Dancing Links and Sudoku

A Java Sudoku Solver

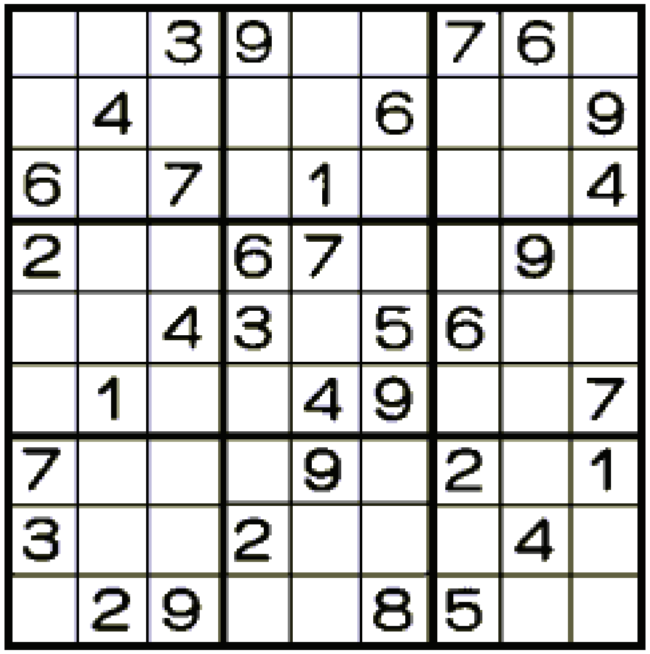
By: Jonathan Chu

Adviser: Mr. Feinberg

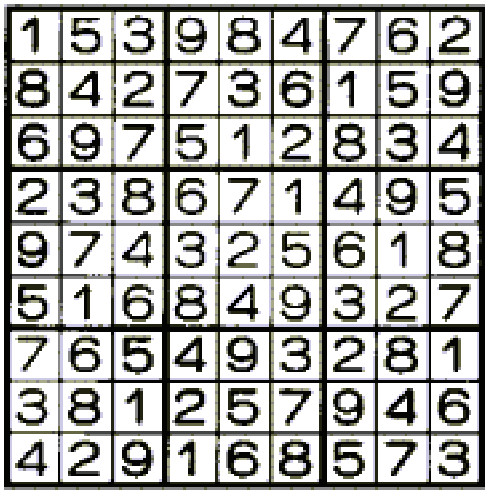
Algorithm by: Dr. Donald Knuth

Sudoku

Sudoku is a logic puzzle. On a 9x9 grid with 3x3 regions, the digits 1-9 must be placed in each cell such that every row, column, and region contains only one instance of the digit. Placing the numbers is simply an exercise of logic and patience. Here is an example of a puzzle and its solution:



		3	9			7	6	
	4				6			9
6		7		1				4
2			6	7			9	
		4	3		5	6		
	1			4	9			7
7				9		2		1
3			2				4	
	2	9			8	5		



1	5	3	9	8	4	7	6	2
8	4	2	7	3	6	1	5	9
6	9	7	5	1	2	8	3	4
2	3	8	6	7	1	4	9	5
9	7	4	3	2	5	6	1	8
5	1	6	8	4	9	3	2	7
7	6	5	4	9	3	2	8	1
3	8	1	2	5	7	9	4	6
4	2	9	1	6	8	5	7	3

Images from web Nikoli

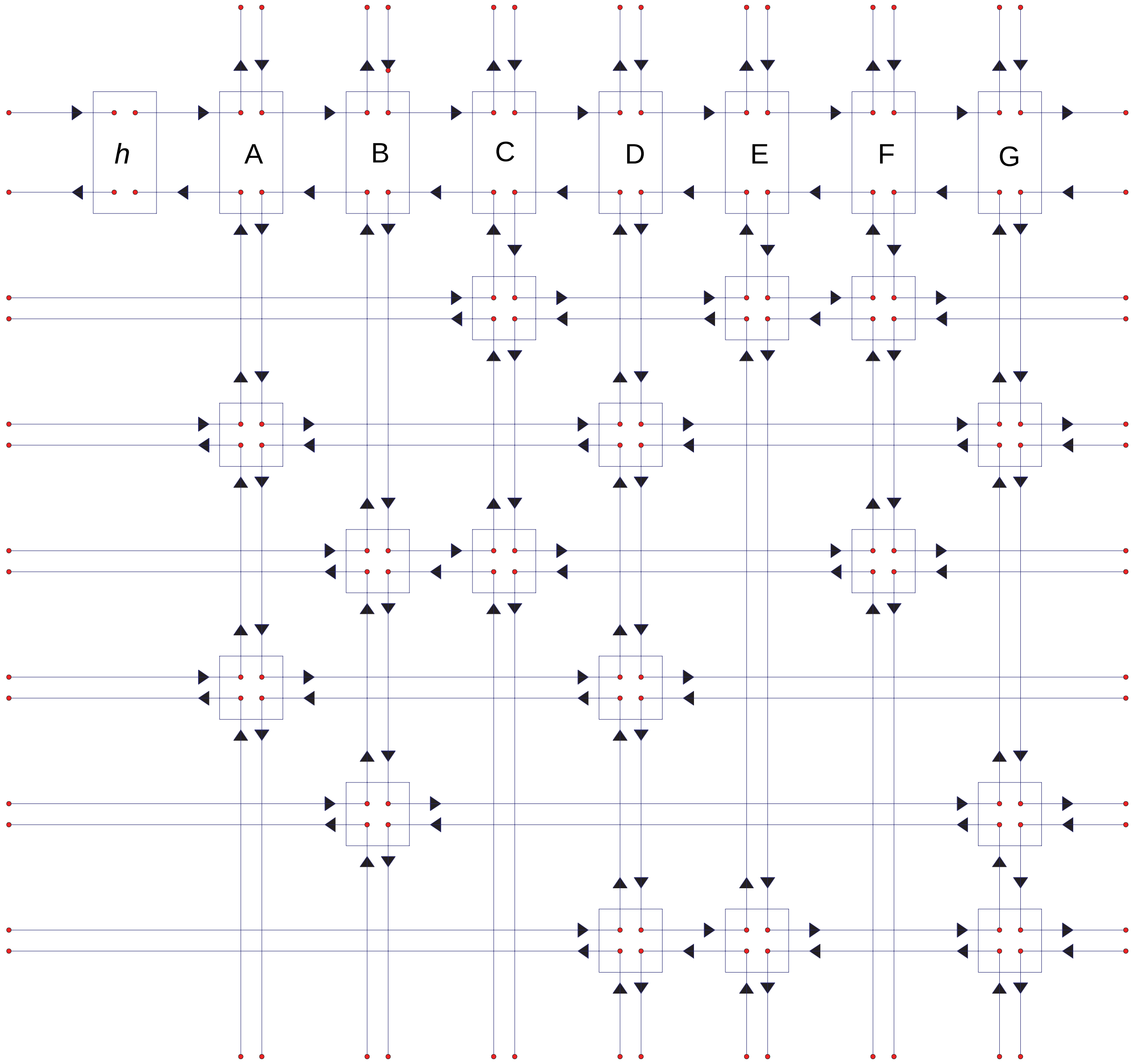
Sudoku is exactly a subset of a more general set of problems called Exact Cover, which is described on the left.
Dr. Donald Knuth's Dancing Links Algorithm solves an Exact Cover situation. The Exact Cover problem can be extended to a variety of applications that need to fill constraints. Sudoku is one such special case of the Exact Cover problem.
I created a Java program that implements Dancing Links to solve Sudoku puzzles.

Exact Cover

Exact Cover describes problems in which a mtrix of 0's and 1's are given. Is there a set of rows that contain exactly one 1 in each column?

