Project-1: Weighted Directed Graph
S15 CSCI 332 - Design and Analysis of Algorithms

Phillip J. Curtiss, Assistant Professor
Computer Science Department, Montana Tech
Museum Building, Room 105

January 22, 2016

Project-1: Due 2016-02-5 by midnight

Purpose: This assignment consists of two parts - implement the weighted digraph class and the graph test driver program.

The WDiGraph class header file is supplied and contains basic graph ADT operations like create graph, destroy graph, copy graph, add edge, remove edge, check if an edge exists, get an edge weight, etc. Most of these operations are left to you to complete. Specifically, graph methods you need to finish are the copy constructor, destructor, check if edge exists, add edge, get edge weight, and remove edge. Specifications for each are provided in the interface header file. Make sure you review the documentation that specifies the input, outputs, and exceptions. A UML diagram is also provided.

A driver program to test the graph class has been started, which you need to be complete. You will add code so all menu options work properly and correctly handle all errors. A Makefile is supplied so you can easily compile your program. You will be expected to create makefiles for all your programs, so make sure you understand it.

Javadoc style documentation is used throughout and should be used to document any code you add. When done, update the version to 1.0 and add your name to the author tag (or add a second @author tag with your name).

The WDiGraph will be used in future programming assignments. You should be familiar with key programming techniques you are expected to use good programming techniques learned in CSCI 232 for file I/O, error handling, exceptions, JavaDoc style documentation, makefiles, etc.

You may create your own driver program, but please use the following numbers for the menu:

1. Print the size of the graph
2. Check if an edge is in the graph
3. Insert an edge into the graph
4. Delete an edge from the graph
5. Retrieve an edge weight
6. Display a copy of the graph
7. Quit the program

Objectives:

- Using UML Diagrams to Specify ADT
- Using existing C++ class methods

Table 1: UML Specification for the Weighted DiGraph ADT

```
+getNumVertices(): int const
+getNumEdges(): int
+getNumberOfNodes(): integer
+edgeExists(u: int, v: int): bool
+getEdgeWeight(in u: int, in v: int): WtType
+add(in u: int, in v: int, in wt: WyType)
+remove(in u: int, in v: int)
```

- Using Class Inheritence
- Understanidng Class Inheritence Relationships
- Create methods that implement the WDiGraph ADTs
- Manipulate Lists of Complex Data Types
- Manipulate Linked List Structures
- Work with Pointers in C++
- Pass objects by reference
- Use C++ source file standards and doxygen to generate documentation
- Modify and add features to the provided main program

Obtaining the Project Files: You will complete the project using your own user account on the department's linux system `katie.mtech.edu`. You should ssh into your account, and execute the command `mkdir -p csci332/proj1`. This will make the directory `proj1` inside the directory `csci332`, and also create the directory `csci332` if it does not yet exist - which it may from any previous project sessions. You should then change the current working directory to proj1 by executing the command `cd csci332/proj1`. You can test that you are in the correct current working directory by executing the command `pwd` which should print something like `/home/students/<username>/csci332/proj1`, where `<username>` is replaced with the username associated with your account. Lastly, execute the command `tar -xzvf ~curtiss/csci332/proj1.tgz` and this will expand the project files from the archive into your current working directory. You may consult `man tar` to learn about the archive utility and its command line options.

Building the Project: The project includes a basic `Makefile` you may use to generate the object files from source code files. If you wisht to provide a test driver for your project, make sure to edit the Makefile as needed. Consult the `Makefile` and understand the rules included and the dependencies and the rule sets that are used to generate the executable program. Use caution when updating the `Makefile` to ensure rule sets make sense.

Helpful Reminders: Study and pay close attention to the provided class(es) and methods. Understand their return types and use them in the code you author to provide robust code that can handle exceptions to inputs and boundary conditions. Look at all the code provided. Read the codes's comments and implement what is required where indicated. Make sure you are reusing code inside methods from inherited classes. Be cognizant of the *best practices we discussed in lecture and abide by good coding style - all of which will be factored into the assessment and grade for this project.* Be sure to review the UML diagram and the `Makefile` and understand how files are being generated and their dependencies.

