The Inequality of 2016 Hollywood, Need We Say Anymore??

@Author Dorrie Peters, @Author Nessa Tong, @Author Isabelle Whetsel

Thank you to our instructor, Professor Takis for teaching us much of what we know and always providing us with funny jokes and reminders such as “Protect your code from little bastards.”

Introduction

In a society fraught with gender inequity and rooted in patriarchal foundations, it is no surprise that the disparities seen in every field across the globe also seep into the film and media industry. From the male-dominated distribution of movie producers and directors, to the rampant sexual misconduct that sparked the #MeToo movement, Hollywood is no stranger to gender gaps and sexism.

The Bechdel Test explores how misogyny affects the characterization of women on screen. Based on a comic strip written by Alison Bechdel in 1985, the Bechdel Test started as a satirical guideline for good female representation in movies. To pass the test, a film must feature at least two women, and the women must speak to each other at least once about something other than a man. Despite the low standard, a shocking number of movies, including *The Avengers*, *Lord of the Rings*, and *Breakfast At Tiffany's*, fail the Bechdel Test, which eventually became a mainstream measurement of media gender representation.

Variations of the Bechdel Test include additional rules, such as ensuring the female characters have names, or that they share at least a minute of screentime. In 2017, *FiveThirtyEight*, an opinion poll website, reached out to women working in film and compiled eleven different alternatives to the Bechdel Test. They found that there are countless conditions that can be used to analyze gender representation in a movie: Did a woman direct it? Was the cast half women? Were the leads half women? What is the percentage of people of color? How about women of color?

In this project, we created a new alternative to the Bechdel Test, which judges a movie based on the proportion of women actors in the cast. To pass, we decided that a movie must have at least 50% of both its leading as well as supporting cast identify as female.

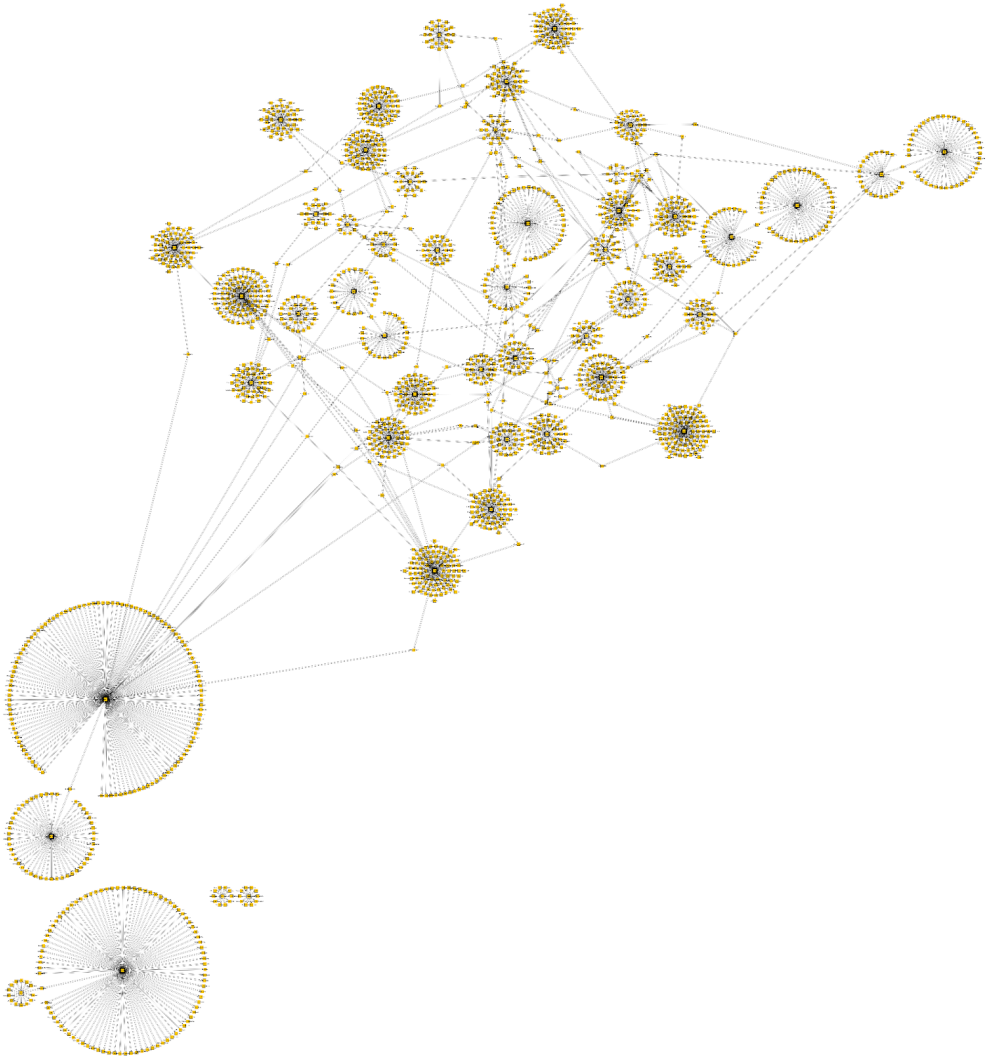
Methods

To begin with, we created an Actor class that kept track of an Actor’s name, gender, and if they were leading or supporting. We also created a Movie class that kept track of the title and cast list, which was filled by an addActor method and stored in a LinkedList of Actors. We decided to create new Actors for every movie in order to denote their role as a lead or support, so some actors were duplicated in our Movie representations.

In the HollywoodGraph class, we created a class that takes in Actor objects and the Movie they star in. We extended our HollywoodGraph class to the Adjacency List Graph class so we had access to all the same methods and also were able to add in our own instance variables and methods. When creating a graphical representation of actors and movies, we decided to store vertices as String objects. This helped ensure that every vertex in the graph was the same object type, allowing us to easily compare different vertices without worrying about their type. This also made it easier to catch duplicate Actors — even though the Actor class includes traits such as gender and leading/supporting roles, we don’t need it in the graph, so it is not important to differentiate between different roles an Actor has played as. Although the graph should be bipartite, since the vertices can be separated between Actors and Movies, it was both

impossible to differentiate them using yEd as well as unnecessary, since the number of actors in a movie is much greater than the number of movies that an actor stars in, so individual movies can be picked out by their large number of arcs (see below).

Because the objects stored in the graph representation were all Strings, this made it simple to create the lists of movies and cast for Tasks 2.0-2.1. For the latter, we needed a way to access the adjacency lists of specific vertices in the graph, so we added a method in AdjListGraph that takes any element in the graph and returns a linked list of all vertices adjacent to it. Back in HollywoodGraph, we used a method that would call this getter for a specific actor. Using these methods in Task 2.2, we were able to find the shortest path between actors by performing a breadth first search and then tracing back the arcs to find the degree of separation between the vertices. Finally, in Task 2.3, since we also store a list of Movies and Actors added to the HollywoodGraph, we simply iterated through the cast list of the Movie that this test was called on. We counted if the number of female leads was greater or equal to the remaining number of leads and if the number of female supporting actors was greater or equal to the remaining number of supporting actors. Unsurprisingly, none of the movies passed our “very rigorous” test. However, we did test it with a made-up movie that had at least 50% of leading and supporting actors that were female, which passed successfully.



Conclusions

Working together on this project, we learned how helpful collaboration can be in designing large scale projects. Throughout our group discussions, we realized that there were multiple possible ways to tackle each problem, and by having each other's ideas to bounce off of, we were able to generate solutions that we found were practical and efficient. We addressed almost every aspect of our program together, from small things like "What kind of collection will we store the list of actors in?" to larger scale items like "How and what items will be inputted into the graph?". For example, we deliberated for a while on how best to represent actors in our Actor class — we couldn't decide whether to make each Actor object represent the human actor as a whole, with a list of all their movies stored, or as a single role in a specific movie. After much debate, we ultimately decided to go with the latter since it was the easiest option for preserving the actor's role in the movie, and because of the ample discussions we had, we all ended up with very clear understandings of what our classes looked like.