The matrix below is an example given by Dr. Knuth in his paper. Rows 1, 4, and 5 are a solution set.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$



We can represent the matrix with toriodal doubly-linked lists as shown above.
The first row uniquely identifies every column.
Each following row represents a row in the matrix in which a 1 is a node.
This representation easily shows the relationships of every node to other nodes,

The following diagrams illustrate how the Dancing Links Algorithm solves this example.

The Dancing Links Algorithm

Given the ColumnNode h, the searching algorithm is then simplified to:

```
if( h.getRight() == h ) {
      printSolution();
      return;
}
else {
      ColumnNode column = chooseNextColumn();
      cover(column);
      for( Node row = column.getDown() ; rowNode != column ; rowNode = rowNode.getDown() ) {
            solutions.add( rowNode );
            for( Node rightNode = row.getRight() ; otherNode != row ; rightNode = rightNode.getRight() )
                 cover( rightNode );
            Search( k+1);
           solutions.remove( rowNode );
           column = rowNode.getColumn();
           for( Node leftNode = rowNode.getLeft() ; leftNode != row ; leftNode = leftNode.getLeft() )
                uncover( leftNode );
      }
     uncover( column );
}
```

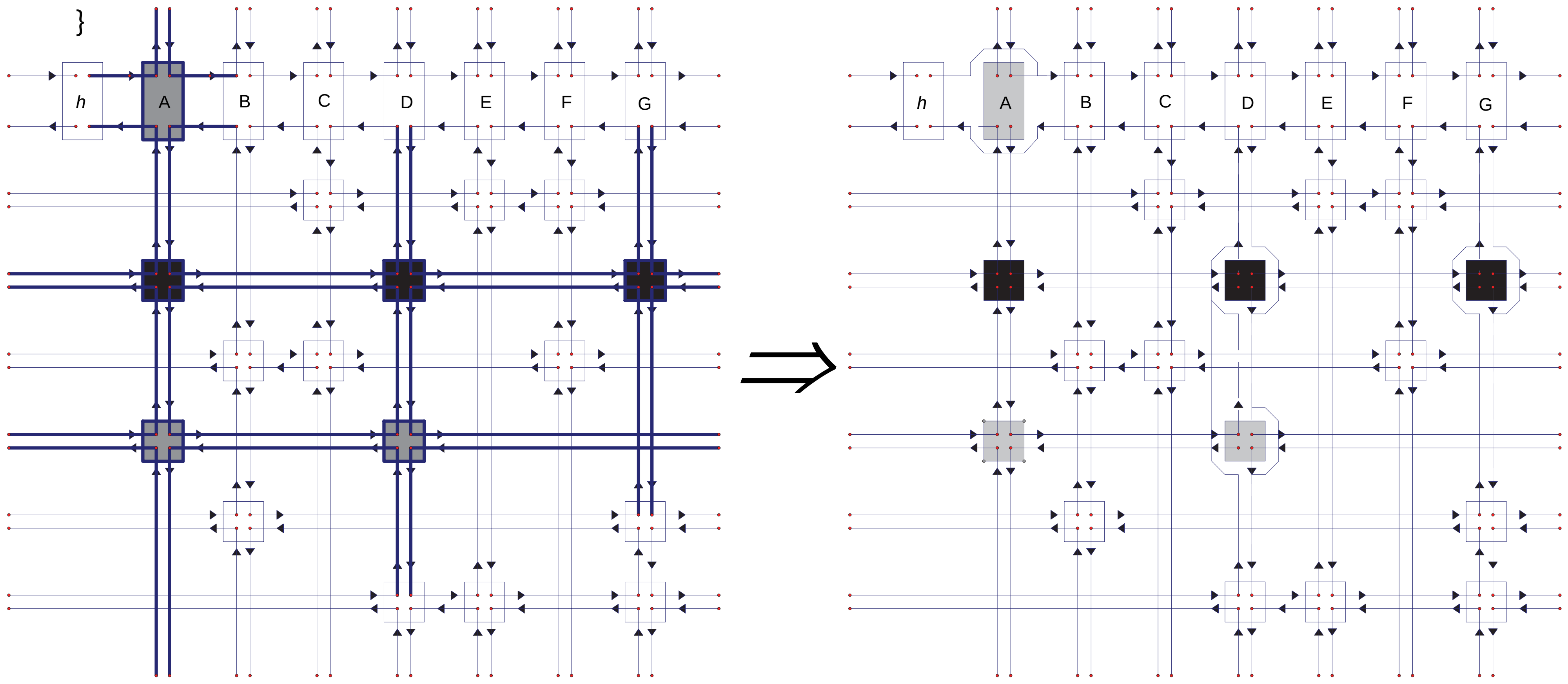
The Cover Method

cover(Node c) This function is the crux of the algorithm. It removes a column from the matrix as well as remove all rows in the column from other columns they are in. The code becomes:

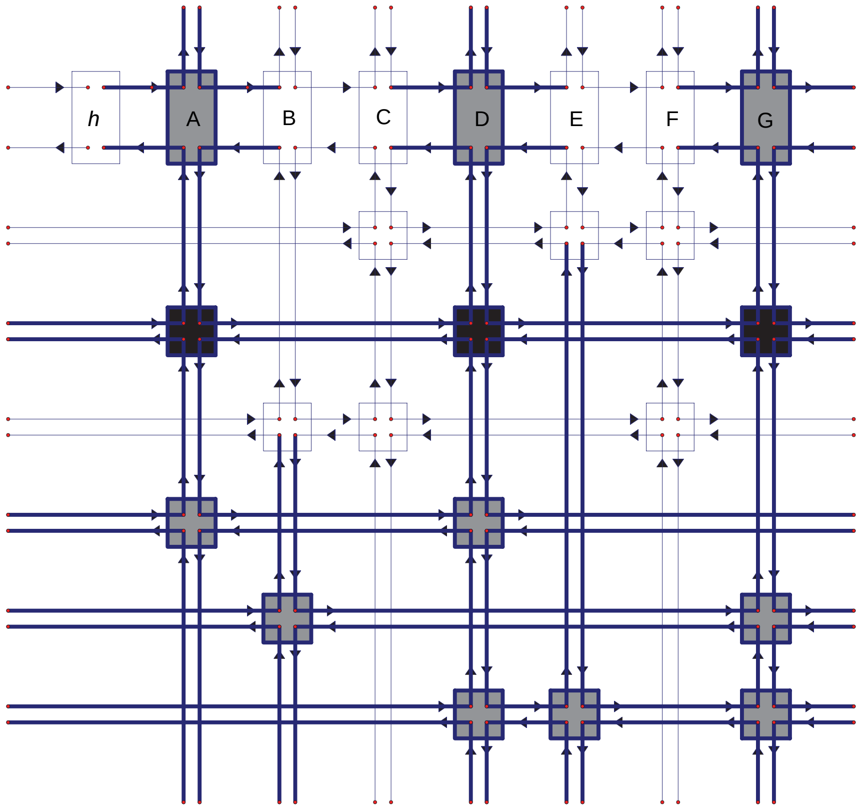
```
Node column = dataNode.getColumn();
column.getRight().setLeft( column.getLeft() );
column.getLeft().setRight( column.getRight() );
for( Node row = column.getDown() ; row != column ; row = row.getDown() )
    for( Node rightNode = row.getRight() ; rightNode != row ; rightNode = rightNode.getRight() ) {
        rightNode.getUp().setDown( rightNode.getDown() );
        rightNode.getDown().setUp( rightNode.getUp() );
    }
}
```

The Uncover Method does the exact oppositve of the cover method by undoing all the changes.

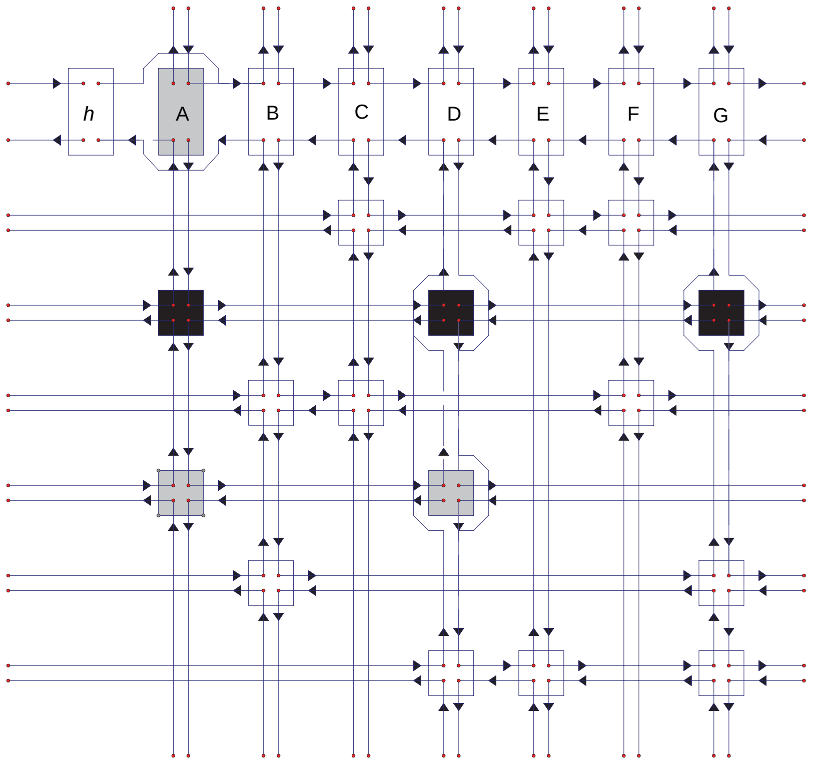
Below, we cover the first row in Column A.



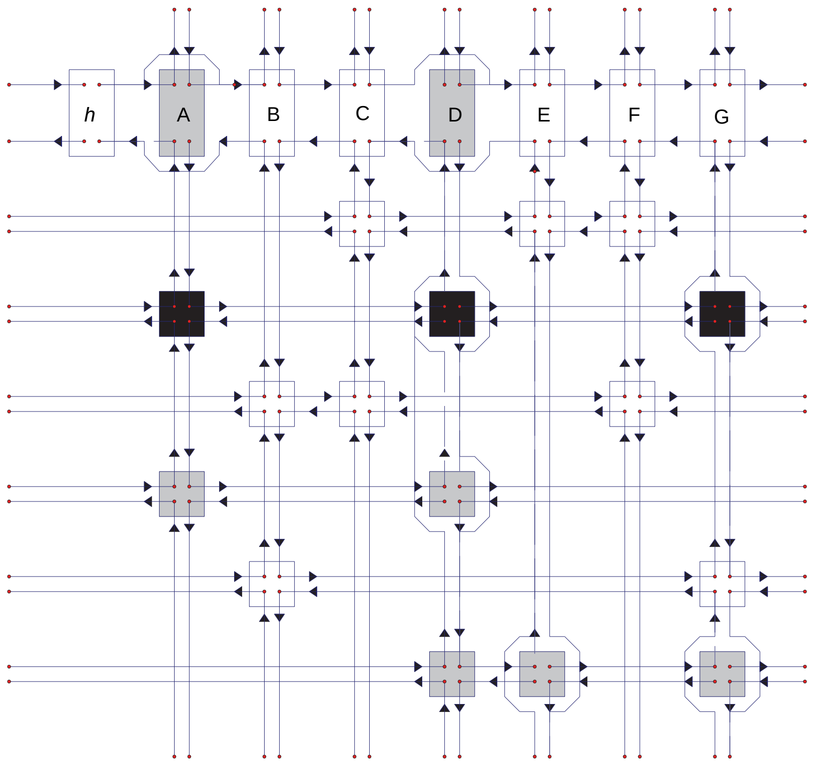
Search(1)