Submission of Project: You have been provided a `Makefile` for this project that will help you not only build your project, but also submit the project in the correct format for assessment and grading. Toward the bottom of the provided `Makefile` you should see lines that look like:

```
# Rule to submit programming assignments to graders
# Make sure you modify the $(subj) $(msg) above and the list of attachment
# files in the following rule - each file needs to be preceeded with an
# -a flag as shown
subj     = "CSCI332 DDA - Proj1"
msg      = "Please review and grade my Project-3 Submission"
submit:  listing-A1.cpp listing-A2.cpp
         $(tar) $(USER)-proj1.tgz $?
         echo $(msg) | $(mail) -s $(subj) -a $(USER)-proj1.tar.gz $(addr)
```

Make sure you update the dependencies on the `submit:` line to ensure all the required files (source files) are included in the archive that gets created and then attached to your email for submission. You do not need to print out any of your program files - submitting them via email will date and time stamp them and we shall know they come from your account. If you submit multiple versions, we will use the latest version up to when the project is due.

Questions: If you have any questions, please do not hesitate to get in contact with either Phil Curtiss (`pjcurtiss@mtech.edu`) or Ross Moon (`rmoon@mtech.edu`) at your convenicne, or stop by during office hours, and/or avail yourself of the time in the MUS lab when Ross is available.

Project File Manifest:

WDiGraph Header

```
/** @file WDiGraph.h
  * Weighted Digraph Class with adjacency list implementation
  *  Vertex indices range from 0 to numVertices-1
  *  Multiple edges from one vertex to another vertex are not allowed
  *
  * @author Phillip J. Curtiss
  * @version 0.95
  * @date 2016-01-22
  *******************************************************/

#ifndef WDGRAPH_H
#define WDGRAPH_H

#include "GraphInterface.h"

/** Weighted DiGraph - matrix implementation using integer vertex labels*/
class WDiGraph : public GraphInterface<int>
{
public:
   /** Default Constructor (default size is 10 vertices)
     * @pre The graph is empty.
     * @post The graph is initialized to hold n vertices. */
   WDiGraph(int n=10);

   /** Copy Constructor
     * @param graph - the graph to copy. */
   WDiGraph(const WDiGraph& graph);

   /** Destructor. */
   virtual ~WDiGraph();

   /** Determines the number of vertices in the graph.
    * @return The number of vertices in the graph. */
   int getNumVertices() const;

   /** Determines the number of edges in the graph.
    * @return The number of edges in the graph. */
   int getNumEdges() const;

   bool add(int start, int end, int edgeweight)
      throw (VertexOutOfRangeException);

   bool remove(int start, int end)
      throw (VertexOutOfRangeException);

   bool edgeExists(int start, int end) const
      throw (VertexOutOfRangeException);

   int getEdgeWeight(int start, int end) const
      throw (VertexOutOfRangeException);

   /** Assignment operator overload for deep copy of rhs graph
    * @pre The lhs and rhs graphs exist
    * @post The lhs graph is deallocated and now contains the rhs graph*/
   WDiGraph& operator=(const WDiGraph& rhs);

private:
   int* data;     /** Dynamicly allocated contiguous data block*/
   int* *matrix; /** 2D matrix that points to rows in data block*/
   int numOfVertices;  /** Number of vertices in the graph. */
   int numOfEdges;     /** Number of edges in the graph. */
}; //end WDiGraph

#endif
```

