CS 2420 Section 2 and Section 3
Spring 2016
Assignment 9: Graph Algorithm (Part II)
Due: 11:59 p.m. April 29 (Friday), 2016
Total Points: 100 points

Part I [35 points]: Submit your solution via Canvas.

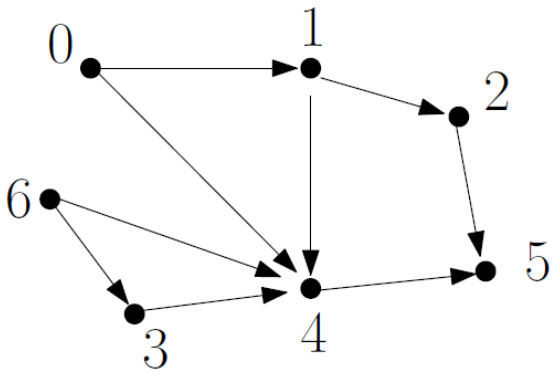

Figure 1: A directed acyclic graph (DAG)

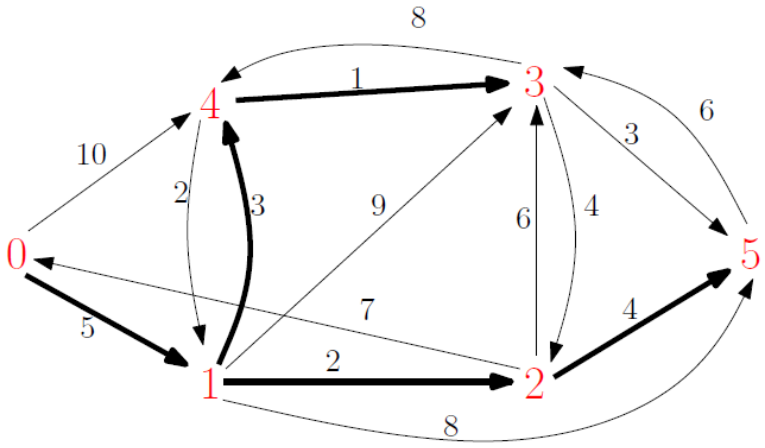

Figure 2: A weighted directed graph: the edges have weights and the thick edges form a shortest path tree from vertex 0.

1. Please show the adjacency lists of the DAG 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 → 4. **(2 points)**
2. Please list the in-degrees for each vertex in the DAG presented in Figure 1. **(1 point)**
3. Apply the topological sort algorithm, whose pseudo-code is summarized on slide 26 of the class notes, on the DAG presented in Figure 1 using the adjacency lists obtained in Question 1 **(15 points)**
    a. Please illustrate the changes in the stack and the changes of the in-degrees of each vertex after each iteration of the "while S is not empty" loop. Refer to slides 9 through 25 of the class notes for details. Note: There is no need to draw the graph at each iteration.
    b. Please give the topological order of the graph based on the flowchart of the pseudocode summarized on slide 26.
4. Please show the adjacency lists of the weighted directed graph presented in Figure 2. For each list, please order the adjacent vertices in the ascending order of their indices. **(2 points)**
5. Apply the Dijkstra's shortest path algorithm, whose pseudo-code is summarized on slides 58 of the class notes, on the weighted directed graph presented in Figure 2. The Dijkstra's shortest path algorithm uses vertex 0 as the source (i.e., starting) vertex and the adjacency lists obtained in Question 4. **(15 points)**
    a. Please show the intermediate results of the distance matrix (e.g., dist in the pseudocode) and the predecessor matrix (e.g., prev in the pseudocode) after each iteration of the "while PQ not empty" loop.
    b. Please give the shortest path from vertex 0 to each of the other vertices.
    c. Please give the shortest path distance from vertex 0 to each of the other vertices.

Part II [65 points; 10 points for the main function]:

In this assignment, you are required to implement the topological sort and the Dijkstra's shortest path algorithm explained in class, respectively. **You have to use the stack data structure to solve the topological sort problem. For the Dijkstra's shortest path algorithm, you may implement the priority queue by an array to find the shortest path distance at each iteration. You will get 10 bonus points if you correctly implemented the priority queue by a heap to find the shortest path distance at each iteration.**

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

```
class Vertex
{
  public:
    int id ;          // the id of the vertex
    int weight ;  // the weight of an edge (u,v), where v is the current node and is in the adj list of vertex u
    Vertex * next;

    Vertex(int id_input, int weight_input, Vertex * input_next = NULL)
    {
          id = id_input ;
          weight = weight_input ;
          next = input_next ;
     }
} ;

class Graph
{
   private: // The first two data members are only used in the shortest path algorithm
     int * pre ;  // record the predecessor for each vertex
     int * dis ;  // record the shortest path distance from each vertex to the source vertex
     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

   public:
     Graph(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 without the edge weights

     void TopologicalSort() ; // perform the topological sort using the stack data structure

     void PrintAdjListsWeight() ;  // print the adjacency lists of the graph with the edge weights
     void Dijkstra(int s); // compute a shortest path tree from the source vertex s in a general graph
     void PrintShortestPath(int source, int v) ; // print the shortest path from the source to v
     int GetShortestPathDis(int v) ;  // return the shortest path distance from the source vertex to v
} ;
```

Points distribution for the above functions are:
Graph [3 points]; SetAdjLists [5 points]; PrintAdjLists[5 points]; PrintAdjListsWeight [5 points]; TopologicalSort [15 points]; Dijkstra [15 points]; PrintShortestPath [5 points]; GetShortestPathDis [2 points]

The main program first reads the graph information from the input file "Assign9TopologicalInput.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 "Assign9TopologicalInput.txt" is actually the DAG shown in Figure 1.

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

With the adjacency lists, your program should call PrintAdjLists and TopologicalSort functions provided in the Graph class to output the following information **to the screen only** (no need to output it to a file).

- The adjacency list for each vertex.
- The topological order of the input graph.

Finally, your main program reads the graph information from the input file "Assign9ShortestPathInput.txt", which contains the total number of vertices and the adjacency matrix of the weighted directed graph shown in Figure 2. For the adjacency matrix, if M[*i*, *j*] is 0, it means that there is no edge from *i* to *j*. Otherwise, there is an edge and its weight is M[*i*, *j*]. The main program then stores the adjacency matrix in an array ***M2***, constructs an instance of Graph using the number of nodes specified in the input file, and builds the adjacency lists from the adjacency matrix ***M2***. It should be noted that there is a data member "weight" in the Vertex class to store the weight information. This weight information may not be used in the Topological Sort Algorithm and will be used in the Dijkstra's Shortest Path Algorithm.

With the adjacency lists, your program should call PrintAdjListsWeight, Dijkstra, PrintShortestPath, and GetShortestPathDis functions provided in the Graph class and output the following information **to the screen only** (no need to output it to a file).

- The adjacency list for each vertex together with the weight information. For example, adjacency list of vertex 0 can be represented as follows: 1(5)→4(10), where the edge weights are listed in the parenthesis.
- The shortest path from vertex 0 to each of the other vertices.
- The shortest path distance from vertex 0 to each of the other vertices.

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: Graph.h, Graph.cpp, and main.cpp.