Architectural information

Variables

Information on constraints is kept in one short int array: vpts (short for viewpoints).

Values from input are stored in vpts in the same order as they are specified in.

Information on solution progress is kept in two 2D short int arrays:

1. solu_grid which keeps records of solved cells

2. possi_grid which keeps records of possible values in each cell.

solu_grid

The values in solu_grid are kept as plain numerical values and can be interpreted as is.

possi_grid

The values in possi_grid are to be interpreted in their bit representations, with each bit being the flag for possiblity of each value, and the least significant bit being the flag for value of 1.

For example, if only 1, 2 and 4 are possible solutions for a certain cell, the equivalent representation for that possibility would be 00... 1011 (or 0b1011), which would appear as integer 5 when printed with printf("%d"). As a side note, this is how chmod figure out permissions to grant with numerical values (x = 0b1 = 1, w = 0b10 = 2, r = 0b100 = 4).

* For a 4*4 grid, a char array should suffice, but a short int was chosen because the program was originally intended to be scaled up.

Constraints

The constraints are written in rules which are broadly categorized in three groups: "edge

rules" (edge_rules), "viewpoint rules" (vpts_rules), and "Sudoku rule" (sudoku_rule). Some of those are admittedly not the best choices of names. Sudoku rule is self-explanatory; hence it won't be discussed in detail in this document. Both viewpoint rules and sudoku rule are called iterative rules.

Also, there are edge rules which are mistakenly categorized as viewpoint rules, but this document will list them as what they are in the source codes rather than what they actually are.

Edge rules

Edge rules are rules that depend only on values of viewpoints and hence are only applied once.

Rule 1. If a viewpoint has value of 1, then the adjacent block must be the tallest block.

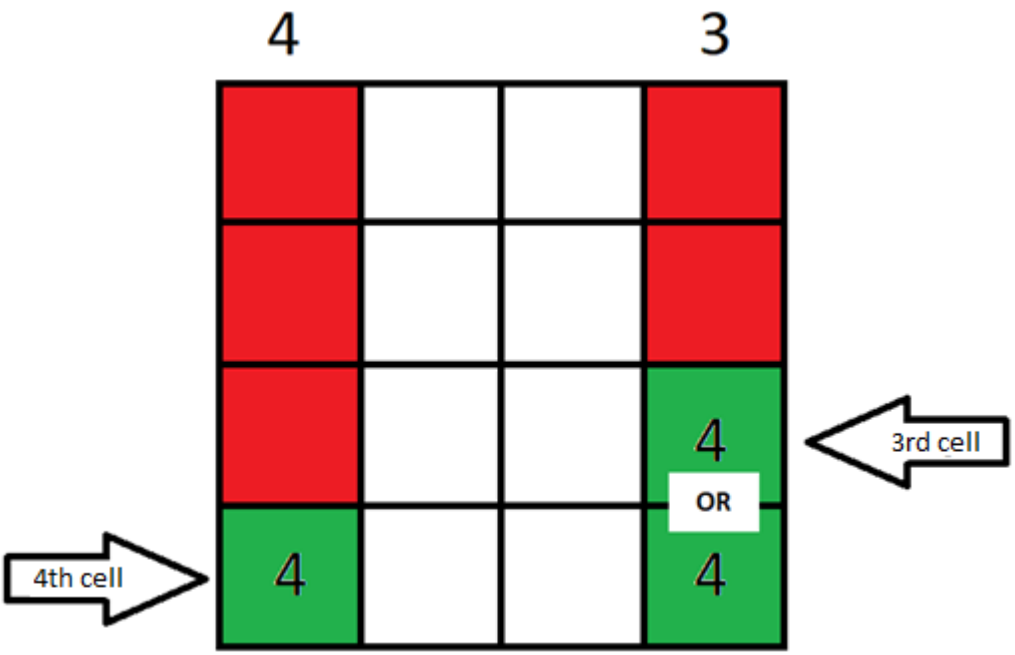
Rule 2. If a viewpoint has a value of 4, then the adjacent block must be a 1 and the height of following blocks must be in ascending order.