WDiGraph Implementation

```
/** @file WDiGraph.cpp
 * @author Phillip J. Curtiss
 * @version 0.95
 * @date 2016-01-22
*****************************************************/

#include "WDiGraph.h"

WDiGraph::WDiGraph(int n) : numOfVertices(n), numOfEdges(0)
{
  //Allocate 2D matrix to contain weighted graph
```

```
   /* assume new always succeeds */
   data = new int[numOfVertices*numOfVertices];
   matrix = new int*[numOfVertices];
   for (int i = 0; i < numOfVertices; ++i)
   {
         matrix[i] = &data[i * numOfVertices];
         for(int j = 0; j<numOfVertices; ++j)
         {
           matrix[i][j] = NOEDGE;
         }
   }
} //end constuctor

WDiGraph::WDiGraph(const WDiGraph& graph)
{
  //Student needs to implement for deep copy
}

WDiGraph::~WDiGraph()
{
  //Student needs to implement
}

bool WDiGraph::edgeExists(int start, int end) const
     throw (VertexOutOfRangeException)
{
  bool found = false;

  //Check if start and end are in range
  if(start < 0 || end < 0 || start >= numOfVertices || end >= numOfVertices){
    throw VertexOutOfRangeException("Invalid Vertex number");
  }

  //Student needs to implement

  return found;
}

int WDiGraph::getNumVertices() const
{
   return numOfVertices;
}  // end getNumVertices

int WDiGraph::getNumEdges() const
{
   return numOfEdges;
}  // end getNumEdges

int WDiGraph::getEdgeWeight(int start, int end) const
   throw (VertexOutOfRangeException)
{
  int weight = 0;

  //Student needs to implement
  return weight;
} // end getWeight


bool WDiGraph::add(int start, int end, int w)
  throw (VertexOutOfRangeException)
{
  //Student needs to implement
  return false;
}

bool WDiGraph::remove(int start, int end)
  throw (VertexOutOfRangeException)

{
  //Student needs to implement
  return false;
}

WDiGraph& WDiGraph::operator=(const WDiGraph& rhs)
{
  //Student needs to implement
  return *this;
}
```

Graph Header

```
//  Created by Frank M. Carrano and Tim Henry.
//  Copyright (c) 2013 __Pearson Education__. All rights reserved.

/** @file Graph.h */

#ifndef _GRAPH
#define _GRAPH

#include "GraphInterface.h"
#include "Vertex.h"

// NOTE <<<<<<<
// Replace SomeADT in the following statements with the name
// of the ADT that stores the vertices.
// Replace SomeADTIterator with the name of the iterator class
// for these vertices.
// These choices can be different from those used in the class Vertex.

#include "SomeADT.h"          // ADT for Vertex List
#include "SomeADTIterator.h" // Iterator for Vertex List

template<class LabelType>
class Graph : public GraphInterface<LabelType>
{
private:
   int numberOfVertices;
   int numberOfEdges;

   SomeADT<LabelType, Vertex<LabelType>* > vertices;
   SomeADTIterator<LabelType, Vertex<LabelType>* > currentVertex;

   // These are optional helper methods that may be useful
   void depthFirstTraversalHelper(Vertex<LabelType>* startVertex,
                                  void visit(LabelType&));

   void breadthFirstTraversalHelper(Vertex<LabelType>* startVertex,
                                    void visit(LabelType&));

   Vertex<LabelType>* findOrCreateVertex(const LabelType& vertexLabel);

public:
   Graph();
   int getNumVertices() const;
   int getNumEdges() const;
   bool add(LabelType start, LabelType end, int edgeWeight = 0);

   // For remove to return true all three of the following must be true:
   // 1) start and end vertices exist
   // 2) Edge start->end is successfully disconnected
   // 3) Edge end->start successfully disconnected
   // Then, if those are successful and either start or end no longer
   // has neighbors, the vertex is removed from the graph
   bool remove(LabelType start, LabelType end);

   int getEdgeWeight(LabelType start, LabelType end) const;
   void depthFirstTraversal(LabelType startLabel, void visit(LabelType&));
   void breadthFirstTraversal(LabelType startLabel, void visit(LabelType&));
}; // end Graph

#include "Graph.cpp"
#endif
```

