Due April 12, 11.59pm

In part 2 of this project you will solve and illustrate the Bacon number problem: find the shortest path from any actor to Kevin Bacon, for for that matter, any other actor. We will again use the IMDB movie/actor dataset to demonstrate this for any pair of actors.

A description of this problem can be found at
`http://introcs.cs.princeton.edu/java/45graph/`

Dataset:

We will continue to use the IMDB dataset from part 1.

Problem Description.

Given that all the graph edges are of the same weight in this problem - we are trying to find the number of links leading to Kevin Bacon, and each link is of the same weight, say 1 - finding the shortest path reduces to the **breadth-first-search(BFS)**. BFS is usually implemented by a queue and a queue implementation will be provided as part of this project.

Thus, the basic idea is to start at the given actor node (via user input) and then search (using BFS) for the destination actor node(provided also via user input). The search progresses from the current node, then its immediate neighbors, then their neighbors, and continues until the target actor is found.

Tasks.

1. **Path Length[15 points].** Given the source and destination actor names, you will need to compute the number of links(path length) that leads from the source to the destination actor. For this you will need to maintain a distance associated with each actor/movie node of the graph(distance for the start node is 0). When a actor or movie node is encountered during the BFS traversal, then that node's distance is updated as follows:

$$D_{node} = D_{parent} + 1$$

   where D_{node} is a child of D_{parent}.

   Implementation Details.

   - **BFS algorithm.** Detailed in Section 3.5(page 125).
   - **Queue Implementation.** You will be provided with an implementation of the Queue data structure that you can use in this project.
   - Use a Java Hashmap, $Distance < String, Integer >$ to keep track of the distances for each actor or movie. Similarly, you can use another hashmap($Mark < String, Boolean >$) to keep track of the nodes that have been visited.
   - **Output.**
     - Highlight the source and destination node in red.
     - Highlight all the remaining nodes that were visited by the BFS traversal in orange.

   - When the destination actor is found, print the length of the path to the console. **Use the visualization to verify your answer.**

2. **[Extra Credit] Path Display[5 points]** Here you will also display the path (in a distinct color) that connects the source and destination actors.

   Implementation Details.

   - For this you need to keep track of the parent node of each node that was visited during the BFS traversal. Again, a hashmap $Parent < String, String >$ is maintained. When the traversal visites 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.
   - When tracing out the path (destination actor to source actor), highlight the nodes and links, similar to what was done in project 2-1.
   - Use a distinct color to highlight the path, using the line thickness element size attributes.

Evaluation:

By interactive demo; a schedule will be set up to test your program by the teaching assistant.

Grading Rubric

- Read in input dataset (large graph) : 3 points
- Implement BFS algorithm with provided queue implementation : 5 points
- Highlight source and destination nodes, visited nodes as specified in description : 5 points
- Documentation : 1 point
- Survey : 1 point
- **[Extra Credit(optional)]**: 5 points

Submission Requirements.

Turn in your source code to Canvas by the deadline; ensure it is well documented.