COMPUTER SCIENCE 311
SPRING 2019
PROGRAMMING ASSIGNMENT 2
TRACING THE SPREAD OF A COMPUTER VIRUS
DUE: 11:59 P.M., APRIL 21

1. Overview

Suppose you are monitoring a collection of n networked computers, labeled $C_1, C_2, \ldots, C_n$, in order to track the spread of an online virus. You are given a collection of trace data indicating the times at which pairs of computers communicated. The data is provided as a sequence of m ordered triples (C_i, C_j, t_k). Such a triple indicates that C_i and C_j communicated at time t_k. For simplicity, let us assume that each pair of computers communicates at most once during the interval under consideration. Your goal is to devise an efficient data structure to answer ***infection queries***; that is, queries of the following form:

> *If the virus was inserted into computer C_a at time x, could it possibly have infected computer C_b by time y?*

The mechanics of infection are simple: if an infected computer C_i communicates with an uninfected computer C_j at time t_k (in other words, if one of the triples (C_i, C_j, t_k) or (C_j, C_i, t_k) appears in the trace data), then computer C_i becomes infected as well, starting at time t_k. Infection can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backward in time. Thus, for example, if C_i is infected by time t_k, and the trace data contains triples (C_i, C_j, t_k) and (C_j, C_q, t_r), where $t_k \leq t_r$, then C_q will become infected via C_j.[1]

Example 1. Suppose that $n = 4$, the trace data consists of the triples

$$(C_1, C_2, 4), (C_2, C_4, 8), (C_3, C_4, 8), (C_1, C_4, 12),$$

and that the virus was inserted into computer C_1 at time 2. Then C_3 would be infected at time 8 by a sequence of three steps: first C_2 becomes infected at time 4, then C_4 gets the virus from C_2 at time 8, and then C_3 gets the virus from C_4 at time 8. □

Example 2. Suppose that $n = 4$, the trace data consists of

$$(C_2, C_3, 8), (C_1, C_4, 12), (C_1, C_2, 14),$$

and that the virus was inserted into computer C_1 at time 2. Then C_3 would not become infected during the period of observation: although C_2 becomes infected at time 14, C_3 only communicates with C_2 *before* C_2 was infected. There is no sequence of communications moving forward in time by which the virus could get from C_1 to C_3. □

Project objective. Write a program that takes a collection R of m triples, corresponding to the trace data for n computers, and preprocesses R in $O(n + m \log m)$ time[2] to build a data structure G

[1] We allow t_k to be equal to t_r. This just means that C_j had open connections to both C_i and C_q at the same time, and so a virus could move from C_i to C_q.

[2] Worst-case or expected time bounds are equally acceptable for this project.

that, given the IDs for any two computers C_a and C_b and two times x, y, where $x \leq y$, returns one of two things in $O(m)$ time.

- If there is a sequence of communications such that a virus inserted into computer C_a at time x could have infected computer C_b by time y, then the program returns a list containing one such sequence.
- If no infection sequence exists, then the program returns `null`.

Teamwork. For this programming assignment, you should work in teams of two or three. It is your responsibility to assemble a team. Note that

- even if you signed up for a group for Project 1 on Canvas, you will need to sign up for a group for Project 2, and
- your group for Project 2 does not have to be the same as for Project 1.

Reference. This project is based on Exercise 11, Chapter 3, of

Jon Kleinberg and Éva Tardos, *Algorithm Design*, Addison-Wesley, 2006.

2. The Data Structure

Our problem can be reduced to a graph connectivity problem that can be solved using breadth-first search or depth-first search. To do this, we use the triples in the trace data to construct a directed graph G as follows.

- Sort the triples by nondecreasing timestamp.
- Maintain a map where the keys are the computers and the associated values are lists. Initially, the list associated with each computer is empty.
- Next, scan the triples in sorted order.
- For each triple (C_i, C_j, t_k) we encounter in the scan do the following.
  - Create nodes (C_i, t_k) and (C_j, t_k), if they do not already exist.
  - Add a directed edge from (C_i, t_k) to (C_j, t_k).
  - Add a directed edge from (C_j, t_k) to (C_i, t_k).
  - Append a reference to (C_i, t_k) to the list for C_i.
  - Append a reference to (C_j, t_k) to the list for C_j.
  - If (C_i, C_j, t_k) is not the first triple involving C_i, then add a directed edge from (C_i, t) to (C_i, t_k), where t is the timestamp of the preceding element (the previously last one) in the list for C_i.
  - Do the analogous thing for C_j.

Figure 1 shows the graph for the trace data of Example 1 in the previous section.

Assume that G has been constructed. To determine whether a virus introduced at computer C_a at time x could have infected computer C_b by time y, we do the following.

- Walk through the list for C_a until we reach the first reference to a node (C_a, x') such that $x' \geq x$.
- Run BFS or DFS on G to determine all nodes reachable from (C_a, x').
- If a node (C_b, y') with $y' \leq y$ is reachable from (C_a, x'), then we declare that C_b could have become infected by time y; otherwise, we declare that this is impossible.

You should verify for yourself that this algorithm indeed answers infection queries correctly in $O(m)$ time.

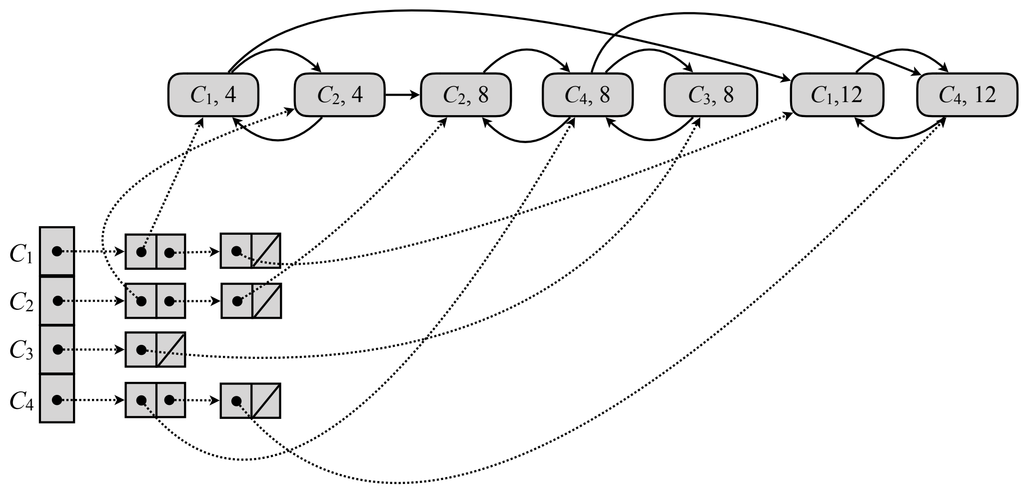

FIGURE 1. The graph G for the trace data of Example 1.

3. SPECIFICATIONS

Your submission must be a Java program that contains the two classes described in this section: `CommunicationsMonitor` and `ComputerNode`. You *must* make these classes and their methods public. Each of the classes may contain additional methods apart from the required ones. Be judicious in designing the classes and the data structures.

3.1. `CommunicationsMonitor`.

The `CommunicationsMonitor` class represents the graph G built to answer infection queries. It has the following methods.

- `CommunicationsMonitor()`: Constructor with no parameters.
- `void addCommunication(int c1, int c2, int timestamp)`: Takes as input two integers `c1`, `c2`, and a `timestamp`. This triple represents the fact that the computers with IDs `c1` and `c2` have communicated at the given `timestamp`. This method should run in $O(1)$ time. Any invocation of this method after `createGraph()` is called will be ignored.
- `void createGraph()`: Constructs the data structure as specified in the Section 2. This method should run in $O(n + m \log m)$ time.
- `List<ComputerNode> queryInfection(int c1, int c2, int x, int y)`: Determines whether computer `c2` could be infected by time `y` if computer `c1` was infected at time `x`. If so, the method returns an ordered list of `ComputerNode` objects that represents the transmission sequence. This sequence is a path in graph G. The first `ComputerNode` object on the path will correspond to `c1`. Similarly, the last `ComputerNode` object on the path will correspond to `c2`. If `c2` cannot be infected, return `null`.

  Example 3. In Example 1, an infection path would be $(C_1, 4)$, $(C_2, 4)$, $(C_2, 8)$, $(C_4, 8)$, $(C_3, 8)$.

  This method can assume that it will be called only after `createGraph()` and that `x` $\leq$ `y`. This method must run in $O(m)$ time. This method can also be called multiple times with different inputs once the graph is constructed (i.e., once `createGraph()` has been invoked).
- `HashMap<Integer, List<ComputerNode>> getComputerMapping()`: Returns a `HashMap` that represents the mapping between an `Integer` and a list of `ComputerNode` objects. The `Integer` represents the ID of some computer C_i, while the list consists of pairs

$(C_i, t_1), (C_i, t_2), \ldots, (C_i, t_k)$, represented by `ComputerNode` objects, that specify that C_i has communicated with other computers at times $t_1, t_2, \ldots, t_k$. The list for each computer must be ordered by time; i.e., $t_1 < t_2 < \cdots < t_k$.

- `List<ComputerNode> getComputerMapping(int c)`: Returns the list of `ComputerNode` objects associated with computer `c` by performing a lookup in the mapping.

3.2. **`ComputerNode`.** The `ComputerNode` class represents the nodes of the graph G, which are pairs (C_i, t).

- `int getID()`: Returns the ID of the associated computer.
- `int getTimestamp()`: Returns the timestamp associated with this node.
- `List<ComputerNode> getOutNeighbors()`: Returns a list of `ComputerNode` objects to which there is outgoing edge from this `ComputerNode` object.

4. Guidelines on Code Submission

- Use the Java default package (unnamed package). While this is not good coding practice for larger applications, it is more convenient for testing.
- Make all the methods and constructors explicitly public.
- You may design helper classes and methods in addition to those listed in Section 3. However, every class and method listed in Section 3 must be implemented ***exactly as specified***. This includes
  - the names of methods and classes (*remember that Java is case-sensitive*),
  - the return types of the methods, and
  - the types and order of parameters of each method/constructor.

  If you fail to follow these requirements, you will lose a significant portion of the points, even if your program is correct.
- You are not allowed to have external JARs as dependencies. You may use inbuilt packages such as `java.util.List`.
- Please include all team members names as a JavaDoc comment in each of the Java files.
- Create the project folder as follows: `<directory-name>/src/*.java`. This folder should nclude *only* `.java` files. Do not include any `.class` or other files.
- Create a zip file of your project folder.
- Upload your zip file on Canvas. Only one submission is necessary per team.

Your grade will depend on adherence to specifications, correctness, and efficiency.

Programs that do not compile will receive zero credit.

Important Note

Some aspects of this specification are subject to change in response to issues detected by students or the course staff. ***Check Canvas and Piazza regularly for updates and clarifications.***