DAA ASSIGNMENT

21071A6722

J.S.RITHVIK

Sudoku using Backtracking

Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning anumber, check whether it is safe to assign.

Check that the same number is not present in the current row, current column and current 3X3subgrid. 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 thenext 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.

Using the backtracking algorithm, we will solve the Sudoku problem.

ALGORITHM:

isPresentInCol(col, num)

Input: The column, and the targeted number.
Output – True when the number is present in the given column.

```
Begin
    for each row r in the grid, doif grid[r, col]
        = num, then
            return true
    done
    return false otherwise
En
d
```

isPresentInRow(row, num)

Input – The row, and the targeted number.

Output – True when the number is present in the given column.

```
Begin
    for each column c in the grid, doif grid[row,
        c] = num, then
            return true
    done
    return false otherwise
En
d
```

isPresentInBox(boxStartRow, boxStartCol, num)

Input – The starting row and column of a 3 x 3 box, and the targeted number.

Output – True when the number is present in the box.

```
Begin
    for each row r in boxStartRow to next 3 rows, do
        for each col r in boxStartCol to next 3 columns, doif grid[r, c] = num,
            then
                return true
        done
    done
    return false otherwise
En
d
```

findEmptyPlace(row, col)

Input: row and column in the grid.

Output – If the grid[row, col] is empty, then return true, otherwise false.

```
Begin
    for each row r in the grid, do
        for each column c in the grid, doif grid[r, c] =
            0, then
                return true
        done
    done
    return false
En
d
```

isValidPlace(row, col, num)

Input: Row, a column of the grid, and number to check.

Output: True, when placing the number at position grid[row, col] is valid.

```
Begin
    if isPresentInRow(row, num) and isPresentInCol(col, num) and isPresntInBox(row – row mod 3,
    col – col mod 3, num) all are false, then
        return true
End
```

solveSudoku(Sudoku Grid)

Input: The unsolved grid of Sudoku.

Output: Grid after solve.

```
Begin
    if no place in the grid is empty, thenreturn true
    for number 1 to 9, do
        if isValidPlace(row, col, number), thengrid[row, col] :
            = number
            if solveSudoku = true, thenreturn
                true
            grid[row, col] := 0
    done
    return false
En
d
```

Follow the steps below to solve the problem:

Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided byusing loops.

Create a recursive function that takes a grid.

Check for any unassigned location.

If present then assigns a number from 1 to 9.

Check if assigning the number to current index makes the grid unsafe or not.

If safe then recursively call the function for all safe cases from 0 to 9.

If any recursive call returns true, end the loop and return true. If no recursive callreturns true then return false.

If there is no unassigned location then return true.

PROGRAM:

```
# A Backtracking programdef

print_grid(arr):

for i in range(9):

for j in range(9):

print (arr[i][j], end = " "),print ()

def find_empty_location(arr, l):for row in

range(9):

for col in range(9):

if(arr[row][col]== 0):

l[0]= row l[1]=

col return True

return False

def used_in_row(arr, row, num):for i in

range(9): if(arr[row][i] == num):

return True

return False

def used_in_col(arr, col, num):for i in

range(9): if(arr[i][col] == num):

return True

return False

def used_in_box(arr, row, col, num):for i in

range(3):

for j in range(3):

if(arr[i + row][j + col] == num):return True
```

```
return False
def check_location_is_safe(arr, row, col, num):return (not
used_in_row(arr, row, num) and
(not used_in_col(arr, col, num) and (not
used_in_box(arr, row - row % 3,col - col % 3,
num))))
def solve_sudoku(arr):
l =[0, 0]
if(not find_empty_location(arr, l)):return True
row = l[0]col
= l[1]
for num in range(1, 10):
if(check_location_is_safe(arr,row, col,
num)):
arr[row][col]= num
if(solve_sudoku(arr)):
return True
arr[row][col] = 0return
False
if __name__=="__main__":
grid =[[0 for x in range(9)]for y in range(9)]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],
[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]]
```

