New Mexico Tech
Department of Computer Science

Programming Assignment 3

CS122 Algorithms and Data Structures

Due 11:00AM, Wednesday, Nov. 24th, 2004

Objectives:

1. Gain familiarity with Object-Oriented programming and design by utilizing C++ objects to solve a problem.

2. Learn about and apply some common graph related techniques.

Specifications:

1. In this assignment, you will be required to derive and implement an algorithm capable of coloring a graph. Specifically:

    - The algorithm must be capable of receiving an arbitrary graph, which it will then traverse, assigning a color to each node in the graph such that (1) no node in the graph is the same color as any of its adjacent nodes, and (2) the fewest possible number of colors are used to color the entire graph. The colored graph should then be returned.

    - Included to help you get started on this project is a .cpp file that contains an implementation of a Graph and a GraphNode class, as well as some simple demonstration code in main() and a framework for implementing the graph coloring algorithm required by this project. It is highly recommended that you familiarize yourself with this code early on and make sure that you understand it (ask any questions that you may have if you don't). It is not required that you use this framework code, but it will likely make your life easier if you do. Note that everything is currently in a single monolithic file for simplified distribution and compilation, and that you may change this if you wish.

- Also included is a picture of the graph which is pre-loaded into the framework code (graph.jpg). Regardless of whether or not you choose to use the framework code or not, you are required to run your finished program on this graph, and then modify the image to reflect the results of your program (or just submit a textual description, such as "Node 0 = red, Node 1 = blue, etc., if you prefer), and submit this along with your code.

- You may implement your coloring algorithm however you wish, however the following may be helpful:

    1. Traverse the entire graph, keeping track of the maximally connected node as you do so (i.e. the node with the highest number of adjacent nodes). This will give you an upper bound on how many colors you will need.

    2. Assign the first color to the maximally connected node, assign the second color to all of its adjacent nodes.

    3. For each adjacent node, check to ensure that none of its adjacent nodes are of the same color. If they are, advance its color by one.

    4. Traverse the graph again, whenever you encounter an uncolored node, assign it the first available color that is not used by any of its adjacent nodes.

- Once your program has completed coloring the graph, it should print the colored graph (a simple text description of the color applied to each node is fine, see the existing print() function in the framework code as an example…it is perfectly acceptable if you want to just use this code as is to print your colored graph).

2. As always, coding and documentation guidelines must be followed, and a README file must be submitted along with your source code. At a minimum, the README file must include **compilation instructions**, and **instructions on how to use your program**. Additionally, a discussion of any design issues you ran into while implementing this project and a description of how the program works (or any parts that don't work) is also appropriate content for the README file.

3. Note that the user-friendliness of your program, which includes things like displaying clear, concise instructions to the user detailing what the program is and how it should be used when your program starts, and the formatting applied to input prompts and output data, **is** important and **does** count. This is another case where it is entirely possible that your program will never display a prompt to the user and read input, but the program should still print instructions describing what it is, and output should still be well formatted.

Submission Guidelines:

Use whatever archiving software you prefer to archive your source and README files together. Also be sure to include your colored copy of the graph image included with this project. The archive should be called '<your name>.Homework3.<ext>'. You may include a directory structure in your archive if desired, but it is not required. E-mail your archive to aroth@nmt.edu with the subject line of "[CS 122] Programming Assignment 3" by the due-date 11/24/04. In the body of your e-mail, be sure to include your name and student-id number

Grading Criteria:

A correct implementation of the graph coloring algorithm is worth 70 points. (Note that efficiency is not a grading criteria in this case. It does not matter if your algorithm makes 1 pass or 100, so long as the end result is correct)

A useful and descriptive README file is worth 15 points.

Adhering to coding style and documentation guidelines is worth 15 points.

Points will be deducted for assignments turned in after the due-date.

Extensions:

For a maximum of an additional 15 bonus points, complete the following enhancement to this project:

Modify your program so that it reads input about the graph to color from a file. The format used for a single line in the input file should be:

[Node #] [Adj. Node 1] [Adj. Node 2] … [Adj. Node n]

So the following input:

```
0 1 2
1 0 3
2 0
3 1
```

…specifies a graph with four nodes, 0, 1, 2, and 3, with node 0 being connected to nodes 1 and 2, node 1 being connected to node 0 and 3, node 2 being connected to node 0, and node 3 being connected to node 1.

Once your program can read input from a file, run it on a new graph layout of your choosing (with no fewer than 8 nodes) and submit the new layout (you may draw the actual graph or just include your input file), along with the results your program returns, with your assignment.