Due Dec 6, 2017

This project will use IMDB's actor/movie dataset to compute the Bacon Number of an actor, which refers to the shortest path through the movie/actor graph that connects two actors – the so called 'Six Degrees of Separation' problem. You will use the BRIDGES s/w to display the path that leads from the actor to Kevin Bacon (or another chosen actor).

In part 1 (this project), you built the graph of all the actors and movies, with actors connected to the movies they were part of, while movies were connected to all actors in that movie. In part 2 (this project), you will **compute the Bacon number for an actor** using the BFS traversal. The Bacon number for an actor is defined as the smallest number of links that connects the given actor to another actor; in particular, the Bacon number for all actors to Kevin Bacon is less than 6, given the number of actors Kevin Bacon has collaborated during his career.

More details of this problem can be found at

`http://introcs.cs.princeton.edu/java/45graph/`

Dataset:

Same as part 1.

GraphAdjList<K, E>

This is the main representation of the graph on BRIDGES, mapping vertex values (in our case, actors or movie names (String)) to an object E (in our case, we will point to lists of edge objects that contain an actor or movie name (String)); Each vertex will point to an adjacency list (which is a singly linked list of type SLelement¡E¿), and which will store the graph edges. Methods to query for a specific vertex by the key K, setting node and edge attributes are illustrated in the graph example. Refer to the full class description at

http://bridgesuncc.github.io/doc/java-api/current/html/annotated.html

Tasks.

1. **Input.** User will specify the source and target actor names; assume the actors exist in the graph.
2. **Implement the BFS traversal.** You will need to use a queue (use any of the Java queue implementations, eg. ArrayQueue, LinkedList, etc) as part of the algorithm (see Section 3.5, Text). You will need to use the adjacency lists of the graph to iterate over the children of a node. See the BRIDGES graph example that illustrates this and how to access the edge vertices. This will be needed to apply attributes(color, size, etc) to nodes and edges in the path from the source to the destination node.
3. **Bacon number - path length computaton.** The Bacon number represents the path length from a start node to a destination node. Let $D(i)$ represents the distance of a node i from its start node. During the BFS traversal, we will update the distance as follows:

   $$D_i = D_{parent(i)} + 1$$

   with the source node $D(source) = 0$.

   Thus, as the BFS traversal progresses from the start node, the distances of the visited nodes are updated, until the destination node is reached. The final distance to the destination node is the Bacon number of the source node.

4. **Displaying/Highlighting the path.** Given that we have a visualization of the actor-movie graph, it makes sense to display the path, once the BFS traversal is completed. This can be done by **keeping track of the parent** of each node that was visited during the BFS traversal. Again, a hashmap $Parent < String, String >$ is maintained. When the traversal visits a node N, and N is a child of P, then *Parent(N)* $= P$. Once the destination actor is found, then we use the hashmap to trace back the path towards the source actor using Parent hashmap (linear in the number of nodes in the path). Once the names of the actors/movies in teh path is known, use the vertices hashmap to access the nodes, so as to set the visual attibutes.

Implementation Details.

1. **BFS Traversal.** Refer to Section 3.5, Text for pseudo code of algorithm. You will need to use a queue as part of the traversal; for this you can use any of Java's queue implementations (LinkedList, ArrayQueue, etc.) In order to ensure that nodes are not repeatedly visited, keep a HashMap, $Mark < String, Boolean >$ that marks nodes as they are visited.
2. **Distance Computations.** Use a Java HashMap $Distance < String, Integer >$ to keep track of the distance at each node.
3. For the path computation, you will need to keep track of the parent of each node; again use a HashMap, $Parent < String, String >$. with the mapping from node to its parent.
4. **Setting attributes to vertices and edges**. The following sequence can be used to get to a vertex for settting attributes:

   ```
   Element<String> vert = graph.getVertices().getVertex(vertex_name);
   vert.getVisualizer().setColor("green")
   ```

   To set the color of a link between two elements $el1$ and $el2$, you can do the following

   ```
   el1.getLinkVisualizer(el2).setColor("green");
   ```

Evaluation:

You will do an interactive demo of your implementation. This should be a fun project.

Submission Requirements.

Turn in all of source code to Canvas; ensure it is well documented.