Intermediate Submission Questions

Spencer Hirsch, Thomas Johnson

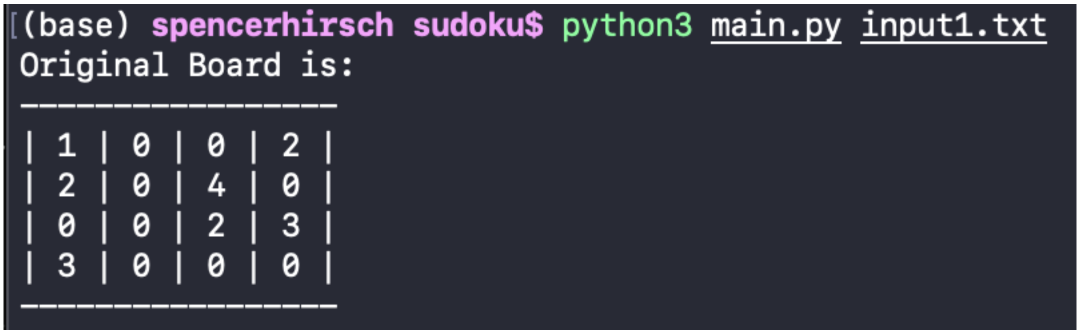
February 4, 2023

Psuedo-Code for Backtracking Algorithm:

```
# Function next empty cell in the matrix


def find_cell(board, size):
       iterate through the board to find the next instance of a 0 value in
       a cell


       If no value is found, return -1, -1


       otherwise return the coordinate of the cell that contains a 0 value



# Function checks for a valid move given the value of the current
# Cell


def valid_move(coordinate_1, coordinate_2, board, size, number):


       Iterate through the size of the board
              Check to see if there is a cell in the column has the current val
                     if so, return false


       Iterate through the size of the board
              Check to see if there is a cell in the row that has the current val
                     If so, return false
```

```
    return true # It is a valid move for the algorithm to make


def solve(board, size):
    Create a list of all possible numbers based on size of the board
    Find the coordinate to the next empty cell

    Check to see if cell is valid

    iterate through all possible numbers to fill a cell
        Check to see if the move is a vlid move

    Continue until all cells are filled or no solution is found
```

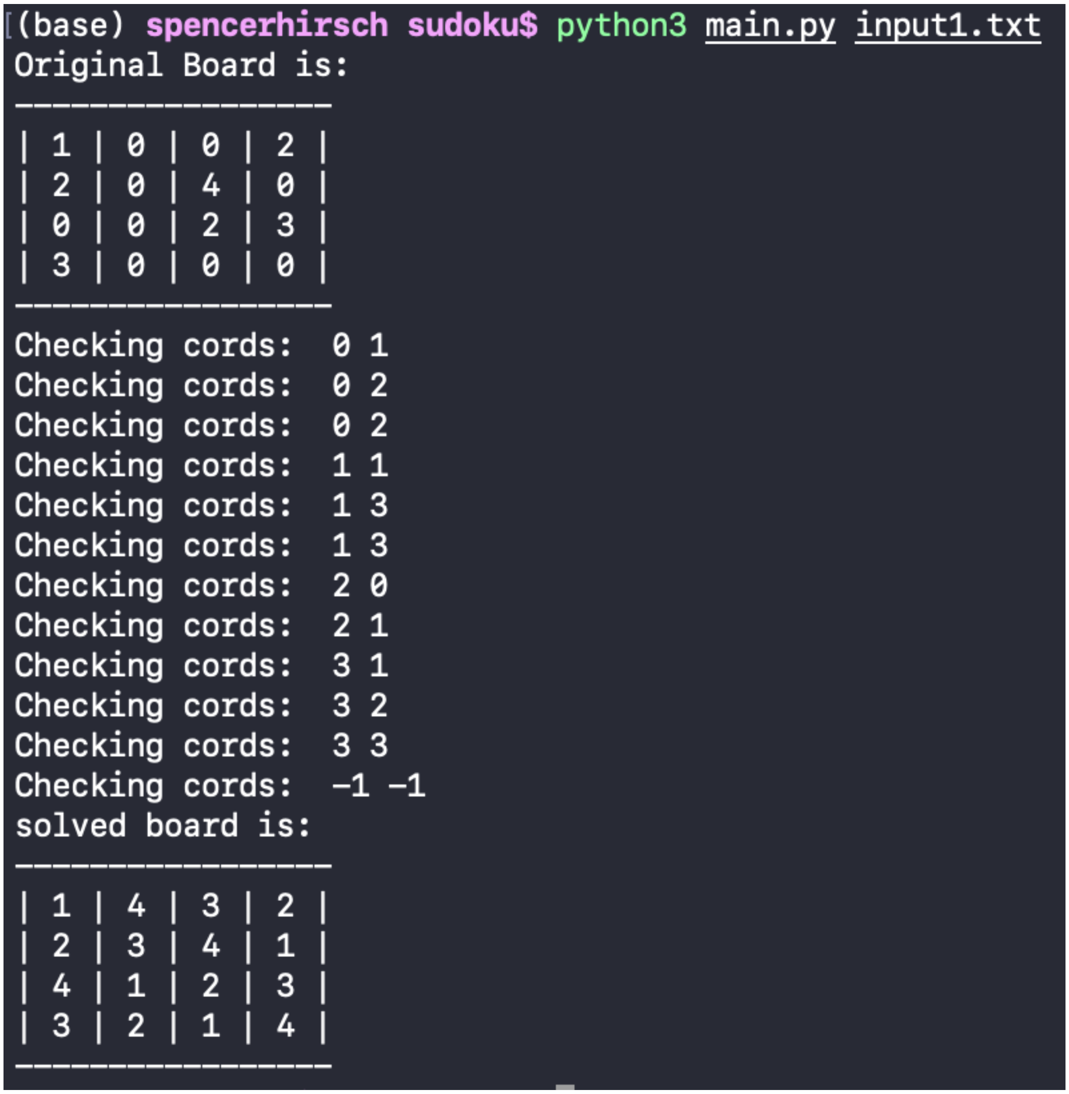
Screenshot demonstrating compilation of code:



```
[(base) spencerhirsch sudoku$ python3 main.py input1.txt
Original Board is:
-----------------
| 1 | 0 | 0 | 2 |
| 2 | 0 | 4 | 0 |
| 0 | 0 | 2 | 3 |
| 3 | 0 | 0 | 0 |
-----------------
```

