ECE250 Project 5: Graphs

Do not proceed until you have a working solution for all prior projects!

In this project, you will use your binary trees to discover relationships between countries based on statistics calculated via their time series. We will represent these relationships in a graph.

Note: in this project, you may use any class in the STL. You may use structs, provided you justify the choice well. That is, something simple like a tuple may be a struct if you wish, but your major classes should still have protected members and therefore should be classes.

Understanding the Relational Graph

We are going to build a graph. Each node represents a country, and it will contain at least that country's name and code, although you may find it much easier to simply store country objects from previous projects there. Two nodes are connected via an edge if they share some relationship on at least one time series. Therefore, edges store information about how the countries are related.

A relationship is defined as a tuple, (Series_Code, Threshold, Relation). The Series_Code is the time series' code that we have been working with so far. The Threshold is a numerical threshold, and the Relation is exactly one of "greater", "less", or "equal". If two countries have an edge between them and that edge contains such a tuple, it means that both countries share the same relation with respect to the mean of the series with the given Series_Code. For example, both Uruguay and Mongolia have >80% of their land area listed for agricultural purposes. These two countries might share an edge with the tuple (AG.LND.AGRI.ZS,80,greater). For equality, we will use the standard 1E-3 tolerance we have been using so far. For simplicity, you may assume that "greater" and "less" both represent strict inequalities (that is, > and <, not >= or <=).

Because we are dealing with graphs, we allow only a single edge to exist between two countries. Therefore, if two countries share multiple relationships, we must add the relationships to the set of relationships that the edge represents. That is to say, each edge must store some kind of data structure that contains all of the relationships discovered so far between the two countries that edge connects. Store only unique relationships – do not store the exact same tuple more than once in the same edge. Tuples are equal if they contain *exactly* the same three components. Therefore, (AG.LND.AGRI.ZS,79,greater) and (AG.LND.AGRI.ZS,80,greater) are unique. Use == here, not tolerance, for equality on the threshold. We will not test such pedantic cases as "what happens if two numbers are entered and they are within floating point precision of each other" or other such nonsense, don't worry.

Program Design and Documentation

You must submit a brief design document for this project that outlines your class design. Guidelines for this document are on Learn.

Input/Output Requirements

The table below was designed so that all commands except LOAD are new and have not been used before. You must start from your previously working project 3 or 4 code. Although we will not be testing any commands from previous projects, the expectation is that your libraries work as specified in all previous projects encountered so far. For example, although we will not be testing Project 3's BUILD command, you are expected to discover the relationships stored in edges by building and searching binary trees. Your code may be subject to static analysis to determine whether or not you are using this functionality.

In ECE250, we will be testing your submissions via scripts written to run on Linux. To do this your solution must follow specific formatting requirements.

You must create a test program that contains your main program. This program must read commands from standard input and write to standard output.

The program must respond to the commands shown in Table 1. The outputs listed in the "Output" column must appear as shown. For instance, the first row has "success" as an output. This means that the string "success", written in lowercase, is expected, and if more input cases exist the output will continue on the next line. The program ends (and destructors are called) when standard input is empty.

Table 1: Testing Commands

Command	Parameters	Description	Output
LOAD	filename	This command works identically to that encountered in lab 3. You may assume it starts every file. You may further assume that once it is encountered it will never be encountered again in the given file, nor will new countries be added. This is done to simplify the project. You may use either your hash map from project 4 or some other data structure to store the country information.	**success**
INITIALIZE		Initialize the graph's nodes. After this command, your graph will be a totally disconnected graph consisting only of nodes (countries) and no edges. If a graph has already been built from the dataset, remove all edges. You may assume that	**success**

		this command will always be called at least once in a file, immediately after a call to LOAD. Therefore, there should always be a graph for subsequent commands.	
UPDATE_EDGES	Series_Code threshold relation	Update edges as described above. If an edge between two countries for which the relationship holds does not already exist, create it. If there are no countries for which this exact relationship holds, or if the edge set already contains all relationships represented here, this command should output failure.	**success** If at least one new relationship has been added **failure** If no new relationships have been added
ADJACENT	Country_Code	Output all countries by *name* that are adjacent to the given country by *code*. Do not print out the country given by code.	**Country1 Country2 … CountryN** A space-separated list of countries. Output the name exactly as in the dataset. The order does not matter. **none** If this country has no adjacent countries **failure** if this country is not in the graph
PATH	Country_Code1 Country_Code2	Determines if the two countries are connected by a path in the graph. You may assume that if this command is called, the two countries will exist in the graph, but they might not be connected.	**true** The two countries are connected by a path in the graph **false** there is no path between the two countries in the graph
RELATIONSHIPS	Country_Code1 Country_Code2	If the two countries are connected by an edge, list all relationships. You may assume that if this command is called, the two countries	**(Series_Code1 Threshold1 Relation1) (Series_Code2 Threshold2 Relation2)…** If these two countries

		will exist in the graph, but they might not share an edge.	share an edge,output a list of all relation tuples. Output the brackets, separate all tuples by spaces. Do not put commas into the brackets. The order doesn't matter. **none** If these two countries do not share an edge
EXIT		All files end with this command. This command produces no output.	

Runtime

Determine the worst-case runtime of your ADJACENT command, then prove it. In this case we are not specifying anything – it's your design. Note that if you used any STL libraries you may need to consider their runtimes in your analysis. How you find that information, and cite it, is up to you. Your answer should start with the sentence "The runtime for my ADJACENT command is O(<whatever your runtime is>)", then a proof should follow.

Submitting your Program

Once you have completed your solution and tested it comprehensively on your own computer or the project computers, produce your tar file according to the general project guidelines and submit to the dropbox on Learn.