Title:	Implementing Two CSP Algorithms for the Map Coloring Problem
Date:	07/28/2023
By:	Ethan Corgatelli

1.0 Table of Contents

2.0 Abstract

This paper describes the two constraint satisfaction problem algorithms I implemented. I used Rust for this project. I believe that this project turned out to be a success; I was able to implement two algorithms suggested by the project specification that solve the map coloring problem incrementally (first it tries with two colors, then with three, then with four). I was also able to create a way to display the final solution in a way that makes it easy to verify manually (although both algorithms already work by looping until the graph is programmatically found to be complete). My code and the full results of this project can be found here: github.com/ETBCOR/cs470/tree/master/proj3

- The Algorithms section explains the details of the algorithms I implemented.
- The Results section shows some of the outputs obtained from running the algorithms with different graphs as input.
- The Conclusions section contains my final thoughts on this project.

3.0 Algorithms

This section describes how my program solves the map coloring problem. 3.1 Naive Coloring serves as a starting point for Local Search. 3.2 Local Search and 3.3 Depth-First Search (with two heuristics) are the two primary algorithms I implemented. Both of the main algorithms have wrapper functions that handle the creation of output files, and the incremental attempts (i.e. first it tries with two colors, then with three, then with four).

- I created an enum *NumColors* whose value is one of: *Two*, *Three*, or *Four*.
- I created a struct *Choices* that simply holds four boolean values, one for each valid color. *Choices* has several associated functions, including a *new* function that takes a *NumColors*, initializing the booleans to have the correct number of colors available.
- I created an enum *Color* whose value is one of: *Empty*, *Red*, *Green*, *Blue*, or *Yellow*.
- I created a type alias *GraphColoring* which is simply a *BTreeMap<usize, Color>*. This is a B-Tree struct from Rust's standard library, using integers (representing the indexes of the variables/nodes of the given graph) as the keys, and *Color*s for the associated values.
- I created a trait *GraphForColoring* to extend the functionality of *UndirectedCsrGraph<usize>* from the graph crate (the crate provides basic graph functionality to build on) that has many functions (including *from_file, is_complete*, *count_confl*, *local_search_itr*, *depth_first_search_itr* and more, which are not described here — and *naive_coloring*, *local_search*, and *depth_first_search*, which are described below).

3.1 Naive Coloring

The *naieve_coloring* function is used as a starting point for local search. It takes a *NumColors* and returns a *GraphColoring* that is not guaranteed to be conflict-free. It first creates an empty *GraphColoring* called *coloring*, a *Vec<bool>* called *visited* with length equal to the number of nodes in the graph all initialized to *false*, and a *VecDeque<usize>* called *queue* initialized to contain *0* so that variable *X1* will be colored first. Then, for as long as there is a new node index to pop off the queue, it does so, then determines the choices available at that node. Then, the first choice available is selected for that node. If no choices are available, then *naive_coloring* chooses a random color for that node (respecting the *NumColors* from its arguments). Finally, all of that node's unvisited neighbors are marked as visited and pushed onto the *queue*. Finally, when the *queue* is empty, the fully colored *coloring* is returned.

3.2 Local Search

The *local_search* function takes a *NumColors* as input and returns a *GraphColoring* which could be conflict-free. It starts by calling *naieve_coloring*, assigning the result to a variable called *coloring*. While *coloring* isn't complete and the number of iterations hasn't exceeded a bailout constant, a random node that is part of at least one conflict is chosen and randomized based on its available color choices (the choice will be random under *NumColors* if there are no colors available). If *coloring* gets completed before the bailout constant, the detailed graph results are output; the failure case's output is more simple.

3.3 Depth-First Search

The *depth_first_search* function takes a *NumColors* as input and returns a *GraphColoring* which will definitely be conflict-free (if there is a valid solution for that number of colors). It first creates an empty *GraphColoring* called *coloring* and a *Vec<Choices>* called *choices_vec* with length equal to the number of nodes in the graph with all the values of the vector initialized using *Choices::new(num_colors)*. Then, for as long as the graph isn't completely filled out, the remaining unassigned node with the fewest available color choices is determined and saved into a variable called *idx_decision* (this is called the least-remaining-values heuristic). If this node is *stuck* (i.e. there are no options) then the graph cannot be colored with the provided *NumColors*, so the function prints some simple output then returns. Otherwise, a *Color* called *color_decision* is determined by creating copies of *choices_vec* for each available color based on the node's neighbors, then adding up the total amount of choices available to all nodes given that color decision, then using the color that yielded the most remaining choices for all the neighbors (this is called the least-constraining-value heuristic). Finally the decision is finalized; the *idx_decision* and the *color_decision* are inserted into the *coloring* as a key-value pair, and the neighbors' options are limited appropriately in the actual *choices_vec*. Again, if the graph was able to be fully colored, the resulting output will be more verbose, showing all the graph's connections with labels for the final color assignments.

