CS  5/7350 – Project

**Graph Coloring Analysis Project**

This project looks at implementing an algorithm multiple ways to solve a problem, analyzing the algorithms’ implementations, along with testing and characterizing your implementations to show they match your analysis.

The particular problem for the project is graph coloring.  For the first part of the project, various coloring orders will be implemented and analyzed for runtime efficiency and coloring efficiency. These efficiencies are intimately related to the data structures used for both the input graph data and the intermediate data necessary for the ordering and coloring algorithms.  For the second part, the coloring algorithms will be used to solve real-world types of problems involving conflict graphs.

Part 1.

INPUT:  A File containing unsigned integers one to a line. The first integers represent the P[] array followed by the E[] array.

* P[] = Pointer for each course I, 1 <= I <= N denoting the starting point in E[] of the list of courses in conflict with course I. That is, the conflicts for course I are indicated in locations E[P[I]], E[P[I]+1], …, E[P[I+1]-1].
* E[] = adjacency list of distinct course conflicts (length = 2M)

OUTPUT:

* For each vertex (course) the color, original degree, (and degree when deleted for the smallest last ordering). These should be printed in the order colored.
* Total number of colors used, the average original degree, and the maximum “degree when deleted” value for the smallest last ordering, and the size of the terminal clique for the smallest last ordering.
* Any other summary data you wish to include.

PROCEDURE:

CS-7350 students are to use six different methods for ordering the vertices in the graph. CS-5350 Students are to use four different methods, and may add two others for +5% extra credit each. One method all students are to use is the smallest last ordering given below, another is the ordering based on the Welch-Powell algorithm and the final one for all students is a uniform random ordering. The other orderings are of your own choosing.  Then you are to assign the minimum color to each vertex in the order determined by each ordering so that it doesn’t conflict with the other vertices already colored.  You will then compare the different ordering methodologies based on the following criteria:

* Asymptotic running time
* Asymptotic space requirement
* Total number of colors needed
* Report any other interesting metric

METHOD 1: Smallest Last Vertex Ordering:  The following format for the values of variables in various fields of the data node for each vertex may be used to save storage space. You may also split fields into different ones to avoid overloading a field for code readability and maintenance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vertex | Field 1 | Field 2 | Field 3 | Field 4 |
| I | Vertex ID | P(I): Pointer to edge list | 1. Current Degree 2. –1 when deleted 3. Color value | 1. Pointers for doubly-linked list of vertices of same current degree 2. Pointer for order deleted list |

* 1. Establish the pointers to the lists of vertices of degree j, 0 <= j <= N-1, and any other pointers or variables needed.
  2. Create the code to iteratively delete a vertex of smallest degree, updating the third and fourth fields of each data node to relate to the remaining graph, adding the vertex deleted to the ordered list of vertices deleted.
  3. After all vertices have been deleted, scan and assign colors (session periods) to each vertex in an order opposite to the order deleted, assigning for a “color” the smallest non-negative integer not previously assigned to any adjacent vertex. The output associated with each vertex may be printed as the coloring is assigned.
  4. Print any further output and summary information for the schedule.

For additional output with METHOD 1 you should include

* A graph indicating the degree when deleted on the vertical axes and the order colored on the x-axis.
* The maximum degree when deleted.
* The size of the terminal clique.
* Any other summary data you wish to include.

Welch-Powell Method

The Welch-Powell method may be in a time other than linear time, there will be extra credit for a liner time solution.

Other Possible Ordering:

You may use an ordering that is the name of the vertex. If you also use the reverse ordering of the name. These two orderings only count as one.

If you do this with a skewed distribution, then the reverse counts as a second ordering.

TESTING:

You should test your program in such a fashion to convince me it operates as expected.  Test input files will also be provided.

PART 1 REPORT:

Your report should describe your computing environment in detail and it should include a description of your algorithms of Part 1, the asymptotic bounding functions (upper and lower bounds) for the **worst case** running times and space required to order the vertices of the graphs for all six algorithms along with the asymptotic bounding functions for the running times and space required to assign colors. **Be prepared to provide runtime examples demonstrating your asymptotic bounds in Part 2.**

You should specifically provide a walkthrough of the smallest last vertex ordering in such a way as to be convincing that it performs as expected along with the different information requested.

Finally, you should be prepared to analyze the capabilities of your different coloring orders for various input graph sets.  In Part 2, you will be generating different sets for this analysis.

GRADING

The grade on this project will be based on the correctness of the code and the completeness and strength of the final report. The strength of the final report will be evaluated on the thoroughness of your algorithm descriptions and analysis along with the presentation of the timing data that supports your analysis.

You should submit the final report, the output for the test cases and the final source code as a single pdf.

DUE DATES

**Part 1** is due March 29th at 5:00pm. For 5% extra credit on this part.

After March 29th at 5:00pm and before April 10th at 5:00pm for full credit

After April 10th at 5:00pm for -5% credit on this part.

**Part 2** is due       April 27th at 5:00pm for 5% extra credit on the entire project

After April 27th   at 5pm and before May 4th at 5pm for full credit

After May 4th may require an incomplete in the course.

If your Part 2 grade is higher than your Part 1 grade, then your Part 1 will be reduced to10% of the project grade and Part 2 will be 90% of your project grade. Otherwise, your project grade will be the average of your Part 1 and Part 2 grades.

PART 2:

For Part 2 of your project, you will create graphs to test your coloring methods in Part 1.

INPUT:

* Number of V
* Number of Edges, E Vertices,
* Type [Complete, Cycle, Uniform, Skewed]

OUTPUT:

* The graph file to accept as input for Part 1.
* You may also pass the graph directly to part 1 through data structure, but still output the file.

PROCEDURE

For the “Complete” type, the number of edges on the input will be ignored and an edge will be placed between every vertex. This will be a total E=V\*(V-1)/2.

For the “Cycle” type, the number of edges on the input will be ignored and an edge will be placed between every vertex Vn and the next vertex Vn+1 . Finally an edge will be placed from the last vertex in the graph back to the first for a total E=V edges.

For the “Uniform” type, you should generate E edges where E < V\*(V-1)/2 and each edge is equally likely to be present in the graph.  Be sure your algorithm works well as E approaches V.

For the “Skewed” type, edges adjacent to vertex 0 should be more likely than edges adjacent to vertex 1, etc.  You will prioritize an edge based on the lowest numbered vertex it is adjacent to.  Be sure your algorithm works well as E approaches V.

Use your graph generation routines to analyze your coloring routines.  Particularly, use them to validate your running time analysis and use them to determine which coloring algorithms are best under various circumstances.

FINAL REPORT

Your final report should include everything from Part I with any **changes clearly marked** with the addition of your **running time analysis verification of your part 1 orderings** and **your Part 1 coloring ordering performance comparison in terms of how well it allows the graph to be colored**.  We have discussed runtime verification in class. The coloring analysis should look at the coloring algorithms vs different graphs.  This should be a substantial addition to your report.

Note, no analysis is required for the graph generation routines. These are simply drivers to assist the analysis of your coloring algorithms.  Unlike the homework, you may use any libraries available to you for these routines.

Turn in your code for part 1 and part 2 as an appendix in your report.