CS 2420 Section 2 and Section 3
Spring 2016
Assignment 8: Graph Algorithm (Part I)
Due: 11:59 p.m. April 20 (Wednesday), 2016
Total Points: 100 points

Part I [25 points]: Submit your solution via Canvas.

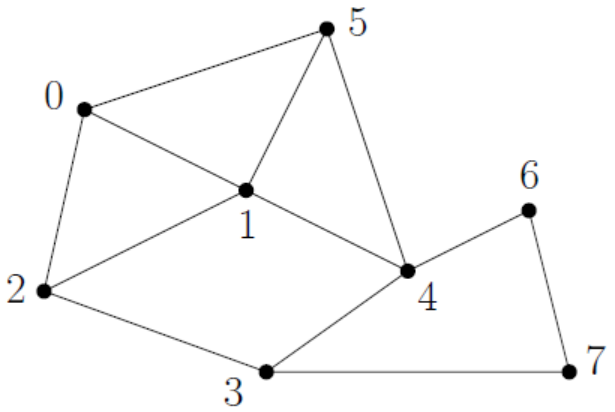

Figure 1: An undirected graph: the number besides each node is the index of the corresponding node.

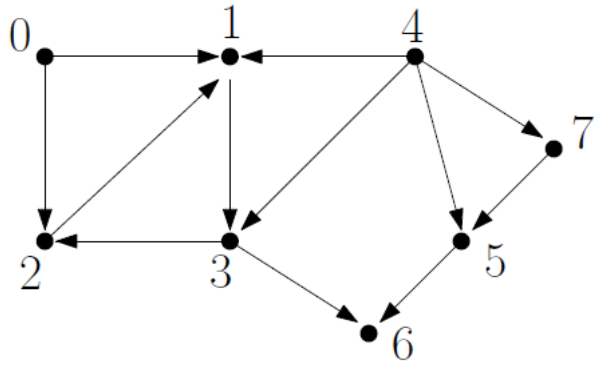

Figure 2: A directed graph: the number besides each node is the index of the corresponding node.

1. Please show the adjacency lists of the undirected graph presented in Figure 1. For each list, please order the adjacent vertices in the ascending order of their indices. For example, the adjacency list of vertex 0 should be $1 \rightarrow 2 \rightarrow 5$. **(2 points)**
2. Please show the adjacency lists of the directed graph presented in Figure 2. For each list, please order the adjacent vertices in the ascending order of their indices. **(3 points)**
3. Apply the Breadth-First Search (BFS) traversal algorithm, whose pseudo-code is summarized on slides 36 and 37 of the class notes, on the undirected graph presented in Figure 1. Your BFS traversal algorithm must use vertex 0 as the source (i.e., starting) vertex and the adjacency lists obtained in Question 1. **(10 points)**
    a. Please give the BFS traversal order of the vertices that are visited by the algorithm.
    b. Please draw the BFS tree (which is also the shortest path tree) produced by the algorithm. The root of the BFS tree is vertex 0. In the meantime, please write down the shortest path distance from each vertex *i* to the source vertex 0, in the parenthesis on the right side of the vertex. For example, vertex 0 is be represented as 0 (0) where the first 0 represents its index and the second 0 within the parenthesis represents the shortest path distance of 0.
4. Apply the Depth-First Search (DFS) traversal algorithm, whose pseudo-code is summarized on slides 61 and 62 of the class notes, on the directed graph presented in Figure 2. Your DFS traversal algorithm must use vertex 0 as the source (i.e., starting) vertex and the adjacency lists obtained in Question 2. Since this is a directed graph, the DFS traversal from vertex 0 may not be able to reach all vertices. Hence, as discussed in class, after the traversal from vertex 0, your algorithm should check other vertices in their index order. If there is an unvisited vertex, your algorithm should start a new traversal from that vertex. **(10 points)**

a. Please show the discovery time $d[i]$ and the finish time $f[i]$ for each vertex i that are visited by the algorithm. Refer to slide 60 of the class notes, where each node contains two numbers $a|b$, with a representing the discovery time and b representing the finish time.
b. Please give the DFS traversal order of the vertices that are visited by the algorithm.
c. Please give the DFS tree/forest generated by the algorithm. Again, if the DFS traversal from vertex 0 cannot reach all vertices, the algorithm will generate a forest (i.e., multiple trees) rather than a single tree. Refer to slide 60 of the class notes, where the red arrows illustrates the tree edges.