4.0 Results

This section contains several of the output files that were generated during this project.

local-bipartite.txt

```
Starting local search for bipartite graph
(first with two colors, then three, then
four).
Graph:
X01 --> [ X02 | X04 ]
X02 --> [ X01 | X03 ]
X03 --> [ X02 | X04 ]
X04 --> [ X01 | X03 ]

Starting a local search with two colors...
completed (iterations: 0)
Final coloring: {0: R, 1: G, 2: R, 3: G}
Detailed graph:
X01: R --> [ X02: G | X04: G ]
X02: G --> [ X01: R | X03: R ]
X03: R --> [ X02: G | X04: G ]
X04: G --> [ X01: R | X03: R ]
```

depth_first-a_triangle.txt

```
Starting depth first search for a_triangle
graph (first with two colors, then three,
then four).
Graph:
X01 --> [ X02 | X03 ]
X02 --> [ X01 | X03 ]
X03 --> [ X01 | X02 ]

Starting a depth first search with two
colors... failed (iterations: 3)
Final coloring: {0: G, 1: R}

Starting a depth first search with three
colors... completed (iterations: 6)
Final coloring: {0: B, 1: G, 2: R}
Detailed graph:
X01: B --> [ X02: G | X03: R ]
X02: G --> [ X01: B | X03: R ]
X03: R --> [ X01: B | X02: G ]
```

local-CSPData.txt

```
Starting local search for CSPData graph (first with two colors, then three, then four).
Graph:
X01 --> [ X02 | X03 | X15 | X29 ]
X02 --> [ X01 | X21 | X22 ]
X03 --> [ X01 | X07 | X10 | X18 | X19 | X24 | X28 ]
X04 --> [ X22 | X29 | X30 ]
X05 --> [ X14 | X15 | X27 | X29 ]
X06 --> [ X12 | X20 | X22 | X28 ]
X07 --> [ X03 | X08 | X27 ]
X08 --> [ X07 | X10 | X11 | X17 | X19 | X30 ]
X09 --> [ X10 | X11 | X23 | X29 ]
X10 --> [ X03 | X08 | X09 | X11 | X24 ]
X11 --> [ X08 | X09 | X10 | X14 | X17 | X18 | X23 ]
X12 --> [ X06 | X13 | X14 | X24 ]
X13 --> [ X12 | X15 | X26 ]
X14 --> [ X05 | X11 | X12 | X18 | X24 | X29 | X30 ]
X15 --> [ X01 | X05 | X13 | X22 | X23 | X27 ]
X16 --> [ X25 ]
X17 --> [ X08 | X11 | X29 | X30 ]
X18 --> [ X03 | X11 | X14 | X21 | X25 | X29 ]
X19 --> [ X03 | X08 | X30 ]
X20 --> [ X06 ]
X21 --> [ X02 | X18 | X23 | X26 ]
X22 --> [ X02 | X04 | X06 | X15 | X24 | X25 | X27 | X30 ]
X23 --> [ X09 | X11 | X15 | X21 | X28 ]
X24 --> [ X03 | X10 | X12 | X14 | X22 ]
X25 --> [ X16 | X18 | X22 | X29 ]
X26 --> [ X13 | X21 ]
X27 --> [ X05 | X07 | X15 | X22 ]
X28 --> [ X03 | X06 | X23 ]
X29 --> [ X01 | X04 | X05 | X09 | X14 | X17 | X18 | X25 | X30 ]
X30 --> [ X04 | X08 | X14 | X17 | X19 | X22 | X29 ]

Starting a local search with two colors... failed with 30 conflicts (iterations: 2048)
Final coloring: {0: G, 1: G, 2: G, 3: R, 4: R, 5: R, 6: R, 7: R, 8: G, 9: R, 10: G, 11:
R, 12: G, 13: G, 14: R, 15: R, 16: R, 17: R, 18: G, 19: G, 20: R, 21: G, 22: R, 23: G,
24: G, 25: R, 26: R, 27: G, 28: G, 29: R}

Starting a local search with three colors... failed with 15 conflicts (iterations: 2048)
Final coloring: {0: B, 1: R, 2: R, 3: R, 4: B, 5: B, 6: B, 7: G, 8: B, 9: R, 10: R, 11:
B, 12: G, 13: G, 14: B, 15: G, 16: B, 17: B, 18: B, 19: R, 20: G, 21: B, 22: G, 23: B,
24: R, 25: R, 26: R, 27: G, 28: G, 29: G}

Starting a local search with four colors... completed (iterations: 11)
Final coloring: {0: R, 1: G, 2: G, 3: B, 4: R, 5: G, 6: R, 7: B, 8: Y, 9: R, 10: G, 11:
R, 12: Y, 13: Y, 14: G, 15: B, 16: Y, 17: B, 18: Y, 19: R, 20: R, 21: Y, 22: B, 23: B,
24: R, 25: G, 26: B, 27: R, 28: G, 29: R}
```

