Devin Hardy

CS 472

Assignment 6

1. N coins placed in a row. Goat is a form of n/2 pairs.

If n is 8 or greater there is a solution.

Pass in a reference to a vector of size n where n must be even. (Vector should have each position filled with a 1)

The very first check is to make sure that the size of the vector is even. If it is it does the following steps.

Now check the number of pairs n will make.

If n/2 is odd

From the end of the vector that holds a coin. Move 3 coins to the left and have that coin jump 2 coins to the right.

After this every 4th coin moves 2 coins to the right. Skipping empty space and checking if a jump would attempt to move past the end.

The next step is that the coin at position 6 moves 2 coins to the left. Skipping over spots were a coin once sat.

The next series of steps is starting from the beginning and checking each single coin and have it attempt to jump over a pair to the right until the end is reached.

At this point each coin should be in a pair with minimum number of moves.

1. Solving Sudoku [1]

C++ code

#include <iostream>

#include <fstream>

using namespace std;

// N is the size of the 2D matrix N\*N

#define N 9

/\* This function was taken from Tutorial Point Which is citation [2] \*/

/\* Taken because it looked nice \*/

void print(int arr[N][N], ostream& fout)

{ //Print out the sudoku both to screen and file

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (j == 3 || j == 6)

{

cout << " | ";

fout << " | ";

}

cout << arr[i][j] << " ";

fout << arr[i][j] << " ";

}

if (i == 2 || i == 5) {

cout << endl;

fout << endl;

for (int i = 0; i < N; i++)

{

cout << "---";

fout << "---";

}

}

cout << endl;

fout << endl;

}

cout << endl << endl;

fout << endl << endl;

return;

}

/\* Taken From Source [1] GeeksforGeeks \*/

// 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)

{

// Checks row for duplicates

// return false for not safe

for (int x = 0; x <= 8; x++)

if (grid[row][x] == num)

return false;

// Checks columns for dups

// return false for not safe

for (int x = 0; x <= 8; x++)

if (grid[x][col] == num)

return false;

// Checks the 3x3 for dups

// return false for not safe

int startRow = row - row % 3,

startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + startRow][j +

startCol] == num)

return false;

return true; // returns true for no dups

/\* Taken From Source [1] GeeksforGeeks \*/

// This is code is contributed by Pradeep Mondal P

}

/\* Taken From Source [1] GeeksforGeeks \*/

/\* 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, int col)

{

// Check if we have reached the 8th

// row and 9th column (0

// indexed matrix) , we are

// returning true to avoid

// further backtracking

if (row == N - 1 && col == N)

return true;

// Check if column value becomes 9 ,

// we move to next row and

// column start from 0

if (col == N) {

row++;

col = 0;

}

// Check if the current position of

// the grid already contains

// value >0, we iterate for next column

if (grid[row][col] > 0)

return solveSudoku(grid, row, col + 1);

for (int num = 1; num <= N; num++)

{

// Check if it is safe to place

// the num (1-9) in the

// given row ,col ->we

// move to next column

if (isSafe(grid, row, col, num))

{

/\* Assigning the num in

the current (row,col)

position of the grid

and assuming our assigned

num in the position

is correct \*/

grid[row][col] = num;

// Checking for next possibility with next

// column

if (solveSudoku(grid, row, col + 1))

return true;

}

// Removing the assigned num ,

// since our assumption

// was wrong , and we go for

// next assumption with

// diff num value

grid[row][col] = 0;

}

return false;

/\* Taken From Source [1] GeeksforGeeks \*/

// This is code is contributed by Pradeep Mondal P

}

int main()

{

int grid[N][N];

int fromFile;

// Attempt at reading in matrix

ifstream myfile1;

ifstream myfile2;

ifstream myfile3;

ifstream myfile4;

ofstream myfile5;

myfile5.open("OutSudoku.txt");

myfile1.open("InSudoku1.txt");

cout << "Sudoku 1 " << endl; myfile5 << "Sudoku 1 " << endl;

for (int i = 0; i < N; i++) { //Should read into the matrix

for (int j = 0; j < N; j++) {

myfile1 >> fromFile;

grid[i][j] = fromFile;

}

}

if (solveSudoku(grid, 0, 0))

print(grid, myfile5);

else

cout << "no solution exists " << endl;

myfile1.close();

myfile2.open("InSudoku2.txt");

cout << "Sudoku 2 " << endl; myfile5 << "Sudoku 2 " << endl;

for (int i = 0; i < N; i++) //Should read into the matrix

for (int j = 0; j < N; j++)

myfile2 >> grid[i][j];

if (solveSudoku(grid, 0, 0))

print(grid, myfile5);

else

cout << "no solution exists " << endl;

myfile2.close();

myfile3.open("InSudoku3.txt");

cout << "Sudoku 3 " << endl; myfile5 << "Sudoku 3 " << endl;

for (int i = 0; i < N; i++) //Should read into the matrix

for (int j = 0; j < N; j++)

myfile3 >> grid[i][j];

if (solveSudoku(grid, 0, 0))

print(grid, myfile5);

else

cout << "no solution exists " << endl;

myfile3.close();

myfile4.open("InSudoku4.txt");

cout << "Sudoku 4 " << endl; myfile5 << "Sudoku 4 " << endl;

for (int i = 0; i < N; i++) //Should read into the matrix

for (int j = 0; j < N; j++)

myfile4 >> grid[i][j];

if (solveSudoku(grid, 0, 0))

print(grid, myfile5);

else

cout << "no solution exists " << endl;

myfile4.close();

myfile5.close();

return 0;

}

From files

InSudoku1.txt

8 0 2 0 4 0 5 0 3

0 3 0 2 0 1 0 6 0

1 0 6 0 9 0 2 0 8

0 1 0 6 0 3 0 5 0

7 0 3 0 5 0 6 0 9

0 5 0 9 0 4 0 7 0

9 0 7 0 3 0 1 0 5

0 2 0 8 0 9 0 3 0

3 0 1 0 6 0 4 0 2

InSudoku2.txt

5 3 0 0 7 0 0 0 0

6 0 0 1 9 5 0 0 0

0 9 8 0 0 0 0 6 0

8 0 0 0 6 0 0 0 3

4 0 0 8 0 3 0 0 1

7 0 0 0 2 0 0 0 6

0 6 0 0 0 0 2 8 0

0 0 0 4 1 9 0 0 5

0 0 0 0 8 0 0 7 9

InSudoku3.txt

9 0 0 0 0 0 4 0 0

4 0 0 0 0 1 0 5 0

3 0 0 7 0 0 0 0 1

8 0 1 0 5 0 0 3 0

0 0 6 0 0 4 0 0 9

7 0 0 0 0 6 0 0 2

0 0 5 2 0 0 0 0 0

0 0 3 0 1 0 0 0 6

0 0 4 0 0 8 7 0 0

InSudoku4.txt

2 0 0 0 0 0 6 9 0

0 5 0 0 0 3 0 0 0

1 7 0 0 0 9 4 0 5

0 0 3 0 2 5 0 1 8

0 0 0 0 4 0 0 0 0

7 2 0 3 8 0 5 0 0

5 0 2 6 0 0 0 4 1

0 0 0 5 0 0 0 7 0

0 6 7 0 0 0 0 0 3

OutSudoku.txt

Sudoku 1

8 9 2 | 7 4 6 | 5 1 3

4 3 5 | 2 8 1 | 9 6 7

1 7 6 | 3 9 5 | 2 4 8

---------------------------

2 1 9 | 6 7 3 | 8 5 4

7 4 3 | 1 5 8 | 6 2 9

6 5 8 | 9 2 4 | 3 7 1

---------------------------

9 6 7 | 4 3 2 | 1 8 5

5 2 4 | 8 1 9 | 7 3 6

3 8 1 | 5 6 7 | 4 9 2

Sudoku 2

5 3 4 | 6 7 8 | 9 1 2

6 7 2 | 1 9 5 | 3 4 8

1 9 8 | 3 4 2 | 5 6 7

---------------------------

8 5 9 | 7 6 1 | 4 2 3

4 2 6 | 8 5 3 | 7 9 1

7 1 3 | 9 2 4 | 8 5 6

---------------------------

9 6 1 | 5 3 7 | 2 8 4

2 8 7 | 4 1 9 | 6 3 5

3 4 5 | 2 8 6 | 1 7 9

Sudoku 3

9 1 2 | 5 6 3 | 4 7 8

4 6 7 | 8 2 1 | 9 5 3

3 5 8 | 7 4 9 | 2 6 1

---------------------------

8 4 1 | 9 5 2 | 6 3 7

5 2 6 | 3 7 4 | 1 8 9

7 3 9 | 1 8 6 | 5 4 2

---------------------------

6 8 5 | 2 9 7 | 3 1 4

2 7 3 | 4 1 5 | 8 9 6

1 9 4 | 6 3 8 | 7 2 5

Sudoku 4

2 3 4 | 1 5 8 | 6 9 7

9 5 6 | 4 7 3 | 1 8 2

1 7 8 | 2 6 9 | 4 3 5

---------------------------

6 4 3 | 9 2 5 | 7 1 8

8 1 5 | 7 4 6 | 3 2 9

7 2 9 | 3 8 1 | 5 6 4

---------------------------

5 9 2 | 6 3 7 | 8 4 1

3 8 1 | 5 9 4 | 2 7 6

4 6 7 | 8 1 2 | 9 5 3

Work Cited

[1] GeeksforGeeks. (2022, March 23). *Sudoku | Backtracking-7*. Retrieved April 18, 2022, from https://www.geeksforgeeks.org/sudoku-backtracking-7/

[2] Chakraborty, A. (2020, May 26). *Sudoku Solver in C++*. Tutorials Point. Retrieved April 18, 2022, from https://www.tutorialspoint.com/sudoku-solver-in-cplusplus