# Build a Graph

## Objectives

* Build a matrix-based graph class.
* Understand the symmetric, antisymmetric, and reflexive properties of a graph.
* Understand how to create the transpose of a graph.

## Part 1

Build a weighted matrix class for an directed, weighted graph. Here's my header file weightedgraph.h.

#pragma once  
#include <stdexcept>  
#include <fstream>  
  
class WeightedGraph  
{  
private:  
 int mVertices; // Number of vertices  
 int\* mGraph; // 2D array of integers or a 1D array of integer.  
 // (Either way, this has mVertices \* mVertices.)  
 // My preference is for a 1D array.  
  
 void copy(const WeightedGraph&); // Copies a graph (requires a for loop!)  
 void checkForVertexInRange(int) const; // Checks for a vertex in range  
  
public:  
 WeightedGraph(int); // Builds a blank matrix that is m \* m  
 // Set mVertices = m  
 // Sets all elements of mGraph to 0.  
  
 WeightedGraph(const WeightedGraph&); // Copy Constructor  
 ~WeightedGraph(); // Destructor  
  
 WeightedGraph& operator=(const WeightedGraph&);  
 // Sets passed graph equal to this graph.  
 // 1. Deletes internal mGraph array  
 // 2. Copies passed graph to this graph  
  
 bool operator==(const WeightedGraph&) const;  
 // Returns true if passed graph's vertices  
 // equals this graph's vertices AND  
 // if every element in passed graph equals  
 // every element in this graph. False otherwise.  
 // Requires a loop.  
  
 int GetNumVertices() const; // Returns mVertices  
  
 void set(int, int, int); // Sets index i,j to weight  
 // First param: i  
 // Second param: j  
 // Third param: weight  
 // Check to make sure that i and j are a vertex in range  
  
 void set(int, int); // Calls "set(i,j,weight)" with weight=1  
 int get(int, int) const; // Returns mGraph at cell (i,j)  
 // Check to make sure that i and j are a vertex in range  
  
 // This method can be written with for-loops (or without if you are clever).  
 bool isSymmetric() const; // Returns true if every elements at (i,j) is identical to (j,i)  
 bool isAntisymmetric() const; // Returns true if element at (i,j) is 1 and element (j,i)  
 // is 0 (provided that i != j)  
  
 // Each of these methods require a for loop  
 WeightedGraph transpose() const; // Builds a new matrix which is transpose of this matrix  
 bool isReflexive() const; // Returns true if all elements along the diagonal are > 0  
 friend std::ostream& operator<<(std::ostream& os, const WeightedGraph& g); // Visualizes the graph  
};

# Part 2. Write the Program.cpp file.

Ask the user for a number of vertices and edges, then create a graph based on those dimensions. Ask the user to read an edge into the graph one-at-a-time with each weight being 1. Afterward, print the original graph, report if it is symmetric, reflexive, and print the transpose of the graph.

Review these sample input and output descriptions.

This matrix has all ones on the diagonal, so it's reflexive. It's symmetric because the transpose is equal to the original.

Number of vertices in our graph: 3  
Number of edges in our graph: 3  
Enter a starting vertex: 0  
Enter an ending vertex: 0  
Enter a starting vertex: 1  
Enter an ending vertex: 1  
Enter a starting vertex: 2  
Enter an ending vertex: 2  
  
Original Graph.  
 1 0 0  
 0 1 0  
 0 0 1  
Reflexive? Yes  
Symmetric? Yes  
Antisymmetric? Yes  
Transpose Graph.  
 1 0 0  
 0 1 0  
 0 0 1

This matrix has at least one zero on the diagonal, so it's not reflexive and it's not symmetric because the transpose it not equal to the original.

Number of vertices in our graph: 3  
Number of edges in our graph: 3  
Enter a starting vertex: 0  
Enter an ending vertex: 1  
Enter a starting vertex: 1  
Enter an ending vertex: 2  
Enter a starting vertex: 2  
Enter an ending vertex: 1  
  
Original Graph.  
 0 1 0  
 0 0 1  
 0 1 0  
Reflexive? No  
Symmetric? No  
Antisymmetric? No  
Transpose Graph.  
 0 0 0  
 1 0 1  
 0 1 0

This matrix is not reflexive (some values on the diagonal are zero) but it is symmetric.

Number of vertices in our graph: 3  
Number of edges in our graph: 7  
Enter a starting vertex: 0  
Enter an ending vertex: 1  
Enter a starting vertex: 0  
Enter an ending vertex: 2  
Enter a starting vertex: 1  
Enter an ending vertex: 0  
Enter a starting vertex: 1  
Enter an ending vertex: 1  
Enter a starting vertex: 1  
Enter an ending vertex: 2  
Enter a starting vertex: 2  
Enter an ending vertex: 0  
Enter a starting vertex: 2  
Enter an ending vertex: 1  
  
Original Graph.  
 0 1 1  
 1 1 1  
 1 1 0  
Reflexive? No  
Symmetric? Yes  
Antisymmetric? No  
Transpose Graph.  
 0 1 1  
 1 1 1  
 1 1 0

## Files

Here is a listing of all files. These files should be included in your final submission.

graph.h  
graph.cpp  
main.cpp