Screenshot demonstrating that the code runs.

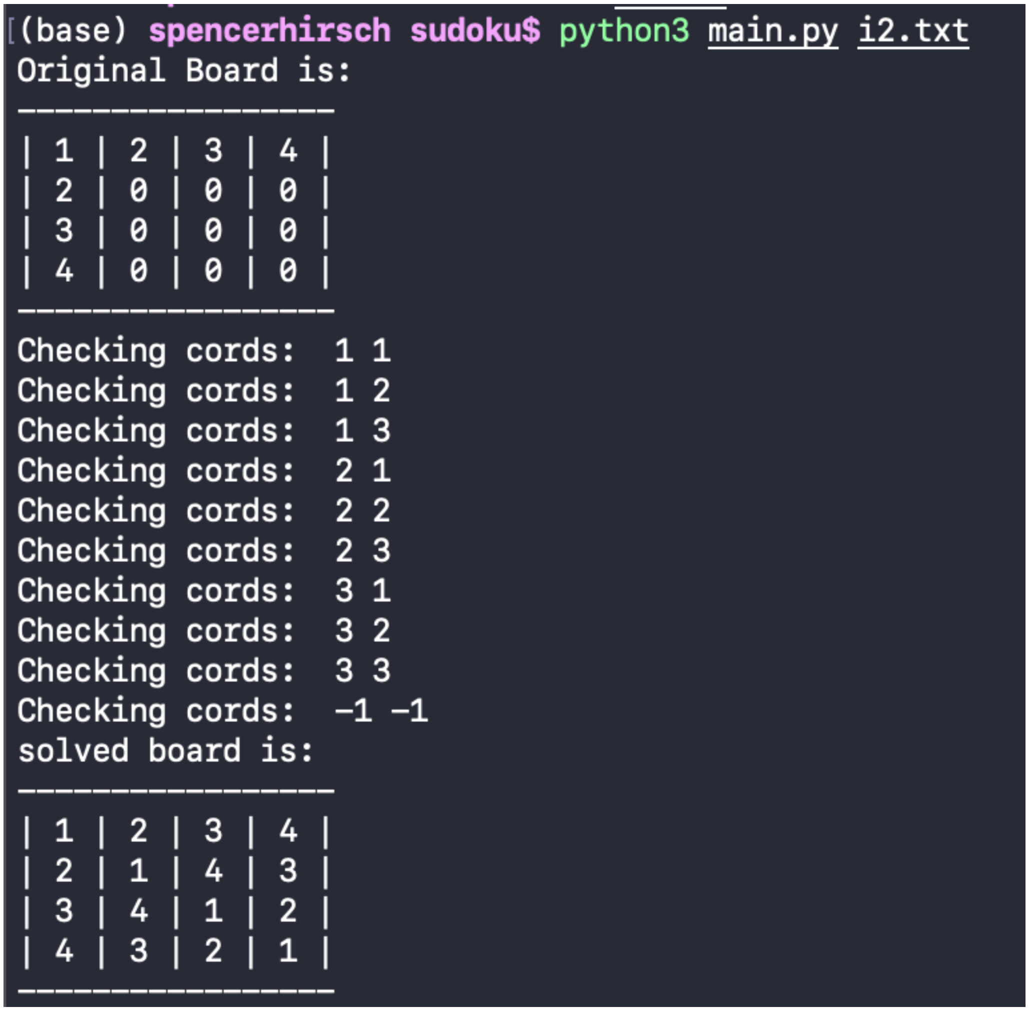
Screenshot showing output for first test case:



```
[(base) spencerhirsch sudoku$ python3 main.py input1.txt
Original Board is:
-----------------
| 1 | 0 | 0 | 2 |
| 2 | 0 | 4 | 0 |
| 0 | 0 | 2 | 3 |
| 3 | 0 | 0 | 0 |
-----------------
Checking cords:  0 1
Checking cords:  0 2
Checking cords:  0 2
Checking cords:  1 1
Checking cords:  1 3
Checking cords:  1 3
Checking cords:  2 0
Checking cords:  2 1
Checking cords:  3 1
Checking cords:  3 2
Checking cords:  3 3
Checking cords:  -1 -1
solved board is:
-----------------
| 1 | 4 | 3 | 2 |
| 2 | 3 | 4 | 1 |
| 4 | 1 | 2 | 3 |
| 3 | 2 | 1 | 4 |
-----------------
```

