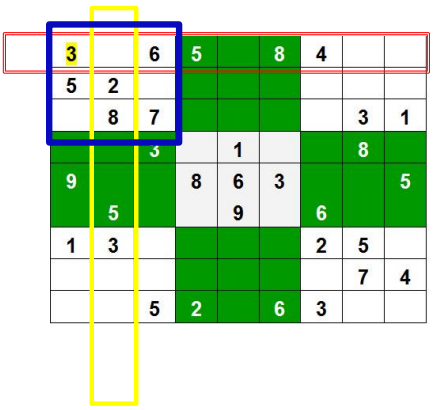


We can solve Sudoku by one by one assigning numbers to empty cells. Before assigning a number, we check whether it is safe to assign.

5

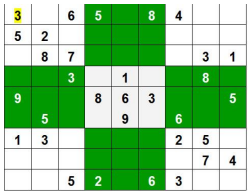
Backtracking **Algorithm** - Sudoku

We basically check that the same number is not present in the 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 the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, we return false.



6

Backtracking **Algorithm** - Sudoku



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

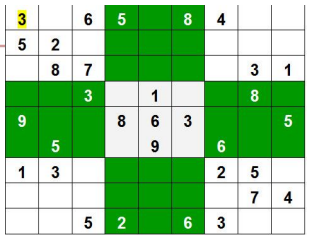
b) If recursion successful, return true

c) Else, remove digit and try another

If all digits have been tried and nothing worked, return false.

7

Backtracking - Sudoku

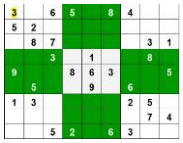


```
bool SolveSudoku(int grid[N][N]) {
   int row, col;
   if (!FindUnassignedLocation(grid, row, col))   return true;
  for (int num = 1; num <= 9; num++)
    if (isSafe(grid, row, col, num))   {
           grid[row][col] = num;
          if (SolveSudoku(grid))     return true;
           grid[row][col] = UNASSIGNED;
     }
    return false;
}
```

```
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;
}
```

8

Backtracking - Sudoku



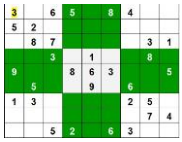
```
bool isSafe(int grid[N][N], int row,  int col, int num)
{
return !UsedInRow(grid, row, num) &&  !UsedInCol(grid, col, num) &&  !UsedInBox(grid, row - row % 3 ,  col - col % 3,
num) &&    grid[row][col] == UNASSIGNED;
}
```

```
bool UsedInBox(int grid[N][N], int SR, int SC, int num)  {
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row +SR]  [col + SC] == num)  return true;
   return false;
}
```

```
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;
}
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;
}
```

9

Backtracking - Sudoku



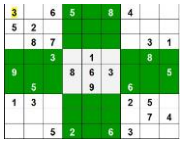
```
bool isSafe(int grid[N][N], int row,  int col, int num)
{
return !UsedInRow(grid, row, num) &&  !UsedInCol(grid, col, num) &&  !UsedInBox(grid, row - row % 3 ,  col - col % 3, num)
&&    grid[row][col] == UNASSIGNED;
}
```

```
bool UsedInBox(int grid[N][N], int SR, int SC, int num)  {
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row +SR]  [col + SC] == num)  return true;
   return false;
}
```

```
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;
}
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;
}
```

10

Backtracking - Sudoku



```
bool isSafe(int grid[N][N], int row,  int col, int num)
{
return !UsedInRow(grid, row, num) &&  !UsedInCol(grid, col, num) &&  !UsedInBox(grid, row - row % 3 ,  col - col % 3,
num) &&    grid[row][col] == UNASSIGNED;
}
```

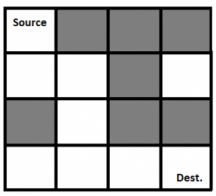
```
bool UsedInBox(int grid[N][N], int SR, int SC, int num)  {
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row +SR]  [col + SC] == num)  return true;
   return false;
}
```

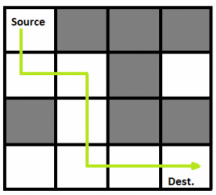
```
bool UsedInRow(int grid[N][N], int row, int num)  {
   for (int col = 0; col < N; col++)
      if (grid[row][col] == num)  return true;
   return false;
}
bool UsedInCol(int grid[N][N], int col, int num)  {
   for (int row = 0; row < N; row++)
      if (grid[row][col] == num)          return true;
   return false;
}
```

11

Backtracking - Rat in a Maze

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and has to reach the destination. The rat can move only in two directions: forward and down.





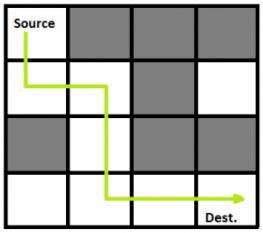
12

Backtracking - Rat in a Maze

In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination. Note that this is a simple version of the typical Maze problem. For example, a more complex version can be that the rat can move in 4 directions and a more complex version can be with a limited number of moves.

{1, 0, 0, 0}
{1, 1, 0, 0}
{0, 1, 0, 0}
{0, 1, 1, 1}

All enteries in solution path are marked as



13

Backtracking **Algorithm** - Rat in a Maze

If destination is reached print the solution matrix
else

a) Mark current cell in solution matrix as 1.
b) Move forward in the horizontal direction and recursively check if this move leads to a solution.
c) If the move chosen in the above step doesn't lead to a solution then move down and check if this move leads to a solution.
d) If none of the above solutions works then unmark this cell as 0 (BACKTRACK) and return false.