Viewpoint rules

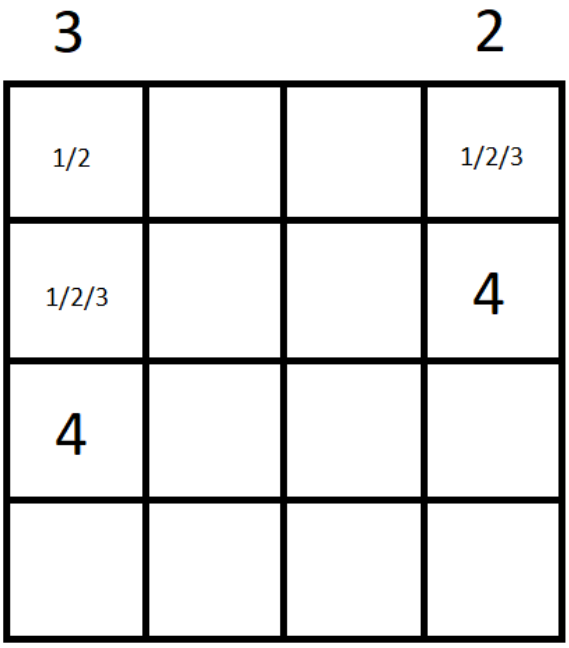
Viewpoint rules depend on both the values of viewpoints and the solutions in the other cells, hence these rules are applied repeatedly until a complete solution is found.

Rule 1. If a viewpoint has value of n, then the tallest box must be no closer than on the

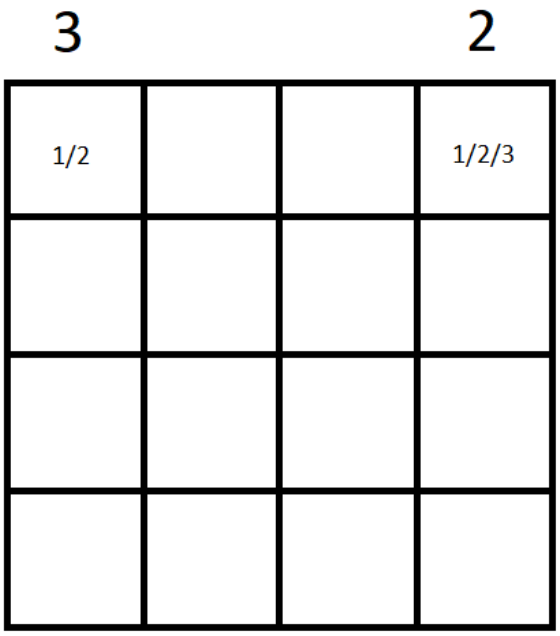
n-th cell from the viewpoint.



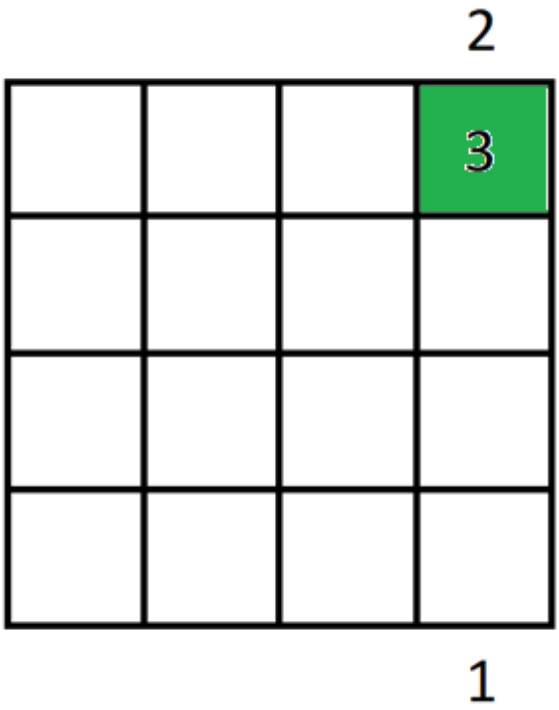
Rule 2. If a viewpoint has a value of n and the tallest block is at the n-th cell, then the height of boxes must get shorter approaching the viewpoint.



