Combinatorics and Graphs

Module code: 502042
Asignment # 2

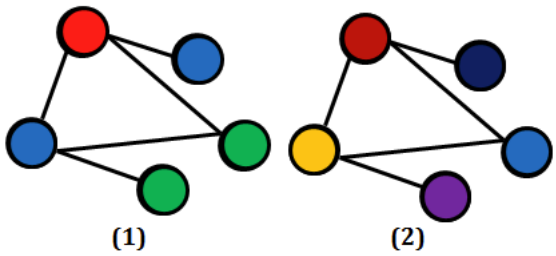
February 12, 2017

0 Objective

In this Assignment #2 you must used coloring graph to find solution for Sodoku puzzle which built in Assignment #1.

1 Coloring Graph

Graph coloring is the assignment of colors to vertices of a graph such that no two adjacent vertices have the same color. The goal is to use the minimum number of colors possible. Below are another two possible colorings for the example graph. Coloring (1) uses the 3 colors while Coloring (2) uses 5 colors, hence, coloring (1) is better than coloring (2).

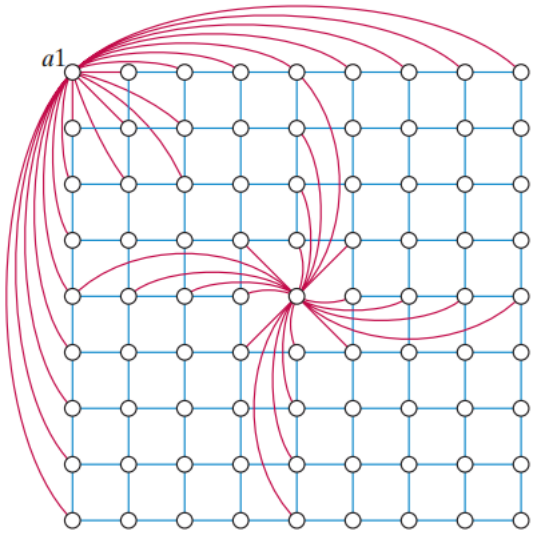


2 Coloring Graph for Sudoku problem

2.1 The Sudoku graph

A Sudoku puzzle (9×9) can be thought of a graph with 81 vertices, one for each cell, and two vertices are connected by an edge if they cannot be assigned the same value. For example, all cells in the same row, column or block will have edges between their corresponding vertices.

Given a Sudoku puzzle we can build the associated graph. The given number in the puzzle can be used to add additional edges to the graph we can then use graph coloring to find a **9-coloring** of this graph (colors 1-9)

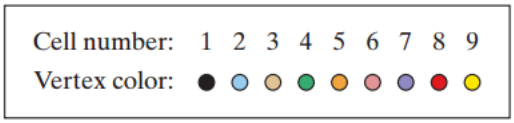


We put an edge connecting the corresponding vertices of the Sudoku graph S. For example, since cells *a3* and *a7* are in the same row, there is an edge joining their corresponding vertices; there is also an edge connecting *a1* and *b3* (they are in the same block), and so on. the graph above shows all **81 vertices** of S, two *a1* and *e5* have their full set of incident edges showing. When everything is said and done, each vertex of the Sudoku graph has **degree 20**, and the graph has a total of **810 edges**.
You can define **Graph** class as following:

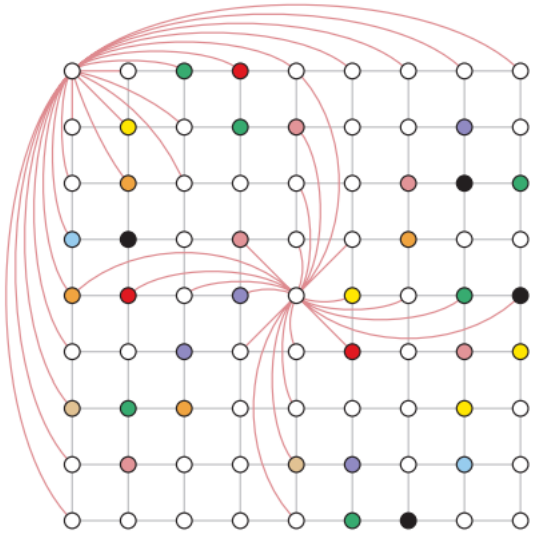
```
public class Graph
{
    private int V;   // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency List
    //Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
        adj[w].add(v); //Graph is undirected
    }
}
```

2.2 Coloring for Soduko graph

The second step in converting a Sudoku puzzle into a graph coloring problem is to assign colors to the numbers 1 through 9. This assignment is arbitrary, and is not a priority ordering of the colors as in the greedy algorithm — it's just a simple correspondence between numbers and colors.



To solve the Sudoku puzzle all we have to do is color the rest of the vertices using the **nine colors** as the graph following:



In this case, there are these colors are used and you will select these candidate colors which have been colored. For example, the candidate colors for *a1* vertex including {1, 2, 6, 7} because these colors as {3, 4, 5, 8, 9} are colored. You can choose '1' color and then continue to *a2* vertex. If at the any vertex there are not any candidate color available, you must be back tracking to the previous vertex and choose the next candidate color.
You can define **SudokuSolver** class:

```
public class SudokuSolver
{
    private Graph g;
    private int cColored; // No. of colored vertices
    int result[]; // list colors for each vertex
}
```

3 Instructions

3.1 Grid Layout

Your program will read the layout for each Sudoku puzzle from a file that contains a 9 × 9 matrix of single characters. The characters will be the digits

from 1 to 9 inclusive, plus the '0' character for any unassigned cell, where all of the cells are separated by space characters. Example of input puzzle given below:

```
0 6 0 2 0 4 0 5 0
4 7 0 0 6 0 0 8 3
0 0 5 0 7 0 1 0 0
9 0 0 1 0 3 0 0 2
0 1 2 0 0 0 3 4 0
6 0 0 7 0 9 0 0 8
0 0 6 0 8 0 7 0 0
1 4 0 0 9 0 0 2 5
0 8 0 3 0 5 0 9 0
```

and output of solution for Sudoku puzzle is presented as formatted file below:

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

3.2 Requirements for your program

- The main class (containing the main function) **must be named as SudokuSolver.java**
- Student do not add other libraries except libraries is imported from code sources.
- Your program **must take command line arguments and must follow this order** of: first your's program, puzzle file name, solution file name. Example: *java SudokuSolver.java puzzle1.txt solution1.txt*
- *Notes:* Student will recieved zero if:
  - Program that **do not compile**.
  - Program that **work only on your machine**.
  - Program that **do not run by command line arguments and your program name is different with requirement**
  - Your code is the same with another student.

3.3 Submission

- Submit your project through website of course.
- **Deadline**: **April 23, 2017**

4 References

1. Khee-Meng Koh, Chuan Chong Chen, [1992], Principles and Techniques in Combinatorics, World Scientific Publishing Company.

2. John Harris, Jeffry L. Hirst, Michael Mossinghoff, [2008], Combinatorics and Graph Theory (2nd Edition), Springer.

3. C. Vasudev, [2007], Combinatorics and Graph Theory, New Age International Pvt Ltd Publishers.

4. Miklos Bona, [2011], A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory (3rd Edition), World Scientific Publishing Company.

5. Sudoku: Bagging a Difficulty Metric & Building Up Puzzles

6. https://www.codeproject.com/Articles/801268/A-Sudoku-Solver-using-Graph-Coloring