UNIVERSITÉ BISHOP'S UNIVERSITY

Summer Project - 2019

Dr. Madjid Allili ∼ J125 ∼ ext. 2740 ∼ mallili@ubishops.ca

Graph Theory: Representation and Algorithms

1 Introduction

In order to complete this project successfully, the student should be familiar with many basic notions of graph theory. The questions below whose answers should be included in the project report will force the student to go over these necessary background notions about graphs. Most of the background can be studied in the following references, specially Cormen textbook where graph definitions can be found in Appendix B.4 and the Graph Algorithms can be read in chapters 22 to 26.

1. *Introduction to Algorithms*, T.H. Cormen et al., 3/e, The MIT Press, 2009.
2. *Data Structures & Algorithms in Java*, R. Lafore, 2/e, Sams Publishing, 2002.
3. *Data Structures & Problem Solving Using C++*, M. Weiss, 2/e, Pearson, 2003.
4. *Data Structures & Algorithm in C++*, M.T. Goodrich et al., 2/e, Wiley, 2011.

1.1 Theoretical Questions

Throughout all this project, we are only concerned with finite graphs.

1. Define the following concepts:
   (a) Undirected graph (ungraph) $G = (V, E)$.
   (b) Directed graph (digraph) $G = (V, E)$.
   (c) Degree (deg) of a vertex in an ungraph.
   (d) The in-degree ($\deg^-$) and the out-degree ($\deg^+$) of a vertex in a digraph.
   (e) A simple graph.
   (f) A path and simple path in an ungraph and a digraph.
   (g) A cycle in ungraph and digraph.
   (h) A (strongly) connected ungraph (digraph) and the (strongly) connected components of a graph (digraph).

(i) A weighted graph.

(j) A directed acyclic graph (DAG).

2. Prove the following (where $|E|$ is the number of edges in G):

   (a) If $G = (V, E)$ is an ungraph then

   $$\sum_{v \in V} \deg v = 2|E|.$$

   (b) If $G = (V, E)$ is a digraph then

   $$\sum_{v \in V} \deg^+ v = \sum_{v \in V} \deg^- v = |E|.$$

3. Prove that any undirected graph has an even number of vertices of odd degree.

4. Show that in a simple graph with at least two vertices there must be two vertices that have the same degree.

5. Give the definition of a complete graph $G = K_n$ with n vertices. What is the number of edges in K_n? Explain how this is computed.

6. Given the following digraph $G = (V, E)$

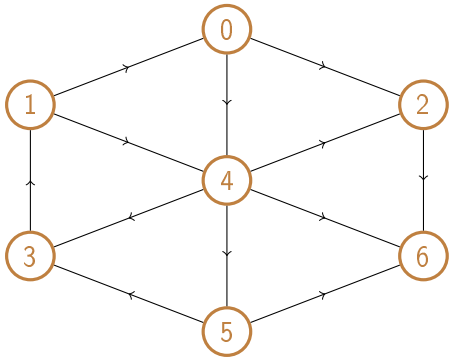


Figure 1: Digraph.

   (a) Give the in-degree and the out-degree of each vertex of G.

   (b) Give its adjacency-list representation.

   (c) Give its adjacency-matrix representation.

