CSI 2372 – Assignment 4

Abdorrahim Bahrami

Linked lists in C++

As for the last assignment of this course, your task is to apply the concepts you learned so far about inheritance in C++. Make sure you have C++ installed, and you are familiar with the header files, and coding files.

Then, you should do the following programming task. Each programming task in this assignment is a design based on the subjects you learned during lectures.

In assignment 2, you designed a class for the concept of directed graphs. You can keep your implementation of graph or change it the way you want but there is a difference here. Your implementation was for a directed graph and you have to make sure you have an implementation for undirected graphs. You can keep the adjacency list implementation or you can change it to adjacency matrix. Note that connectivity_type method is removed because it is for directed graphs. You can keep BFS and DFS or remove them.

Class Graph	
Method	**Description**
Graph	The default constructor that initializes a graph of any default number of nodes without any edges
Graph	The user constructor, which initializes the graph with a given number of nodes without any edges
Graph	The copy constructor
~Graph	The destructor
add_edge	Add an edge between two given vertices (Direction from the first to the second vertex)
remove_edge	Remove an edge between two given vertices (Direction from the first to the second vertex)
edge_exist	Check if there is an edge between two given vertices
get_degree	returns the out degree of a vertex (The number of vertices it is connected to)
operator ++	In both forms for adding a vertex to the graph
operator --	In both forms for removing the last vertex from the graph
path_exist	Checks if a path exists between two given vertices
BFS	Returns a list of integers, which shows the breadth first search of the graph starting from a given node

DFS	Returns a list of integers, which shows the depth first search of the graph starting from a given node

operator <<	For printing the graph in the following format,
	The example from the graph in the picture,
	V = {1, 2, 3, 4, 5, 6}
	E =
	{
	1 => 2
	2 => 1, 3, 4
	3 => 3
	4 => 1, 3
	5 => 6
	6 => None
	}

Now, let's get to the main task. Your main task is to implement two classes, which are children of the graph. The first class is to represent the concept of a Forest. A forest is a graph, which does not have a cycle. See, what methods you need to override to make this happen. Then, inherit a Tree from the Forest graph. A Tree is a forest, which is connected. Note that this is not necessarily a binary tree. A tree needs many methods to be overridden. There is no user constructor for a tree. Only there is a default constructor, which creates a tree with one vertex and no edge. Operator ++ can only add a leaf to the tree and it has to add an edge too to keep it as a tree. This edge should connect the new vertex to the vertex that is last vertex added to the tree. Operator -- should remove the last vertex added to the tree, which is definitely a leaf. You need to add a method set_root to the tree class, which sets a vertex as the root of the tree. Also, operator << should print the tree in a tree style. For example, if the root is vertex 1, it is similar to the following, which means 1 is connected to 2, 5, and 8. 2 is connected to 3 and 4. 5 is connected to 6 and 6 is connected to 7.

1

--- 2

    --- 3

    --- 4

--- 5

    --- 6

        --- 7

--- 8

You can add any method you need.

The bonus part for this assignment is to inherit a class for representing bipartite graphs from the graph class.

Rules

You can do this assignment in a group of two to learn team work. Make sure you collaborate both in thinking and brainstorming about this problem and programming with your partner. Any similarity between your programs to other groups is considered plagiarism. Yes, if you do not like team work, you can do it alone. Do not use any code or program from the Internet because it is also considered plagiarism. See the university policies for plagiarism in the following link.

https://www2.uottawa.ca/about-us/provost

When you submit, on the top of every header file and cpp file, include your firstname, lastname, and student ID of yourself and your partner as follows.

// firstname lastname Student_ID

// firstname lastname Student_ID

Measures that we take to detect plagiarism

Teaching assistants have been instructed to report to the professor any suspicion of plagiarism they find when they mark assignments.

If plagiarism has been detected in any part or in the whole assignment, the professor will take appropriate measures. Recall that it is equally bad to copy a solution and to let someone else copy your solution.