**Administrative:**

Team Name: The Jam Club

Team Members: Bishoy Pramanik, Brayden Smith, Johnathan Gaskin-Paulsen

GitHub URL: https://github.com/bishyboi/DSAProj_3

Link to Video: https://youtu.be/IVO-nNCpO8U

**Proposal:**

Problem: How common have swears become over time in music?

Motivation:

> As new music comes along with each new generation, older generations often claim that music has become more inappropriate over time. To test this assumption, we realized that we could analyze the prevalence of explicit lyrics in songs.

Features:

> Our program is able to find the average number of explicit lyrics in a song per year, across several different languages. Our program supports a search across thirty-six languages, including English, Spanish, Portuguese, Romanian, German, and many more. The search ranges from 1950 to 2020 for each language.

Description of Data and Tools:

> We used the WASABI database, a collection of songs and their metadata from several other databases on the internet, that have already been processed for lyrics, chord sequences, and emotions using natural language processing techniques. For quick overarching scans of data, we used the pandas library for Python to determine other characteristics of the dataset, as well as filter out the years that did not have a year attached, as well as songs that were not created within the timeline we were looking for, (1950-2020).
>
> Using SFML, we created the GUI for the program and the rest was implemented using the STL library in C++.

Additional Data Structures implemented:

> A Max Heap and HashMap were used to store the song data.
>
> The HashMap used a Hash function which combined sets of two characters in ascii and then summed up all pairs E.X. if the first two letters were represented as 70 and 71 then 7071 would be added to the sum. Finally doing the modulo operator on the final number gives you index placement of that language. Each element of the hash container is a list containing all songs that belong to that language. Since there are not many languages and all songs of the same language go to the same home index we used a linear collision policy of one as collisions are unlikely.

The Max Heap stored node structs which held a list of song structs for each year. These node structs were sorted by their year values. Beyond each node being a list of songs that year this is a normal max heap. There is no sorting of the data in each nodes list beyond the order they were added to the structure.

Distribution of Roles:

Brayden: Wrote code for the GUI using SFML and Hashmap.

Bishoy: Wrote report, algorithm for the Heap, and helped preprocess data from database.

Jonathan: Wrote the Heap data structure from scratch, wrote the report.

**Analysis:**

Changes and Rationale:

- Problem statement
    - After analyzing the data provided from the database, the problem statement was revised from average amount of english words in other languages to due the lyrics of songs no longer being provided in the data because they were copyrighted and the WASABI database is open source. This challenge made it impossible to test with the given data.
- Motivation
    - Due to the revision of the problem statement, the motivation was changed to match this new goal.
- Features
    - The graph idea was maintained, however, the data on the graph was changed to match the new problem statement, showing average explicit language in given time periods versus number of english words.
- Data:
    - Due to not having access to the song lyrics, the previous dataset needed for the list of english words was no longer needed.
- Tools:
    - The Qt GUI library was swapped in preference for the SFML GUI library due to having a better understanding of SFML.
- Data Structure/Algorithms used
    - The B-Tree was traded in favor of the heap due to a better understanding of the heap and time constraints.
- Distribution of Work:
    - Tasks were shifted around and allocated on the basis of who was the best fit for each task and who was available for them. For example, initially Bishoy was responsible for parsing data, but due to fortunate timing, it became easier for Brayden to finish the task.

**Time Complexity:**

<u>Heap:</u>

HeapifyUp:

In the worst-case scenario, the node is at the end of the heap array, and is the smallest node in the max-heap. Because the algorithm will repeatedly swap child and parent nodes and traverse the tree through ancestors, *heapifyUp* has a time complexity of O(log(*n*)) where n is the number of years in the array.

Insert:

In the worst-case scenario for an insertion, the year of the node does not exist in the heap, requiring the function to check each index in the heap array, giving at least an O($n$) time complexity. If the node does not exist, then a new node is created, O(1), and then *heapifyUp* is called, which has a time complexity of O(log($n$)). Because these are not nested functions, the overall time complexity is O($n$ + 1 + log($n$)), reducing to a time complexity of O($n$) where n is the number of years in the heap array.