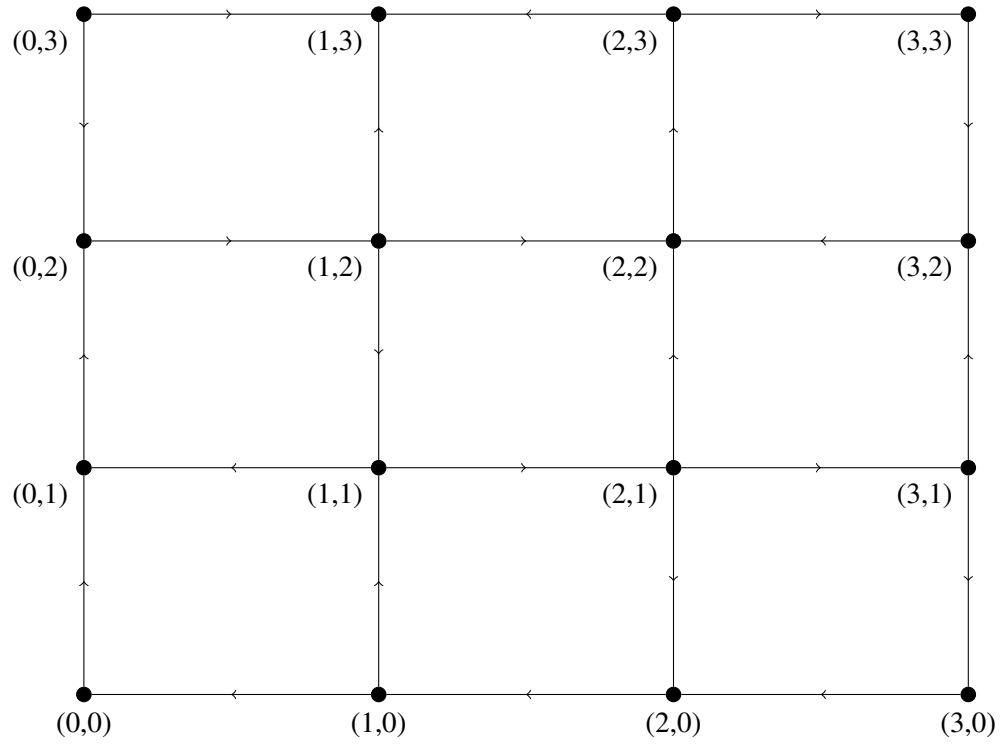
   (d) List all the directed cycles in G.

   (e) List all the cycles in the underlying undirected graph G_u of G.

   (f) Is G strongly connected?

   (g) What are its strongly connected components?

7. Consider the digraph below.



   (a) How many different directed simple paths are there from vertex $(3,1)$ to vertex $(3,3)$? List all of them by giving the ordered sequence of vertices.

   (b) How many strongly connected components are there in this graph? Specify all of them by giving the list of vertices.

8. Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

9. The transpose of a directed graph $G = (V,E)$ is the graph $G^T = (V,E^T)$, where

$$E^T = \{(v,u) \in V \times V : (u,v) \in E\}.$$

   Thus, G^T is G with all its edges reversed. Describe efficient algorithms for computing G^T from G, for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

10. Give examples of:

   (a) Digraphs that are not DAGs

   (b) DAGs whose underlying ungraph has (undirected) cycles.

   (c) DAGS whose underlying ungraph has no cycles.

11. Describe the Depth-First-Traversal and the Breadth-First-Traversal procedures (as described in Cormen). Show the results of the DFT on the digraph in Figure **??**. Assume that the for loop of lines 5-7 of the DFT procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

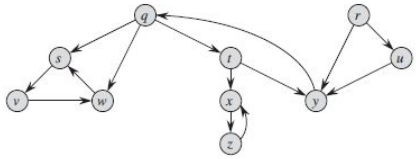


Figure 2: A digraph for DFS.

12. Describe the algorithm based on the DFS for producing a topological sort order when the digraph is a DAG.

13. Show the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the dag of Figure **??**, under the assumption of the previous exercise.

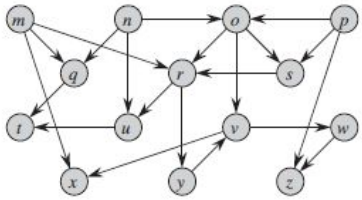


Figure 3: DAG for Topological Sorting.

14. Describe the Dijkstra's single-source shortest-paths algorithm and run it on the directed graph of Figure **??**, first using vertex s as the source and then using vertex z as the source. Show all the results.

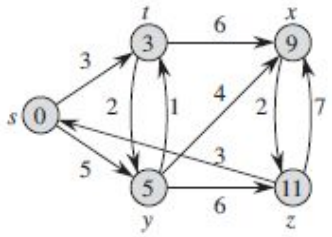


Figure 4: A digraph for shortest path.