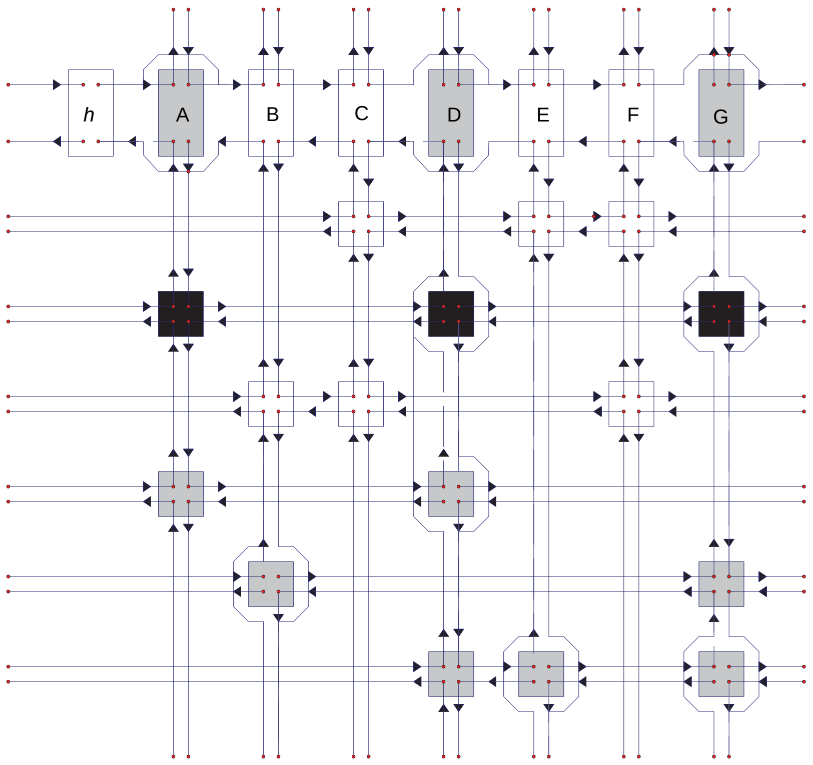
Choose Column A
Row ADG



Cover
Column A



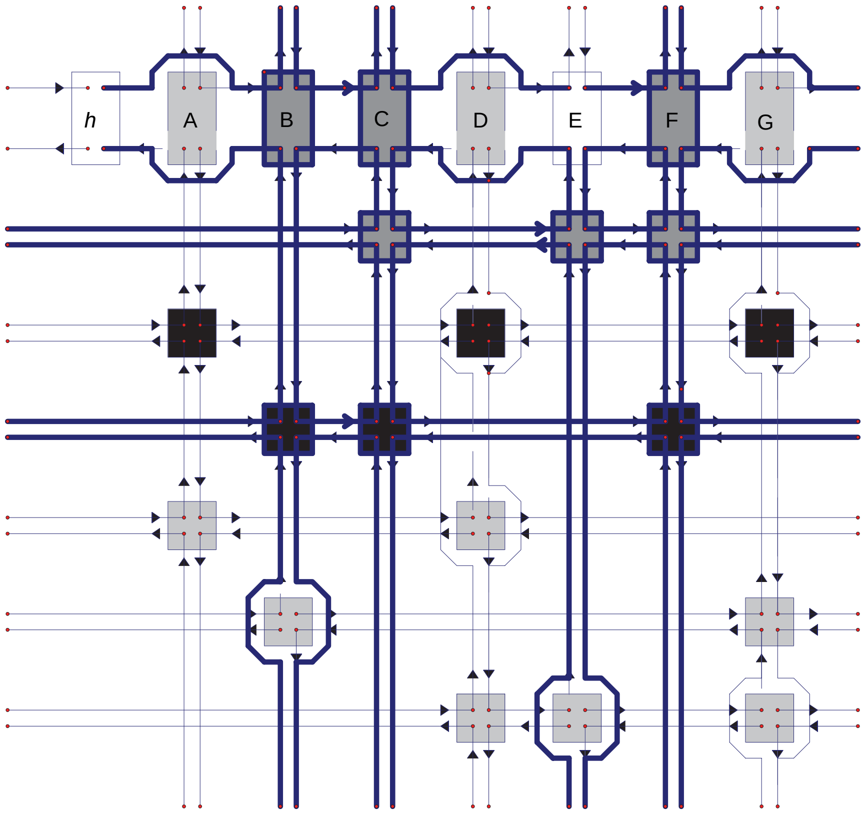
Cover
Column D



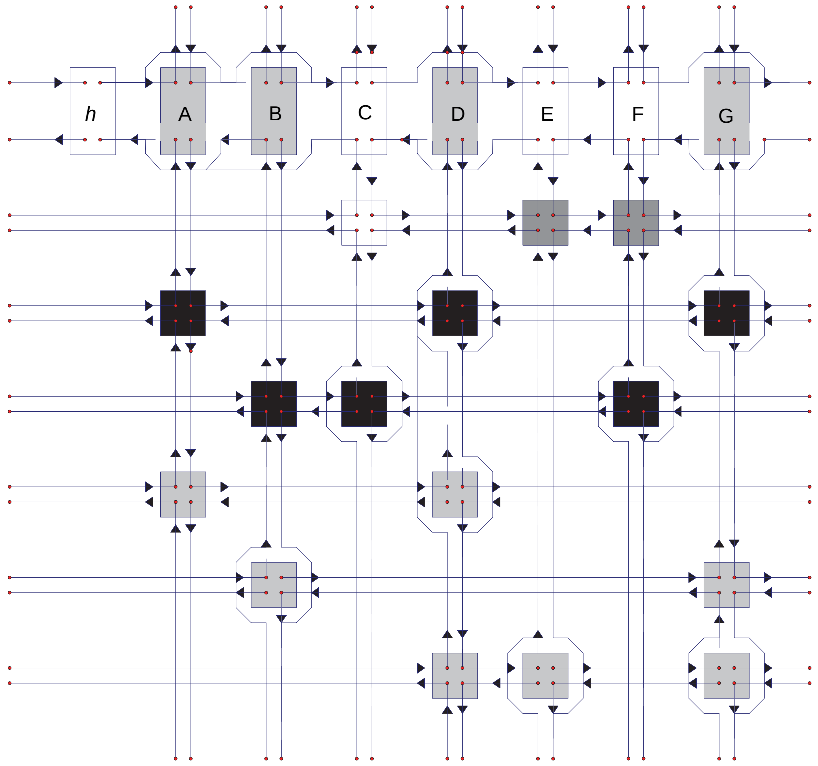
Cover
Column G

Search(2)

