University of Illinois at Chicago
CS 202: Data Structures and Discrete Mathematics II — Handout 9
Professor Robert H. Sloan — November 15, 2002

Program 5(a): The Graph class

(With minor corrections).

Due: Tuesday, November 26, 10:00 p.m.
Extension: You may have until 10:00 p.m. Sunday night.

Please create a class called `Graph`, which will provide an (edge-)weighted, directed graph class. You will also be providing the client with several iterators on this class. You will turin two files, Graph.cc and Graph.hh.

We will *not* make the Graph class a template class, but you will put two `typedef` statements at the top of the `Graph.hh` file as a sort of "poor man's template."

```
typedef string VertexName;
typedef float  Weight;
```

and in the rest of your class you will use `VertexName` and `Weight`.

Note that your `Graph` class may have only one map as its only data member that is not a simple, primitive, non-pointer type. That is, other than the map, the `Graph` class will have only data members that are ints or floats or chars, but not any vectors or lists or arrays or pointers.

However, your *other* classes, definitely including some of the iterator classes, and maybe including a Vertex class if you choose to use one, *are allowed* to have complicated data members.

1 Required public member functions

1.1 Not involving iterators

- `Graph()` A zero argument constructor.
- `insertEdge`. Takes 3 arguments: VertexName, VertexName, Weight. Return type void.

  The Weight argument should be defaulted to 1.0. (So we can ignore it if we want and have an unweighted graph. The fancier way to do this would be to have an unweighted graph class inherit from our weighted graph class.)
- `insertVertex`. Takes VertexName argument, returns bool, which is false if and only if that vertex is already in the graph.

- `edgeCount`, `vertexCount`. Both take zero arguments and return an unsigned int with the current number of edges or vertices in the graph.
- `edgeWeight`. Takes two VertexName arguments and returns a Weight.

  Returns -1.0 if edge is not in the graph.
- `bool empty()`. Tells if graph is empty.
- `getNeighbors`. Takes a VertexName, and returns a list of its successors, so return type is `list<VertexName>`.
- `bool containsEdge`. Takes two VertexName arguments and tells if that edge is in the graph.
- `bool containsVertex`. Takes a VertexName argument and tells if it is in the graph.

1.2 Graph member functions Involving iterators

You will have 3 iterator classes: `iterator`, `dfs_reach_iterator`, and d `dfs_all_iterator`.

The plain one is supposed to just use the map and give you all the vertices in arbitrary order when you walk through it.

The dfs one's are supposed to give you all the vertices in dfs order from a specified starting vertex. `dfs_reach` will give only those vertices that can be reached from the starting vertex; `dfs_all` will give all vertices eventually.

There are the same sorts of operators for each of the 3 kinds iterator; here I'll mostly just tell you about the vanilla iterator.

1.2.1 Graph member functions returning iterator

For each of those 3 iterator classes, you will have two *Graph* class member functions, a begin one and an end one.

- `iterator begin()`
  Returns an iterator positioned at an arbitrary beginning vertex.

  For the two dfs iterators, we need different names and an argument is taken:
- `dfs_reach_iterator dfs_reach_begin(const VertexName& start)`
  takes a VertexName as an argument and returns a `dfs_reach_iterator` positioned at that vertex.
-
- `dfs_all_iterator dfs_all_begin(const VertexName& start)`
  Same deal with begin and `dfs_all`.

- `iterator end()`
  gives an iterator that can be used in comparisons to terminate an iteration through this graph.
- `dfs_reach_iterator dfs_reach_end()`
- `dfs_all_iterator dfs_all_end()`

1.2.2 Iterator member functions

You will need to create a post-increment operator, `++` (postfix), to move your iterators forward, and a dereference operator, unary `*`, that will return a VertexName for any iterator other than end (for which it is an error).

(A slick `++` would return the pre-incremented iterator, but we will not use your return value so you can do whatever you like.)

You will also need to define the the equality and inequality operators, `operator==` and `operator!=`.

Finally, it will have to be possible for the user to declare an iterator, which *may* mean you need to write a constructor. In any event, the line

```
Graph::dfs_reach_iterator itr;
```

should compile (and so should the similar line for the other two flavors of iterators.)