Object Oriented Programming 2009-2010 Spring Term
2nd Homework Assignment

Due On: 03.05.2010 23:00[1]

You are going to elaborate your graph class implementations and prepare simple graph traversal classes that can be used polymorphically on your graph implementations. An incomplete UML class diagram of the desired implementation can be found in Appendix A. Students are free to complete and implement the design regarding the restrictions and rules below.

Please keep in mind that you are free to use and modify first homework's solutions in implementing the second homework. You can find the solution under "Class Files" section in ninova.

a) Graph class
Your graph class should contain most of the functionality you have implemented in the first assignment. Your classes should represent graphs using its name and an array of nodes. Also each node in a graph should hold references to the adjacent nodes in the graph.

This time a graph can be created in one of the four ways.

- Having its content from a file
- An empty graph that doesn't contain any node or edge
- A graph as a copy of other

Graphs should contain all the operations and functionality in the first assignemnt. Additionaly, graph class should contain an acceptTraverse() method to let the traverser object operate on the graph.

b) Directed Graph Class – Tree Class
These two classes will be derived from Graph class and should expose all the operations of graphs except the following restrictions:

- A directed graph can only be united and intersected with another directed graph. Otherwise there should be a compiler error.
- A tree can only be united and intersected with another tree. Otherwise there should be a compiler error.
- Directed graph's and tree's addEdge and deleteEdge operations are quite different from Graph's. It only adds directional adjacents.
- A tree doesn't have addEdge or deleteEdge operations, instead it has addChild and deleteChild which also adds directional adjacents to nodes.
- Bidirectional graphs can only be traversed breadth first. Otherwise you should give an error message.
- Trees can both be traversed breadth first or depth first.

1 Due date is strict, e-mail submissions will be subject to penalty. Be cautious and submit a version before last minute rush.

c)Breadth First(BF) – Depth First(DF) Traversal classes

- Both these classes should implement Traversal interface using the algorithm in Appendix B and Appendix C[2]. An object of an abstract Traversal interface cannot be constructed!!
- BF and DF traversal classes should implement traverse method which operates on a graph and return the consecutive node names in a string.
- Traversal classes can use visitedNodes array to keep the visited nodes during the traversal.

A very preliminary test case is provided below. More concrete ones will be provided as the due time gets close.

```
int main(){

      A* aGraph= new Graph("aGraph.txt");
      B* bGraph= new Directed("bGraph.txt");
      C* cGraph= new Tree("cGraph.txt");

      aGraph->addEdge("A","F");          //F should be added to A's
                                               //adjacents and vice versa
      bGraph->addEdge("A","F");          //F node should be added to A's
                                               //adjacents

      cGraph->addEdge("A","F");          //Compiler error
      cGraph->addChild("A","F");         //F node should be added to A's
                                               //adjacents

      aGraph->unite(bGraph);             //Allowed
      bGraph->unite(cGraph);             //Compiler error
      cGraph->unite(aGraph);             //Compiler error

      Graph** someGraphs= new Graph*[3];
      someGraphs[0]=aGraph;
      someGraphs[1]=bGraph;
      someGraphs[2]=cGraph;

      for(int i=0;i<3;i++)
            someGraphs[i]->acceptTraverse(new BreadthFirst());    //Allowed

      someGraphs[0]->acceptTraverse(new DepthFirst());            //Allowed
      someGraphs[1]->acceptTraverse(new DepthFirst());            //Error Message
      someGraphs[2]->acceptTraverse(new DepthFirst());            //Allowed


      return 0;
}
```

Pay attention:

- You should extend the class definitions above, also you may perform minimal changes if required.
- **No logging on graph operations is needed this time! Only traversal outputs(as successing noded) and graph representations (graph name-nodes-edges) is enough.**
- The program must be written using object oriented principles. Try to use C++ built-in classes as

2 Examples taken from: http://www.ics.uci.edu/~eppstein/161/960215.html

much as possible.

- Academic dishonesty including but not limited to cheating, plagiarism, collaboration is unacceptable and subject to disciplinary actions. Any student found guilty will have -200 penalty points.

Evaluation Criteria:

- Programming(70 Points) :
  - Compilation: Your program must compile under g++. Programs that doesn't compile will be subject to a penalty of 20 points.
  - Usage of OO Principles(55 Points):
    - Class/object usage (15 Points)
    - Polymorphism usage (20 Points)
    - Inheritance usage (20 Points)
  - General programming(10 points): Your program shouldn't have any memory leaks, poor programming discipline .You should declare each class in a seperate (.h - .cpp) pair. Including Makefile is appreciated.
  - Program Comprehension/Comments(5 Points): You should at least include a few words for important methods.

