Bhavana Yerraguntla

Bhagya Dhome

SUDOKU SOLVER

Approach: Representing Sudoku as an exact cover problem and using Algorithm X and Dancing Links to solve the represented problem.

Before we start, let us try to understand what **Exact Cover** is;

Exact Cover:

Given a set S and another set X where each element is a subset to S, select a set of subsets S* such that every element in X exist in exactly one of the selected sets. This selection of sets is said to be a cover of the set S. The exact cover problem is a decision problem to find an exact cover.

E.g. Let S = {A, B, C, D, E, F} be a collection of subsets of a set X = {1, 2, 3, 4, 5, 6, 7} s t:

A = {1, 4, 7}

B = {1, 4}

C = {4, 5, 7}

D = {3, 5, 6}

E = {2, 3, 6, 7}

F = {2, 7}

Then the set of subsets S* = {B, D, F} is an exact cover.

Before we represent the Sudoku problem into an exact cover, let's explore an example smaller.

The basic approach towards solving an exact cover problem is simple

1) List the constraints that needs to be satisfied. (columns)
2) List the different ways to satisfy the constraints. (rows)

Simple example: Imagine a game of Cricket (a batting innings), the batting order is almost filled with three batsmen A, B, C are yet to be placed and with three exact spots unfilled. One opener, one top order and one middle order.

Each batsman must play in one order, and no two batsman must be in the same order.

For example: A as opener, B as top order and C as middle order.

Constraints: opening order, top order and middle order.

Different possible ways: A as opener, A as top, A as middle, B as opener, B as top, B as middle, C as opener, C as top and C as middle.

Binary Representation:

	Opener	Top	Middle
A as opener	1	0	0
A as top	0	1	0
A as middle	0	0	1
B as opener	1	0	0
B as top	0	1	0
B as middle	0	0	1
C as opener	1	0	0
C as top	0	1	0
C as middle	0	0	1

From the binary matrix, select a set of row s.t there exactly 1 in the column.

There can be even more constraints like: A placed, B placed, C placed

	Opener	Top	Middle	A placed	B placed	C placed
A as opener	1	0	0	1	0	0
A as top	0	1	0	1	0	0
A as middle	0	0	1	1	0	0
B as opener	1	0	0	0	1	0
B as top	0	1	0	0	1	0
B as middle	0	0	1	0	1	0
C as opener	1	0	0	0	0	1
C as top	0	1	0	0	0	1
C as middle	0	0	1	0	0	1

This binary matrix representation is used in Donald Knuth‘s Algorithm X.

We will later reduce Sudoku to this representation.

Algorithm X

‘Algorithm X’ is “the most obvious trial and error approach” for finding all solutions to the exact cover problem. It is a recursive, non-deterministic, depth-first, backtracking algorithm.

1. If the matrix M has no columns, the current partial solution is a valid solution; terminate successfully.
2. Otherwise choose a column c (deterministically).
3. Choose a row r such that Mr, c = 1 (non-deterministically).
4. Include row r in the partial solution.
5. For each column j such that Mr, j = 1,
   for each row i such that Mi, j = 1;
   delete row i from matrix M.
   delete column j from matrix M.
6. Repeat this algorithm recursively on the reduced matrix M.

Let's run this on our previous batting innings example;

1. As the matrix has columns, run.
2. Choose a column (constraint) (opener)
3. Choose a row M1,1 -> 1 (A as opener)
4. And add it to the partial solution.

Solution:
A as opener

5. For each column (constraint) (opener) M1,1=1, remove Mi,1; I.e. remove any rows that also satisfy this constraint. And M1, 4=1 (A placed) (Note: I.e. Any column j and row 1 = 1)

	Opener	Top	Middle	A placed	B placed	C placed
A as opener	1	0	0	1	0	0
A as top	0	1	0	1	0	0
A as middle	0	0	1	1	0	0
B as opener	1	0	0	0	1	0
B as top	0	1	0	0	1	0
B as middle	0	0	1	0	1	0
C as opener	1	0	0	0	0	1
C as top	0	1	0	0	0	1
C as middle	0	0	1	0	0	1

Remove column 1 (Opener) and column 4 (A placed)

	Top	Middle	B placed	C placed
B as top	1	0	1	0
B as middle	0	1	1	0
C as top	1	0	0	1
C as middle	0	1	0	1

6. Repeat
1. Matrix not empty yet.
2. Choose a column constraint -> 2 (Top)
3. Choose a row -> M2,2 (B as top)
4. Add it to the solution.