14

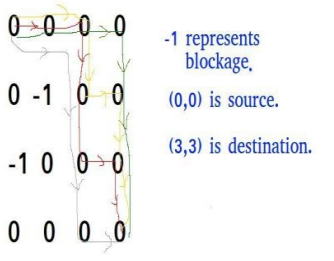
Backtracking - Rat in a Maze

```
bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]) {
    if (x == N - 1 && y == N - 1) {        sol[x][y] = 1;        return true;     }
     if (isSafe(maze, x, y) == true) {
      sol[x][y] = 1;          /* Move forward in x direction */
      if (solveMazeUtil(maze, x + 1, y, sol) == true)    return true;
        /* If moving in x direction doesn't give solution then move down in y direction  */
      if (solveMazeUtil(maze, x, y + 1, sol) == true)   return true;
        /* If none of the above movements work then BACKTRACK:    unmark x, y as part of solution path
*/
      sol[x][y] = 0;
      return false;
   }
    return false;
}
```

15

Count number of ways to reach destination in a Maze

Given a maze with obstacles (*chướng ngại vật*), count number of paths to reach rightmost-bottommost cell from topmost-leftmost cell. A cell in given maze has value -1 if it is a blockage or dead end, else 0.



There are total four paths from source to destination

16

Count number of ways to reach destination in a Maze

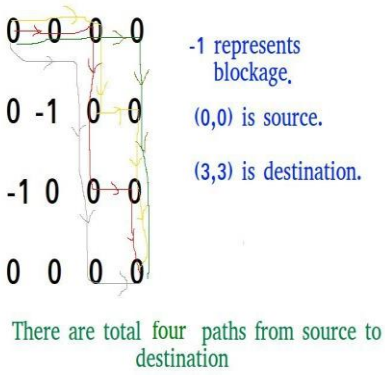
From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only.

Input: maze[R][C] =

```
{{0,  0, 0, 0},
 {0, -1, 0, 0},
 {-1, 0, 0, 0},
 {0,  0, 0, 0}};
```

Output: 4

There are four possible paths.



There are total four paths from source to destination

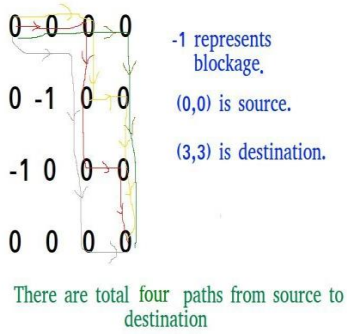
17

Count number of ways to reach destination in a Maze

The idea is to modify the given grid[][] so that **grid[i][j] contains count of paths to reach (i, j) from (0, 0) if (i, j) is not a blockage, else grid[i][j] remains -1.**

We can recursively compute grid[i][j] using below formula and finally return grid[R-1][C-1]

```
// If current cell is a blockage
if (maze[i][j] == -1)
   maze[i][j] = -1; //  Do not change
// If we can reach maze[i][j] from maze[i-1][j] then increment count.
else if (maze[i-1][j] > 0)
   maze[i][j] = (maze[i][j] + maze[i-1][j]);
// If we can reach maze[i][j] from maze[i][j-1] then increment count.
else if (maze[i][j-1] > 0)
   maze[i][j] = (maze[i][j] + maze[i][j-1]);
```



There are total four paths from source to destination

18

Count number of ways to reach destination in a Maze

```
int countPaths(int maze[][C]) {
        if (maze[0][0]==-1) return 0;
        for (int i=0; i<R; i++)
            if (maze[i][0] == 0)  maze[i][0] = 1; else break;
        for (int i=1; i<C; i++)
            if (maze[0][i] == 0) maze[0][i] = 1;  else break;
        for (int i=1; i<R; i++)
            for (int j=1; j<C; j++)
            {
                if (maze[i][j] == -1) continue;
                if (maze[i-1][j] > 0) maze[i][j] = (maze[i][j] + maze[i-1][j]);
                if (maze[i][j-1] > 0) maze[i][j] = (maze[i][j] + maze[i][j-1]);
            }
        return (maze[R-1][C-1] > 0)? maze[R-1][C-1] : 0;
}
```

19

Backtracking - ***Count number of ways to reach destination in a maze 2***

Given a maze of 0 and -1 cells, the task is to find all the paths from (0, 0) to (n-1, m-1), and every path should pass through at least one cell which contains -1. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only.

Input: *maze[][] = {*
{0, 0, 0, 0},
{0, -1, 0, 0},
{-1, 0, 0, 0},
{0, 0, 0, 0}}
Output: *16*

20

Backtracking - *Count number of ways to reach destination in a maze 2*

Approach:

To find all the paths which go through at least one marked cell (cell containing -1). If we find the paths that do not go through any of the marked cells and all the possible paths from (0, 0) to (n-1, m-1) then we can find all the paths that go through at least one of the marks cells.

Number of paths that pass through at least one marked cell = (Total number of paths – Number of paths that do not pass through any marked cell)

21

Homework

1/ Implement a program to *count number of ways to reach destination in a maze 2.*

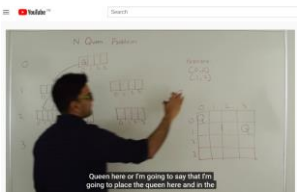
2/ Find the solution for the Knight Tour Problem.

https://en.wikipedia.org/wiki/Knight%27s_tour

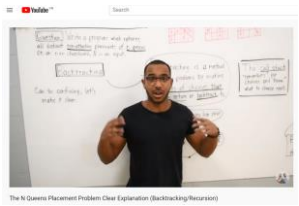
22

Link YouTube

https://www.youtube.com/watch?v=xouin83ebxE



https://www.youtube.com/watch?v=wGbuCyNpxIg



23