BT-sudoku

May 18, 2020

1 Sudoku solver using backtrack

1.1 How backtracking works:

todo: - [ ] this section...

1.2 Implementation

We only need to implement 2 functions:
1) Select next (empty) element:

[1]:
```
def pickEmpty(grid):
    for i in range(9):
        for j in range(9):
            if grid[i][j] == 0:
                return i,j
    return False
```

2) Can ***value*** be placed in ***pos***?

[2]:
```
def isPossible(pos, value, grid):
    # check line
    for a in range(9):
        if grid[pos[0]][a] == value and pos[1] != a:
            return False

    # check column
    for b in range(9):
        if grid[b][pos[1]] == value and pos[0] != b:
            return False

    # check cell
    y, x = (pos[0]//3)*3, (pos[1]//3)*3
    for c in range(3):
        for d in range(3):
```

```
            if grid[ y + c ][ x + d ] == value and pos != [c, d]:
                return False

    return True
```

Main function: 1. If *pickEmpty()* returns False then we have reached the last element and the board has been solved and the functions can immediately return *True*.

2. Else, we take the next *pos*ition and run through values 1-9 untill one fits.
    1. If it fits we move on to the next empty element. Call the function again and we just find ourselves in **step 1.** again.
        1. If no value fits, then we must backtrack: We exit the loop, clear the current element, and return *False* to the line `if solve(board):` one level above, wich makes it try the next *value* and we just find ourselves in **step 2.A.** again.

[3]:
```
def solve(board):
    if not pickEmpty(board):
        return True

    else:
        pos = pickEmpty(board)

        for val in range(1, 10):
            if isPossible(pos, val, board):
                board[pos[0]][pos[1]] = val

                if solve(board):
                    return True

        board[pos[0]][pos[1]] = 0
        return False
```

1.3 Running the code

Right now numpy is only used to print the grid as a matrix.

[4]:
```
import numpy as np

sudoku = [[0, 4, 0, 0, 0, 0, 1, 7, 9],
          [0, 0, 2, 0, 0, 8, 0, 5, 4],
          [0, 0, 6, 0, 0, 5, 0, 0, 8],
          [0, 8, 0, 0, 7, 0, 9, 1, 0],
          [0, 5, 0, 0, 9, 0, 0, 3, 0],
          [0, 1, 9, 0, 6, 0, 0, 4, 0],
          [3, 0, 0, 4, 0, 0, 7, 0, 0],
          [5, 7, 0, 1, 0, 0, 2, 0, 0],
```

```
          [9, 2, 8, 0, 0, 0, 0, 6, 0]]

print("Sudoku:")
print(np.matrix(sudoku), "\n")

solve(sudoku)

print("Solution:")
print(np.matrix(sudoku))
```

```
Sudoku:
[[0 4 0 0 0 0 1 7 9]
 [0 0 2 0 0 8 0 5 4]
 [0 0 6 0 0 5 0 0 8]
 [0 8 0 0 7 0 9 1 0]
 [0 5 0 0 9 0 0 3 0]
 [0 1 9 0 6 0 0 4 0]
 [3 0 0 4 0 0 7 0 0]
 [5 7 0 1 0 0 2 0 0]
 [9 2 8 0 0 0 0 6 0]]

Solution:
[[8 4 5 6 3 2 1 7 9]
 [7 3 2 9 1 8 6 5 4]
 [1 9 6 7 4 5 3 2 8]
 [6 8 3 5 7 4 9 1 2]
 [4 5 7 2 9 1 8 3 6]
 [2 1 9 8 6 3 5 4 7]
 [3 6 1 4 2 9 7 8 5]
 [5 7 4 1 8 6 2 9 3]
 [9 2 8 3 5 7 4 6 1]]
```

todo: - [ ] Print grid in a nice way without using numpy; - [ ] Read grid directly fom a .txt file; - [ ] Create GUI in pygame that shows progress in real time;