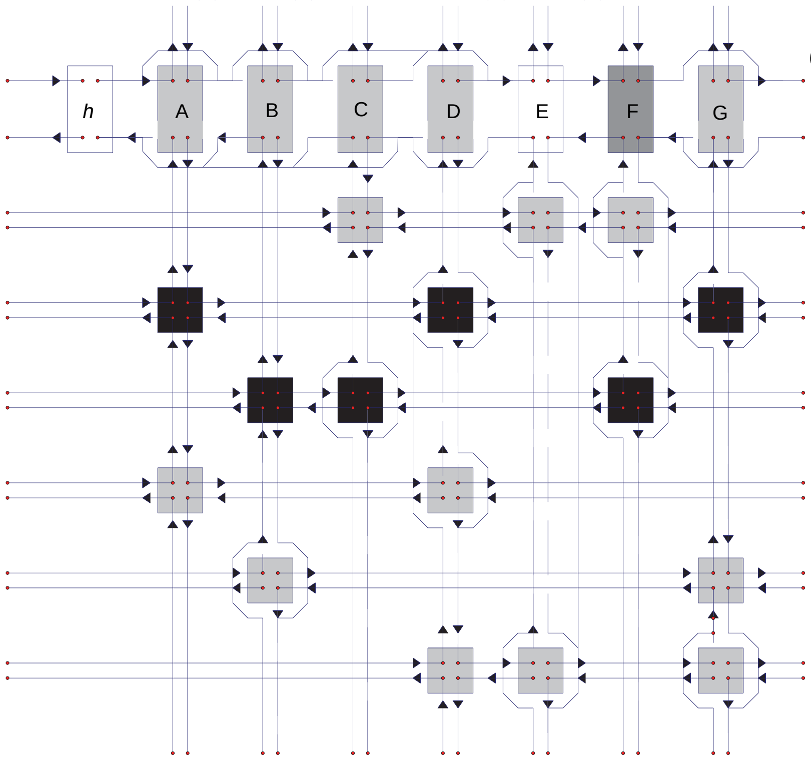
Choose Column B
Row BCF



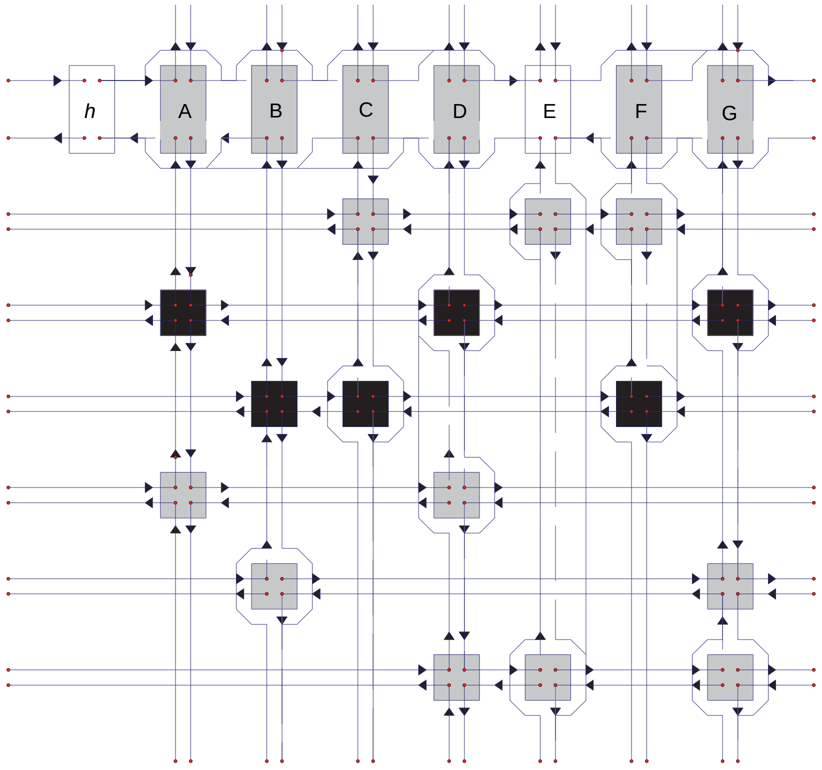
Cover B



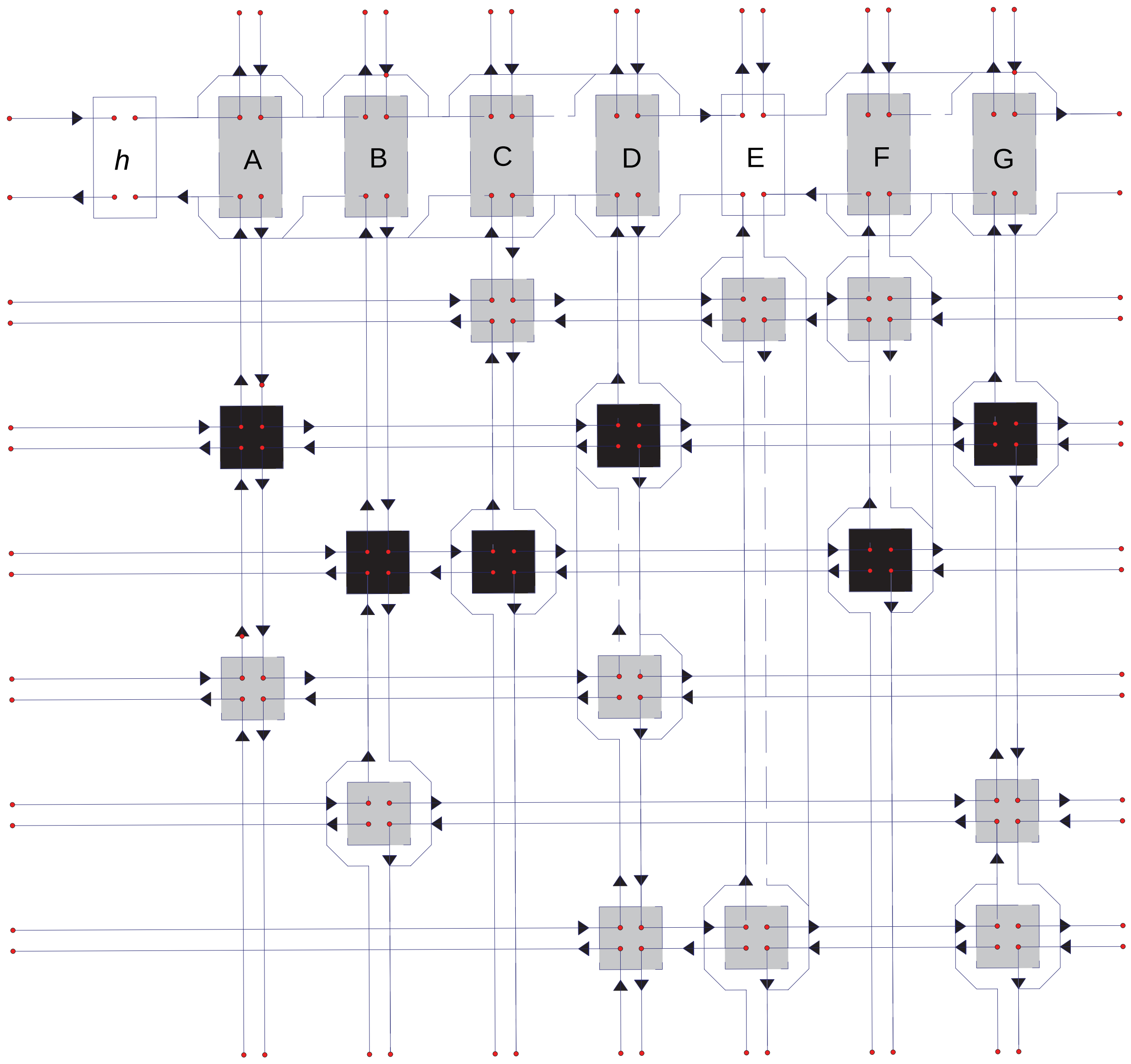
Cover C



Cover F



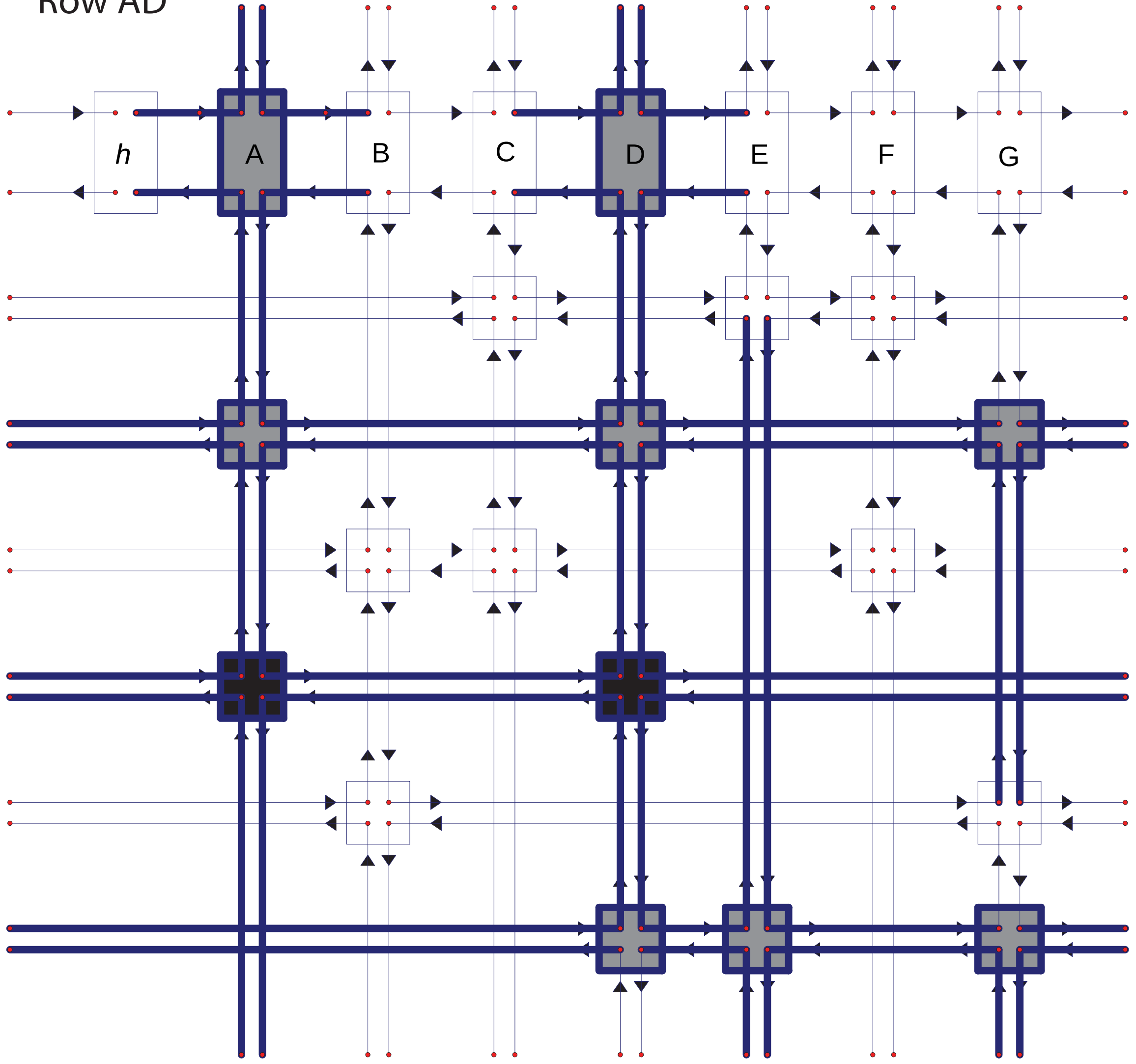
Search(3)



