Información arquitectónica

Variables

La información sobre las restricciones se mantiene en una matriz int corta: vpts (abreviatura de puntos de vista).

Los valores de entrada se almacenan en vpts en el mismo orden en que se especifican.

La información sobre el progreso de la solución se mantiene en dos matrices 2D de int corto:

1. solu_grid que mantiene registros de celdas resueltas

2. possi_grid que mantiene registros de los valores posibles en cada celda.

solu_grid

Los valores en solu_grid se mantienen como valores numéricos simples y se pueden interpretar tal cual.

possi_grid

Los valores en possi_grid deben interpretarse en sus representaciones de bits, con cada bit
siendo el indicador de posibilidad de cada valor, y siendo el bit menos significativo el indicador
por valor de 1.

Por ejemplo, si solo 1, 2 y 4 son soluciones posibles para una determinada celda, el equivalente
representación para esa posibilidad sería 00... 1011 (o 0b1011), que aparecería como
entero 5 cuando se imprime con printf("%d"). Como nota al margen, así es como chmod se da cuenta
permisos a otorgar con valores numéricos (x = 0b1 = 1, w = 0b10 = 2, r = 0b100 = 4).

* Para una cuadrícula de 4*4, una matriz de caracteres debería ser suficiente, pero se eligió un int corto porque el
El programa originalmente estaba destinado a ser ampliado.

Restricciones

Las restricciones están escritas en reglas que se clasifican ampliamente en tres grupos: "borde

reglas" (edge_rules), "reglas de punto de vista" (vpts_rules) y "regla de Sudoku" (sudoku_rule).
Es cierto que no son las mejores opciones de nombres. La regla de Sudoku se explica por sí misma; por lo tanto, no se discutirá en detalle en este documento. Tanto las reglas de punto de vista como la regla de sudoku se denominan reglas iterativas.

Además, hay reglas de borde que se clasifican erróneamente como reglas de punto de vista, pero este documento las enumerará como lo que son en los códigos fuente en lugar de lo que realmente son.

Reglas de borde

Las reglas de borde son reglas que dependen solo de los valores de los puntos de vista y, por lo tanto, solo se aplican una vez.

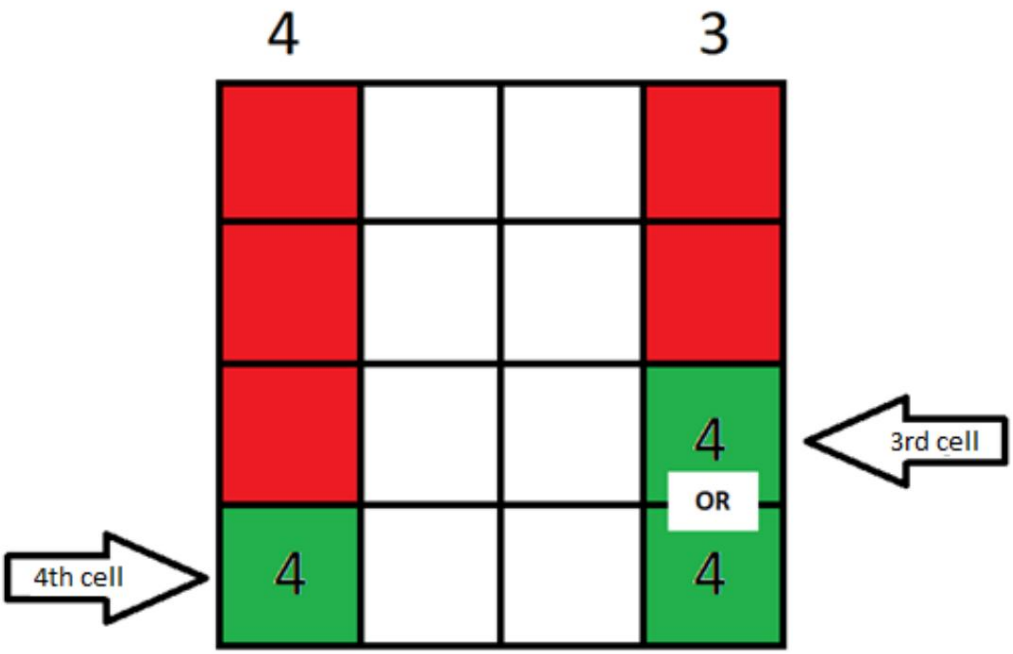
Regla 1. Si un punto de vista tiene valor de 1, entonces el bloque adyacente debe ser el bloque más alto.

Regla 2. Si un punto de vista tiene un valor de 4, entonces el bloque adyacente debe ser un 1 y la altura de
Los siguientes bloques deben estar en orden ascendente.