Part II [75 points; 35 points for BFSGraph class; 30 points for DFSGraph class; 10 points for the main function]:

In this assignment, you are required to implement BFSGraph and DFSGraph classes to perform Breadth-First Search (BFS) and Depth-First Search (DFS), respectively.

Below is the BFSGraph.h file, which contains C++ BFSGraph class interface, for your reference. Please refer to slide 19 of the class notes for the definition of the Vertex structure.

```
enum colorType {WHITE, GRAY, BLACK } ;

class BFSGraph
{
        private:
                colorType * color ; // record the colors of the vertices during BFS
                int * pre ;             // record the predecessors during BFS
                int * dis ;             // record the shortest path distances during BFS

        public:
                int n ;             // the number of vertices, the ids of the vertices are from 0 to n-1
                Vertex ** adj ; // adj[i] points the head of the adjacency list of vertex i, for i from 0 to n-1

                BFSGraph(int n_input) ;          // constructor
                void SetAdjLists(int * adjM) ; // build the adjacency lists from the adjacency matrix adjM
                void PrintAdjLists() ;             // print the adjacency lists of the graph

                // the following two functions are for the BFS traversal as we discussed in class
                void BFS(int id = 0) ;      // BFS traversal, id is the source vertex, with default 0
                void BFSVisit(int id) ;     // actually does BFS, search a connected component from id

                void PrintSP(int source, int v) ;  // Print the shortest path from the source to v and the
                                                   // shortest path distance from source to v
} ;
```

Points distribution for the above functions are:
BFSGraph [3 points]; SetAdjList [5 points]; PrintAdjLists[5 points]; BFS [5 points]; BFSVisit [10 points]; PrintSP [7 points];

Below is the DFSGraph.h file, which contains C++ DFSGraph class interface, for your reference.

```
enum colorType {WHITE, GRAY, BLACK } ;

class DFSGraph
{
        private:
                colorType * color ;   // record the colors of the vertices during DFS
                int * pre ;           // record the predecessors during DFS

        public:
                int n ;             // the number of vertices, the ids of the vertices are from 0 to n-1
                Vertex ** adj ;   // adj[i] points the head of the adjacency list of vertex i

                DFSGraph(int n_input);          // constructor
                void setAdjLists(int * adjM);  // build the adjacency lists from the adjacency matrix adjM
                void printAdjLists();           // print the adjacency lists of the graph

                // the following two functions are for the DFS traversal as we discussed in class
                void DFS(int id = 0) ;  // DFS traversal, id is the source vertex, with default 0
                void DFSVisit(int id ); // actually does DFS, search a connected component from id

                void PrintReachableNodes(int source) ; // Print all nodes that can be reached by source
};
```

Points distribution for the functions are:
DFSGraph [3 points]; PrintAdjList [3 points]; PrintAdjLists[3 points]; (Note: These three functions should be the same as the ones in BFSGraph class)
DFS [5 points]; DFSVisit [10 points]; PrintReachableNodes [6 points]

The main program first reads the graph information from the input file "Assign8BFSInput.txt", whose first line is the number of vertices of the input graph. The remaining numbers in the file are the values of the adjacency matrix of the input graph. The input graph stored in "Assign8BFSInput.txt" is actually the undirected graph shown in Figure 1.

The main program then stores the adjacency matrix in an array ***M1*** and constructs an instance of BFSGraph using the number of nodes specified in the input file and the adjacency lists built from the adjacency matrix ***M1***.

With the adjacency lists, your program should output the following information **to the screen only** (no need to output it to a file).

- Output the BFS traversal order of the vertices that are visited by the algorithm.
- Output a shortest path from vertex 0 to vertex *i* for every vertex $i \neq 0$ and its shortest path distance to vertex 0

Finally, your main program reads the graph information from the input file "Assign8DFSInput.txt", which contains the total number of vertices and the adjacency matrix of the directed graph shown in Figure 2. It then stores the adjacency matrix in an array ***M2*** and constructs an instance of DFSGraph using the number of nodes specified in the input file and the adjacency lists built from the adjacency matrix ***M2***.

With the adjacency lists, your program should output the following information **to the screen only** (no need to output it to a file).

- Output the DFS traversal order of the vertices that are visited by the algorithm.
- Output all the nodes that can be reached from vertex 0.

The grader will use his own driver code and input files to test the correctness of all the public member functions.

Programming files to be submitted: BFSGraph.h, BFSGraph.cpp, DFSGraph.h, DFSGraph.cpp, and main.cpp.