Solution:
A as opener
B as top

5. For each column (constraint) (top) M2,2=1, remove Mi,2; I.e. remove any rows that also satisfy this constraint. And M2, 5=1 (B placed)

	Top	Middle	B placed	C placed

B as top	1	0	1	0
B as middle	0	1	1	0
C as top	1	0	0	1
C as middle	0	1	0	1

Remove column 2(Top) and column 5(B placed)

	Middle	C placed
C as middle	1	1

6. Repeat.
1. Matrix not empty, continue;
2. Choose a column constraint -> 3 (Middle full)
3. Choose a row -> M3,3 (C as middle)
4. Add it to the solution.

Solution:
A as opener
B as top
C as middle

5. For each column (constraint) (top) M3,3=1, remove Mi,3; I.e. remove any rows that also satisfy this constraint. And M3, 6=1 (C placed)

C as middle	1	1

Remove column 3(Middle) and column 6(C placed)

6. Repeat.
7. Matrix empty, I.e. has no columns, the current partial solution is a valid solution; terminate successfully.

Reducing Sudoku to an Exact Cover Problem

Like the batting innings problem, Sudoku can be represented as an exact cover problem, the matrix is going to be huge! (Exact Cover Matrix for 9x9 Sudoku.)

Rules:

1. Cell: Each cell can only contain one integer (1-9)
2. Row: Each row can only contain 9 unique integers.
3. Column: Each column can only contain 9 unique integers.
4. Box: Each box(3x3 cells) only contains 9 integers (1-9)

In the binary matrix representation (exact cover), columns are constraints, while rows are the possible values for the cells.

For simplicity, let's forget the structure of the Sudoku block and just remember the constraints and binary representation of an exact cover problem.

(Assuming 9x9 Sudoku)

Rows of the Matrix: Every possible combination;

(9 (rows) x 9 (columns)) = 81(or cells) x 9 (Possible values) = 729 rows

Let's talk about columns, this is where it gets a little complicated:

Row – r, column – c & value – n (in reference to the above link)

Cell: Each cell can only contain one integer (1-9)

This constraint needs 81 columns, one per cell.

Like in the Batting Innings example where each Batsman could be in opener or top or middle order, each cell could have 1 to 9 possible values I.e. r1c1n1... r1c1n9 belongs to r1c1.

Row: Each row can only contain 9 unique integers.

This constraint needs 81 columns, one per row/value combination.

r1c1n1... r1c9n1 -> r1n1

Column: Each column can only contain 9 unique integers.

This constraint needs 81 columns, one per column/value combination.

r1c1n1... r9c1n1 -> c1n1

Box: Each box (3x3 cells) only contains 9 integers (1-9)

This constraint needs 81 columns, one per box/value combination.

r1c1n1, r1c2n1, r1c3n1

r2c1n1, r2c2n1, r2c3n1

r3c1n1, r3c2n1, r3c3n1 … belongs to b1n1

Total columns -> 81 x 4 = 324

I.e 729 x 324 exact cover matrix.

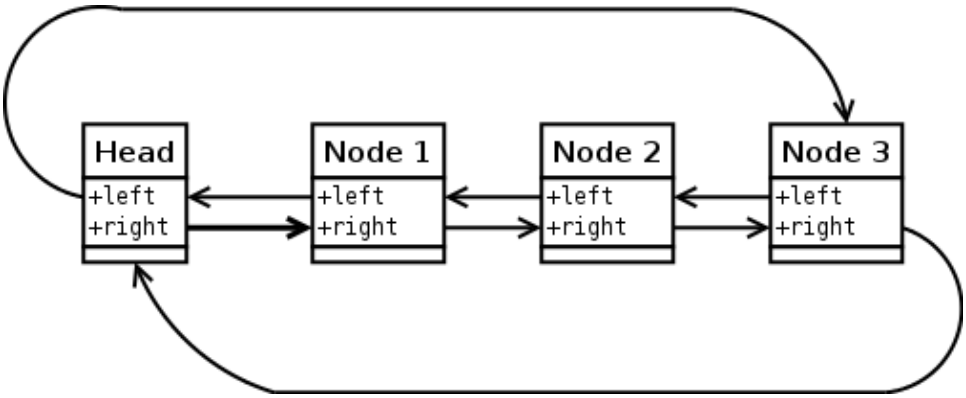
Dancing Links

Donald Knuth created an efficient method for implementing Algorithm X; called Dancing Links.