Screenshot demonstrating our first test case, as well as correct output.

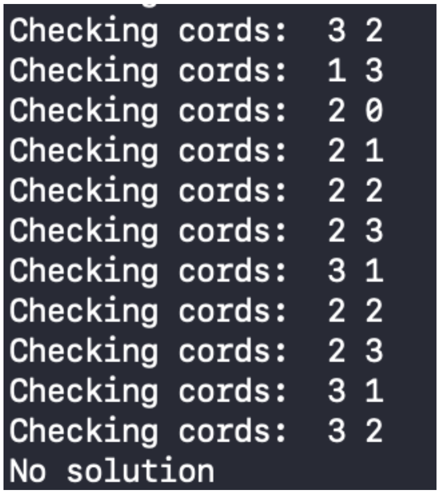
Screenshot showing output for second test case:



```
[(base) spencerhirsch sudoku$ python3 main.py i2.txt
Original Board is:
-----------------
| 1 | 2 | 3 | 4 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 |
-----------------
Checking cords:  1 1
Checking cords:  1 2
Checking cords:  1 3
Checking cords:  2 1
Checking cords:  2 2
Checking cords:  2 3
Checking cords:  3 1
Checking cords:  3 2
Checking cords:  3 3
Checking cords:  -1 -1
solved board is:
-----------------
| 1 | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 3 | 4 | 1 | 2 |
| 4 | 3 | 2 | 1 |
-----------------
```

Screenshot demonstrating our second test case, as well as the correct output.

