Map coloring as a CSP
William Craske #300180386
Assignment #1
Due: February 25th 2024

Abstract:

Constraint satisfaction problems find solutions for imposed limitations and restrictions on variables. In this case, our variables are the provinces of Canada, and the goal is to colour the map with colours that differ from adjacent provinces. To solve this issue, the implementation of a recursive backtrack algorithm on an adjacent matrix can provide a solution. Despite the constraints given, this algorithm solves a 13 node graph and outputs the answer in a text format. This algorithm accurately solved the graph on many test trials.

Introduction:

The objective defined by the assignment was to write a program with constraints that specified no neighboring provinces and territories in Canada will be assigned the same colour. This is a relatively small graph, however this algorithm could be implemented for larger countries as well.

This solution implemented a graph based approach where each province is considered as a node or vertex, and each border as an edge.To ensure the constraints are satisfactory for the node size, there are only three colours that are in use: red, blue, and green. For a larger graph, more colours may be necessary, as there could potentially be more neighbouring states than colours available.

In this scenario, the algorithm implemented was a recursive backtracking algorithm. This algorithm iterated through the list of provinces and assigned a colour to each, ensuring the constraints remain unbroken. For easy user implementation, a gui was also used to streamline the experience.

Experimental Setup:

To be able to provide constraints to the variables, an adjacency matrix was implemented to define where each province was in relation to the other. This was hard-coded into the program, and provides a connection for the graph. By using an adjacency matrix, we are able to check the province borders much easier than another data structure. Since we used this structure, where each province represents a node, we can then implement constraints and recursive backtracking. The recursive backtracking algorithm allows the program to run and assign, then reassign colours to ensure that the variables conform to the constraints. This is done by iteratively checking the nodes besides the current node. This algorithm mostly takes place in the colorRecursive function. Below is the pseudocode of the algorithm.

Function

If all the provinces have a color:

Return true//exit case

For each color in the list of all colors:

If province has adjacent node && if adjacent node has been assigned a color && if adjacent node's color == current color:

assign color to node

if there is a next province in the list then assigned a color //recursively calling function and allows for backtracking implicitly

return true

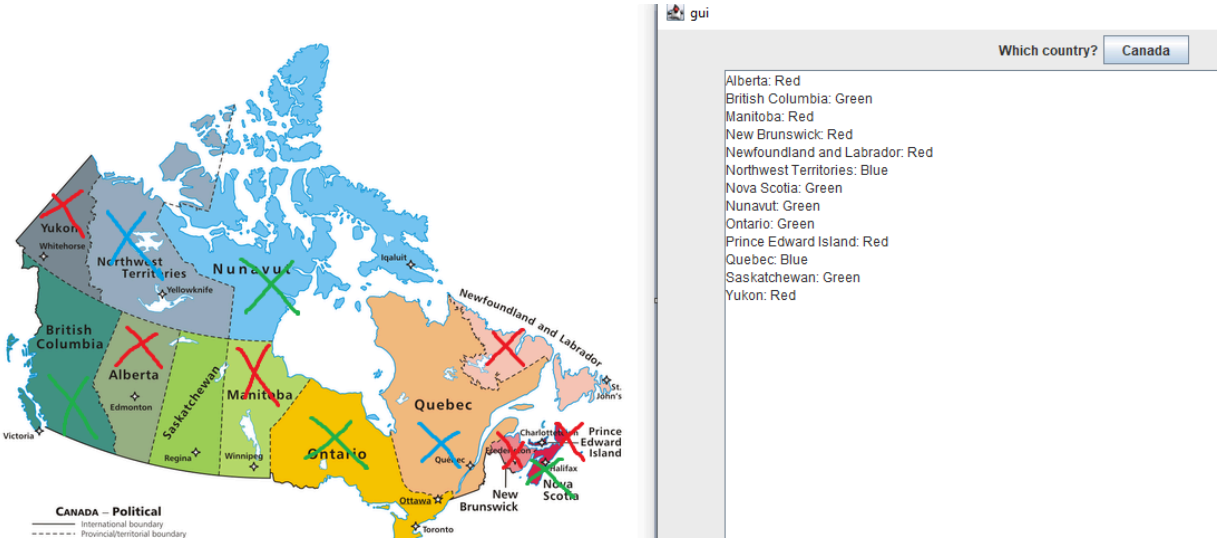
return false //if no colours can be used

From the beginning of the code, the function checks if all the provinces have a color. This is the exit case for the recursion. Then by iterating through the list of colors it checks the adjacent provinces for their existence and colour, and if the current colour is different. Then checks the next province by calling itself with the next province in the list. This pseudo code allows for the backtracking necessary due to the nature of recursive functions. If the function returns true, we print out the colours that were assigned to the provinces.

The user interacts with this program in a very simple way. Since the adjacency matrix is hard coded into the program, the user simply hits a button, indicating the country they wish to solve. In this case the only button is Canada. Then the output and solution of the program is outputted to the window. The user is expected to know that only three colours are being used. It is a given that the design is intuitive, and it is assumed that the program functions correctly.

Results and discussion:

The program successfully provided a colour for each of the provinces that were different from the adjacent provinces. The output of the program, alongside a manually translated map where the X's represent the colour that was assigned to each province.



From the given map, one might say there is an error from the Prince Edward Island being adjacent to Newfoundland and New Brunswick, however that is a user-design choice, since they are separated by water, the data entry of the adjacency matrix specified that they are separate.

The code has a time complexity overall of O(n^2). This is due to the colorRecursive method, which has a nested loop, iterating through each colour and province. For larger graphs, this algorithm would not be efficient, as it would take a long time. However, since there are only 13 nodes, the calculations are performed quickly.

Summary:

This graph based approach with the recursive backtracking algorithm is an effective solution for a graph of this specific size. By implementing an adjacency matrix and recursive methods to search each adjacent node, this proved to be a successful solution. For larger datasets, potentially using the United States instead, the algorithm will certainly be not as efficient. By incorporating heuristics to optimize the efficiency of the algorithm, larger datasets will be far more efficient.

References:

https://www.geeksforgeeks.org/graph-and-its-representations/
https://www.geeksforgeeks.org/constraint-satisfaction-problems-csp-in-artificial-intelligence/