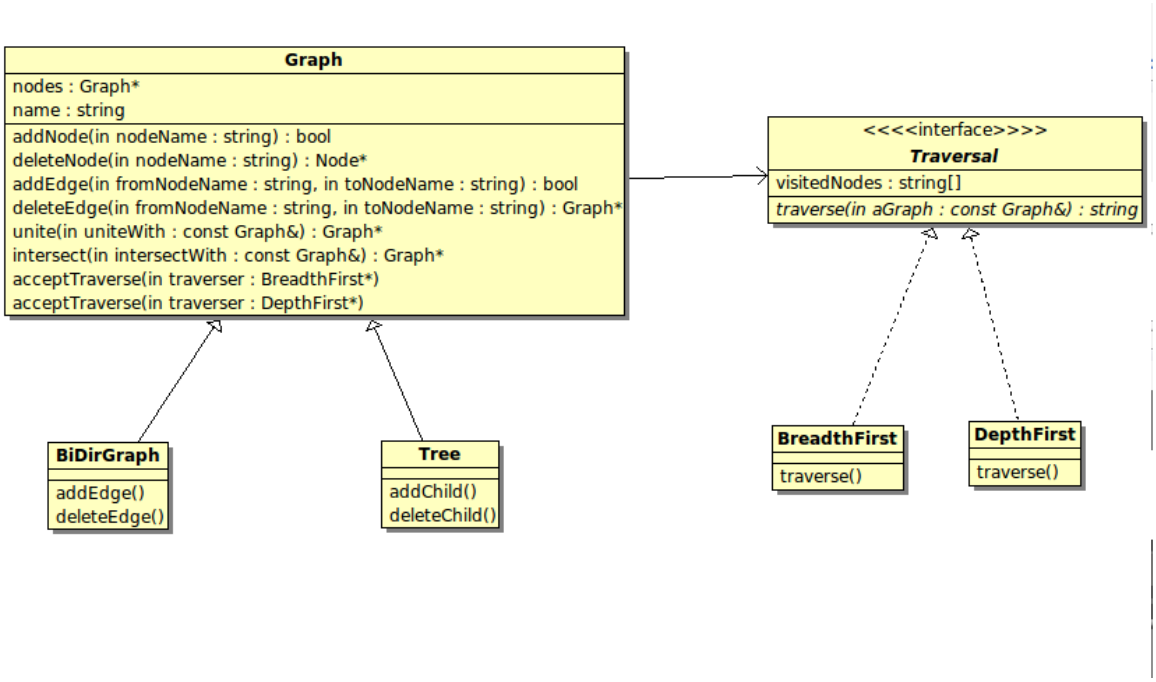
- Execution(20 Points) : Programs that produce no output or doesn't work at all receive 0-5 points; programs that work erroneously or by giving segmentation faults receive 5-15 points; programs that work without problem receive 15-20 points

- Report(10 Points) : **Programs without reports will not be evaluated.** A good report should include explanations regarding "general software structure" "program workflow" "important classes and data structures" "important methods" in a structured fashion. A good report shall explain the most with fewest words, remember that "A picture is worth a thousand words". A good report shall not include copy-paste code blocks or comments.

How to submit:

- You should archive your report(.pdf), program files(.cpp, .h, Makefile etc.) and input files(graph files for this assignment) in a .zip or .rar file named after your student number (e.g. 504052505.rar)
- Submit your work to ninova. Submit a working copy at least half an hour before the due time to avoid last minute rush accidents.
- For any questions-comments please contact only with R.A. Tolga Ovatman.

Appendix A – Incomplete UML Class diagram of the assignment



Appendix B – Breadth First Traversal Algorithm

```
unmark all vertices
choose some starting vertex x
mark x
list L = x
tree T = x
while L nonempty
choose some vertex v from front of list
visit v
for each unmarked neighbor w
        mark w
                add it to end of list
        add edge vw to T
```

You may also want to check wikipedia too : http://en.wikipedia.org/wiki/Breadth-first_search

Appendix C – Depth First Traversal Algorithm

```
dfs(vertex v)
{
        visit(v);
        for each neighbor w of v
             if w is unvisited
             {
                dfs(w);
                add edge vw to tree T
             }
}
```

You may also want to check wikipedia too : http://en.wikipedia.org/wiki/Depth-first_search