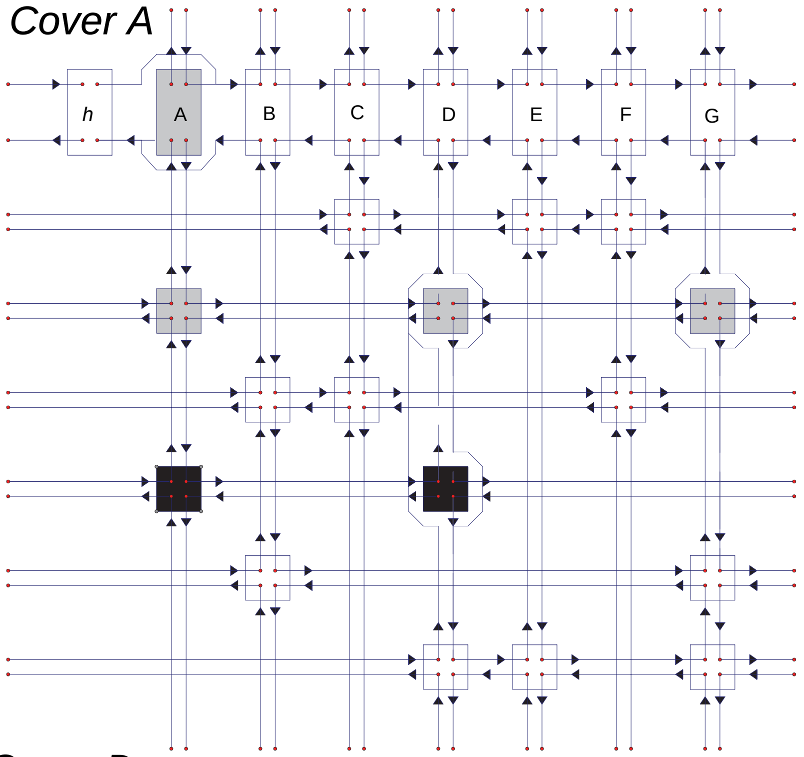
Column E has no nodes.
Return to search(1).

Search(1)

