CSCI 3901 Winter 2021

Assignment 3

Due date: 23:59 AST Friday, February 26th, 2021 in Brightspace

Problem 1

Goal

Work with graphs and minimum weight spanning trees.

Background

A common task in machine learning and big data is clustering data. Clustering data means that we have many initial data points and we want to group the data points so that all of the points in the same cluster are similar in some way and points in different clusters are deemed sufficiently different from one another. There are many ways to do clustering. We are going to approximate one of them.

We are going to think about grouping documents around common topics. Given a set of documents, we will compare each pair of documents and will create a number that reflects how similar the two documents are to one another.

The basic idea of the clustering is to start with each document as its own cluster and to repeatedly merge clusters until we reach a balance point. How we merge the clusters is the point of this assignment.

Problem

You will have a method that lets us build an **undirected**, **weighted graph** between vertices. Each vertex has a string name/identifier (like a document name). The weight on each edge is a similarity measure between the vertices. For this assignment, that measure will be a positive integer, with a lower value meaning that the two vertices are very similar.

We will want to identify sets of vertices that are all similar. We call these sets *clusters*. For each cluster, we remember the weight of the heaviest edge that was used to create this cluster from previous ones (the maximum weight starts at 1 for clusters of a single vertex).

We begin with each vertex as its own cluster. We then proceed with merges in a way similar to making a minimum weight spanning tree, using Kruskal's algorithm[1]. Find the edge e with the smallest weight that connects two different clusters C and D. If there are several edges of the same low weight then take the edge with the smallest-named vertex at the end of an edge. Let the heavy edge that we're tracking for clusters C and D be edges c and d. Here is where we differ from Kruskal a bit: if the ratio

$$\text{(weight of } e\text{) / min(weight of } c\text{, weight of } d\text{)}$$

is less than the given tolerance value, then merge clusters C and D. Otherwise, don't consider edge e to merge clusters in the future. Stop when none of the clusters can be merged.

[1] https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

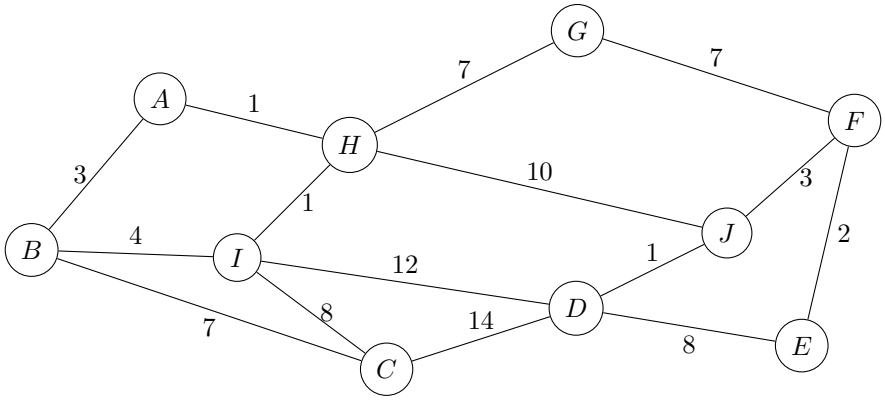

Figure 1: Sample graph

Your task is to create a class called `VertexCluster`. The class has at least 2 methods:

`boolean addEdge(String vertex1, String vertex2, int weight)` Record an edge going between the two vertices with the given weight. Return `true` if the edge was added. Return `false` if some error prevented you from adding the edge.

`Set<Set<String>> clusterVertices (float tolerance)` Return the set of clusters that you get from the current graph, using the given `tolerance` parameter for deciding if an edge is good enough to merge two clusters (as described earlier). Return `null` if some error happened while calculating the clusters.

Example

Consider the weighted graph of fig. 1, created with the following sequence of `addEdge` commands:

```
addEdge( "A", "B", 3);
addEdge( "A", "H", 1);
addEdge( "I", "H", 1);
addEdge( "I", "B", 4);
addEdge( "B", "C", 7);
addEdge( "I", "C", 8);
addEdge( "G", "H", 7);
addEdge( "H", "J", 10);
addEdge( "D", "I", 12);
addEdge( "D", "C", 14);
addEdge( "D", "E", 8);
addEdge( "J", "F", 3);
addEdge( "E", "F", 2);
addEdge( "F", "G", 7);
addEdge( "J", "D", 1);
```

In the algorithm, we process the edges in the following order of increasing weight and then tie-breaker. We use a maximum ratio of 3 for the calculation on whether or not to merge the components.

Edge	Weight	Calculation	Outcome
(A, H)	1	$1/\min(1,1) = 1$	Merge components
(D, J)	1	$1/\min(1,1) = 1$	Merge components
(H, I)	1	$1/\min(1,1) = 1$	Merge components
(E, F)	2	$2/\min(1,1) = 2$	Merge components
(A, B)	3	$3/\min(1,1) = 3$	Merge components
(F, J)	3	$3/\min(1,1) = 3$	Merge components
(B, I)	4	Vertices in same component	Disregard edge
(B, C)	7	$7/\min(3,1) = 7$	Don't merge
(F, G)	7	$7/\min(3,1) = 7$	Don't merge
(G, H)	7	$7/\min(3,1) = 7$	Don't merge
(C, I)	8	$8/\min(3,1) = 8$	Don't merge
(D, E)	8	Vertices in same component	Disregard edge
(H, J)	10	$10/\min(3,3) = 3.3$	Don't merge
(D, I)	12	$12/\min(3,3) = 4$	Don't merge
(C, D)	14	$14/\min(3,1) = 14$	Don't merge

We end with the 4 components:

- A, B, H, I
- C
- D, E, F, J
- G

Inputs

All the inputs will be determined by the parameters used in calling your methods.

Assumptions

You may (but don't need to) assume that

- You do not need to worry about round off error in the computations.
- If two vertex strings are different in any case-invariant way then they are different vertices.
- Not all vertices will necessarily be connected to one another in the graph.

Constraints

- You **may** use any data structures from the Java Collection Framework.
- You **may not** use a library package to for your graph or for the algorithm to do matchings on your graph.
- If in doubt for testing, I will be running your program on `timberlea.cs.dal.ca`. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- You do not need to implement and use a disjoint-set data structure to store the components, though it may be a fun exercise to do so if you choose.
- You will need an internal class to store each vertex of the graph. That class will want to store some representation of the cluster that the vertex belongs to. One option is to use an integer for the cluster number. Start each vertex with its own unique cluster number. When clusters merge, give everyone the cluster number that is the smaller number of the clusters being combined.
- You will need to choose whether to store the graph as an adjacency matrix, as an adjacency list, or as an incidence matrix. Your best bet will likely be an adjacency list.
- You will need to decide on how to store the weights of edges.
- Rather than find the smallest edge between clusters, you might consider running through all edges and ignoring the edges that join vertices in the same cluster (which you can tell by them having the same cluster number in their endpoint vertices).
- You might consider writing your own method to print the graph. It might help simplify your debugging.

What to submit

- Your `VertexCluster.java` and any other support files.
  - Do **not** include a `package` statement in your files. This makes it easier to mark.
- A list of tests cases for the problem, either in a separate `.pdf` or as internal documentation in your code.

Marking scheme (out of 20)

- Documentation, program organization, clarity, modularity, style – 4 marks
- List of test cases for the problem - 3 marks
- Ability to create the graph from the inputs and to return appropriately in error conditions - 3 marks
- Ability to calculate and report on the clusters - 10 marks