Graph Interface

```
/** An interface for the ADT undirected, connected graph.
 Carrano and Henry Code  Listing 20-1.
 @file GraphInterface.h */

#ifndef _GRAPH_INTERFACE
#define _GRAPH_INTERFACE
#include "VertexOutOfRangeException.h"
#include "NotFoundException.h"

const int NOEDGE = 0;

template<class LabelType>
class GraphInterface
{
public:
   /** Gets the number of vertices in this graph.
```

```
      @pre    None.
      @return  The number of vertices in the graph. */
   virtual int getNumVertices() const = 0;

   /** Gets the number of edges in this graph.
      @pre    None.
      @return  The number of edges in the graph. */
   virtual int getNumEdges() const = 0;

   /** Creates an undirected edge in this graph between two vertices
        that have the given labels. If such vertices do not exist, creates
        them and adds them to the graph before creating the edge.
      @param start  A label for the first vertex.
      @param end  A label for the second vertex.
      @param edgeWeight  The integer weight of the edge.
      @throw VertexOutOfRangeException if either vertex is not in range.
      @return  True if the edge is created, or false otherwise. */
   virtual bool add(LabelType start, LabelType end, int edgeWeight) = 0;

   /** Removes an edge from this graph. If a vertex has no other edges,
        it is removed from the graph since this is a connected graph.
      @pre   None.
      @param start  A label for the first vertex.
      @param end  A label for the second vertex.
      @throw VertexOutOfRangeException if either vertex is not in range.
      @return  True if the edge is removed, or false otherwise. */
   virtual bool remove(LabelType start, LabelType end) = 0;

   /** Determine if edge is in graph
      @param start - edge start vertex
      @param end - edge end vertex
      @throw VertexOutOfRangeException if either vertex is not in range.
      @return true if edge (start, end) is in the graph */
   virtual bool edgeExists(LabelType start, LabelType end) const = 0;

   /** Gets the weight of an edge in this graph.
      @throw VertexOutOfRangeException if either vertex is not in range.
      @return  The weight of the specified edge.
        If no such edge exists, returns a negative integer. */
   virtual int getEdgeWeight(LabelType start, LabelType end) const = 0;

   /** Performs a depth-first search of this graph beginning at the given
        vertex and calls a given function once for each vertex visited.
      @param start  A label for the first vertex.
      @param visit  A client-defined function that performs an operation on
        or with each visited vertex. */
//    virtual void depthFirstTraversal(LabelType start, void visit(LabelType&)) = 0;

   /** Performs a breadth-first search of this graph beginning at the given
        vertex and calls a given function once for each vertex visited.
      @param start  A label for the first vertex.
      @param visit  A client-defined function that performs an operation on
        or with each visited vertex. */
//    virtual void breadthFirstTraversal(LabelType start, void visit(LabelType&)) = 0;
```

```
}; // end GraphInterface
#endif
```

Custom Exception Classes

```
/** @file NotFoundException.h */

#ifndef _NOT_FOUND_EXCEPTION
#define _NOT_FOUND_EXCEPTION

#include <stdexcept>
#include <string>

using namespace std;

class NotFoundException : public logic_error
{
public:
   NotFoundException(const string& message = "");
}; // end NotFoundException
#endif

/** @file NotFoundException.cpp */
#include "NotFoundException.h"

NotFoundException::NotFoundException(const string& message)
         : logic_error("Not Found Exception: " + message)
{
}  // end constructor

// End of implementation file.

/** @file VertexOutOfRangeException.h
  */

#ifndef VERTEX_EXCEPTS_H
#define VERTEX_EXCEPTS_H

#include <stdexcept>
#include <string>

using namespace std;

/** Vertex Out of Range Exception class
  * Exception class for vertices that are not in the graph.
  */
class VertexOutOfRangeException: public std::out_of_range
{
public:
  /**
   * Creates exception initialized with a message.
   * @param message exception message string
   */
   VertexOutOfRangeException(const std::string & message = "")
```

```
                            : std::out_of_range(message.c_str())
   { }
}; // end VertexOutOfRangeException

#endif
```

