COMP 322: Assignment 3 - Winter 2014

Due at 11:30pm, Mar 21th 2014

1 Introduction

In this assignment we would like to illustrate the use of classes in C++ by explicitly inplementing the ideas we built in the previous two assignments using inheritance. As you already know, the programming assignments for this course aim at exploiting the graph structure of wikipedia pages in making use of graph algorithms to discover interesting information about different wikipedia articles and the subjects they describe. As such, we would like to create two different classes: one for the abstract graph structure (which we will enhance in the last homework) and one for the wikipedia graph. We will focus on improving the modularity and the safaty of our code by exploiting inheritance, therefore we would like the class of Wikipedia graphs to be derived from the class of abstract graphs. We will use this strucutre in the last homework to concentrate separately on popular graph algorithms and on how these can be used to extract useful information about different wikipedia articles.

Important reference:

- `http://www.cplusplus.com/doc/tutorial/classes/`
- `http://www.cplusplus.com/doc/tutorial/templates/`
- `http://www.cplusplus.com/reference/stl/vector/`
- `http://www.cplusplus.com/reference/stl/list/`
- `http://www.cplusplus.com/reference/stl/set/`
- `http://www.cplusplus.com/reference/iostream/fstream/`

2 Assignment Requirements

Please add all your code in a file named `wikiClassesStudent.cpp`. Test your code using these and the other files provided: `wikiClasses.h`, `wikiClasses.cpp` and `simpleClassTest.cpp`. The TAs will test your code by compiling your submission using the provided `Makefile` and an additional `gradingClassTest.cpp` (not provided). NOTE: You do not have to use any code from the first two assignments. All you need is already in `wikiClasses.cpp`.

`wikiClasses.h` contains class definition both for `class Graph` (which implements the concept of an abstract graph strucutre) and `class WikiGraph` (which implements the concept of a concrete graph of Wikipedia articles). **Please read the class definitions before you read the remainder of the document**. Note that some of the constructors/functions are already implemented. Make sure you understand how they work and don't waste your time re-implementing these. You are responsible for the follwing:

- In `class Graph`:

  1. `Graph::Graph(list<Edge> lst, int num_vertices)`, which should perform the same task as `organizeList` in the previous homework, the main difference being that now we also have the secondary effect of builing an instance of the class `Graph` around the adjacency list (as opposed to simply returning an adjacency list which could be altered in inimaginable ways).
  2. `Graph::Graph(ifstream& in_file)`, which does not have the explicit list of edges as the previous constructor, but it has access to a file that contains the edges. This method is similar to `readGraphFromFile` from the previous homework.
  3. `~Graph::Graph()`, which is the destructor for this class. Make sure all memory is deallocated properly.
  4. `void Graph::save_to_output_file(ofstream& o_edges) const`, which is the same as `saveGraphToFile` in the previous assignment.
  5. `void Graph::push_node(list<Edge>& lst)`, which should create a new node in the already existing graph, and add the edges in `lst` to the adjacency list. Assume that the edges are valid: the origin of each edge is the ID of the new node, and the destination of each edge has an ID that is already in the graph. **Make sure** the edges are added in the adjacency list of the newly created node as well as the adjacency list of their destinations.

- In `class WikiGraph`:

  1. `WikiGraph::~WikiGraph()`, which is the destructor for this class. Make sure all memory is deallocated properly, and remember that the destructor of the super class is called by default. (We have provided code for all the constructors).
  2. `void WikiGraph::push_page(WikiPage& wp)`, which should add a `WikiPage` the the graph. Note that the input page `wp` does not have an `ID` defined yet. You have to set this `ID` as the next available ID in the graph (which is the size of the graph +1). Next, this function should call `push_node` from the super class. To do this, you first have to build the list of all edges that are related to `wp`. Other pages are related to `wp` if:
     - There exists a hyperlink from the `wp` to this other page in the `HTML` source file of the article of `wp`. You can use `allAssociations` (provided in `wikiClasses.cpp`) to obtain the set of all pages linked in a source `HTML` file.
     - The other page is already in the `WikiGraph` with which we are working. Use the `find` method on the `map<string, node> title_to_node` to determine whether or not a particular page exists in the graph.

     Last but not least, if an edge should be created between two pages, remember to compute the weight associated with the edge! This is done just as in `createEdge` of the previous assignment. We provide code for `countOccurences` in `wikiClasses.cpp`.

3. `void WikiGraph::save_to_output_files(ofstream& out_edges, ofstream& out_title_to_node)` This method should save the wiki graph info to two separate files: in one file we save the edges in the graph, using the similar method in the super class. The second output file contains the information of each wiki page of the current graph: all fields of the struct WikiPage on a separate line. The output should be compatible with the constructor +`WikiGraph(ifstream&, ifstream&)`. Please read its code, provided in `wikiClasses.cpp`, for more details.

- Other methods:

  1. `ostream& operator<< (ostream& o, Graph const& g)`, which should implement the `<<` operator for a `Graph g` in such a way that all the data of the graph is printed in the format required in the **first** homework. Please see the following article on more details on how to overwrite the `<<` operator

     `http://en.wikibooks.org/wiki/C%2B%2B_Programming/Operators/Operator_Overloading`

     Note that this method was purposely added as a `friend` of `Graph`, so that it can access that data to be printed.
  2. `ostream& operator<< (ostream& o, WikiGraph const& g)`, , which should implement the `<<` operator for a `WikiGraph g` in such a way that all the data of the graph is printed in the format required in the **second** homework.
  3. `WikiGraph::WikiPage& make_wiki_page(string path_to_html, string path_to_txt)`. This method should prepare a `struct` of type `WikiGraph::WikiPage` when only the path to the `HTML` and `TXT` source files are provied. You might find the method `getPageTitle` (provided in `wikiClasses.cpp`) quite useful. Note that you don't have to set the ID of the page, as it will be changed anyway when the object will be inserted in the wiki graph.