Before discussing more about this let's see the data structure of doubly-linked lists; [8]

struct node {
Int data;
node * left;
node * right;};

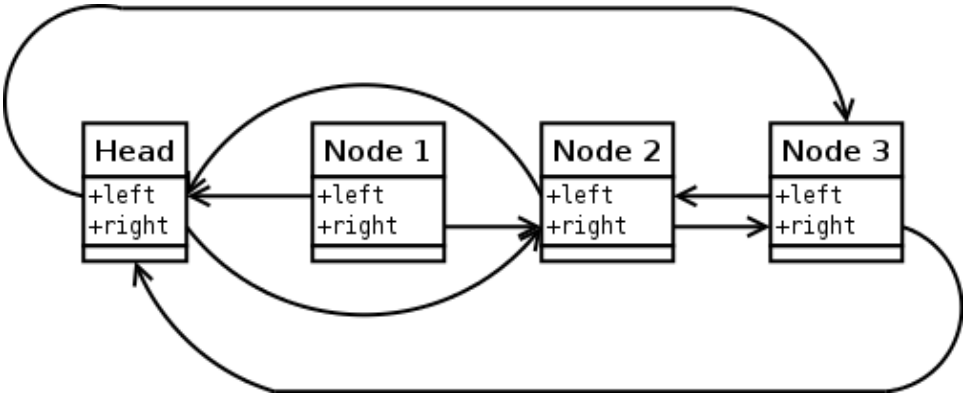
Where a list always consists of at least one element, the head of the list. If the list contains no data show left and right on the head itself.



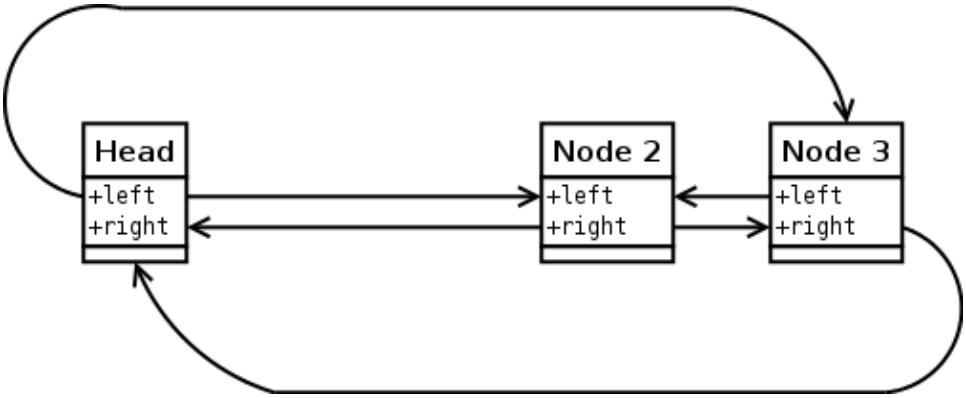
If you have a pointer to an element node1and you want to delete it, just re-assign two pointers ("bend"):

node-> left-> right = node-> right;

node-> right-> left = node-> left;



It node1 still points to its neighbors, but if you start from the top of the list and walk along the pointers, you cannot get there. It is effectively deleted.



The graphics omit the data that can be stored in the individual elements, since they have no influence on the bending of the pointer.

Incidentally, a deleted item can be restored very easily because it still has pointers to the right and left. This is enough

node1-> left-> right = node1;

node1-> right-> left = node1;

Matrix using doubly linked list:

struct node {
node * left,
node * right,
node * top,
node * bottom};
Each node has 4 pointers: left, right, top, bottom.

And, it also points to its column header node.

This helps avoid searching through 0s to find 1s (as only non-empty nodes exist) and allows removal or re-inserting of nodes.