2 Programming Project

This project involves implementing an adjacency list representation of a weighted graph, and using it to apply Dijkstra's shortest paths algorithm (single-source, all destinations(SSAD)) to a weighted directed graph. The program will read the specification of the problem from a given file named for instance "File.txt ". The input file will begin with two lines specifying the number of vertices in the graph, $G = (V, E)$, and the source vertex. Each of those values will be preceded by a label that ends with a colon character (':'). The next line will be blank. The remainder of the file will be a matrix (similar to the adjacency-matrix representation) of $|V|$ rows, each containing $|V|$ integer values for the weights of the edges in the graph. Weights will be integers in the range 1 to 99. Missing edges will be designated by the value 0. Here's an example:

```
Number of vertices:  11
Start vertex:  4

0   0   0  10   0  94   0   0  73   0   0
0   0   0  13   0   0   0   0   2   0   0
0   0   0   0   0  43   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0
0   0   0  87   0   0   0   0   0  68   0
0   0   0   0   0   0   0   0   0   0   0
0  97   0   0   0   0   0   0   0   0   0
0   0   0   0   0  91   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0
0  17   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0  55   0   0   0   0
```

Output will be written to a file named for instance "Result.txt ". The output file should consist of two sections, one displaying information about the given graph and the other displaying the results of running the Dijkstra's shortest paths SSAD algorithm.

The graph information display consists of a row of labels (see example), followed by $|V|$ rows of output displaying the graph information in a format similar to the adjacency-list representation of the graph. For each vertex, there will be one row of data listing the vertex number, followed by a list of neighbor records. A neighbor record consists of a neighbor (vertex) number, followed by a colon character, followed by the weight of the corresponding edge. The graph information display is followed by a blank line.

The SSAD results display begins with two lines of labels (see example), one showing the source vertex and a blank line, followed by $|V|$ rows showing the results for each possible destination vertex (including the source vertex itself). Each of these lines consists of three sections, the destination vertex number, the length (total weight) of the shortest path found, and the actual path (a sequence of vertex numbers separated by spaces). If a vertex is not reachable from the source vertex, you should display "`inf`" instead of the length.

Here's an output example that corresponds to the input example above:

```
 Node | Out-neighbors
------+--------------------
   0  | 3: 10   5: 94   8: 73
   1  | 3: 13   8: 2
   2  | 5: 43
   3  |
   4  | 3: 87   9: 68
   5  |
   6  | 1: 97
   7  | 5: 91
   8  |
   9  | 1: 17
  10  | 6: 55
```

```
Start vertex is:  4
```

```
 Dest | Total Weight | Path
------+--------------+-------------
   0  |      inf     |
   1  |       85     | 9   1
   2  |      inf     |
   3  |       87     | 3
   4  |        0     |
   5  |      inf     |
   6  |      inf     |
   7  |      inf     |
   8  |       87     | 9   1   8
   9  |       68     | 9
  10  |      inf     |
```

2.1 Notes

There are some explicit requirements for the programming project:

- Code has to be written in C++, no other platform is accepted.
- You must implement a C++ class for the weighted adjacency list representation of a directed graph; whether it is a template or not is entirely up to you.
- Your main class must be called **digraph**.
- The data read from a file and specifying the graph can be stored in a dynamic one dimensional or two dimensional array as a private member of the class together with the source vertex.

- That class must include a member function that writes the display of the graph in the proper format indicated above.
- You have to write an extensive REPORT containing the answers to the 14 theoretical questions and explaining the programming project, and how to operate your program. For each input file specification, you have to draw the corresponding digraph in your report and HIGHLIGHT the shorted path found by the Dijkstra's SSAD algorithm for each destination vertex (when the path exists obviously).
- Dijkstra's SSAD algorithm must be implemented as a separate function, not as a member of the digraph class. That function should take an initialized graph object as one of its parameters (and the source vertex as the second), and compute the path lengths and shortest paths, but it should not write any of the display to the output file. That should be done by a different function that creates the necessary digraph object and calls the SSAD method.
- Since you'll be implementing the complete program, the interfaces of the digraph class and the SSAD function are up to you.

2.2 Testing

We will include a number of files specifying weighted graphs, and the corresponding SSAD solutions. You may want to develop additional test cases yourself (there will be some bonus points for doing that). For marking purposes, it is important to format your output as specified above.