Documentation of GossipTravel

Feng Zhou
fengzhou810@gmail.com

Description of the Problem

Background

This challenge is about gossip. 662 high school students are spreading juicy bits of social news among each other. Your challenge is to determine how long it will take, in seconds, for the "Victim" of each rumor to hear the news after it is started by the "Gossiper."

A few factors add complexity to this challenge:

· Each student ("Talker") only talks to their closest and most trusted friends ("Listener"). Talkers can have anywhere between 1 and 10 trusted Listeners with whom they share their secrets. Thus, in order for a rumor to reach the Victim, the rumor will most likely have to propagate across the network of friend pairs (the Victim will probably not hear it directly from the Gossiper).

· Every Talker/Listener pair has conversations of different lengths. The shortest conversation is only 8 seconds and the longest is over 45 minutes.

· Conversation lengths between the same two friends differ depending on which one is the Talker and which is the Listener.

Data

The attached file "Programming Challenge.xlsx" contains two tabs:

· Gossip Chain – list of Talker/Listener pairs and their Chat Time in seconds

· Inputs & Outputs – List of rumors for which to determine the travel time based on who is specified as the Gossiper and who is the Victim

Answer Format

Record your responses on the Inputs & Outputs tab. Assume this program will be implemented as a web application. It accepts and sends JSON strings as its inputs and outputs. Thus, each rumor should be encoded as a JSON string (using the specifications in cell H4) and answers should follow the JSON specification in cell H5.

Code Guidelines

Your program should:

1. accept a set of JSON strings as inputs and produce a set of JSON strings as outputs

2. process the set of inputs on multiple concurrent threads (how you divide up the work among threads is up to you)
3. use a test suite (jUnit, for example, if using Java) to ensure that the program behaves as desired (include your tests with your code submission)

In addition, provide a brief plan for deployment of this application to a secure, cloud-based environment. Assume Git is being used for version control and the repository can be cloned to both local and remote environments, but that we do not have a cloud server running yet – setting that up would be your job. Include enough detail that we're confident you could execute this task if needed. Names of specific platforms, for instance, would be useful. But don't worry about writing a step-by-step tutorial on the subject.

Overview

In this project, I implemented Dijkstra's algorithm to find the shortest path of a rumor from the gossiper to the victim. By utilizing different data structures, I improved the calculation time in cost of more space. More details will be discussed in the part "Documentation of Classes" when talking about the Class GossipTravelTimeCalculation which implemented Dijkstra's algorithm. I used unit test mainly in GossipSolution.java and GossipServerThread.java to check some feature of the built graph (GossipChainBuilder class)and the gossip travel time and path (GossipTravelTimeCalculation class). The calculation time may vary when you run the program again due to different CPU working conditions like temperature, load etc.

How to Run

1.Record the Input and Output in xlsx file:

Put the file *"Programming Challenge.xlsx"* under the directory of "GossipTravel/", run GossipSolution.java, it will fill in the input and output part of the xlsx file with required JSON strings. BTW, I think the terms *gossiper* and *victim* would be more accurate than *talker* and *listener* in the input field. So I made this change. It will also print the path for each rumor from the gossiper to victim in the console along with the total and average calculation time.

2.Web Application Which Receives and Sends JSON Strings:

I wrote a GossipServer to function as a web application to accept and send JSON strings. Run GossipServer.java to wait for connection, then run GossipClient.java to send a request. For each request the server receives, it will create a GossipServerThread to process that request. So you can run GossipClient several times to simulate multiple clients. I map the set of JSON strings to the field of "Rumor" in a JSONObject. The server only accepts strings in such format. Here I didn't take care of security communication stuff to make it simple and I just want to show an idea of what the web application would be like.

Documentation of Classes

1.GossipVertex

Each student is modeled as a GossipVertex. Its contains information of the name of the student and the list of GossipEdges(asTalkerEdges) when this students serves as a talker and the list of GossipEdges(asListenerEdges) when the student serves as a listener.

Instance variables:
private String name;
private ArrayList<GossipEdge> asListenerEdges;
private ArrayList<GossipEdge> asTalkerEdges;

Important Methods:
public boolean addEdge(GossipEdge e):
add a GossipEdge to either asTalkerEdges or asListenerEdges.

public GossipEdge findEdge(GossipVertex listener):
to find the GossipEdge in asTalkerEdges that contains the given listener.

2.GossipEdge

The chat between a talker and a listener is modeled as a GossipEdge. Its contains the information of the talker(GossipVertex), the listener(GossipVertex) and the chat time.

Instance Variables:
private GossipVertex talker, listener;
private int chatTime;

3.GossipGraph

The GossipGraph contains the information of all the students(GossipVertex) and the chat(GossipEdge) between every pair of talker and listener.

Instance Variables:
private HashMap<String,GossipVertex> gossipVerticies;
private ArrayList<GossipEdge> gossipEdges;

Important Methods:
public boolean addEdgeAndVertex(String talkerName,String listenerName, int chatTime):
for a given input from the xlsx file, we use this method to add and update the corresponding GossipVertex and GossipEdge.

4.GossipChainBuilder

This class is used to build the gossip graph based on the information from the xlsx file.

Constructor Method:

public GossipChainBuilder(String fileName, GossipGraph gg):
build the GossipGraph according to the content about the GossipVertex and GossipEdge in the given file.

5.GossipTravelTimeCalculation

It implemented Dijkstra's algorithm to get the shortest path for a rumor to travel from the gossiper to the victim. For more detail about Dijkstra's algorithm, please visit the following webpage http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. I used HashMap<String,GossipVertex> for the vertices in the graph to quickly get the corresponding GossipVertex when what I get is just a string of the talker or listener's name. I used PriorityQueue<GossipVertex> for the unSettledVertices and write the comparator for it to quickly retrieve the vertex with the minimal gossip travel time which is a key step (greedy strategy) in the Dijstra's algorithm. I also improved the performance of the method getListeners by retrieving listeners in the talker's asTalkerEdges list instead of finding the edge from the graph. So there are two warnings in this class about the instance variables vertices and edges are not used. To make the structure clear and maybe for later uses in the graph, I kept them.

Instance Variables:

private final HashMap<String,GossipVertex> vertices;
private final ArrayList<GossipEdge> edges;
private Set<GossipVertex> settledVertices;
private PriorityQueue<GossipVertex> unSettledVertices;
private Map<GossipVertex, GossipVertex> predecessors;
private Map<GossipVertex, Integer> gossipTravelTime;

Important Methods:

public int calculateGossipTravelTime(GossipVertex gossiper, GossipVertex victim):
implemented Dijstra's algorithm to calculate the shortest path from gossiper to victim

private void findMinimalGossipTravelTime(GossipVertex talker):
relax the listeners of a given talker, i.e. update the minimal gossip travel time for each unsettled listener of a given talker.

private int getChatTime(GossipVertex talker, GossipVertex listener):
get the chat time for a given pair of talker and listener

private List<GossipVertex> getListeners(GossipVertex talker):

get all the listeners for a given talker

private class shortestDistanceComparator implements Comparator<GossipVertex>:
write the comparator for the PriorityQueue unSettledVertices

private boolean isSettled(GossipVertex vertex):
decide whether a vertex is settled

private int getMinimalGossipTravelTime(GossipVertex v):
get the minimal gossip travel time from the gossiper to a given GossipVertex.

public LinkedList<GossipVertex> getPath(GossipVertex v):
get the path from the gossiper to the victim and null if no path exists

6.GossipInputFormatter

It is used to process the input from the xlsx file and get the required input json fromat.

7.GossipSolution

This class is used to fill in the JSON Request String and JSON Response String part in the xlsx file. It calls the calculateGossipTravelTime method from the GossipTravelTimeCalculation class to calculate gossip travel time and print the shortest path for each rumor to travel from gossiper to victim. For the calculation time I used System.nanoTime() before and after the calculation method is called to get the time used in calculation. I didn't use multiple threads for the process in this class since there would be multiple write operations on the same xlsx file. Apache POI 3.9 library is used for parse of the xlxx file. JSONParsing library is used for the parse of JSON Object. Junit 4.11 is used for unit test.

8.GossipServer

The GossipServer would wait for client request and once there's a request it will create a GossipServerThread to process that request. It supports multiple client requests by creating a new thread for each client request.

9.GossipServerThread

A GossipServerThread is used to process a client request by calculating the result for a set of JSON strings. It only accepts request with the correct JSON format. It creates multiple GossipCalculateThreads to calculate for each rumor which inputs as a JSON String.

Important Methods:
public void calculateWithMultipleThreads(String str, JSONObject outputJson) :
create a GossipCalculateThread for each rumor, calculate them concurrently and wait for all of the threads to finish to get the final result for the output JSON.

10.GossipCalculateThread

The GossipCalculateThread is used to calculate the shortest travel time for a rumor to travel from the gossiper to the victim. It calls GossipTravelTimeCalculation's calculateGossipTravelTime method. Here I used synchronized on outputJson since it is shared in multiple threads.

11.GossipClient

This class calls method in GossipInputFormatter to turn the data from the xlsx file to required request JSON string. Then send it to server and receives the result from the server.

Brief Plan for Deployment on Cloud

I would choose Amazon Web Service for this task. Use Amazon EC2 to launch an instance (a machine) and then we can use SSH client to connect to that instance. So we can upload the .class files of my program to run the code on that EC2 instance. After we get the IP address of the instance and specify the port for the server. We can run clients on other machines in the networks and connect to that server, send it JSON strings and receive its output JSON string.