Sebastian Crowell

Brief statement of contribution:

This program was coded by myself but I did receive input from other students like Chase (I think that's how it's spelled) for logical information and how to describe things given the odd way that C tends to work. If you look at his code, it is unrelated and in fact in a different language entirely as he made many comments like "Why on earth would you use C?"

Backtracking:

The idea behind backtracking is fairly simple, you go through a depth first search looking at the nodes/vertices/etc. along a list/map/etc. and you apply some kind of value to each. That number shows a behavior more than anything else as it describes more about what the surrounding points should do rather than itself. This logic continues until the search reaches a point when it is confronted with a decision that doesn't allow it to progress because some number of nodes disagree with the value that should be placed at the current spot. This is where you back track up the branch of the search, in order to change a node, and then continue in a new or the same direction as before. The implementation of this is often done with for loops that represent the number of choices for the values of the states. Being unsure that this satisfied our project given the file containing three choices and the file containing four choices, I decided to work towards a recursive solution that allows any number of values and becomes faster/easier with more values. First, I tried to read in all the strings as there decimal representation to feed them into an array, which works for highly distinct strings (lots of characters) but failed ultimately in the US map because they are two characters a piece and often they result in a situation where two states like ME and FL are equal (both are 146). The solution to this was still a number representation for each state but instead of allowing the states to decide, I just used their position in the document. The purpose of this was to be able to value check the left side of the relation and the right side, using the space in between as a barrier, so that I could apply the bi-conditional property to each and then just store them as a Boolean value in a simple array. The hope of this was to then only have a running check of the relations between the states as they are colored so a red flag could be thrown up in the event that two states that are related had the same value. This structure made it very difficult to allow random values for the first state in the search as it often wants to get stuck there and then slowly work its way onwards, which is not so random anymore. It also may still be bugged as it seems AL, FL, and GA often have a shared color, AL being the problem, and that doesn't determine whether or not the map is completed because once AL has been changed a certain number of times, the program moves on regardless and then comes back to it. The run time varies between 2 million steps counted for the US map (AL is green) and I think 130 million for the longest (AL was yellow and it caused an immense amount of lag), but if AL is red then it is around 4 million every time. In the turned in code, I have removed the random first state but if it is wanted, change the 1 in line 41 to rand() % range + min.

Local search:

This program seems to function fairly similarly at a basic level, you do a lot of the same functions and actions in order to get the map working but just fill the map with random colors and have an added step to check the map for validity. For this I used the previous backtracking code and modified a check utility function to look at states and see that they have an acceptable color, if not then set it to no color, then check the next neighbor working like a depth first search at a point of failure. This causes a lot of problems for runtime as well as amount of solvable graphs. It may take 10-15 tries to get it to solve a graph and I have seen it solve ones in 2000 steps and some in 1 billion steps. It was very difficult to get another solution to work, this is the logic I was asking Chase about, and because jumping randomly in the C arrays often had a chance at causing a segmentation fault or a failure to escape a point. This is part of the problem with my choice of reading in the .txt files but I also don't know another solution to doing so. My final thought on this, and what I will continue to work towards, is that it is solvable by randomly jumping if I create a second undirected graph that is a copy of the relations that I can't jump to, and jump back from because I can then check that graph compared to my current ones to see what the next node I should jump to is. With this my solution would be a simple change to look at the number of relations, remove one relation from it, and then jump within that number of relations. Once this is done I can just copy the colors from that completed graph with zero relations to the second graph as the indexes of the colors will match the states.