Broadcasting a message within a Social Network

Overview

This project investigates the problem of making an announcement which I want to reach everybody within a social network. How many people do I need to announce that? I just want to find the smallest number of people to essentially announce to the social network that something's true, and everyone will hear it.

Data

First I will use some tiny data sample in order to test the algorithm and check if it will be working fine, afterward I will test the algorithm with a huge data sample, the provided USCD Facebook data. The Facebook data represents friendships between students at USCD in 2005. Every line in the file specifies a particular friendship relationship between two users, just numbers, which are the user IDs.

So, for example, the first line in this file says that user 0 is friends with user 22. It would be an undirected graph, because the friendship has to exist in both directions, but the algorithm will create a directed graph.

Questions

I want to find the group of people highly connected, so with this group I can reach everyone in the network. This will be a dominating set, afterward I would like to find the mínimum group of people in order to reach everyone in the network "The mínimum dominating set in the network".

Algorithms and Data Structures progress

Main Data Structure: The network has been laid out as a classic graph using an adjacency list. Each individual in the graph is a vertex and an edge between vertices represents a relationship. In order to represent the Graph I use a map, each entry in the map is a vertex(key) and the value is a list with its neighbors.

I have already done the first part of the Project with a greedy algorithm that is described in (Johnson, 1973). Basically, at each stage, the greedy algorithm chooses the set which contains the largest number of uncovered elements. Starting from an empty set, if the current sub-set of vertices is not the dominating set, a new vertex which has the most number of the ad-jacent vertices that are not adjacent to any vertex in the current set will be added.

The greedy algorithm would work as follow:

- Mark all vértices as "uncovered"

- Initialize an empty dominant set
- While there are uncovered vértices:
  - Find the vertex, v, which would cover the most uncovered vértices
  - Add v to the dominant set
  - Mark that vertex and all of its uncovered neighbors as covered

Now I have to cope with the hard part of the problem finding the mínimum group of people in order to reach everyone in the network "The mínimum dominating set in the network" that is a NP hard problem taking no polynomial time instead exponential or factorial. For this part I will try the "Big   Step   Greedy   Set   Cover Algorithm" depicted in: https://arxiv.org/pdf/1506.04220.pdf

Code Overview

Class name: [Graph.java]

Purpose and description of the class: public interface that represents a Graph with the necessary methods and class variables in order to reproduce the Graph's behaviour.

Class name: [CapGraph.java]

Purpose and description of the class: This class is the implementation of the Graph Interface. Fort he purpouse of this part of the Project, the method that resolves the Dominant Set problema is: public Set<Node> findDominantSet()

Class name: [Node.java]

Purpose and description of the class: This class represents a Node in the Graph. This class overrides hashCode and equals method in order to find a Node in collections.

Class name: [Edge.java]

Purpose and description of the class: This class represents an Edge in the Graph. An Edge represents a relation between two nodes in the Graph. This class overrides hashCode and equals method in order to find an Edge in collections.

Class name: [GraphLoader.java]

Purpose and description of the class: This class is an utility class that calls the methods in CapGraph in order to load data in the Graph.

Class name: [TestSetCover.java]

Purpose and description of the class: This class if for the purpose of testing the behavior of the CapGraph and GraphLoader class to see if the Graph behavior is as expected. The framework I used for testing is JUnit.

Overall Design Justification

The network has been laid out as a classic graph using an adjacency list. Each individual in the graph is a vertex and an edge between vertices represents a relationship. In order to represent the Graph I use a map, each entry in the map is a vertex(key) and the value is a list with its neighbors which is the main data structure. This has worked fine for this problem.

The main method in order to resolve the easy part of the Project is findDominantSet(). A method that return a Set of nodes which are a dominant set in the Graph.

Testing Plan

In order to test my implementation of the algorithm for the easy part of the Project I developed a JUnit class to test the basic cases, toy examples of graphs with a few vértices and edges, and in all cases it worked as I expected.

I developed a static main method in order to check the behavior of the algorithm with a big data example (Facebook data).

Reflection

The Greedy algorithm worked in each case as I expected, it is a very well known and tested algorithm. In some cases it found out the mínimum dominant set, but in other cases only a dominant set. I think that the hard part of the problem will be more problematic.