CIS 291 – Data Structures in C++ - Spring 2005
Week 15 Lab Exercise
Week 15 Lab Exercise due: by END of LAB on Tuesday, May 3rd

Purpose:
To become more familiar with depth-first-search and breadth-first-search of a graph

Week 14 Lab Exercise
Answer the following questions on paper; after coming up with your initial answers, you may discuss them with another student before getting your answers checked, if you wish. When you are ready, put your name on the "Next:" list on the board so that your answers can be checked.

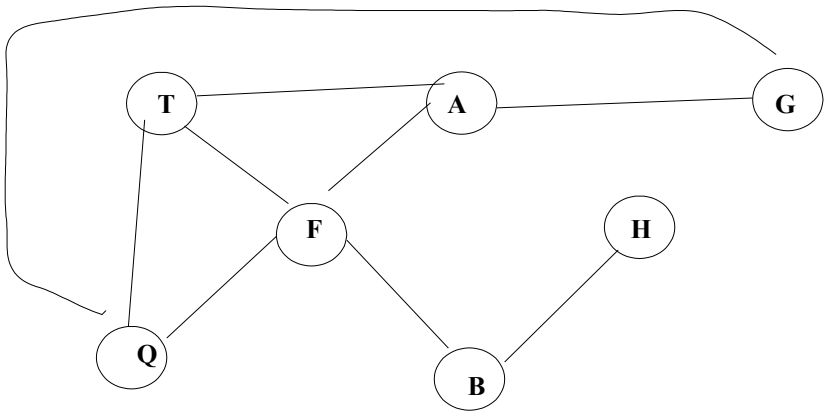
[Note: check **carefully**! There will be **points docked** for errors, this time, to encourage you to double-check carefully *before* getting your work checked.]

Consider the following **pseudocode** for **depth-first search:**

```
// PSEUDOCODE - RECURSIVE VERSION
// dfs
// Purpose: traverses a graph g beginning at vertex v by using a depth-first
//          search:
//          Recursive version
template <typename Item>
void dfs (graph<Item> g, Item v)
{
    // mark v as visited
    g.mark(v);
    cout << "visited: " << v << endl;

    for (each unvisited vertex u adjacent to v)
    {
        dfs(g, u);
    }
}
```

Assume further that you have the following graph g1:



1. Write this graph in **G = {V, E}** form.

_____

_____

_____

2. Assume that, for professorial sanity, when you have a choice of adjacent unvisited nodes, you choose the unvisited node in that is earliest in the alphabet.

Then, write what could be printed to the screen as a result of:

(a) **dfs(g1, 'G');**

_____

_____

_____

_____

_____

_____

_____

(b) **dfs(g1, 'F');**

_____

_____

_____

_____

_____

_____

_____

3. In graph **g1**, which nodes are **adjacent** to node **T**?

_____

Give an example of a **simple path** in graph **g1**.

_____

Give an example of a **simple cycle** in graph **g1**.

_____

Is graph **g1** connected? _____ Is graph **g1** complete? _____

(go to NEXT PAGE for #4!!!)

Now consider this pseudocode for breadth-first-search:

```
// PSEUDOCODE
// bfs
// Purpose: traverses a graph beginning at vertex v by using a breadth-first
//          search

template <typename Item>
void bfs(graph<Item> g, Item v)
{
    queue<Item> myQ;
    Item w, u;

    // add v to queue and mark it
    myQ.enqueue(v);
    g.mark(v);
    cout << "visited: " << v << endl;

    while (!myQ.empty())
    {
        w = myQ.dequeue( );

        // loop invariant: there is a path from vertex w to every vertex in
        // the queue myQ
        for (each unvisited vertex u adjacent to w)
        {
            // mark u as visited
            g.mark(u);
            cout << "visited: " << u << endl;
            myQ.enqueue(u);
        }
    }
}
```

4. Again, assume that, for professorial sanity, when you have a choice of adjacent unvisited nodes, you choose the unvisited node in that is earliest in the alphabet.

And now show what would be printed for the calls:

(a) **bfs(g1, 'G');**

_____

_____

_____

_____

_____

_____

_____

(b) **bfs(g1, 'F');**

_____

_____

_____

_____

_____

_____

_____