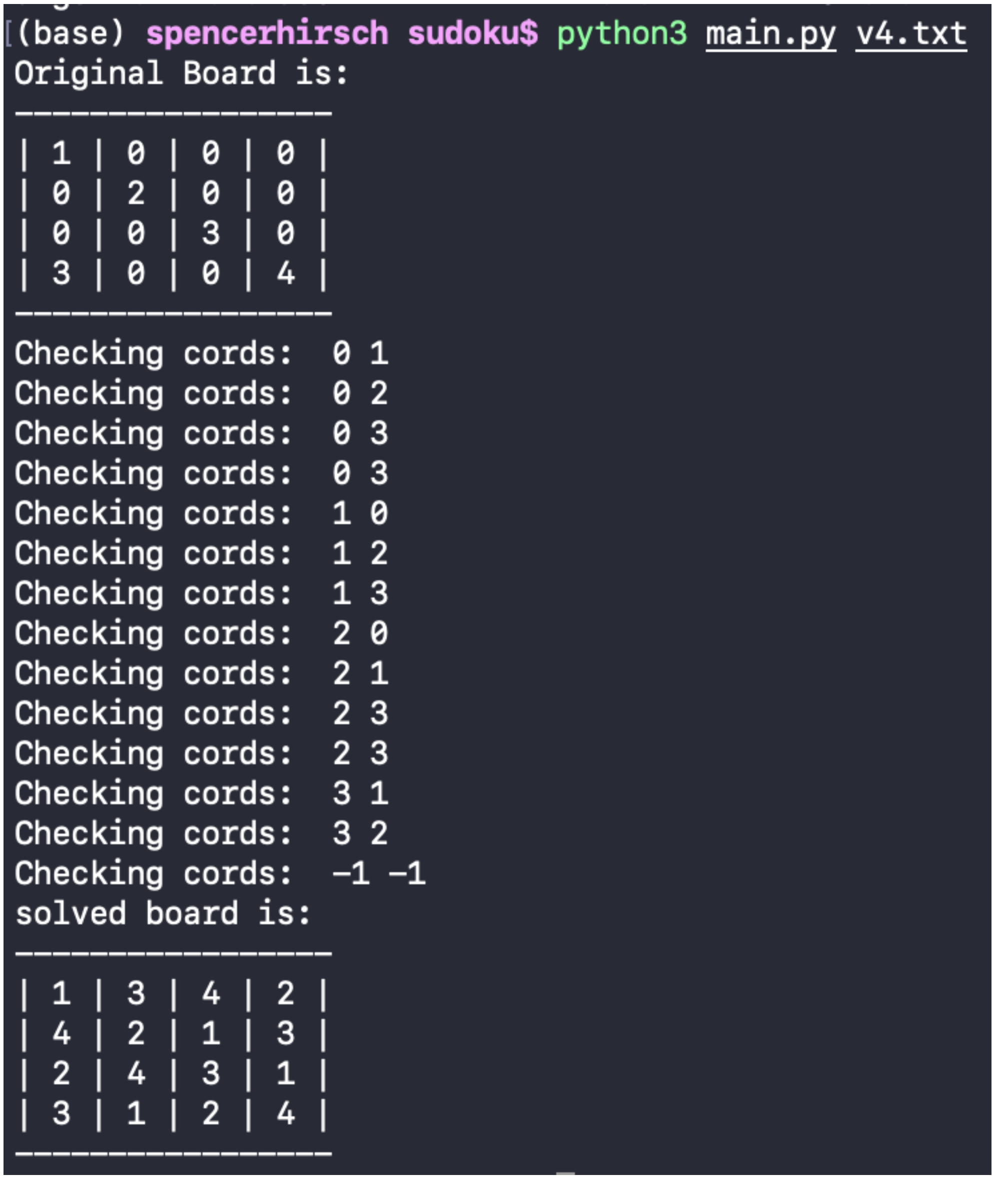
Screenshot showing output for third test case (impossible):



```
Checking cords:  3 2
Checking cords:  1 3
Checking cords:  2 0
Checking cords:  2 1
Checking cords:  2 2
Checking cords:  2 3
Checking cords:  3 1
Checking cords:  2 2
Checking cords:  2 3
Checking cords:  3 1
Checking cords:  3 2
No solution
```

Screenshot demonstrating the output for an impossible puzzle given to the program.

Screenshot showing output for fourth test case:



```
[(base) spencerhirsch sudoku$ python3 main.py v4.txt
Original Board is:
-----------------
| 1 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 0 | 0 | 3 | 0 |
| 3 | 0 | 0 | 4 |
-----------------
Checking cords:  0 1
Checking cords:  0 2
Checking cords:  0 3
Checking cords:  0 3
Checking cords:  1 0
Checking cords:  1 2
Checking cords:  1 3
Checking cords:  2 0
Checking cords:  2 1
Checking cords:  2 3
Checking cords:  2 3
Checking cords:  3 1
Checking cords:  3 2
Checking cords:  -1 -1
solved board is:
-----------------
| 1 | 3 | 4 | 2 |
| 4 | 2 | 1 | 3 |
| 2 | 4 | 3 | 1 |
| 3 | 1 | 2 | 4 |
-----------------
```

Screenshot demonstrating the output for a fourth test case, for good measure.

Summary of the intermediate submission:

For this assignment, my partner and I used the backtracking algorithm in order to solve for the problem. Our program reads in a text file that conatins the number of rows and columns as well as a list of the values that the matrix will be made up of. The values will be read in by row,

$$R_{00}, R_{01}, R_{02}, R_{03}, R_{10}, R_{11}, R_{12}, R_{13}, R_{20}, R_{21}, R_{22}, R_{23}, R_{30}, R_{31}, R_{32}, R_{33}$$

The values are then placed in their repsective places in the n × n matrix. The initial empty spaces hold a value of 0, the algorithm will search for the 0's in the matrix and replace them with their correct value. We chose to use a backtracking algorithm in order to solve this problem. The psuedo-code for our solution is posted above. The solution that we came to is our original code.