For every column; Column Nodes:

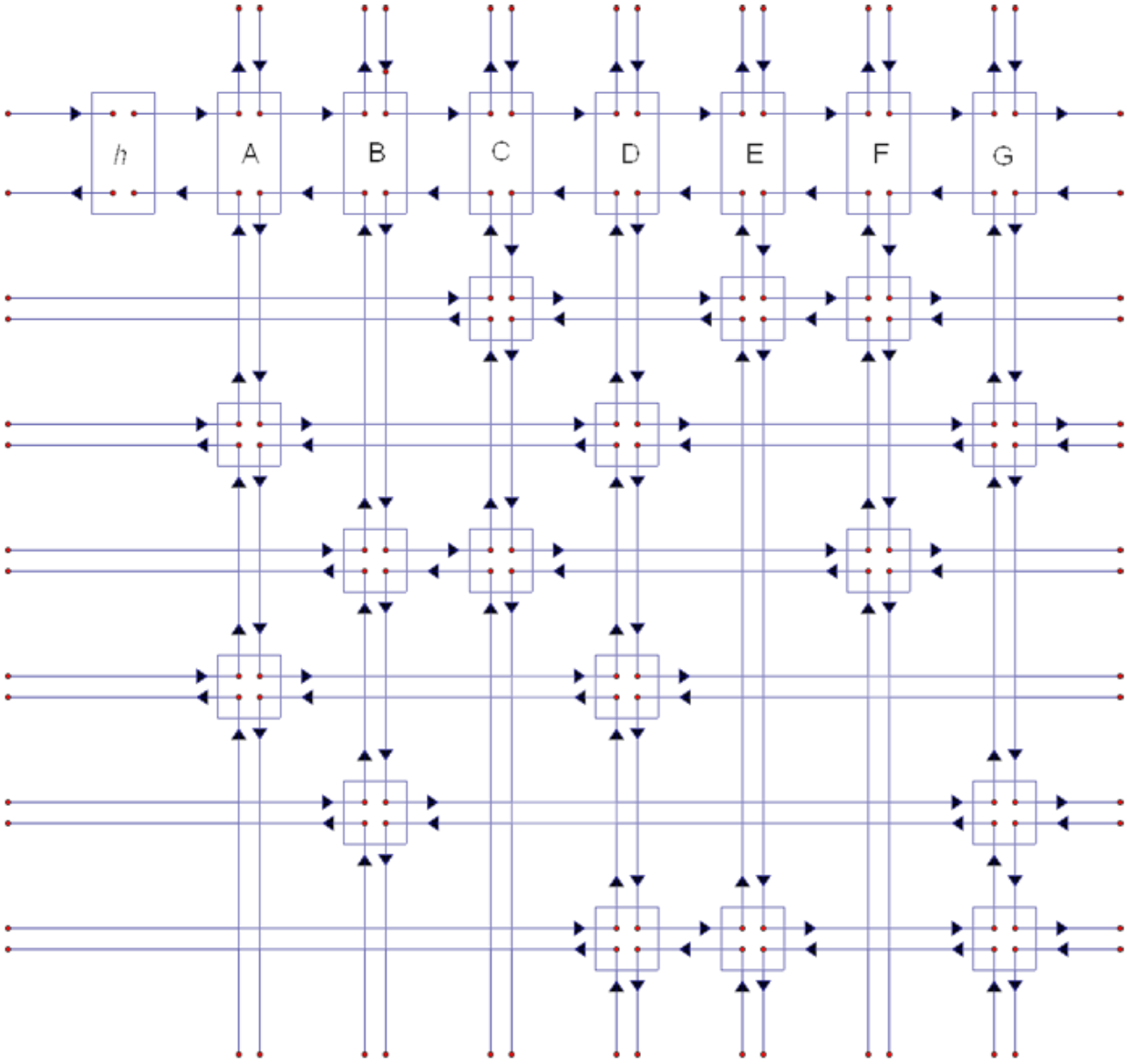
column headers store the no. Of cells occupied I.e. every node with 1.

Every node:

Has one. And pointers: LRTB and to its Column Nodes

Root header:

to find column headers of the unsatisfied constraints. Points to the first column node.



