**CS311 Yoshii - HW6 Intro to Graphs (based on Week 10 - Week 12)**

**DUE: Week 13 Tuesday at the beginning of class**

**Total: 38 points Your score is:**

**Your Name: Jer Cherng Law**

**Date Turned In: Nov 18 2014**

**-------------------------------------------------------**

**Purpose: To form fundamental understanding about graphs**

**--------------------------------------------------------**

**Review Questions from Notes-10B:[8pts] Your score:**

A **complete graph** has every vertex connected **directly** to every other vertex.

**\*Inter2\* Draw an example with 3 vertices.**

**a. undirected graph**

B

C

A

**b. directed graph**

A

B

C

A **connected graph** has to have a **path** between every pair of vertices.

**\*Inter3\* Draw a smallest connected graph with 3 vertices.**

**a. undirected graph**

A

B

C

**b. directed graph**

C

A

B

**\*Inter4\* How many edges does a tree of N nodes have? Why?**

**N-1 Because a tree is a connected graph there must be an edge connecting every other vertex. Everybody except the root has a root going in.**

**\*Inter6\* If my directed graph has 200 cities, and I want to make sure there is a direct flight from any city to any other city, how many flights are needed?**

**200\*199 = 39800**

**\*Inter7\* If my undirected graph has 200 cities, and I want to make sure I can drive from any city to any other city, how many road are needed? 199**

For an adjacency list representation:

**\*Inter12\* If you have N vertices and M edges,**

* **how many list heads (array slots are you going to need?) are there in G?? N**
* **how many linked list nodes are required altogether:**

**for directed G? Why? M**

**for undirected G? Why? 2M**

**Review Questions from Notes-11A:[10pts] Your score:**

**Depth First Traversal from the notes ends like this:**

Pop I. [F G top]

I has not been marked yet.

Mark I.

Adjacent vertices are A G and E.

push E push G push A [F G E G A top]

**\*Inter2\* Complete this trace from this point using exactly the same wording and the same format until the stack becomes empty.**

Pop A. [F G E G top]

A has been marked.

Pop G. [F G E top]

G has been marked.

Pop E. [F G top]

E has not been marked yet.

Mark E.

E has no adjacent vertices—nothing to push to stack.

Pop G. [F top]

G has been marked.

Pop F. [- top]

F has no been marked yet.

Mark F.

Adjacent vertices are I and E.

push E push I. [E I top]

Pop I. [E top]

I has been marked.

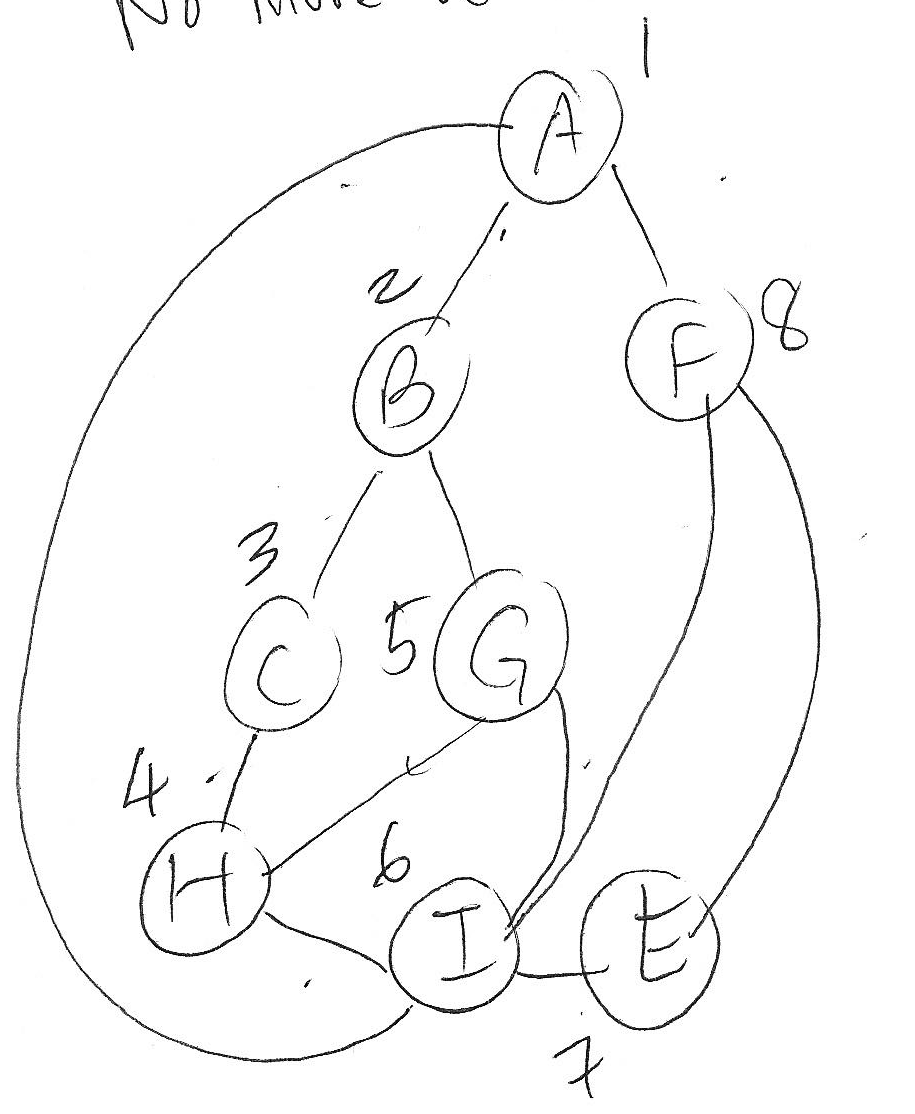
Pop E. [- top]

E has been marked.

No more vertices, done!

**\*Inter3\* On the graph you drew in Inter1, number the vertices**

**in the order they are marked by DFS**

**(i.e. A is 1, B is 2, etc.)**

**Program Graph Implementation:[20pts] Your score:**

**dgraph.h, dgraph.C (see HW6help)**

Create a **directed graph** **class** which has the following data member:

**I provided for you HW3 files in case your HW3 is not working.**

**Gtable[20]**(an array) which contains the following (struct)in each slot:

**Struct Gvertex: (this is declared outside the class)**

- a vertex name (char)

- the mark/visit number (int)

- the out degree (int)

- a linked list object for adjacent vertices**(from HW3P3 slist)**

You may use other data members such as size (how many slots are used) as necessary.

**And the following methods/member functions:**

dgraph Constructor - initializes the table entries

[ Make sure the names are initialized to be ‘ ‘ and visit number is 0]

dgraph Destructor - destroys the table

[ Does this call the list destructor automatically? If not, you have to destroy the lists. Test and see. ]

displayGraph() – displays the table content in a very readable format

But make sure you do not display unused slots

fillTable() - reads the input file **table.txt** to fill the table

Open and close the input file table.txt in here

int findOutDegree(char) – returns the out degree of the vertex

whose name is given as an argument

slist findAdjacency(char) – returns the linked list of

adjacent vertices of

the vertex whose name is given as an argument

[ This one calls your HW3P3 copy constructor automatically because a list is being returned.]

**Note that the mark/visit number is not being used yet by these functions. It will be used in the next HW.**

**Note that the linked list of adjacent vertices is of type slist and thus, you can use any of the slist member functions on it.**

**table.txt** should have the following format:

Each line is

name out-degree a-list-of-its-adjacent-vertices-separated-by-blanks

e.g.

A 2 B F

**I have provided you with the input file based on Notes-11A.doc**.

**Write a client program which will:**

1. fillTable()
2. displayGraph()
3. findOutDegree(char) for various vertices in the graph (a loop)
   1. the user will specify which vertex
   2. displays the returned result
   3. gtable[i].name == ‘A’; return gtable[i].out-degree
4. findAdjacency(char) for various vertices in the graph (a loop)
   1. the user will specify which vertex
   2. displays the returned list (use HW3P3 function)
   3. gtable[i].adj == ‘A’; return gtable[i].adj

**Q) The state of the program statement [2pts]**

* **Does your program compile without errors? No errors**
* **List any bugs you are aware of, or state “No bugs”: No Bugs**

**Submit: Stapled in this order**

1. **This assignment sheet with your answers**
2. **dgraph.h**
3. **dgraph.C**
4. **client.C**
5. **table.txt that I have provided**
6. **the results of thorough test cases (screen dump or script)**

**Whether working or not, test result must include the lines for compiling your files or we will not grade our program i.e. 0 points for the program.**

**Did you check your comments and style against CS311 How To Comment.doc??**

**Here is the screen dump. Highlighted lines are not output, they’re post-edit comments.**

Jers-MacBook-Pro:Directed Graphs jerchernglaw$ g++ -o a main.cpp dgraph.cpp llist.cpp slist.cpp (this is the compile line. It produces executable file named “a”)

Jers-MacBook-Pro:Directed Graphs jerchernglaw$ ./a

A

2

B F

B

2

C G

C

1

H

E

0

Empty linked list

F

2

I E

G

0

Empty linked list

H

2

G I

I

3

A G E

Enter vertex letter to find outDegree or 'q' to quit: A

The degree at 'A' is: 2

Enter vertex letter to find outDegree or 'q' to quit: a

The degree at 'a' is: Error: Vertex does not exist.