Algo:

The function first iterates through every node in the heap, where nodes describe a list of every song that was made that year from every language. Within each node there is $s$ songs made for that year, and $s_{total}$ is the total amount of songs made through out all years or the number of songs stored in the data structure. At each song, if the language matches what it is searching for, the function increments a variable to keep track of the explicit lyrics for each year accordingly. For these reasons, this first half of the program gives a time complexity of O($s_{total}$).

The second half of the function iterates through the explicit lyrics found for each year and finds the average explicit lyrics per song, iterating over $y$ years. For this reason, the second half of the function has a time complexity of O($y$).

Because these halves are not coupled with each other, the overall time complexity of the function is O($s_{total}$ + $y$).

<u>HashMap:</u>

Insert:

In the Worst-Case Scenario, the function will resize the hashmap due to the load factor being 1.0, followed by inserting a new element at the index of the new hashmap as the current language is not in the list. For resizing the hashmap, this would be O($C$), with $C$ in this case being the current capacity of the hash map, as in this scenario, the list is full and would need to be placed into a temporary list while the main container is resized. Following this, each index of the temporary list is checked and if a language list exists there the entire list is moved over to the new bigger hash map. Once the resizing is done the song to be inserted creates a new list for a new language in the worst case. Since this is constant the final complexity is O($C$).

Hash:

The function sums the ASCII values for each set of two characters in the string, and then finds the modulo of the sum with respect to the capacity of the hash function. Because the function iterates through every character in the string and modulo is a constant-time operation, the time complexity of the hash function is O($c$), where $c$ is the number of characters in the string.

Search:

In the worst-case scenario, a linear collision occurs when getting the hash value for the specified language and linear collisions keep occurring, causing the function to iterate through every index in the hash map until it finds an empty index. For this reason, the search function has a worst-case time complexity of O($C$), where $C$ is the capacity of the hash map.

Algo:

The function first calls *search*. Since we are using the worst case time complexity we get a time complexity of O($C$) where $C$ is the capacity of the hash map. It then stores the value into a vector, with $S$ many songs that are in the specified language. The function then iterates through each song in the vector of songs and increments the variables storing the average number of explicit lyrics accordingly, giving O($S$) time complexity. The function then iterates through each year of songs and takes the average of the explicit lyrics in a song per year, giving O($y$) time complexity where y is the number of years in the range of data. Because these time complexities are not coupled with each other we can sum them, giving a time complexity of O($C + S + y$), where $s$ is number of songs in a specified language and $y$ is the number of years in the timeline being analyzed and $C$ is the capacity of the hashmap.

**Reflection:**

Bishoy:

Overall, I enjoyed the freedom of the project and the direction we were allowed to take it in. My team communicated very clearly the goals and we met frequently to adjust time frames, expectations, and goals making the workflow very seamless. Each of us brought different skills to the table, allowing us to make up for each other's weakness in the project. If I were to do this project again, I would not make any changes to the workflow, as I think using GitHub for code sharing and using Zoom to meet frequently was extremely effective. In terms of changes to the project, I only wish that we had a more holistic database for songs that allowed for more complex lyric analysis, so that we could have asked deeper statistical questions to find trends in the data. I learned that constant and clear communication is vital in creating an effective environment for teamwork.

Brayden:

Experience overall went pretty smoothly. The biggest challenge that I faced was trying to parse the original massive csv file as it was near 5 GB and in a formatting that was original to itself. Once I could cut away most of the unused data it became easy to use. The other hard part of the project was that I didn’t know if a part of the code worked until the full functionality was done and I could see the data loaded into the graph.

Johnathan:

The overall experience was very smooth. We were able to delegate work to each member of the team to get the project done efficiently. The Biggest Challenge for me was software challenges. I had trouble attempting to load the code from github and while I was able to contribute the final product, I never was able to load the full project on my Laptop.

**Reference:**

Michel Buffa, Elena Cabrio, Michael Fell, Fabien Gandon, Alain Giboin, et al.. The WASABI Dataset: Cultural, Lyrics and Audio Analysis Metadata About 2 Million Popular Commercially Released Songs. The Semantic Web. ESWC 2021. Lecture Notes in Computer Science, vol 12731., pp.515-531, 2021, ⟨10.1007/978-3-030-77385-4_31⟩. ⟨hal-03282619⟩

https://www.sfml-dev.org