CS 310 ‘Algorithms’ **Assignment-1**

Due date: **Thursday, October 3, 2019 at 11:55 PM**. [Total Marks = 50]

IMPORTANT: Read the instructions at the end of this document and follow the naming conventions.

Q1. **[14 marks]**

You are given a set of ‘n’ chemical compounds which have to be packed in two boxes. Some of the chemicals may react with other chemicals and should not be packed in the same box.

1. Code an **efficient** algorithm in C/C++ that determines if it is possible to safely pack these chemicals in two boxes and if so, then print which chemicals should go in Box-1 and which in Box-2. [6]
2. If it is not possible to safely pack the chemicals in two boxes then print at least one sequence of chemical interactions that prevents such a division. [6]
3. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm?** You should include clear comments that explain your algorithm’s time complexity. [2]
4. You should create at least three different input files to test your algorithm for different scenarios (see format below). Creating test cases helps you think about the problem and different corner cases. For testing your code, you may share your test cases with other classmates. Your code should be able to scale up to larger input sizes .

Your code will read input from a text file. The format of the file is as follows. The first line contains the number of chemicals. This will be followed by n lines (one for each of the n chemicals) and each line will contain the chemical number followed by a colon and a list of chemicals with which it reacts. Some examples are given below.

Input :

```
n 6
0 : 1 2
1 : 0 3 5
2 : 0
3 : 4 1
4 : 3
5 : 1
```

Output Format:

```
Yes
0 3 5
1 2 4
```

Input :

```
n 6
0 : 1 2
```

```
1 : 0 3 5
2 : 0 4
3 : 4 1
4 : 3 2
5 : 1
```

Output Format:

```
No
0->1->3->4->2->0
```

Q2. **[14 marks]**

Suppose there exists a large transportation road network in a country that is prone to flooding. This road network is modeled as a connected graph, where the cities are represented as vertices and the bi-directional roads as undirected edges. Some roads in this transportation network are especially vulnerable, because they form the *only* connection between one part of the network with another and if they are damaged the network will become disconnected. You, as a computer scientist, are given the task of identifying ***all*** such vulnerable roads in this large network. See some examples below.

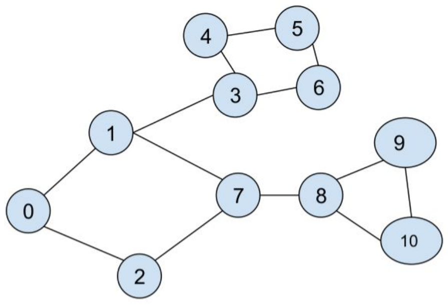

Figure-1

In Figure-1, **two** edges (1,3) and (7,8) are vulnerable edges.

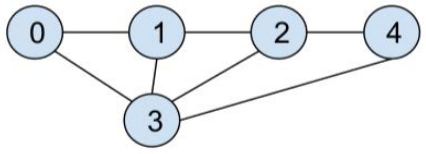

Figure-2

In Figure-2, **none** of the edges are vulnerable.

Your code will read input from a text file. The problem will be modeled as an undirected graph, where the vertices represent cities and the edges represent the roads that connect the cities. The graph nodes are numbered from V_0 to V_{n-1}. The file will have the value of n on the first line. This will be followed by n lines (one for each of the n nodes in the graph) and each line will contain a node number (say V_i) followed by a colon and a list of nodes to which node i has an edge. For example, if there is an edge between node V_0 and node V_1 and edge between node V_0 and node V_3, then that will be represented as 0:1 3

For example, the input graph in Figure-2 will be represented as below.

Input (for Figure-2):

```
n 5
0 : 1 3
1 : 0 2 3
2 : 1 3 4
3 : 0 1 2 4
4 : 2 3
```

Output (for Figure-2):

```
0
```

Output (for Figure-1):

```
2
(1,3)
(7,8)
```

1. Code an **efficient** O(|V|+|E|) algorithm in C/C++ that determines the number of vulnerable edges in the given graph. You also have to output all such vulnerable edges, if they exist. See the given output format. [12]
2. Give a clear description of your algorithm and data-structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm?** You should include clear comments that explain your algorithm's time complexity. [2]
3. You should create at least three different input files to test your algorithm for different scenarios (see format below). Creating test cases helps you think about the problem and different corner cases. For testing your code, you may share your test cases with other classmates. Your code should be able to scale up to larger input sizes .

Q3a. **[12 marks]**

A group of world leaders are attending a United Nations meeting. Some of the leaders hold a grudge against some of the others. You are given the set of world leaders and a set of **ordered** pairs (L_i, L_j) of leaders such that L_i holds a grudge against L_j. Note that the feeling is not mutual, i.e., L_j does not have a grudge against L_i. The UN organizers have to take them to the meeting hall and they will be walking in a line down a passage. The organizers are afraid that L_i might try to throw something at L_j's back while they are in the line and want that those that hold a grudge (L_i) are not **somewhere** behind those they grudge (L_j) in the line (note the phrase "somewhere behind in the line"). The organizers have asked you for help.

1. Code an **efficient** algorithm in C/C++ that prints a safe ordering of the leaders in the line or lets the organizers know that it is not possible. [5]

2. If a safe ordering is not possible then print at least one sequence of grudges to illustrate that. [5]
3. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm?** You should include clear comments that explain your algorithm's time complexity. [2]
4. You should create at least three different input files to test your algorithm for different scenarios (see format below). Creating test cases helps you think about the problem and different corner cases. For testing your code, you may share your test cases with other classmates. Your code should be able to scale up to larger input sizes .

Your code will read input from a text file. The format of the file is as follows. The first line contains the number of leaders. This will be followed by n lines (one for each of the n leaders) and each line will contain the leader number followed by a colon and a list of leaders they hold a grudge against. Some examples are given below.

Input :

```
n 5
0 : 1 2 3
1 : 3
2 : 1
3 :
4 : 0
```

Output Format:

```
Yes
4 0 2 1 3
```

Input :

```
n 5
0 : 1 2 3
1 : 3 4
2 : 1
3 :
4 : 0
```

Output Format:

```
No
4->0->1->4
```

Q3b. **[10 marks]**

This part is a continuation of Q3a. Instead of lining up the leaders, you have to seat the leaders in rows while they are on the stage. Let's call the front row of the stage as Row-1 and second row as Row-2 and so-on. If L_i holds a grudge against L_j then L_i **must be in a lower numbered row** than L_j. Note: lengths of different rows (i.e. number of people sitting in that row) need not be the same. The organizers have asked you for help.

1. Code an **efficient** algorithm in C/C++ that finds the **minimum** number of rows needed. You have to print the number of rows and the order in which the leaders are seated in rows. If it is not possible to safely place the leaders in rows then mention that in the output. [8]
2. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. The running time should not exceed that of Q3a. You should include clear comments that explain your algorithm's time complexity. [2]
3. You may reuse the input files you created in Q3a above to test your algorithm.

Input :

```
n 5
0 : 1 2 3
1 : 3
2 :
3 :
4 : 0
```

Output Format:

```
Yes
R 4
4
0
1 2
3
```

Input :

```
n 5
0 : 1 2 3
1 : 3 4
2 : 1
3 :
4 : 0
```

Output Format:

```
No
```

Instructions and policies

1. When submitting, please ***rename the folder*** according to your ***roll number***.
2. Do ***delete all executables*** and ***test files*** before submitting your assignment on LMS.
3. Folder convention ***should not be changed.*** If you make any changes, the auto grader will ***grade your assignment 0***.
4. You must submit your ***own*** work. You may discuss the problems with other classmates but must not reveal the solution to others or copy someone's work. Remember to acknowledge other classmates if discussions with them has helped you.
5. You should name your code files using the following convention: qx.cpp
6. If the assignment includes any theoretical questions, then type your answer to those questions and submit a separate pdf file for each using the naming convention above.
7. Upload all your files in the corresponding assignment folder on LMS. **There will be a 20% deduction for assignments submitted up to one day late (the late deduction is only applicable to the questions submitted late, not on the whole assignment). Assignments submitted 24 hours after the deadline will not be marked.**
8. There will be vivas during grading of the assignment. The TAs will announce a schedule and ask you to sign up for viva time slots. Failure to show up for vivas will result in a **70% marks reduction** in the assignment.
9. In the questions where you are asked to create test cases. Think carefully about good test cases that check different conditions and corner cases. The examples given in the assignment are for clarity and illustration purposes. You should not assume that those are the only test cases your code should work for. Your code should be able to scale up to larger input sizes and more complex scenarios.
10. Do not make arbitrary assumptions about the input or the structure of the problem without clarifying them first with the Instructor or the TAs.
11. **We will use automated code testing so pay close attention to the input and output formats.**
12. **Make sure that your code compiles and runs on Ubuntu. You may choose to develop your code in your favourite OS environment, however, the code must be properly tested on Linux before submission. During vivas, your code should not have any compatibility issues. It's a good idea to use gcc -Wall option flag to generate the compiler's warnings. Pay attention to those warnings as fixing them will lead to a much better and robust code.**
13. For full credit, comment your code clearly and state any assumptions that you have made. There are marks for writing well-structured and efficient code.
14. Familiarize yourself with LUMS policy on plagiarism and the penalties associated with it. **We will use a tool to check for plagiarism in the submissions.**