Project #07: Part 1 of 2: updated graph class

Complete By: **On-time: Wednesday 4/29 @ 11:59pm**
Late: Thursday 4/30 @ 11:59pm (-10%)

Assignment: **"graph.h" file**

Policy: **Individual work only, late work *is* accepted**

Submission: **"graph.h" file via Gradescope; the first 12 submissions are free, each additional submission 1 pt**

Assignment

We've been working with a **graph** class limited to at most 100 vertices. This limitation is due to the underlying implementation based on an adjacency matrix. Your assignment here in part 1 is to remove this limitation by rewriting the graph class to use an adjacency list representation. The "list" does not have to be a linked-list, you are free to use whatever data structure you want to represent a "list of edges". And you are free to use any of the built-in C++ data structures: map, set, list, deque, etc.

A current version of the "graph.h" file is available on Codio: **cs251-project07-graphs**. A main program is also provided that reads a graph from an input file, and then outputs the graph to see if it was built correctly. Use this as an initial testing platform if you wish. A sample input file is available in "graph.txt". If you prefer to work outside of Codio, these same files are available on the course dropbox under Projects, **project07-part01-files**.

You are going to completely rewrite the class, which means e.g. that you should delete the **EdgeData** structure, the **AdjMatrix** data member, and the **MatrixSize** constant. Your new class will have no size limit. You can even delete and replace the **Vertices** vector, it's up to you. The class must remain templated with VertexT and WeightT, and you must retain all public functions as currently defined. Your job is to replace how they are implemented, but you must keep all existing public functions. In particular, here are the major steps (and requirements):

1. Delete all aspects of the adjacency matrix.
2. Replace with an implementation based on an adjacency list; see zybooks section 10.13.
3. Delete the constructor graph(int n).
4. Add a default constructor graph().
5. If you decide to dynamically-allocate memory, add a destructor. You'll also need to add a

copy constructor and operator= to properly make deep copies.

6. Re-implement all other functions. For **addVertex**, delete the "if matrix is full" check.
7. NO LINEAR SEARCH --- you are better than that. Any instances of linear search will result in an automatic 0. Example: a call to **addVertex(v)** has to check if v exists, and this must be done in O(lgN) worst-case time (where N is the # of vertices). You may assume the graph is sparse, which implies a vertex has a small number E of edges and thus E is significantly less than the total # of edges M. This allows you to use a linked-list for your adjacency list, and it's legal to search this list in "linear" O(E) time since E is very small. What you cannot do is have a single list of *all* the graph edges, since this would require an expensive linear search of O(M) time.
8. Finally, update the **dump()** function to properly output the vertices and edges, based on your final implementation. When you output the edges, output in a readable format such as:

```
A: (A,B,80) (A,C,100), …
B: (B,A,100) (B,F,123), …
```

Grading, electronic submission, and Gradescope

Submission: "graph.h".

Your score on this project is based on two factors: (1) correctness of graph.h" as determined by Gradescope, and (2) manual review of "graph.h" for commenting, style, and approach (e.g. adherence to requirements). Since this is part 1, it is worth 50 points for correctness, and 10 points for manual review of commenting and style. In this class we expect all submissions to compile, run, and pass at least some of the test cases; do not expect partial credit with regards to correctness. Example: if you submit to Gradescope and your score is a reported as a 0, then that's your correctness score. The only way to raise your correctness score is to re-submit.

In this project, your "graph.h" will be allowed **12 free submissions**. After 12 submissions, each additional submission will cost 1 point. Example: suppose you score 50 after 15 submissions, and you activate this submission for grading. Your autograder score will be 47: 50 – 3 extra submissions. Note that you cannot use another student's account to test your work; this is considered academic misconduct because you have given your code to another student for submission on their account.

By default, we grade your **last** submission. Gradescope keeps a complete submission history, so you can **activate** an earlier submission if you want us to grade a different one; this must be done before the due date. We assume *every* submission on your Gradescope account is your own work; do not submit someone else's work for any reason, otherwise it will be considered academic misconduct.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 10%. After 24 hours, no submissions will be accepted.

All work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them (e.g. your "iClicker teammates"), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml .