CS163 Lab #8 – Programming Graphs

Check-off each step as you proceed! Submit the code at the end of lab.

Adjacency Lists: The purpose of adjacency lists is to allow for direct access to each vertex (using an array of vertices) and all other vertices connected (with an edge list). We use a **combination of arrays and linked lists**. Consider in your design where the "visit flag" will be placed.

Getting Set Up – Have your answers checked for this first page before logging in

Group Activity

Level 1 - Introductory

_____Step 1. Assume that we will be storing a journal entry in each vertex. Create the **vertex** struct:

_____Step 2. **Now,** create the "node" struct for each edge. An edge needs information about where the current vertex is connected (via an index, vertex identifier, or pointer), and a next pointer.

Level 2 - Intermediate

_____Step 3. **Next,** the adjacency list is an array of vertices, where each has a head pointer and a pointer to a journal entry. Create that array, dynamically allocated of size N:

____

_____Step 4. What is the underlying type of each element?_________________________

_____Step 5. Indices range from index ____________ to index ____________ for the array

****** RECEIVE feedback from Lab assistants before continuing with the next steps***

Individual Coding a Graph

Coding: Now that we are set up, we will be working with an existing class implementing a graph ADT for a journal. The data structure is an **adjacency list** of journal entries. You have access to the .h class interface to see what data members and member functions are available. Your job will be to implement functions to experience manipulating the graph.

<u>Level 1 - Introductory</u>

- Login to **cs163lab.cs.pdx.edu** using your assigned login and password
- Change into the CS163/Lab8 directory **cd CS163/Lab8**
- Use a **linux editor** such as **vi, vim, or emacs** to type in a program.
- **Compile** and link to my object code on **linux**.

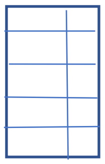
```
g++  *.cpp  *.o  -g  -Wall
```

- **Always** fix the **warnings** found by the -Wall. Typically, these mean that there are more fundamental issues with the structure of the code. They should not be ignored!

_____Step 6. **In the table class (cs163_graph.cpp),** implement these member functions.

_____a. Implement the **Constructor** table(int size);

Allocate the an array of vertices and set each Head pointer (to an edge list) to nullptr



_____b. Implement the **insert_vertex** function (insert into the array)
Since we are working with a pointer to a journal_entry, make sure to find the first available spot in the array of vertices and then perform a "new journal_entry" to store this new journal into the array of vertices

_____c. **Compile (g++ *.cpp *.o -g -Wall)** and **fix all warnings!**

_____d. **Evaluate** using a pointer diagram:

Proficiency Scale after Step #6:

_____Step 7. **In the table class (cs163_graph.cpp),** implement these member functions.

_____a. Implement the **find_location** function (to return the index for a matching journal title). Use the compare function from the journal class to determine if there is a match.

_____b. **Evaluate** using a pointer diagram:

Proficiency Scale after Step #7:

<u>Level 2 - Intermediate</u>

_____Step 8. Implement the **insert_edge** function (to insert connections between vertices) Given two key values, use the find location function in the journal class to determine how to connect up. Once that is determined, add a node in the edge list.

_____a. **Evaluate** using a pointer diagram:

_____b. **Compile (g++ *.cpp *.o -g -Wall)** and **fix all warnings!**

_____c. What cases did you test?______________________________

____

Proficiency Scale after Step #8:

_____Step 9. **Display the adjacent journal entries for a particular vertex**

_____a. Implement the function, passing in the desired vertex as an argument
_____b. Compile **(g++ *.cpp *.o -g -Wall)** and **fix all warnings!**
_____c. **Evaluate** using a pointer diagram:

_____d. **What cases did you test?**

____

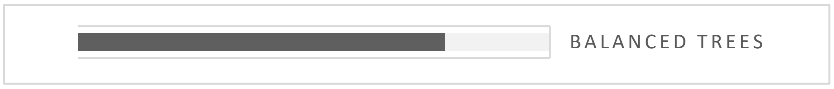
Proficiency Scale after Step #9:

Level 3 - Proficient

_____Step 10. **Challenge. Implement depth first traversal**

_____a. **First,** plan how to handle keeping track of visiting the vertices only once. DO NOT ADD DATA MEMBERS TO THE CLASS (this will not work with pre-compiled code!): ____________________________
_____b. Implement the approach. Use both iteration and recursion in your approach.
_____c. Compile **(g++ *.cpp *.o -g -Wall)** and **fix all warnings!**
_____d. **Evaluate** using a pointer diagram:

Proficiency Scale after Step #10:



_____Step 11. **Tar the files** for the lab by typing:

```
tar   -cvf    CS163_Lab8.tar   *.cpp   *.h
```

_____Step 12. **Submit** your program by typing **./submit** at the linux prompt