CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 30 April 3, 2013

Coding Graphs
DFS and BFS

Lecture Goals

1. Updates to HW 8
2. Writin' Code

Upcoming Homework Assignment

HW #8 **Due: Mon, Apr 8**

Graphs: DFS and BFS

The driver, header file, and cpp have all changed (slightly but importantly) since they were first released. Last change was this morning around 7:20am.

Due to my derpage, and to the too-short lecture on Monday, the **due date is pushed to Mon April 8, 6pm**.

Also, this involves more programming than usual, but it isn't all that brain-melty.

What Changed

The **Node** class now has:

Node predecessor;*

bool isAncestor(Node& other);

void setPredecessor(Node& other);

The Graphs-DFS-BFS document has been updated to talk about these.

What Changed

The **Graph** class now has a *getClock()* function. This is implemented, so you don't have to do anything except get the code.

Writing Member Functions

The class is declared in the graph.hpp header file, but the functions are implemented elsewhere (graph.cpp). All the members (variables and functions) are implicitly available whenever you implement a member function.

E.g. if you implement Node's 'clear' function, you can use the private members like *color*, *discovery_time*, and *predecessor*.

You also have a Node* called 'this' that refers to the Node instance you are working with.

graph.cpp:homeworks

```
public:
  Graph();
  ~Graph();
  vector<Node*> getNodes();
  vector<Edge*> getEdges();
  int getClock();
  void addNode(Node& n);
  void addEdge(Edge& e);
  void removeNode(Node& n);
  void removeEdge(Edge& e);
  bool isDirected();
  void setDirected(bool val);
  set<Edge*> getAdjacentEdges(Node& n);
  friend std::ostream &operator << (std::ostream& out, Graph graph);

  /**
   * Resets all nodes to have WHITE color, with -1 discovery and
   * finish times and rank. Resets all edges to type
   * UNDISCOVERED_EDGE. Resets the clock to 0.
   **/
  void clear();

  /**
   * OPTIONAL debugging method. Use this after every time you
-:---  graph.hpp:homeworks   10% (41,0)    Git-master  (C++/l Abbrev)
  return data;
}

void Node::setRank(int r) {
  rank = r;
}

void Node::clear() {
  color = WHITE;
  discovery_time = -1;
  completion_time = -1;
  rank = -1;
  predecessor = NULL;
}

void Node::setColor(int search_color, int time) {
  color = search_color;
  if (color == GRAY) {
    discovery_time = time;
  } else if (color == BLACK) {
    completion_time = time;
  }
}

-:---  graph.cpp:homeworks   75% (323,16)  Git:master  (C++/l Abbrev)
```

I usually have both the header file and implementation file visible whenever I'm writing C++ code.

The header (top) documents the classes and says what specifically to do.

The implementation (bottom) just fleshes it out.

Coding: Clear

We'll start with the Graph's 'clear' member function and progress from there.

This is actually a big deal, because initializing your graph (nodes and edges) correctly also touches on most of the Node and Edge member variables as well.

I'll put the results of today's hacking session on GitHub in **ROOT/code/graphs**.