Writing programs for such a complicated task also taught us the importance of pseudocode. During our discussions, we would periodically type our ideas into pseudocode. Although we didn't end up using every planned method or body of code, the pseudocode gave us a guideline for our actual programming. It also allowed us to view our entire program from a macro perspective, again leading to full understanding and agreement on the structure of our code. Additionally, doing so allowed us to analyze the efficiency of our program — because we could see from the big picture, we were able to spot redundancies that might waste space or increase the runtime of our code.

This project showed us how object-oriented programming can be used to model real world subjects, giving us the tools to analyze data relating to these representations. Knowing that we only wanted to know an actor's name, gender, and whether or not they had a leading role in a movie, we filtered through extraneous data, such as the name of the movie character and their billing. So, while our representations of movies and actors were simplified, they retained the exact amount of information that we needed to complete our tasks.

The results we produced very clearly demonstrated the disparities between men and women in the film industry. The standard we used was considerably higher than the original Bechdel Test — we required that the leading and supporting cast of a film have at least 50% women. In a perfect world, around 50% of movies should pass this test, with some being male cast dominated and others being female cast dominated. Unsurprisingly, not a single movie passed our test. Although it is not inherently uninclusive to have a cast with less than 50% women, the fact that no movies were able to achieve this benchmark shows an undeniable imbalance between gender representations. Oftentimes, the default casting of a character is male, even if gender has no impact on their character. For example, why do *The Secret Life of Pets*, *Zootopia*, and *Sing*, all movies starring talking animals, need so many "boy" animals? While the original Bechdel Test analyzes the complexity of female characterization in movies, our new test aspires for arbitrary female casting of characters that have anywhere from no to shallow to deep characterization. We hope that someday the default will no longer be male casting, and that women playing any role is normalized, rather than mainly to fulfill traditionally feminine roles (e.g. mothers, love interests, sexy spies, maidens that need saving, etc).

Collaboration

In order to properly discuss the tasks we needed to complete for this project, we met up multiple times in person and worked on most of the coding together.

For Task 1, we discussed in detail what our classes were going to look like and each drew basic UML diagrams of our ideas. Once we decided on the specific make up of each class, we used LiveShare on VS Code to work on the program simultaneously. Since most of the methods in the Actor and Movie classes were simple and already planned out, each member of the group took on a couple methods on our own, although we continued to discuss while writing code. Next, we worked together to create the HollywoodGraph class. Since this class was much more complex than the Actor and Movie classes, we spent much longer discussing the structure of the class as a whole as well as how we were going to implement each method. Throughout our meeting, Isabelle noted our ideas in pseudocode, and once we nailed down our ideas, we used LiveShare and also passed around Isabelle's laptop (since Nessa's could no longer connect to LiveShare for unknown reasons) to write the code. We then split up the documentation for each class, with Dorrie, Nessa, and Isabelle writing the Javadoc for Actor, HollywoodGraph, and Movie respectively. Finally, Isabelle ran the code from her computer and created the yEd graph, which was uploaded to Gradescope.

For Task 2, we again met up to discuss the planning of our next methods. We narrowed down the specific approaches we would use for each method, such as adding a getAdjList method to the AdjacencyListGraph class for 2.1 and using Breadth First Search for 2.2, before outlining the methods from our laptops individually (we gave up on LiveShare). We all agreed on creating a new Bechdel Test based on the proportion of female actors in the leading and supporting cast lists for 2.3. Then, Dorrie, Nessa, and Isabelle typed out the code for Task 2.1, 2.2, and 2.3 respectively based on our planned outline. Once all the programming was completed, we sent all the code to Isabelle, who added them to the HollywoodGraph class and did the final debugging and running of our program. Finally, we all wrote in the Final Report.

Sources

Bechdel, Allison. "The Rule." Dykes To Watch Out For, 1985, dykestowatchoutfor.com/the-rule/.

Hickey, Walt. "Creating the next Bechdel Test." FiveThirtyEight, 21 Dec. 2017, projects.fivethirtyeight.com/next-bechdel/.

Neville, Lois. "What Is the Bechdel Test? Movies That Pass & Fail | Backstage." *Backstage*, 30 May 2023, www.backstage.com/magazine/article/what-is-the-bechdel-test-75534/.