Ann, Bob, Charlie (*replace with your names*)
COSC 336
3/19/2020 (*replace with the current date*)

Assignment 9

Instructions.

1. Due Nov 23.

2. This is a team assignment. Work in teams of 3-4 students. Submit one assignment per team, with the names of all students making the team.

3. Your programs must be written in Java.

4. Write your programs neatly - imagine yourself grading your program and see if it is easy to read and understand. At the very beginning present your algorithm in plain English or in pseudo-code (or both). Comment your programs reasonably: there is no need to comment lines like "i++" but do include brief comments describing the main purpose of a specific block of lines.

5. You will submit on Blackboard 3 files. The first file should be a .pdf file with the short descriptions in English or in pseudocode of the algorithms for the programming tasks you are required to do and the results that you are required to report. Files 2 and 3 will contain the java codes of the program for task1 and the program for task 2.

   For editing the above document with Latex, see the template posted on the course website.

   assignment-template.tex and

   assignment-template.pdf

   To append in the latex file a pdf file, place it in the same folder and then include them in the latex file with

   ```
   \includepdf[pages=-,pagecommand={},width=\textwidth]{file.pdf}
   ```

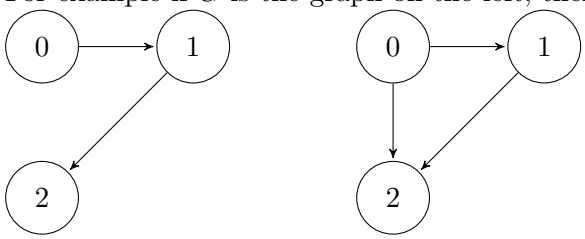
   To append in the latex file a .jpg file (for a photo), use

   ```
   \includegraphics[width=\linewidth]{file.jpg}
   ```

Programming Task 1. This is related to Exercise 22.1-5, page 593 in the textbook.

Part (a). Write a program that computes the adjacency list of the directed graph G^2 (defined in the exercise; this is called the square of G), given the adjacency list of the directed graph G. The graph G^2 has the same nodes as G, and (u, v) is an edge of G^2 if and only if there is path of length 1 or 2 from u to v.

For example if G is the graph on the left, then G^2 is the graph on the right.



For the adjacency list, you **must** use the Java class Adj_List_Graph given in the file Adj_List_Graph.java (see Test_Adj.java for a very simple example of using this class).

You will read the input graph G from a file which contains 2 lines. The first line contains the number n of vertices of the graph. The second line contains a sequence of n^2 bits (values 0 and 1). The n nodes are labeled $0, 1, \ldots, n-1$. If the $i \times n + j$-th bit in the sequence is 1, then there is an edge from node i to node j, and if the $i \times n + j$-th bit in the sequence is 0, then there is no edge from node i to node j.

The program has to create the adjacency list of the graph G^2 and then use the printGraph function of the class Adj_List_Graph to print the edges of G^2.

Run your program on two data sets from the files

input-9.1

input-9.2

Describe briefly your program and report the results in the .pdf file.

Programming Task 2.

Write the program that modifies Breadth First Search (see for example the basic version of BFS in Notes 10) in such a way that given an undirected connected graph G, and a starting node s, it will print for every node v the length of the shortest path from s to v and also the number of shortest paths from s to v. Thus, you'll have two arrays $dist$ and $npath$, and for each vertex v, at the end of the program, $dist[v]$ will be equal to the length of a shortest path from s to v, and $npath[v]$ will be equal to the number of shortest paths from s to v.

(For example, if G_1 is the first graph below, the length of a shortest path from 1 to 7 is 3, and there are three shortest paths from 1 to 7, namely

1 - 2 - 5 - 7,

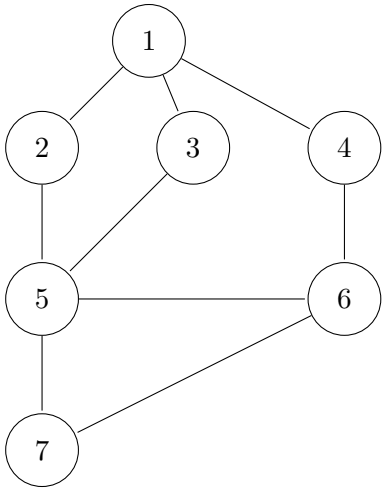
1 - 3 - 5 - 7 and

1 - 4 - 6 -7).

So, $dist[7] = 3$, and $npath[7] = 3$.

Test your program on the graphs G_1 and G_2 (see the figures) using 1 as the starting node and report the results you have obtained for these two graphs. You must use the adjacency list representation of a graph. As in the other programming task, for the adjacency list, you **must** use the Java class Adj_List_Graph given in the file Adj_List_Graph.java (see Test_Adj.java for a very simple example of using this class).

graph G_1



graph G_2: