Graphs: DFS and BFS

Objectives

- Be familiar with graph representations, DFS and BFS.
- Be able to implement graph and search algorithms using linked list.

Instructions

In this programming assignment, you are about to implement graphs using adjacency list represenation and two popular search algorithms: Depth First Search (DFS) and Breadth First Search (BFS).

You will use the classes provided below (also available in the course shared folder). Please use the diagram given on page 744 in the textbook (Chapter 13) for the graph representation. Stack and queue with linked list implementaton are also used in DFS and BFS aglorithms. Upon completion of coding, you need to test your programs using the graph: graph.dat provided in the shared folder (starting from vertex Austin). The order of adjacent vertices of a vertex could be listed in either alphabetically or reverse-alphabetically in the input file.

Submissions

Zip all the source code into a single file and submit to the Blackboard. Don't include any project configuration files. We will use run your program in our own project.

General Advice

General advice about this programing assignment: a) Be clear on how to represent graphs using adjacency list. b) Do some program design before coding. c) Learn to use debugger to find program problems. d) Use conventional coding format to help check your programs easily.

Classes

```
class AdjVertex {
public:
//Constructor
AdjVertex( int index, int w )
{
indexOfAdjVertex = index;
weight = w;
next = NULL;
}

void setNext( AdjVertex * e ) {next = e;}
AdjVertex * getNext() { return next; }
int getAdjIndex() {return indexOfAdjVertex; }
int getWeight() {return weight;}

private:
int indexOfAdjVertex;  //Index of this adjacent vertex in array vertices
int weight;    //weight of an edge
AdjVertex * next;       //pointer to next adjacent vertex in linked list fashion
};

class Vertex {
public:
//Constructor
Vertex( string s )
{
info = s;
mark = false;
first = NULL;
}

//Getters and setters
string getInfo() {return info;}
AdjVertex * getFirstAdjVertex() {return first;}
void setAdjVertex( AdjVertex * e ) {first = e;}
void setInfo ( string s ) { info = s;}

//Add an adjacent node
void addAdjVertex(AdjVertex * en );

void setMark( bool b) { mark = b; }
bool isMarked() { return mark; }
```

```
void clearMark () { mark = false;}

//Print the info of an adjacent node
void print() { cout << info << "->"; }

private:
string info;       //Vertex info
bool mark;         //true if this vertex has been visited or brought to stack/queue
AdjVertex * first;  //Pointer to the first adjacent node
};

class Graph
{
public:
  Graph();
  Graph(int max);
  ~Graph(){}

  void addVertex(Vertex * );
  void addEdge(string from, string to, int weight);
  int  getNumVertices() {return counter;}
  int  getIndex( string st);
  Vertex * getVertex(int index) { return vertices[index]; }
  Vertex * getVertex(string st)
  {
  int index = getIndex(st);
  return vertices[index];
  }
  void clearMarks();

private:
  int numVertices; //number of vertices declared
  Vertex ** vertices; //array of pointers to vertices
  int counter; //number of vertices added into this graph at a point of time
};
```