IO Functions

```
/** @file io_functions.h
 *
 * Contains C++ functions for printing menus, reading input values, etc.
 * @author Braun
 * @date 9/16/13
 */

#ifndef IO_FUNCTIONS_H
#define IO_FUNCTIONS_H
#include <iostream>
#include <sstream>

using namespace std;

/**Prints main program menu
 * @pre  none
 * @post none
 */
void printMenu();

/**Reads and returns an integer
 * @pre  User has been prompted to enter an integer
 * @post none
 * @return integer value
 */
int getInt();

/**
  * Prompts for a list item and returns it to caller
  * @return list item
  * @remarks Accepts strings with spaces
  */
string getItem();

#endif
```

```
/** @file io_functions.cpp
 *
 * Contains C++ functions for printing menus, reading input values, etc.
 * @author Braun
 * @date 1/12/14
 */

#include "io_functions.h"
```

```
using namespace std;

/**Prints main program menu
 * @pre  none
 * @post none
 */
void printMenu(){
  cout << "1) Print the size of graph" << endl;
  cout << "2) Check if an edge is in the graph" << endl;
  cout << "3) Insert an edge into the graph" << endl;
  cout << "4) Delete an edge from the graph" << endl;
  cout << "5) Retrieve an edge weight" << endl;
  cout << "6) Display the graph" << endl;
  cout << "7) Quit the program" << endl;
}

/**Reads, validates, and returns a positive integer
 * @pre  User has been prompted to enter an integer
 * @post none
 * @return nonnegative integer value
 */
int getInt(){
  int intValue = 0;
  string inputStr;
  do{
    getline(cin, inputStr);
    stringstream(inputStr) >> intValue;
    if(intValue < 0) cout << "ERROR - please enter a nonnegative integer> ";
  } while(intValue < 0);
  return intValue;
}
```

Makefile

```
#
# Makefile for Generating C++ executables
#
# S16 CSCI 332 - Design and Analysis of Algorithms
# Phillip J. Curtiss, Associate Professor
# Computer Science Department, Montana Tech
# Museum Buildings, Room 105
#
# Project-1: Weighted Directed Graph
# Date Assigned: 2016-01-22
# Date Due: 2016-02-05 by Midnight

# Define Macros related to printing and submitting project
a2ps    = a2ps -T 2
mail    = mail
addr    = pcurtiss@mtech.edu #rmoon@mtech.edu
tar             = tar -cvzf

# Define Macros related to object code and program generation
```

```
DIA = dia2code
C++  = g++ -std=c++11
CFLAGS = -g -Wall -Werror
LD = g++
LDFLAGS =
LIBS =
SRCS = Driver.cpp WDiGraph.cpp WDiGraph.h \
           VertexOutOfRangeException.h NotFoundException.cpp NotFoundException.h \
           Graph.h GraphInterface.h \
           io_functions.cpp io_functions.h
OBJS = Driver.o WDiGraph.o io_functions.o
EXEC = Driver

# Provide Make with additionally known suffixes
.SUFFIXES:          .dia

# Default rule to make if now target specified on command line
all:                $(EXEC)

# Dependency Rules
io_functions.o: io_functions.h io_functions.cpp
WDiGraph.o:         VertexOutOfRangeException.h WDiGraph.h WDiGraph.cpp
Driver.o:           Driver.cpp WDiGraph.h

#Rules to create target file Driver
# If any files on line with colon are modified, then recompile the object file
# Target Rules for Make
$(EXEC):            $(OBJS)
                    $(LD) $(LDFLAGS) -o $(EXEC) $(OBJS) $(LIBS)

clean:
                    rm -f $(OBJS) $(EXEC)

subj =   "CSCI332 DAA - Proj1"
msg =    "Please review and grade my Project-1 Submission"
submit: $(SRCS)
        $(tar) $(USER)-proj1.tgz $?
        echo $(msg) | $(mail) -s $(subj) -a $(USER)-proj1.tgz $(addr)

print:  $(SRCS)
        $(a2ps) $?

# Implcit Rules based on file extension dependencies
.cpp.o:
        $(C++) $(CFLAGS) -c $<

.dia.h:
        $(DIA) -t cpp $<

.dia.cpp:
        $(DIA) -t cpp $<
```