Detailed graph:

X01: R --> [X02: G | X03: G | X15: G | X29: G]
X02: G --> [X01: R | X21: R | X22: Y]
X03: G --> [X01: R | X07: R | X10: R | X18: B | X19: Y | X24: B | X28: R]
X04: B --> [X22: Y | X29: G | X30: R]
X05: R --> [X14: Y | X15: G | X27: B | X29: G]
X06: G --> [X12: R | X20: R | X22: Y | X28: R]
X07: R --> [X03: G | X08: B | X27: B]
X08: B --> [X07: R | X10: R | X11: G | X17: Y | X19: Y | X30: R]
X09: Y --> [X10: R | X11: G | X23: B | X29: G]
X10: R --> [X03: G | X08: B | X09: Y | X11: G | X24: B]
X11: G --> [X08: B | X09: Y | X10: R | X14: Y | X17: Y | X18: B | X23: B]
X12: R --> [X06: G | X13: Y | X14: Y | X24: B]
X13: Y --> [X12: R | X15: G | X26: G]
X14: Y --> [X05: R | X11: G | X12: R | X18: B | X24: B | X29: G | X30: R]
X15: G --> [X01: R | X05: R | X13: Y | X22: Y | X23: B | X27: B]
X16: B --> [X25: R]
X17: Y --> [X08: B | X11: G | X29: G | X30: R]
X18: B --> [X03: G | X11: G | X14: Y | X21: R | X25: R | X29: G]
X19: Y --> [X03: G | X08: B | X30: R]
X20: R --> [X06: G]
X21: R --> [X02: G | X18: B | X23: B | X26: G]
X22: Y --> [X02: G | X04: B | X06: G | X15: G | X24: B | X25: R | X27: B | X30: R]
X23: B --> [X09: Y | X11: G | X15: G | X21: R | X28: R]
X24: B --> [X03: G | X10: R | X12: R | X14: Y | X22: Y]
X25: R --> [X16: B | X18: B | X22: Y | X29: G]
X26: G --> [X13: Y | X21: R]
X27: B --> [X05: R | X07: R | X15: G | X22: Y]
X28: R --> [X03: G | X06: G | X23: B]
X29: G --> [X01: R | X04: B | X05: R | X09: Y | X14: Y | X17: Y | X18: B | X25: R | X30: R]
X30: R --> [X04: B | X08: B | X14: Y | X17: Y | X19: Y | X22: Y | X29: G]

