COMP 560 Homework 1
Group Members: Zachary Reyes, Jacob Chandran, Jed Padoa

Description:
This project is two algorithms that both try and solve the map coloring problem. The map coloring problem is as follows: given an undirected graph and a certain colors, color the map in a way such that no connected nodes are the same color. If no solution is present, output false.

Input:
The input is a txt file that starts with a list of colors, then gives a list of nodes, then links between nodes (undirected so A B means A is connected to B both ways). These three lists are separated by a blank line, and each data point in the list is a new line. Our algorithm then converts this into a Dictionary (a representation of an adjacency list).

Output:
The output will be a Dictionary (a representation of the abstract concept of an adjacency list), that has the node and the color assigned to it. If the nodes follow the rule that no connected nodes are the same color, the next output will be true, otherwise the next output would be false.

AC Backtracking search:

At a very high level, a backtracking algorithm increments candidates to a solution, and then backtracks(ends path) when the candidate cannot be a solution. In this scenario, this "backtracking" is in reference to the implicit call stack for the recursive backtrack algorithm. Our algorithm is detailed below:

1. Construct a graph dictionary (an implementation of the abstract concept of an adjacency list) based on the input text.
2. Start at a node (somewhat random - based on the first node in the graph dict).
3. If color is consistent with adjacent nodes, assign it to that node.
4. The next node to "search", is decided by most restraining variable (i.e. the state with the least amount of colors left).
5. Iterate amongst the possible colors at each node (breath a decent way to think about it), "search" in terms of depth (dfs a decent way to think about it). Implemented with the help of a global ledger and a remaining variables list based on that global ledger and solution matrix.
6. Keep going until it hits a valid solution, if it hits a valid solution return the solution, output True.
7. If exhausted all valid possibilities (breath * depth) return, output False.

Local Search Algorithm:

At a very high level, a local search algorithm moves from solution to solution in the space of candidate solutions by applying changes until a solution is reached, or it does not find a solution in time and then terminates. The algorithm initialized a random color to every node in the graph as the starting state. Then, the algorithm found the node with the greatest amount of conflicts, then assigned the color that made that node had the least amount of conflicts. It does this iteratively until a solution is reached or a minute is finished, which it then times out. A numbered list is found below:

1. Assign random state
2. Find a value with greatest amount of conflicts
3. Assign to that value the color that has results in the least amount of conflicts with adjacent values(from graph)
4. Repeat until all conflicts are solved or time limit reached (1 minute)

Group Member contributions:
Zachary Reyes: Helped derive logic for AC Algorithm, developed test cases for both parts of the assignment. Pair programmed Backtracking and Arc Consistency with Jacob

Jacob Chandran: Helped derive logic for AC Algorithm. Pair programmed Backtracking and Arc Consistency with Zach. Pair programmed Hillclimbing with Jed

Jed Padoa: Derived logic for Hillclimbing, pair programmed Hillclimbing with Jacob, tested final program.