

Project Summary

In this project, I used verbs in the movie script as the relevant phrases related to the movie “The Interview”. The movie script text file can be found in the project folder. I’ve also made a video walkthrough, the video is in unlisted mode and can be accessed only via this link: <https://www.youtube.com/watch?v=rHdO4oIXWH8&t=1s>

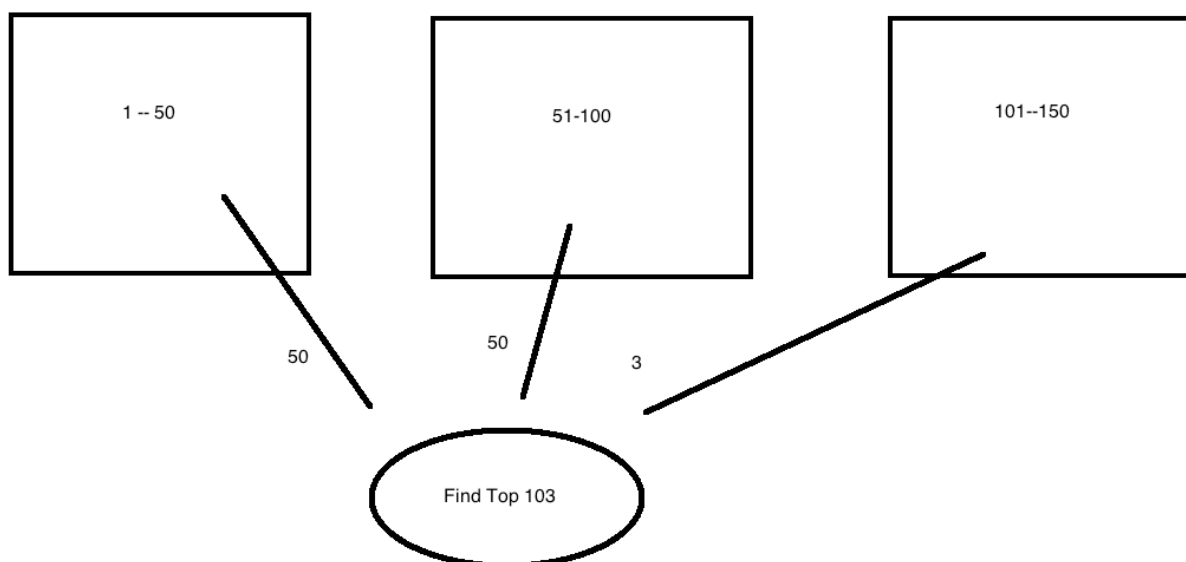
Preprocessing Stage:

To filter out irrelevant words and keep only verbs, I used Stanford coreNLP library to assign parts of speech to each word. After filtering out the words, I used a Hashmap to keep track of word occurrence. To sort our verbs in descending order, I first wrapped the words and their occurrence up as a class node and then feed into a priorityQueue, whose comparator is rewritten to reorder words based on their number of occurrence. Lastly, I output the sorted words into a file. (Doing NLP filtering and sorting took about 30-40 minutes on my Mac Air laptop, saving the result can be reusable for later development and testing)

Find x most relevant algorithm:

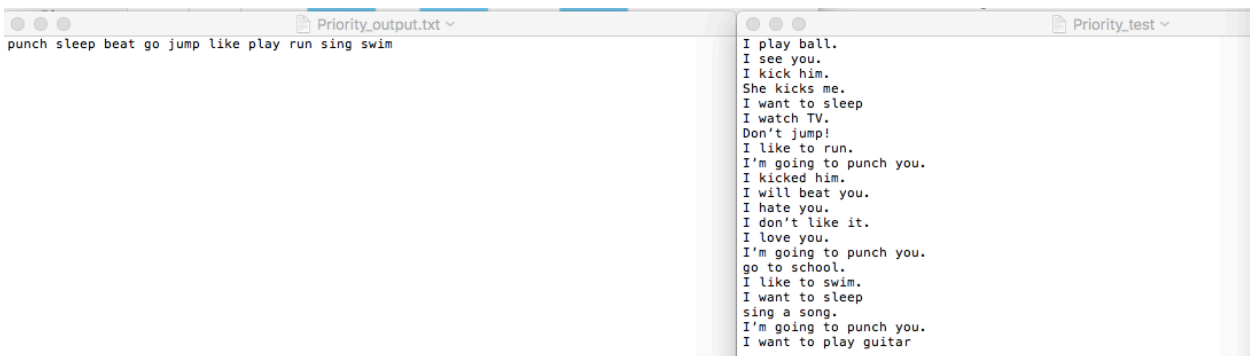
The naive top x relevant algorithm would just be loading the sorted words from previous output file and then iterate x elements. The downside of this is that consecutive calls on the function could result in redundant work. For example, if we first call the method to print the top 200 verbs and then the second time we call the method to print the top 199 verbs, the only one element difference would lead us to do the same work again.

To avoid this problem, we need to do some kind of memorization. So my approach is to partition the total number of words into different intervals and use a Hashmap to store intervals of words. For example, I stored the string of words whose occurrence between top 1 to top 50 in bucketID 1, and range between top 51 to 100 in bucketID2. So if we want to find top 101 words, we can just load top 100 words from bucket 1 and bucket 2 combined, and then get one extra word from bucket 3. The interval bucket is filled during the initialization of object instance, so consecutive calls on the method would be pretty fast.



NLP ANALYSIS:

Inside my project folder, I included a sample Priority_test file and a Priority_output file to test if the NLP algorithm correctly labels all the action verbs and also test if my algorithm successfully sorted the words. Unfortunately, even Stanford coreNLP library cannot 100% label verbs correctly. It assigns part of speech to word based on the context of the sentence and makes mistakes sometimes. I would say the verbs it identifies are mostly correct, but the number of occurrence might be slightly off for some words. Thus, I decided to use the same sentence for each verb occurrence in this small testing file so that either the verb doesn't count or it would give me the correct occurrence. With the NLP flaw in mind, I designed the following small test case, and it's showing me that the functionality of sorting words based on occurrence is working as expected.



```
Priority_output.txt
punch sleep beat go jump like play run sing swim

Priority_test
I play ball.
I see you.
I kick him.
She kicks me.
I want to sleep
I watch TV.
Don't jump!
I like to run.
I'm going to punch you.
I kicked him.
I will beat you.
I hate you.
I don't like it.
I love you.
I'm going to punch you.
go to school.
I like to swim.
I want to sleep
sing a song.
I'm going to punch you.
I want to play guitar
```