|  |
| --- |
| LOG515 – TP2 |
| Rapport de laboratoire |
| Résolveur de sudoku |

|  |
| --- |
| Marc-André Destrempes – DESM14119009 Samuel Beauchemin – BEAS99999999  6/13/2013 |

Table des matières

[Description de l’algorithme 2](#_Toc358760168)

[Description des problèmes rencontrés 3](#_Toc358760169)

[Description des améliorations que vous avez implémentées 3](#_Toc358760170)

[Autres éléments que vous jugez importants 3](#_Toc358760171)

[Références 3](#_Toc358760172)

Table des figures

[Figure 1 - Sparse Matrix 2](#_Toc358759887)

[Figure 2 - Sparse Matrix avec Column A de caché 2](#_Toc358759888)

# Description de l’algorithme

Nous avons fait des recherches pour essayer de trouver comment résoudre un sudoku de façon mathématique et nous sommes tombés sur le type de problème « Exact Cover ». Ce type de problème a été repris par la suite par Donald E. Knuth avec son algorithme « Dancing Links » que nous avons utilisé pour trouver toutes les solutions possibles d’un sudoku. Pour utiliser cet algorithme, nous devons commencer par créer ce que l’on appelle une « sparse matrix » comme dans la figure 1.

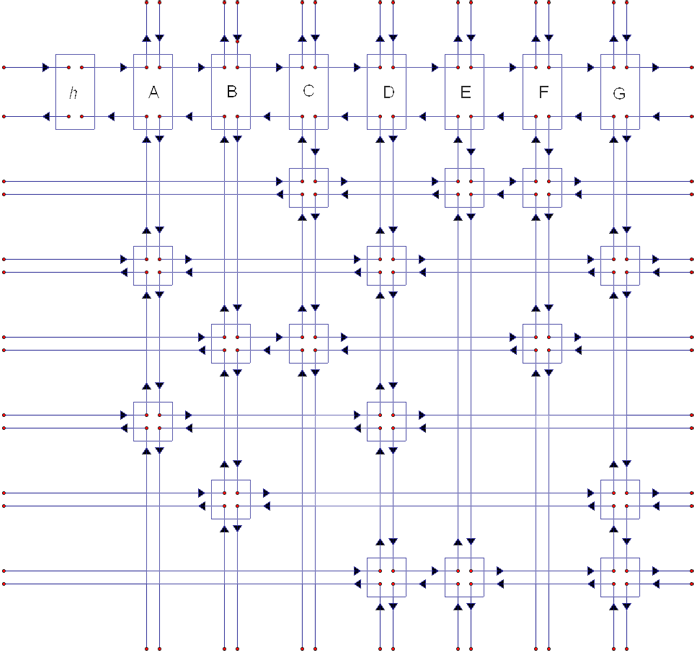


Figure - Sparse Matrix

Chacun des nodes (linked list double) représente un nombre dans la grille du sudoku et quand il n’y a pas de nombre, le node est vide. Le but de cet algorithme est d’ensuite cacher les colonnes jusqu’à ce dont on obtienne une seule colonne comme le démontre la figure 2.

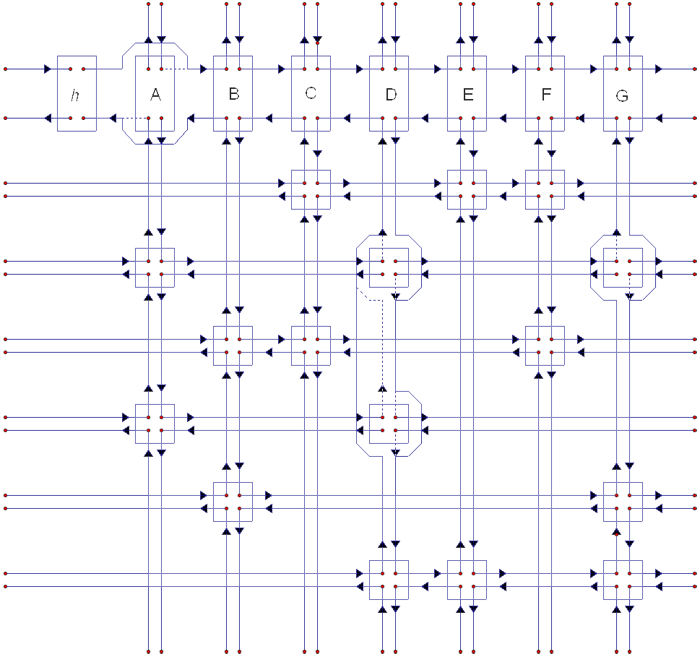


Figure 2 - Sparse Matrix avec Column A de caché

Lorsque nous obtenons une seule colonne, nous avons une solution.

# Description des problèmes rencontrés

Nous avons commencé par implémenter notre solution maison pour résoudre un sudoku efficacement, mais il s’est avéré que notre solution était moins performante que la solution non optimise que la version avec backtracking seulement. C’est à partir de là que nous avons décidé de faire des recherches et que nous avons trouvé l’algorithme « Dancing Links » de Donald E. Knuth.

Nous avons aussi eu de la difficulté à implémenter la « sparse matrix » pour respecter les 4 contraintes d’un sudoku qui sont, 1 chiffre unique par ligne, 1 chiffre unique par colonne, 1 chiffre unique par carre et 1 chiffre par case.

# Description des améliorations que vous avez implémentées

Nous avons fait un tri des colonnes de la « sparse matrix » du plus petit nombre d’éléments au plus grand pour augmenter la vitesse de l’algorithme.

Nous avons stoppé des boucles lorsque l’élément recherche était trouvé pour augmenter la vitesse de l’algorithme et éviter des recherches inutiles.

# Autres éléments que vous jugez importants

## Références

* http://en.wikipedia.org/wiki/Exact\_cover#Sudoku
* http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf
* http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/sudoku.paper.html
* http://cgi.cse.unsw.edu.au/~xche635/dlx\_sodoku/
* http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/presentationboard.pdf
* http://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm
* http://www.ams.org/samplings/feature-column/fcarc-kanoodle
* http://stackoverflow.com/questions/1935120/representing-a-100k-x-100k-matrix-in-java