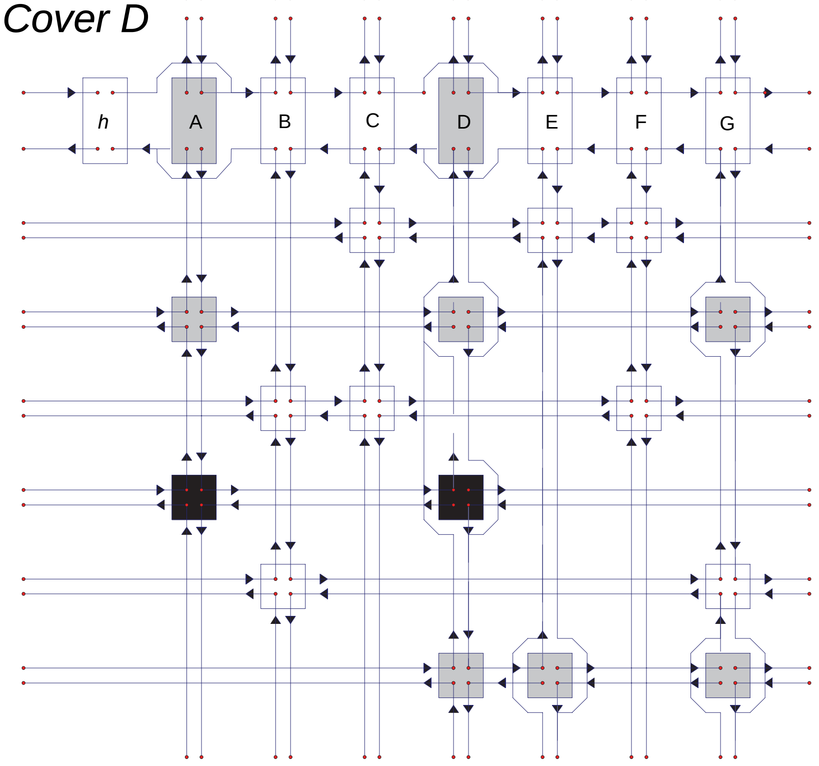
Choose Column A
Row AD



Cover A

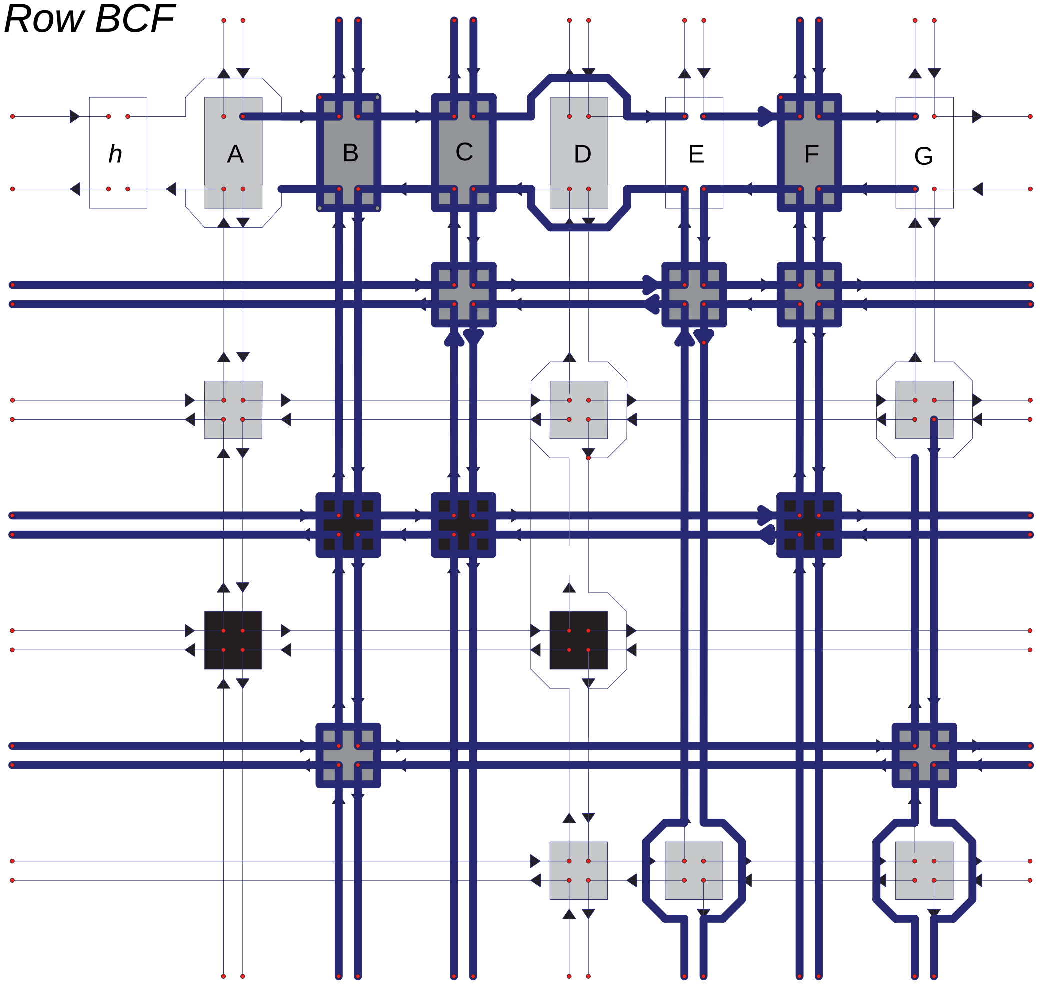


Cover D

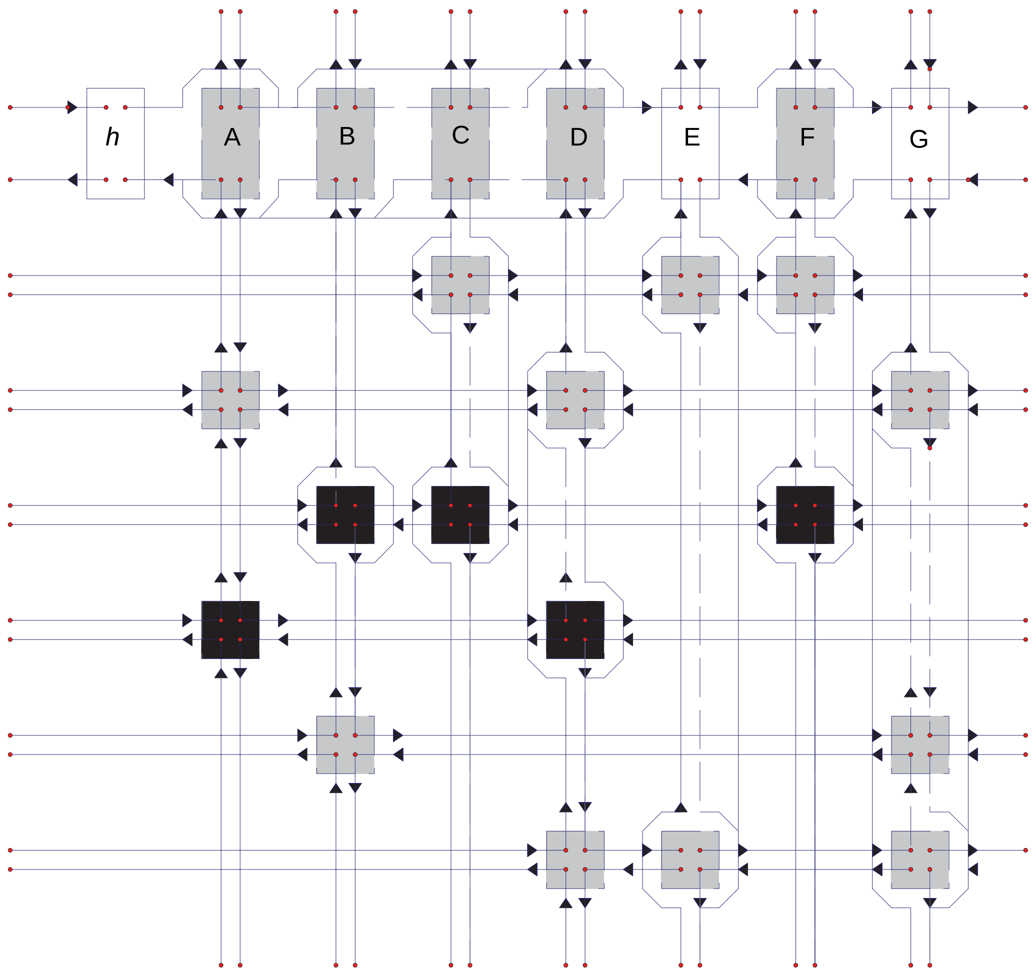


Search(2)

Choose Column B
Row BCF



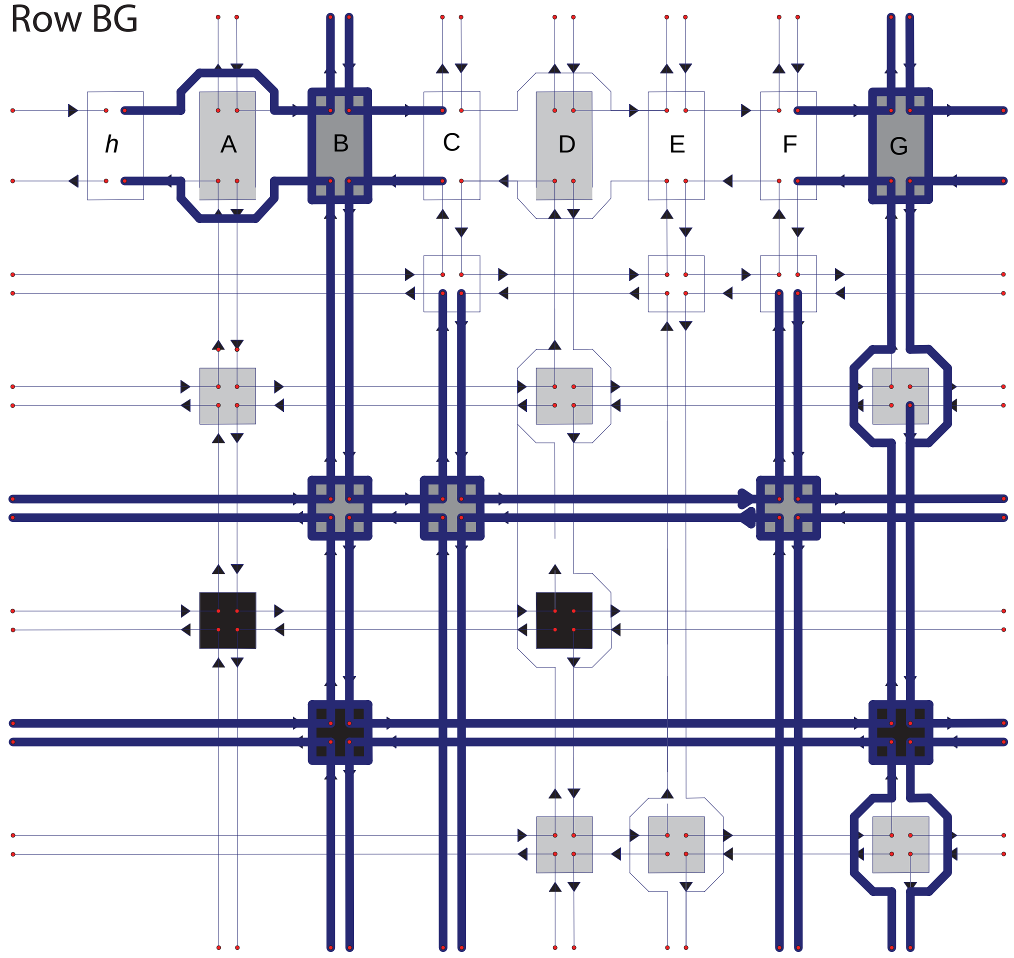
Search(3)



Column E has no nodes.
Return to search(2).

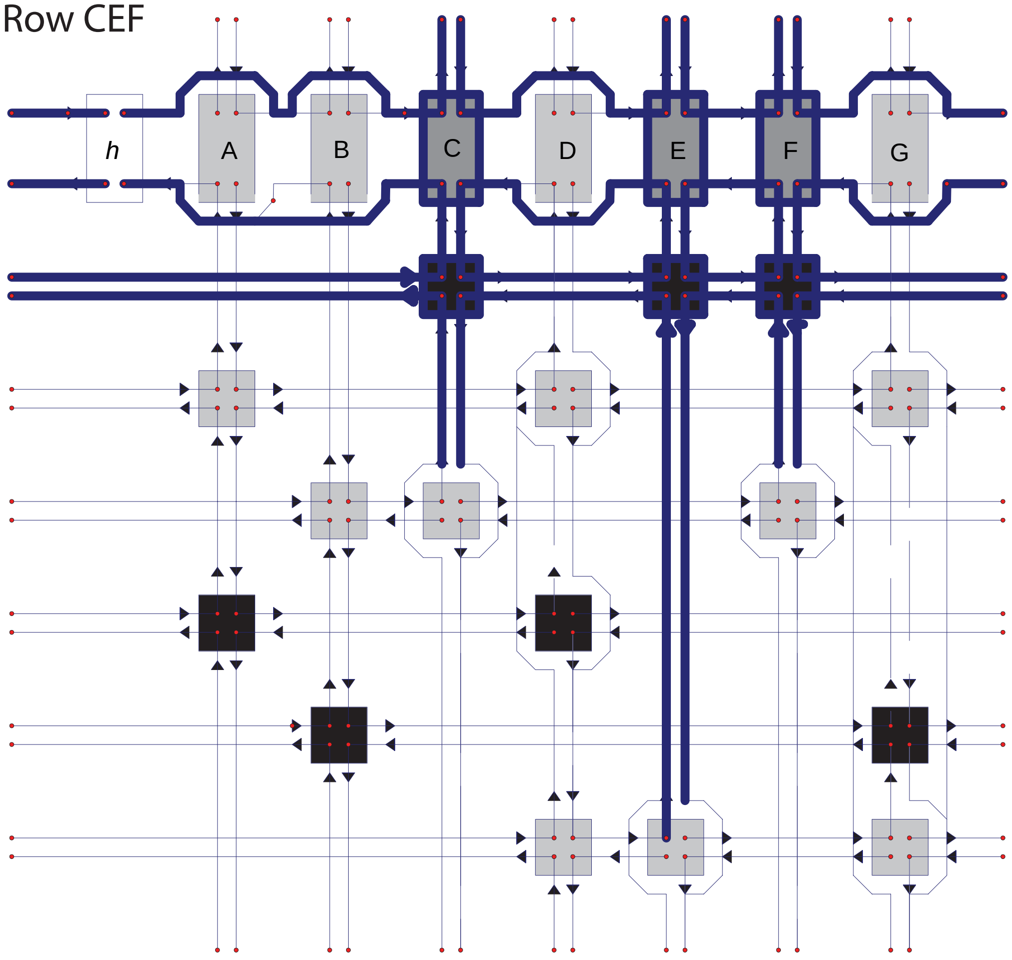
Search(2)