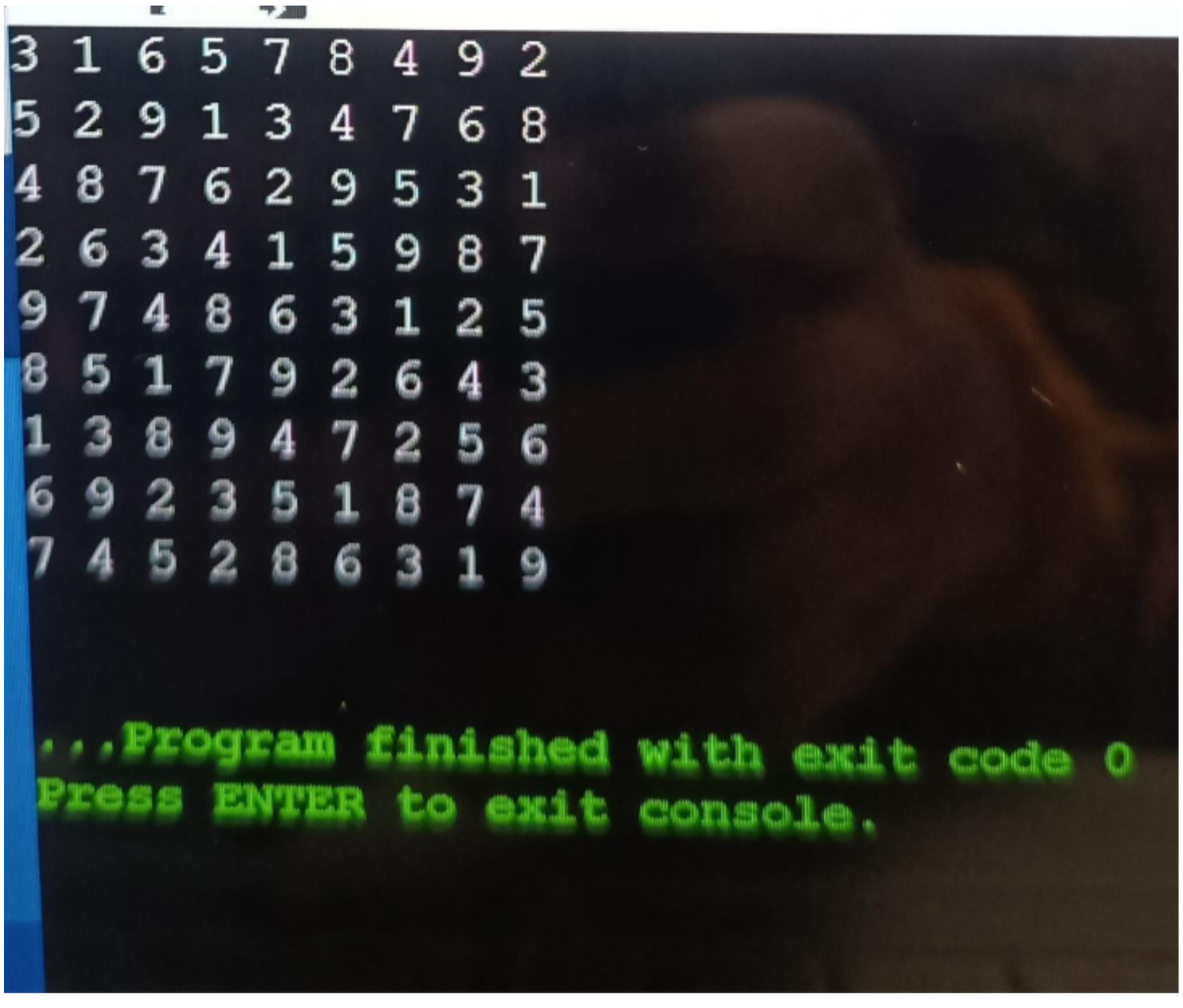
if(solve_sudoku(grid)):

print_grid(grid)else:

print ("No solution exists")

The above code has been contributed by Harshit Sidhwa.

OUTPUT:



o