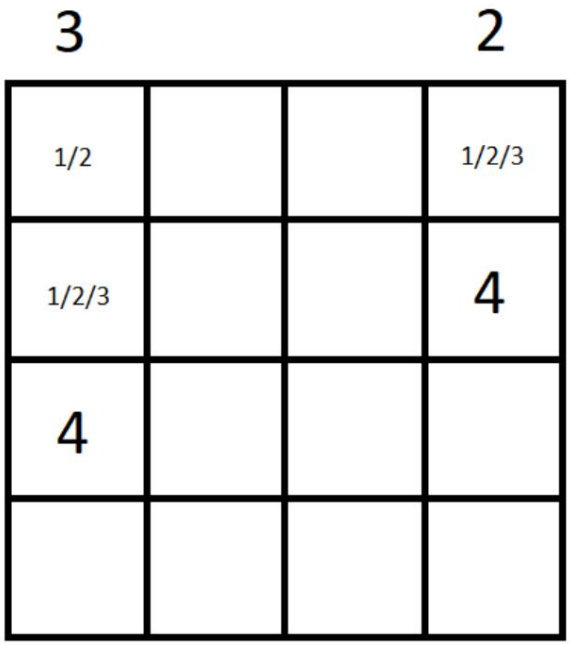
Reglas de punto de vista

Las reglas de puntos de vista dependen tanto de los valores de los puntos de vista como de las soluciones en las otras celdas, por lo tanto, estas reglas se aplican repetidamente hasta que se encuentra una solución completa.

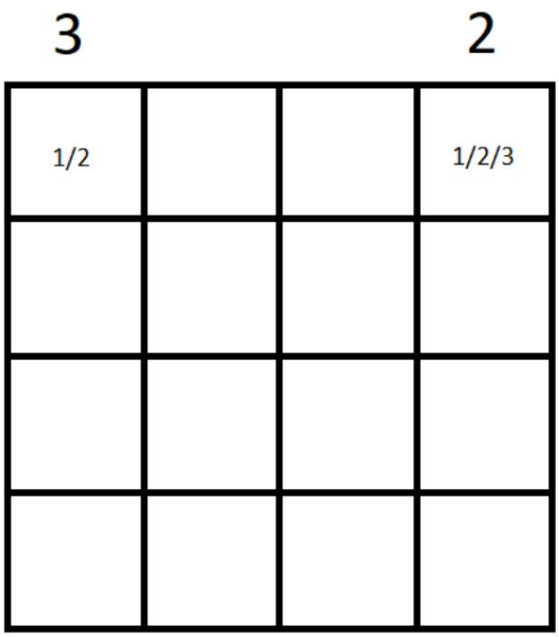
Regla 1. Si un punto de vista tiene un valor de n, entonces la caja más alta no debe estar más cerca que en el
n-ésima celda desde el punto de vista.



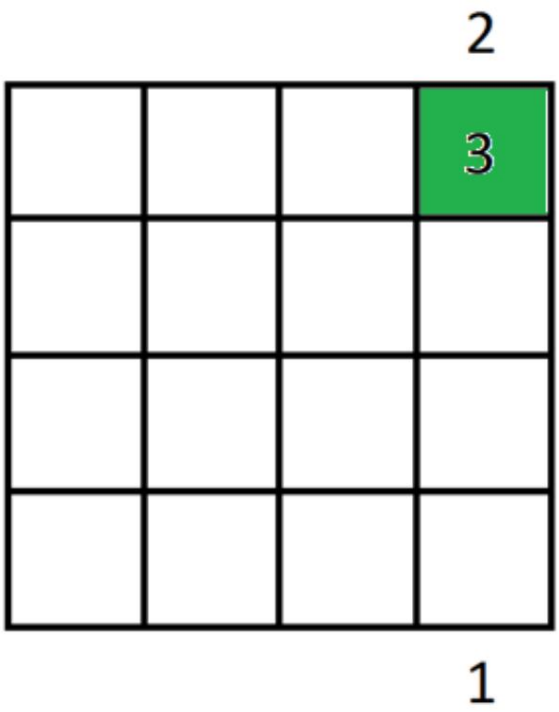
Regla 2. Si un punto de vista tiene un valor de n y el bloque más alto está en la n-ésima celda, entonces la altura de las cajas debe acortarse al acercarse al punto de vista.



