CS 5/7350 – Project

Graph Coloring Analysis Project

This project looks at implementing an algorithm in multiple ways to solve a problem, analyzing the algorithms' implementations for running time, and testing and your implementations to show they match your analysis. The project also looks at testing to determine how well the multiple implementations solve the problem.

The particular problem for the project is scheduling via graph coloring. For the first part of the project, different conflict graphs which can represent real world problems will be created and saved in files. For the second part of the problem, carious 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.

Part 1.

PURPOSE

Part 1 of the project is used to create different conflict graphs. Conflict graphs represent real-world problems. For example, one may be thought of as a set of vertices representing courses that students may take. An edge is present between two vertices (courses) in the graph if a student is taking both courses. This edge indicates they may not be scheduled at the same time. Once the graph is colored, the color of each vertex represents its timeslot and the total number of colors would represent the total number of required timeslots for that coloring.

The vertices which have a conflict in this project will be determined based on graph types and a random number generator with different distributions.

INPUT:

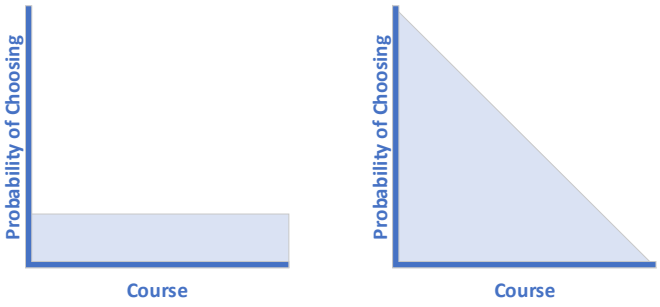
- V = Number of vertices. (MAX = 10,000)
- E = Number of conflicts between pairs of vertices for random graphs. (MAX = 2,000,000)
- G = COMPLETE | CYCLE | RANDOM (with DIST below)
- DIST = UNIFORM | SKEWED | YOURS

OUTPUT:

- E[] = adjacency list of distinct course conflicts (length = 2M)
- 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].

PROCEDURE:

The vertices for conflicts shall be chosen using a pseudo random number generator according to one of the specified distributions. These distributions are uniform, skewed, and one other of your choosing. The uniform distribution assumes each vertex is equally likely to be chosen for a conflict. The skewed distribution assumes lower numbered vertices are linearly more likely than higher numbered vertices. Graphs of these distributions are shown below.



You may use a built-in uniform distribution pseudo-random number generator, but must base the other distributions on the output of this built-in generator.

Part 2.

INPUT: A file like the output in Part 1:

- 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 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.
- An output file of the format where each line is VERTEX_NUMBER, COLOR_NUMBER

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 smallest original degree last 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 report on the following criteria for the different ordering methodologies:

- Asymptotic running time for Smallest Last Ordering and Coloring
- Asymptotic space requirement for Smallest Last Ordering and Coloring
- Total number of colors needed for each ordering
- Report any other interesting metric for each ordering

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

Smallest Original Degree Last

The Smallest Original Degree Last method is a subset of the smallest last ordering. You should determine the vertices to color based on their original degree, but not remove them from the graph as you do it. This should run in THETA(V+E).

TESTING:

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

FINAL REPORT:

The final report should be the equivalent of 10-15 typewritten double spaced pages and contain the following material.

Your report should discuss:

- Your computing environment in detail
- A description and analysis of your algorithms for generating the conflict graphs including
    - Asymptotic running time analysis, runtime tables and graphs for each graph generation method
    - Histograms showing how many conflicts each vertex has for each method.
- Vertex Ordering
    - Include a description of your vertex ordering implementations
    - Include a walkthrough of your Smallest Last Ordering on a small graph in such a way as to be convincing that it performs as expected.
    - Analyze your Smallest Last Vertex Ordering Implementation to include running time analysis, tables and graphs for it.
- Coloring Algorithm
    - Asymptotic running time analysis, tables and graphs for the routine that assigns colors to the vertices in the order specified.
- Vertex Ordering Capabilities
    - Total number of colors needed for a variety of different graphs based on ordering method.
    - An in-depth analysis of the capabilities different orderings based on your results.
    - Other output required for smallest last ordering
        - 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.
        - A discussion of how these bound the colors needed

In general, this report should present a good case to “sell” your scheduling program by listing its range of applicability and expected behavior in the most favorable light, while also acknowledging the limitations in an appropriate manner.

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. A key point in your report will also be the comparison of the different orders and conclusions you draw relating to how well they perform coloring different graphs.

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

DUE DATES

Some aspects of the project will be due as Homework Grades prior to these deadlines:

April 26th at 5:00pm for 5% extra credit on the project
After April 26th at 5pm & before May 3rd at 5pm full credit
After May 3rd at 5pm and before May 10th at 5pm for -5%
After May 10th is -10% and may require an incomplete in the course.