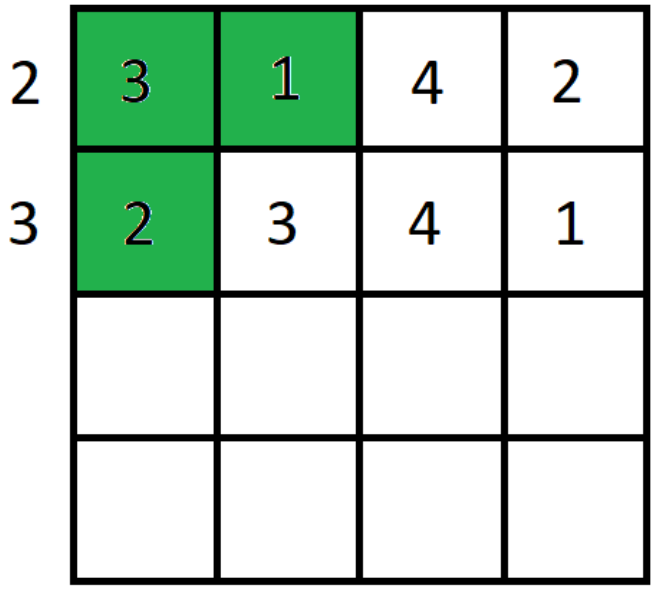
Rule 3. If a viewpoint has a value of n, then the cell adjacent to it can only be from 1 to [MAX – (n – 1)] or [5 – n] in case of a 4*4 grid.



Rule 4. If a viewpoint has a value of 1 and the viewpoint opposite to it has a value of 2, then the cell adjacent to that viewpoint must be [MAX – 1] or a 3.



Rule 5. If a viewpoint has value n, then (MAX – n) blocks must be hidden from view. If there are as many unfilled cells as there are remaining hidden boxes + 1, then the adjacent cell must take the highest possible value.



Process overview

1. Input validation

2. Input parsing

3. Apply edge rules

4. Apply viewpoint rules and sudoku rules until all cells are solved

5. Output solution

Implementation details

1. Input validation

Associated file(s): io_funcs.c

Checks for the correct length of input string, then checks for valid characters based on

their indices in the string, i.e. string should contain only numerical characters from '1'

to '4' at even indices, and only space characters at odd indices. These are both done by

is_input_valid function..

2. Input parsing

The input string is split and each numerical character is converted into an integer before

being stored in vpts array. This is done by str_split_to_int function.

3. Apply edge rules

The solutions are written into solu_grid directly. All functions handling these rules are included in edge_rules.c

4. Apply viewpoint rules and sudoku rules until all cells are solved

This stage involves working with possi_grid. There are basically 2 operations that can be done on a bit: set, or clear.

To set n-th least significant bit in any variable foo, one can do:

foo = foo | (1 << n);

To clear the n-th least significant bit, one can instead do:

foo = foo & ~(1 << n);

Not that | and & are used instead of || and &&. | and & are bitwise operators which will operate on every bit of the value, which || and && will only give one of two results: TRUE (1) or FALSE (0).

<< is a bitshift operator, while ~ is a bitwise negation operator.

All the viewpoint rules are handled by functions inside vpts_rules*.c, with the help from functions within other .c files such as bit_funcs.c, info_*.c, etc..

After application of each viewpoint rules, the possi_grid is updated by calling apply_elimination_rules which in turn calls update_possi_grid.

apply_elimination_rules searches for values that can only be in one cell within any row or any column. This is done by counting the number of set bits at a particular location within a row or a column. For example, if a particular row in possi_grid has the following values:

0b1100 0b1110 0b1110 0b0001

Then the value 1 can only be in the rightmost column.

update_possi_grid checks the solu_grid and clear all solutions from the list of possibilities in other cells within the same row and column.

For each iteration, the solu_grid is checked for completeness (absence of 0 in the whole array) and possi_grid is checked for cells with all bits cleared (implying no possible solution for that particular cell). If solu_grid is complete, the solution is validated, and if it's valid, the solution is output, otherwise an error message is displayed. If possi_grid contains 0, then no valid solution is found and an error message is displayed.

5. Output solution

Straight forward, would not go into details.