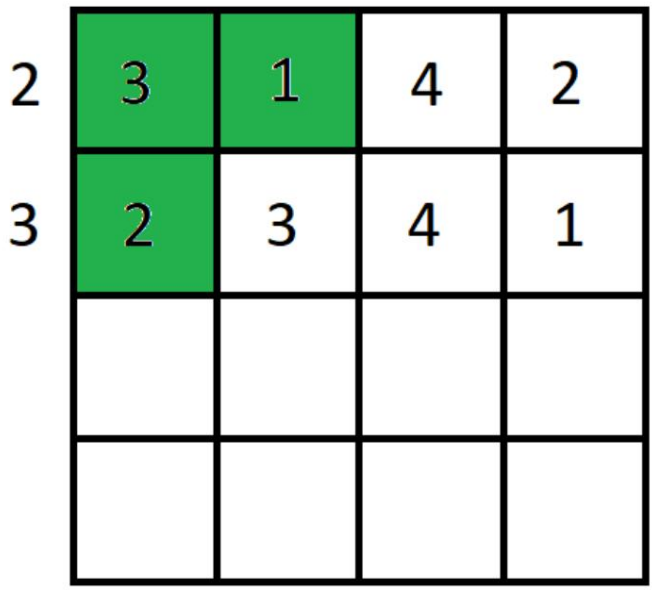
Regla 3. Si un punto de vista tiene un valor de n, entonces la celda adyacente solo puede ser de 1 a [MAX – (n – 1)] o [5 – n] en el caso de una cuadrícula de 4x4.



Regla 4. Si un punto de vista tiene un valor de 1 y el punto de vista opuesto tiene un valor de 2, entonces la celda adyacente a ese punto de vista debe ser [MAX - 1] o un 3.



Regla 5. Si un punto de vista tiene un valor n, entonces (MAX – n) bloques deben estar ocultos a la vista. si hay como tantas celdas vacías como casillas ocultas restantes + 1, entonces la celda adyacente debe tomar el valor más alto posible.



Vista general del proceso

1. Validación de entrada

2. Análisis de entrada

3. Aplicar reglas de borde

4. Aplicar reglas de punto de vista y reglas de sudoku hasta que se resuelvan todas las celdas

5. Solución de salida

Detalles de implementacion

1. Validación de entrada

Archivo(s) asociado(s): io_funcs.c

Verifica la longitud correcta de la cadena de entrada, luego verifica los caracteres válidos según

sus índices en la cadena, es decir, la cadena debe contener solo caracteres numéricos desde '1'

a '4' en índices pares, y solo caracteres de espacio en índices impares. Ambos son realizados por

función is_input_valid..

2. Análisis de entrada

La cadena de entrada se divide y cada carácter numérico se convierte en un número entero antes

siendo almacenado en la matriz vpts. Esto se hace mediante la función str_split_to_int.

3. Aplicar reglas de borde

Las soluciones se escriben directamente en solu_grid. Todas las funciones que manejan estas reglas están incluidas en edge_rules.c

4. Aplicar reglas de punto de vista y reglas de sudoku hasta que se resuelvan todas las celdas

Esta etapa consiste en trabajar con possi_grid. Básicamente, hay 2 operaciones que se pueden hacer en un bit: establecer o borrar.

Para establecer el n-ésimo bit menos significativo en cualquier variable foo, se puede hacer:

foo = foo | (1 << n);

Para borrar el n-ésimo bit menos significativo, se puede hacer:

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

eso no | y & se utilizan en lugar de || y &&. | y & son operadores bit a bit que operarán en cada bit del valor, que || y && solo dará uno de dos resultados: VERDADERO (1) o FALSO (0).

<< es un operador de desplazamiento de bits, mientras que ~ es un operador de negación bit a bit.

Todas las reglas de punto de vista son manejadas por funciones dentro de vpts_rules*.c, con la ayuda de funciones dentro de otros archivos .c como bit_funcs.c, info_*.c, etc.

Después de la aplicación de las reglas de cada punto de vista, possi_grid se actualiza llamando a apply_elimination_rules que, a su vez, llama a update_possi_grid.

apply_elimination_rules busca valores que solo pueden estar en una celda dentro de cualquier fila o columna. Esto se hace contando el número de bits establecidos en una ubicación particular dentro de una fila o columna. Por ejemplo, si una fila en particular en possi_grid tiene los siguientes valores:

0b1100 0b1110 0b1110 0b0001

Entonces el valor 1 solo puede estar en la columna más a la derecha.

update_possi_grid comprueba solu_grid y borra todas las soluciones de la lista de posibilidades en otras celdas dentro de la misma fila y columna.

Para cada iteración, se verifica que solu_grid esté completo (ausencia de 0 en toda la matriz) y possi_grid se verifica en busca de celdas con todos los bits borrados (lo que implica que no hay una solución posible para esa celda en particular). Si solu_grid está completo, la solución se valida y, si es válida, se genera la solución; de lo contrario, se muestra un mensaje de error. Si possi_grid contiene 0, no se encuentra una solución válida y se muestra un mensaje de error.

5. Solución de salida

Sencillo, no entraría en detalles.