Exercise – Representing Graphs

Representing a graph:

Graphs aren't always represented by a network of nodes in a 2D or 3D environment, a graph is a general collection of data connected in some way. Here are a 2 examples:

- Friends connected on Facebook.
- Connections on Google +

Friends on Facebook

Each person on Facebook is represented as a "node" a connection between friends must be reciprocated. If Tom is friends with Bob, than Bob must also be friends with Tom.

Connections of Google +

Each person can add someone to their "circles" eg: associates, work, home, students etc… This connection does not need to be reciprocated, for example: Tom may have added Bob to his "work" circle, whereas bob is not forced to include tom in any of his circles.

With the above examples, what type of graph would these be?

a) Non Directed Graph
b) Directed Acyclic
c) Directed Cyclic

With the below pseudo code structures, which one would be best suited for representing the above graphs?

Method 1

You've got a list of node objects. Each node contains data, EG: a person name representing a Facebook or google plus user. Each node contains a list of edges. The edge contains some data and a reference to the node its connecting to. This type of structure would be best suited for a Directed graph, as two edges would need to be added to freely travel back and forth between nodes.

```
List of Nodes

Structure Node
    Data
    List of Edges

Structure Edge
    Connection to Node
    Data
```

Method 2

With this method, we have a single list of nodes and a single list of edges. This could be used to represent a non-directed graph.

```
List of Nodes
List of Edges

Structure Node
    Data

Structure Edge
    Node A
    Node B
    Data
```

Method 3

Again, we have a list of nodes and an adjacency matrix used to represent the connections.

```
List of Nodes
AdjacencyMatrix[n][n]

Structure Node
    Data
```

Of the above Methods, answer the following questions:

1. With each of the above methods, what collection types would you use for nodes and edges? std::vector, std::list, std::unorderedmap. What are the pro's and cons for each.

2. Which method above would be most efficient for adding and removing connections? O(1), O(n), O(n^2), O(log^n)? And how much additional memory may be required?

3. Which method would be most efficient for adding and removing nodes? O(1), O(n), O(n^2), O(log^n)? And how much additional memory may be required?

Graph Implementation

Open the template project provided. This project contains functions that will allow you to load textures and fonts

To this project, create a "Graph" class in a new *.h and *.cpp file. Provided here are some functions and empty structures. Implement the functions.

```
class Graph
{
public:
	class Node
	{ public:
		Vector2 data;
		// Add the approprate data
	};

	class Edge
	{ public: // Add the approprate data
	};
public:

	Graph();
	virtual ~Graph();

	Node *AddNode(Vector2 data);
	Node *FindNode(Vector2 data); // find node with value
	void RemoveNode(Node *pNode); // remove the given node

	void ConnectNodes(Node *nodeA, Node *nodeB,
			/* edge data (cost or something) */ );

protected:
	// List of Nodes
	// (which container type are you going to use)
};
```

Now that you've created a graph class, you should be able to represent a simple graph, add the following.

```
Graph *pGraph = new Graph();

Node *a = pGraph->AddNode( Vector2(100, 100) );
Node *b = pGraph->AddNode( Vector2(150, 100) );
Node *c = pGraph->AddNode( Vector2(200, 100) );
Node *d = pGraph->AddNode( Vector2(150, 150) );
Node *e = pGraph->AddNode( Vector2(100, 200) );
Node *f = pGraph->AddNode( Vector2(150, 200) );
Node *g = pGraph->AddNode( Vector2(200, 200) );
Node *h = pGraph->AddNode( Vector2(300, 150) );
Node *i = pGraph->AddNode( Vector2(250, 100) );
Node *j = pGraph->AddNode( Vector2(300, 100) );
Node *k = pGraph->AddNode( Vector2(350, 100) );

pGraph->ConnectNodes(a, d, 1);
pGraph->ConnectNodes(b, d, 1);
pGraph->ConnectNodes(c, d, 1);
pGraph->ConnectNodes(d, h, 1);
pGraph->ConnectNodes(d, e, 1);
pGraph->ConnectNodes(d, f, 1);
pGraph->ConnectNodes(d, g, 1);
pGraph->ConnectNodes(i, h, 1);
pGraph->ConnectNodes(j, h, 1);
pGraph->ConnectNodes(k, h, 1);

// make sure to delete pGraph somewhere sensible
```

In the provided framework, there are methods for rendering a texture and lines between textures.

Assets should be loaded in the constructor of the "Game1" class, for example:

```
Texture *texture = new Texture("./Images/myImage.png");
```

To draw the texture, use the sprite batch object.

```
m_spritebatch->Begin();

m_spritebatch->DrawSprite( texture, xPos, yPos, width, height, rotation );

// TODO: more drawing operations.

m_spritebatch->End();
```

- Add functionality to the graph class to allow you to iterate over nodes and edges.
  Use this added functionality to iterate over the nodes and edges to and draw each one.

- Using the Framework, add a node at the mouse position when you left mouse click. And remove a node when from the graph when you right mouse click on one of the nodes.

- If the newly added node is within 50 pixels of another node, connect them.