Choose Column B
Row BG

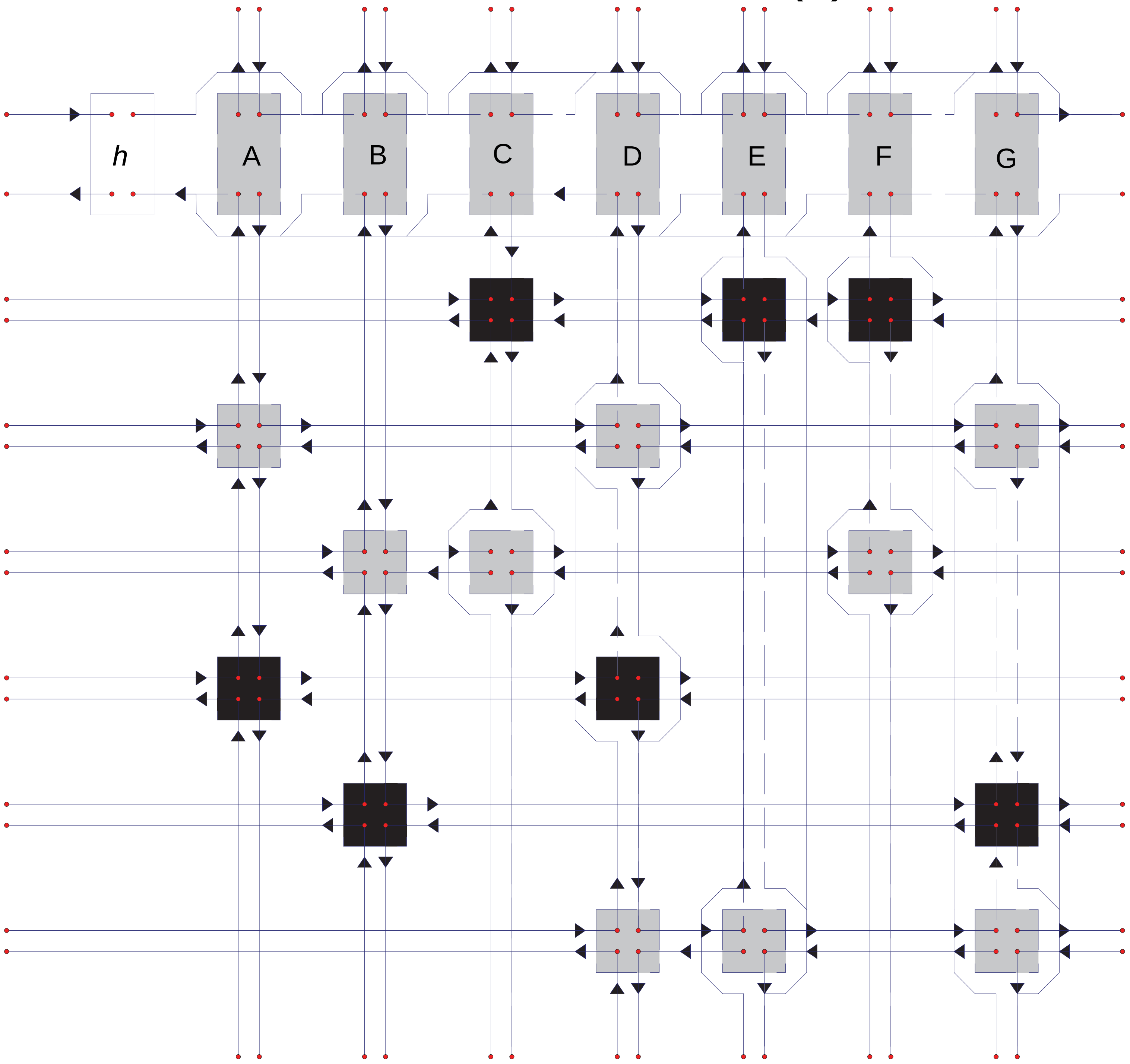


Search(3)

Choose Column C
Row CEF



Search(4)



All columns have been removed.

The solutions is the set:
Row AD
Row BG
Row CEF

Sudoku Solver

In order to apply the Dancing Links Aglorithm to Sudoku, we need a sparse matrix. To create the sparse matrix of Sudoku, we need to recognize what the rows and columns represent.

The columns represent the constraints of the puzzle. In Sudoku, we have 4:

* A position constraint: Only 1 number can occupy a cell
* A row constraint: Only 1 instance of a number can be in the row
* A column constraint: Only 1 instance of a number can be in a column
* A region constraint: Only 1 instance of a number can be in a region

Therefore there are SIZE^2 * 4 columns., where SIZE is the number of candidates/rows/cols.
In a 4x4, there are 64 columns. In a 9x9, there are 324 columns.

The rows represent every single possible position for every number.
Therefore, there are SIZE ^ 3 rows.
In a 4x4, this would be 64 columns. In a 9x9, this would be 729 rows.

Credits

Sudoku Puzzles - http://www.nikoli.co.jp/puzzles/1/index_text-e.htm
Exact Cover Example - Knuth Figure 3
Exact Cover Dancing Links Representation based on Knuth's models
Visualization inspired by SudoCue by ruud van derWerf

Dr. Donald Knuth and his lecture series: Computer Musings and his paper: Dancing Links
[http://www-cs-faculty.stanford.edu/~knuth/]
Standford University: [http://www.stanford.edu/]
Stanford Center for Professional Development [http://scpd.stanford.edu/scpd/default.htm]
Wikipedia [http://en.wikipedia.org/wiki/Main_Page]
Sudoku Programmer's Forum [http://www.setbb.com/phpbb/index.php]
Stan Chesnutt [http://www.bluechromis.com:8080/stan/chesnutt.html]
Ruud van der Werf and his program SudoCue [http://www.sudocue.net/]
Bob Hanson and his Sudoku Solver [http://www.stolaf.edu/people/hansonr/sudoku/]
The American Computer Science League (ACSL) [http://www.acsl.org/]
The Harker Research Symposium sponsored by WiSTEM [http://web.harker.org/WiSTEM/]
And finally my teacher, Dave Feinburg [http://faculty.harker.org/DaveF/index.htm]

For more information, visit my website at:
www.ocf.berkeley.edu/~ritac/didi/public portal
Or see my paper at
www.ocf.berkeley.edu/~ritac/didi/sudoku