1 <— this is the error code returned

Enter vertex letter to find outDegree or 'q' to quit: B

The degree at 'B' is: 2

Enter vertex letter to find outDegree or 'q' to quit: C

The degree at 'C' is: 1

Enter vertex letter to find outDegree or 'q' to quit: E

The degree at 'E' is: 0

Enter vertex letter to find outDegree or 'q' to quit: F

The degree at 'F' is: 2

Enter vertex letter to find outDegree or 'q' to quit: G

The degree at 'G' is: 0

Enter vertex letter to find outDegree or 'q' to quit: H

The degree at 'H' is: 2

Enter vertex letter to find outDegree or 'q' to quit: I

The degree at 'I' is: 3

Enter vertex letter to find outDegree or 'q' to quit: z

The degree at 'z' is: Error: Vertex does not exist.

1 (testing for vertices that do not exist.)

Enter vertex letter to find outDegree or 'q' to quit: 3

The degree at '3' is: Error: Vertex does not exist.

1 (testing for vertices that do not exist.)

Enter vertex letter to find outDegree or 'q' to quit: 2

The degree at '2' is: Error: Vertex does not exist.

1 (testing for vertices that do not exist.)

Enter vertex letter to find outDegree or 'q' to quit: 0

The degree at '0' is: Error: Vertex does not exist.

1 (testing for vertices that do not exist.)

Enter vertex letter to find outDegree or 'q' to quit: -1

The degree at '-' is: Error: Vertex does not exist.

1 (reads as char process ‘-‘ and ‘1’ separately. Does not crash)

Enter vertex letter to find outDegree or 'q' to quit: The degree at '1' is: Error: Vertex does not exist.

1 (testing for vertices that do not exist.)

Enter vertex letter to find outDegree or 'q' to quit: q

Enter vertex letter to find adjacency list or 'q' to quit: A

The adjacency list at 'A' is:

Calling the linked list destructor.

B F

Enter vertex letter to find adjacency list or 'q' to quit: B

The adjacency list at 'B' is:

Calling the linked list destructor.

C G

Enter vertex letter to find adjacency list or 'q' to quit: C

The adjacency list at 'C' is:

Calling the linked list destructor.

H

Enter vertex letter to find adjacency list or 'q' to quit: E

The adjacency list at 'E' is:

Calling the linked list destructor.

Empty linked list

Enter vertex letter to find adjacency list or 'q' to quit: F

The adjacency list at 'F' is:

Calling the linked list destructor.

I E

Enter vertex letter to find adjacency list or 'q' to quit: G

The adjacency list at 'G' is:

Calling the linked list destructor.

Empty linked list

Enter vertex letter to find adjacency list or 'q' to quit: H

The adjacency list at 'H' is:

Calling the linked list destructor.

G I

Enter vertex letter to find adjacency list or 'q' to quit: I

The adjacency list at 'I' is:

Calling the linked list destructor.

A G E

Enter vertex letter to find adjacency list or 'q' to quit: i

The adjacency list at 'i' is:

Error: Vertex does not exist.

Calling the linked list destructor.

Calling the linked list destructor.

Empty linked list

(testing for vertices that do not exist.)

Enter vertex letter to find adjacency list or 'q' to quit: 1

The adjacency list at '1' is:

Error: Vertex does not exist.

Calling the linked list destructor.

Calling the linked list destructor.

Empty linked list

(testing for vertices that do not exist.)

Enter vertex letter to find adjacency list or 'q' to quit: 3

The adjacency list at '3' is:

Error: Vertex does not exist.

Calling the linked list destructor.

Calling the linked list destructor.

Empty linked list

(testing for vertices that do not exist.)

Enter vertex letter to find adjacency list or 'q' to quit: q

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Calling the linked list destructor.

Jers-MacBook-Pro:Directed Graphs jerchernglaw$