depth_first-CSPData.txt

```
Starting depth first search for CSPData graph (first with two colors, then three, then four).
Graph:
X01 --> [ X02 | X03 | X15 | X29 ]
X02 --> [ X01 | X21 | X22 ]
X03 --> [ X01 | X07 | X10 | X18 | X19 | X24 | X28 ]
X04 --> [ X22 | X29 | X30 ]
X05 --> [ X14 | X15 | X27 | X29 ]
X06 --> [ X12 | X20 | X22 | X28 ]
X07 --> [ X03 | X08 | X27 ]
X08 --> [ X07 | X10 | X11 | X17 | X19 | X30 ]
X09 --> [ X10 | X11 | X23 | X29 ]
X10 --> [ X03 | X08 | X09 | X11 | X24 ]
X11 --> [ X08 | X09 | X10 | X14 | X17 | X18 | X23 ]
X12 --> [ X06 | X13 | X14 | X24 ]
X13 --> [ X12 | X15 | X26 ]
X14 --> [ X05 | X11 | X12 | X18 | X24 | X29 | X30 ]
X15 --> [ X01 | X05 | X13 | X22 | X23 | X27 ]
X16 --> [ X25 ]
X17 --> [ X08 | X11 | X29 | X30 ]
X18 --> [ X03 | X11 | X14 | X21 | X25 | X29 ]
X19 --> [ X03 | X08 | X30 ]
X20 --> [ X06 ]
X21 --> [ X02 | X18 | X23 | X26 ]
X22 --> [ X02 | X04 | X06 | X15 | X24 | X25 | X27 | X30 ]
X23 --> [ X09 | X11 | X15 | X21 | X28 ]
X24 --> [ X03 | X10 | X12 | X14 | X22 ]
X25 --> [ X16 | X18 | X22 | X29 ]
X26 --> [ X13 | X21 ]
X27 --> [ X05 | X07 | X15 | X22 ]
X28 --> [ X03 | X06 | X23 ]
X29 --> [ X01 | X04 | X05 | X09 | X14 | X17 | X18 | X25 | X30 ]
X30 --> [ X04 | X08 | X14 | X17 | X19 | X22 | X29 ]

Starting a depth first search with two colors... failed (iterations: 7)
Final coloring: {0: G, 1: R, 2: R, 6: G, 7: R, 9: G}

Starting a depth first search with three colors... failed (iterations: 26)
Final coloring: {0: B, 1: G, 2: G, 5: G, 6: B, 7: G, 8: G, 9: B, 10: R, 11: B, 13: G, 16: B, 17: B, 20: R, 21: B, 22: B, 23: R, 27: R, 28: R}

Starting a depth first search with four colors... completed (iterations: 69)
Final coloring: {0: Y, 1: B, 2: B, 3: B, 4: Y, 5: B, 6: Y, 7: B, 8: B, 9: Y, 10: G, 11: Y, 12: G, 13: B, 14: B, 15: Y, 16: Y, 17: Y, 18: Y, 19: Y, 20: G, 21: Y, 22: Y, 23: G, 24: B, 25: Y, 26: G, 27: G, 28: G, 29: R}
Detailed graph:
X01: Y --> [ X02: B | X03: B | X15: B | X29: G ]
X02: B --> [ X01: Y | X21: G | X22: Y ]
X03: B --> [ X01: Y | X07: Y | X10: Y | X18: Y | X19: Y | X24: G | X28: G ]
X04: B --> [ X22: Y | X29: G | X30: R ]
X05: Y --> [ X14: B | X15: B | X27: G | X29: G ]
```

```
X06: B --> [ X12: Y | X20: Y | X22: Y | X28: G ]
X07: Y --> [ X03: B | X08: B | X27: G ]
X08: B --> [ X07: Y | X10: Y | X11: G | X17: Y | X19: Y | X30: R ]
X09: B --> [ X10: Y | X11: G | X23: Y | X29: G ]
X10: Y --> [ X03: B | X08: B | X09: B | X11: G | X24: G ]
X11: G --> [ X08: B | X09: B | X10: Y | X14: B | X17: Y | X18: Y | X23: Y ]
X12: Y --> [ X06: B | X13: G | X14: B | X24: G ]
X13: G --> [ X12: Y | X15: B | X26: Y ]
X14: B --> [ X05: Y | X11: G | X12: Y | X18: Y | X24: G | X29: G | X30: R ]
X15: B --> [ X01: Y | X05: Y | X13: G | X22: Y | X23: Y | X27: G ]
X16: Y --> [ X25: B ]
X17: Y --> [ X08: B | X11: G | X29: G | X30: R ]
X18: Y --> [ X03: B | X11: G | X14: B | X21: G | X25: B | X29: G ]
X19: Y --> [ X03: B | X08: B | X30: R ]
X20: Y --> [ X06: B ]
X21: G --> [ X02: B | X18: Y | X23: Y | X26: Y ]
X22: Y --> [ X02: B | X04: B | X06: B | X15: B | X24: G | X25: B | X27: G | X30: R ]
X23: Y --> [ X09: B | X11: G | X15: B | X21: G | X28: G ]
X24: G --> [ X03: B | X10: Y | X12: Y | X14: B | X22: Y ]
X25: B --> [ X16: Y | X18: Y | X22: Y | X29: G ]
X26: Y --> [ X13: G | X21: G ]
X27: G --> [ X05: Y | X07: Y | X15: B | X22: Y ]
X28: G --> [ X03: B | X06: B | X23: Y ]
X29: G --> [ X01: Y | X04: B | X05: Y | X09: B | X14: B | X17: Y | X18: Y | X25: B | X30:
R ]
X30: R --> [ X04: B | X08: B | X14: B | X17: Y | X19: Y | X22: Y | X29: G ]
```

5.0 Conclusions

local-bipartite.txt shows the local search algorithm running on a graph designed to need two colors. This can be done by the naive coloring algorithm, leading to the output that the result was found in 0 iterations.

depth_first-a_triangle.txt shows the depth-first search algorithm running on a graph that is a triangle (3 nodes, each connected to the other two). We can see that the algorithm fails with two colors, then tries with three. The depth-first algorithm runs just once, so there is no iteration count in the output.

local-CSPData.txt shows the local search algorithm running on the data provided in the project specification. We can see that it fails at two and three colors, reaching the max iteration count of 2048, then succeeds with four colors after just 11 iterations.

I'm quite happy with the results of this project; I'm proud of what I was able to create around this problem.

depth_first-CSPData.txt shows the depth-first search algorithm running on the provided data. Because failure is detected early in this algorithm, we can see that when it tries to run with too few colors, it bails out early, and doesn't need to rely on a bailout constant like local search.