CS1713 ALGORITHM DESIGN AND ANALYSIS

# Assignment 6 – Programming Graph Algorithms

## Idea

To create a class to represent graphs and write some graph algorithms.

## Method

There are two tasks.

1. Create a class to encapsulate information about a graph. Call this class **MyGraph**. This class requires the following **private** data members:

**numVertices** – number of vertices

**numEdges** – number of edges

**adjMatrix** – **numVertices** x **numVertices** integer adjacency matrix

**val** – array used in searches

**count** – int used in searches

The **public** methods required in this class are as follows:

|  |  |
| --- | --- |
| **Method** | **What it does** |
| void **MyGraph**(int v, int e) | Creates a **MyGraph** object with v vertices and e edges. Initializes other data members. |
| void **setCount**(int c) | Sets **count** to c |
| void **setNumVertice**(int v) | Sets **numVertices** to **v** |
| void **setnumEdges**(int e) | Sets **numEdges** to **e** |
| void **setVal**(int i, int x) | Sets entry i in **val** to x |
| void **setAdjMatrix**(int i, int j, int x) | Sets entry (i, j) in array **adjMatrix** to x |
| int **getNumVertices**() | Returns value of **numVertices** |
| int **getNumEdges**() | Returns value of **numEdges** |
| int **getCount**() | Returns value of **count** |
| int **getVal**(int i) | Returns entry i in **val** |
| int **getAdjmatrix(**int i, int j) | Returns entry at (i, j) in **adjMatrix** |
| void **printGraph**() | Prints **adjMatrix** in matrix form with 0s and 1s |
| void **BFS**() | Carries out BFS on whole graph by invoking **visitBFS** for each unseen vertex |
| void **visitBFS**(int k) | Carries out BFS starting at vertex k. Uses some kind of queue data structure |
| void **DFS**() | Carries out DFS on whole graph by invoking **visitDFS** for each unseen vertex |
| void **visitDFS**(int k) | Carries out recursive DFS starting at vertex k. |
| void **ArtPts**() | Carries out search for articulation points on whole graph by invoking **visitDFSArtPts** for each unseen vertex |
| int **visitDFSArtPts**(int k) | Carries out search for articulation points starting at vertex k. |

During both BFS and DFS you should print out the sequence of edges traversed, separated and labeled into connected components (this occurs naturally). Edges should be displayed as ordered pairs.

During the search for articulation points you should print out the articulation points only. An articulation point may print out twice because of the way the algorithm works – that is fine. A vertex is considered to be an articulation point only in regards to the connected component that it is part of.

See attached sample output.

2. Create a driver program **GraphTest** that will do the following things using the methods created in the **MyGraph** class:

* Instantiate a **MyGraph** object and add in the actual edges (using **setAdjMatrix**())
* Print out the adjacency matrix
* Print out all the neighbors of the second node.
* Carry out BFS and print out the sequence of edges traversed.
* Carry out DFS and print out the sequence of edges traversed.
* Print out articulation points.

This process should be repeated for 5 different graphs. **You will need to pause during output so that everything can be seen.**

#### Turning in the Assignment

This assignment is due on Friday December 6 (bonus) or Monday December 9 at 1:30pm (bring to exam). Upload all source and compiled files to CourseWeb (preferably zipped). Turn in a folder containing printouts of all source code

**Details**

1. Do not change the headers in **MyGraph**. If you are unsure of what specific methods are for please ask. Not all methods will be used in **GraphTest.** You may add more methods to the **MyGraph** class if you want

2. Refer to the vertices as 0, 1, 2, 3…..(V – 1). Start searches at vertex 0.

3. I will specify what 5 graphs I want you to test on the website. You will just copy and paste the lines of code into your **MyGraph** program. The lines will involve instantiating graphs and adding in hardcoded edges.

4. The **adjMatrix** matrix will be initialized to contain all 0s. The **val** array will be initialized to have all entries "unvisited" (you can use –1). Use the **count** variable to track the order in which nodes are visited and the **val** array to determine which elements have been visited, which are on the fringe and which have not been "seen" at all.

5. BFS requires use of a queue data structure. You do not have to build your own – you can use a predefined one in the Collections Framework.

6. When finding articulation points do not forget that the root of the search tree has to be checked in a special way. That is not taken into account in my handout. You must code that yourself.

Sample Output for one graph:

