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

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

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

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

**\*Inter2\* Draw an example with 3 vertices.[1]**

**a. undirected graph**

**b. directed graph –**

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

**\*Inter3\* Draw a smallest connected graph with 3 vertices. [1]**

**a. undirected graph**

**b. directed graph**

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

**N - 1**

**\*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?[1]**

**N\*(N-1) = 39800 flights;**

**\*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?[1]**

**N\*(N-1)/2 flights = 39800 / 2**

**= 19900 flights;**

**For an adjacency list representation, G is the table:**

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

**- how many array slots are needed in G?[1] N**

**- how many linked list nodes are required altogether**

**for directed G? Why? [1] M**

**for undirected G? Why? [1] 2M**

**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.[7]**

Pop A

A has already been marked; ignore

[F G E G top]

Pop G

G has already been marked; ignore

[F G E top]

Pop E

Mark E

But there are no adjacent vertices to push

[F G top]

Pop G

G has already been marked; ignore

[F top]

Pop F

Mark F

F has I and E adjacent; push them

[E I top]

Pop I

I has already been marked; ignore

[ E top]

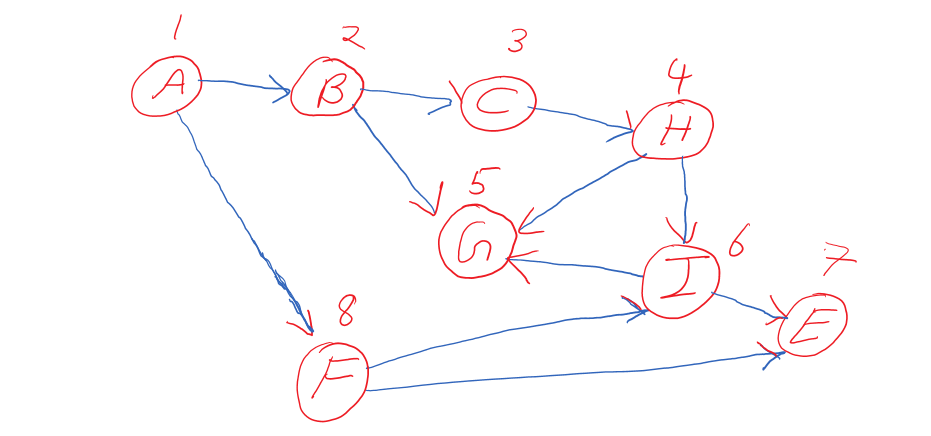
Pop E

E has already been marked; ignore

[empty]

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

**in the order they are marked by DFS <insert drawing here>[3]**

****

**dgraph.h, dgraph.cpp and hw6Client.cpp must be used.**

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

**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

**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

Use a loop but search through used slots only.

May throw an exception.

slist findAdjacency(char) – returns the linked list of

adjacent vertices of

the vertex whose name is given as an argument

Use a loop but search through used slots only.

May throw an exception.

[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

**Client program (hw6Client.cpp) 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
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)

And catches exceptions.