CIS 291 – Data Structures in C++ - Spring 2005
Homework #12
HW #12 due: THURSDAY, May 5th, BEGINNING of lecture

Purpose: Thinking/experience with graphs

How this will be turned in:
Use **~st10/291submit**, called from the directory on cs-server where the files you wish to submit are stored.

HOMEWORK #11:

1. Copy all of the files accompanying this assignment handout on the public course web page into your desired current working directory on cs-server.

You now have a (possibly-buggy, hopefully-not) implementation of a template class **graph**, implemented using adjacency lists (which can be done in a way rather reminiscent of a buckets-and-chaining hash table! But I digress...) You also have some other handy ADT's from previous assignments.

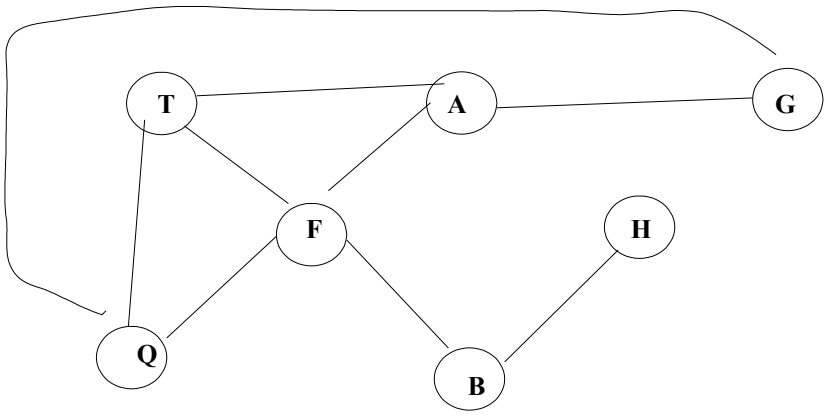
Show that you have everything you need for the graph implementation, at least, by:
(a) Adding a a **cout** statement containing your name to **try_graph.cpp**,

(b) compiling it, running it, and running it redirecting its output into **try_graph_out1**:

try_graph > try_graph_out1

...and submitting **try_graph_out1**.

2. I want to make sure you are at least a little familiar with the capabilities of this provided **graph.h** and **graph.template**. So, write a small program **graph_ex.cpp** that simply uses **graph.h** and **graph.template** to create the following graph, and to print it using **graph**'s **print_graph** member function:



[Of course, you should precede that **print_graph** call with a printout saying what you expect to see --- but, you may use the style used in **try_graph.cpp** for this. That is, it is sufficient to list the expected vertices and edges; they needn't be formatted/look exactly the same as print_graph depicts

them, nor must each edge be listed twice as print_graph does. The point is that the reader can tell if the expected graph and actual graph are the same in essence.]

Run:

graph_ex > graph_ex_out

...and submit your resulting **graph_ex.cpp** and **graph_ex_out .**

3. If I wanted you to *implement* bfs and dfs --- I couldn't, yet, with the provided **graph**. It has no way to **mark** nodes as visited.

SO --- we're going to **modify** it so that it does.

Modify **graph.h** and **graph.template** such that you:

* add a separate array of type **bool** and size **MAXIMUM** called **vertex_markings** that is initially all false --- it represents the current markings for all vertices in the graph.

* (note that get_vert_index(label) returns the index into array **vertices** for vertex label --- this value should also be label's index into **vertex_markings**.)

* add a member function **unmark_all** which re-marks all vertices as false;

* add a member function **mark** which takes an Item **vert** and sets the mark for the vertex **vert** to true.

* add a member function **get_mark** which takes an Item **vert** and returns the current marking for **vert**.

Add appropriate tests of **mark**, **unmark_all**, and **get_mark** to **try_graph.cpp**. Run:

try_graph > try_graph_out2

...and submit versions of your modified **graph.h**, **graph.template**, **try_graph.cpp**, and **try_graph_out2**

4. 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);
    }
}
```

Implement this pseudocode for **dfs** as a stand-alone template function.

[HINT: you'll find an implementation of **set** with the provided code --- that's because one of graph's methods returns a set of vertices. Which one? And how might that method be USEFUL in implementing **dfs** --- especially if you make use of set's internal iterator methods as well?]

In **babytest_dfs.cpp**, run your **dfs** function on the graph from homework problem #2 **two** times once starting at node G and once starting at node F, putting the output into **babytest_dfs_out** and submitting your **dfs.template**, **babytest_dfs.cpp** and **babytest_dfs_out**.

[yes, **babytest_dfs_out** should still print out actual and expected results --- here, though, just summarizing the order that you expect the nodes to be visited before each **dfs** call will suffice. The "expected" doesn't have to put 1 node per line with "visited:" as **dfs** will actually do. ASK ME if you are not sure what I mean by this.]

5. 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);
        }
    }
}
```

Implement this pseudocode for **bfs** as a stand-alone template function.

[HINT: the same comment regarding **set** applies here as well as it does in problem #4...]

In **babytest_bfs.cpp**, run your **bfs** function on the graph from homework problem #2 **two** times once starting at node G and once starting at node F, putting the output into **babytest_bfs_out** and submitting your **bfs.template**, **babytest_bfs.cpp** and **babytest_bfs_out**.

[yes, **babytest_bfs_out** should still print out actual and expected results --- here, though, just summarizing the order that you expect the nodes to be visited before each **bfs** call will suffice. The "expected" doesn't have to put 1 node per line with "visited:" as **bfs** will actually do. ASK ME if you are not sure what I mean by this.]

And, when you are satisfied with all of the above, submit them using **~st10/291submit** on cs-server:

try_graph_out1
graph_ex.cpp, graph_ex_out
graph.h, **graph.template, try_graph.cpp**, and **try_graph_out2**
dfs.template, **babytest_dfs.cpp, babytest_dfs_out**
bfs.template, babytest_bfs.cpp, babytest_bfs_out