The basic operation is to remove a column from the matrix, together with all the rows that intersect that column. (Implementing Algorithm X)

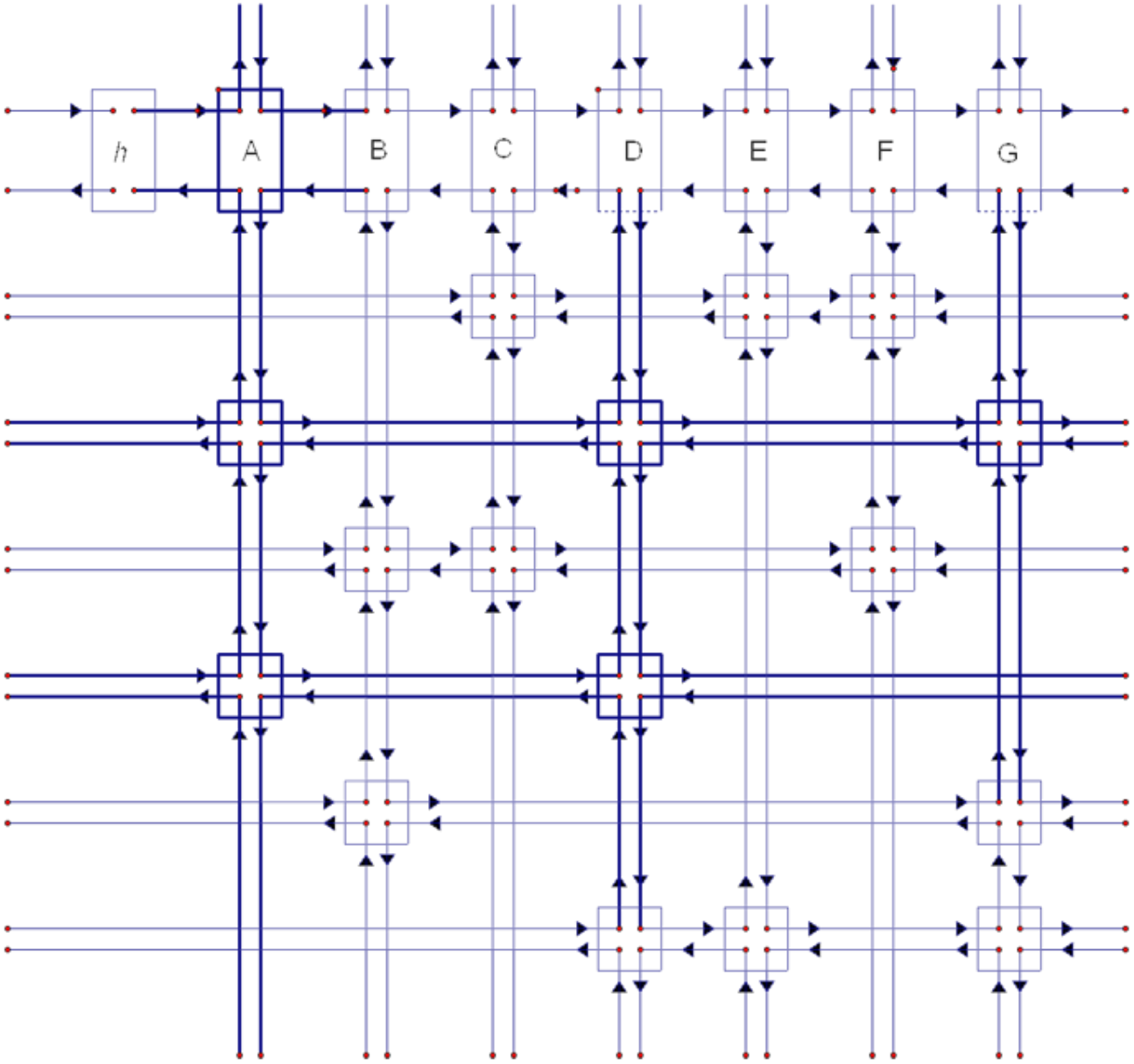
How to remove contrainst and solution in DLX, I.e covering:

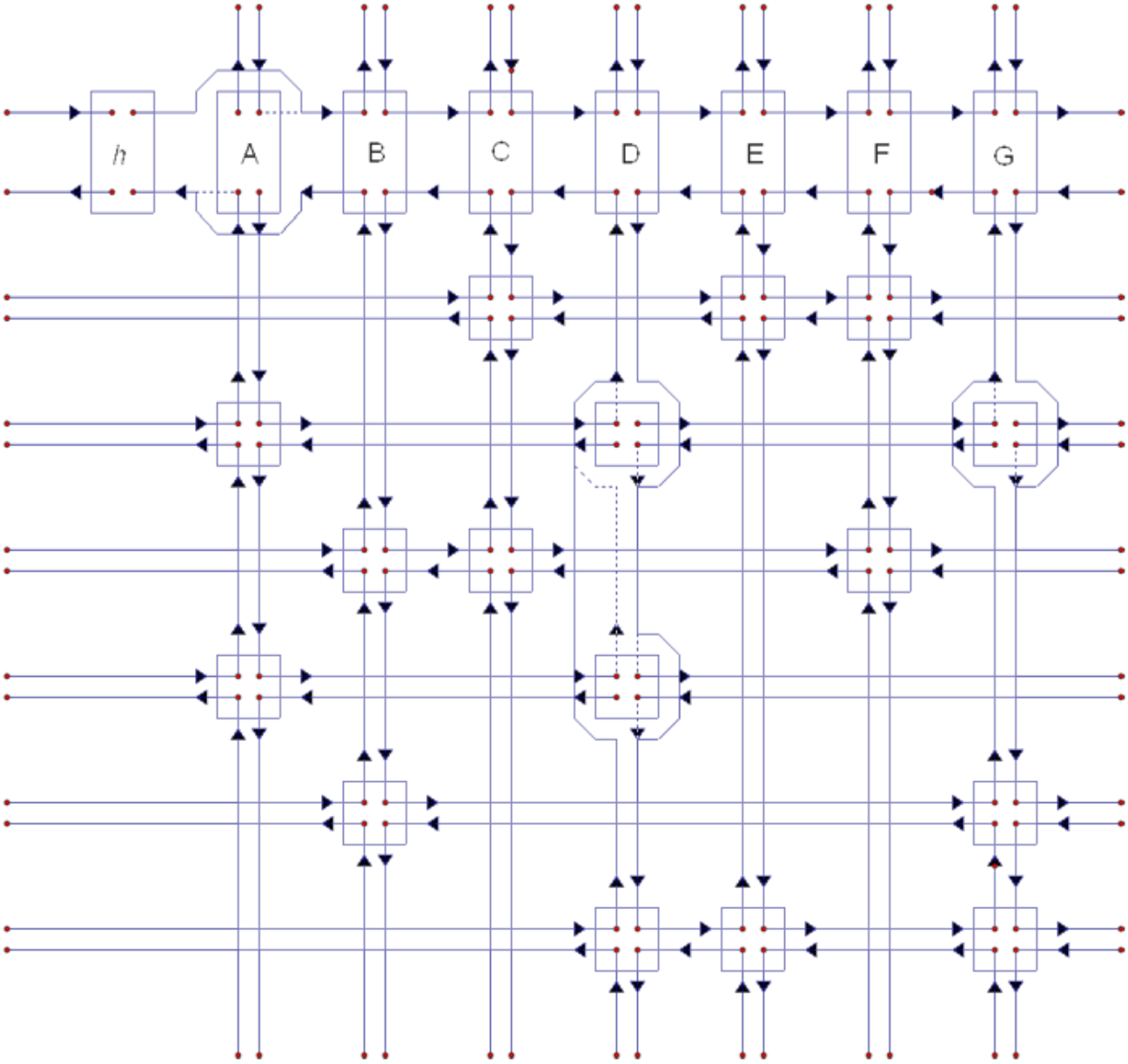
1. For each 1 in the row;
2. Cover the column header node with 1
3. Cover any rows with 1 in the same column

Cover(columnA)

It removes column A from the columns; goes down and removes rows traversing to the right.

Each node still points to the elements inside the matrix to allows backtracking.





Figures [3]

Uncovering: (for backtracking)

Because of the pointer to the column header, covering is easily reversed, as the cells still have pointers to their neighbours.

The nodes are put back into the matrix using the reverse operation of cover.

First put back the rows by traveling up the column and to the left of the row, and then put column back.

REFERENCES

[1]

The Algorithm Design Manual by Steven Skiena (Section 7.3)

[2]

http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group6Alexander/final/Patrik_Bergren_David_Nilsson.report.pdf

[3]

https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf

[4]

https://www.kth.se/social/files/58861771f276547fe1dbf8d1/HLaestanderMHarrysson_dkand14.pdf

[5]

http://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm

[6]

https://en.wikipedia.org/wiki/Knuth%27s_Algorithm_X

[7]

https://en.wikipedia.org/wiki/Exact_cover

[8]

http://sudokugarden.de/de/loesen/dancing-links