Nathan Liriano - calluste

Alex Jamison - AlexKJ101

https://github.com/calluste/Project-3-DSA

https://youtu.be/W6xKw5zn4qc

DSA Project Report

Extended and Refined Proposal

Our project aims to make an efficient way to search for music and organize a massive dataset of music. As time passes more and more music is made and this creates a problem because as more songs are made the longer it will take to search up a song that you want because the computer will have to check through more and more songs to find the one that you were looking for. Our Song search engine helps to make it much easier and more efficient to find the song you're looking for while giving the users an easy to use interface. The main feature of our project is the ability to search any song available on spotify and get a list of songs with that name really quickly. Another one of the features is the dynamic line cursor that helps the user indicate where they are writing. On top of that the Graphical User Interface (GUI) has a text box for a more easy to use interface and updates the input field in real time. The search engine gives you tons of results and we condense it down to 5 in a very easy to read list. For our data set we used Spotify's million song dataset from kaggle. This dataset contains an enormous amount of songs along with their author, album, and lyrics. For our project we focused on the name and author of the songs. For the project the language we used was C++ because we both feel the most comfortable with it and it allows us to use

SFML which we used to build the GUI. It was coded in Clion to make debugging easier, and Github to allow for us to collaborate on the project. The data structures used were a Trie built from scratch and a standard library unordered map.

Nathan took the lead in designing the Graphical User Interface using SFML, making a seamless and easy to use interface for users. He also managed loading in the dataset and putting it into a container. Along with that he also contributed a significant part of the analysis. Alex's main focus was the development of the algorithm used in the search engine, the most critical part of the project's functionality. Also Alex took charge of the video by recording and producing the video demo, showing all the features and functionality of the search engine.

Analysis

After the initial project proposal we began working on the project and found several areas for improvement and made pretty significant changes. The main change made was instead of just a simple console based UI we went with a fully functional GUI. A GUI allows for a better user experience and lets the users do real time searches along with allowing us to implement many of the other features included like the line cursor to tell you where you are typing or any of the other interactive features. Another major change made was a restructuring of the dataset. We removed unnecessary columns with information not needed and that slowed the system. Their removal allowed for a much faster load time and much lower memory usage. This optimization leads to a much better and more efficient search engine, which is fairly important with a dataset of this size. The Big O Time complexity of the loadSongs function is O(n). It's

O(n) because the function reads each line in the csv, parses the string and creates a single object for each one, which would make it a time complexity of O(n) because it does all that for each line exactly once. The n being the number of songs. The insertion of items into the Trie had a time complexity of O(L) L being the length of the song name. This is because you traverse the tree from the root and create child nodes for each character in the song name. For searching the tree it is O(L+N*K) where L is the length of the song name, N is the number of song names that match the prefix, and K is the average length of the remaining characters in each matching word beyond the prefix. This is because searching for a prefix in a trie involves traversing from the root node down to the node corresponding to the last character of the prefix and then you must collect all the words with the given prefix. This involves traversing all descendant nodes of the prefix node. For the unordered map, the worst case time complexity for insertion is O(N) assuming there are collisions. The worst case for searching the map is O(N*L) where N is the number of keys and L is the amount of character comparisons per key. When none of the keys match early in the comparison, you end up comparing all L characters for each of the N keys.

Reflection

As a group the overall experience of this project was very challenging but also really cool. It was cool because it allowed us to take the things that we learned in class and apply them on a real world project with a real life dataset. It also served as a way to challenge us in our coding and help better it with just more experience. The project taught us a ton on designing a GUI that's friendly to use while also being functional.

Along with the main portion, processing and using an extremely large dataset similar to those we will be using in the field. Overall our experience with this project was very positive and we learned a ton of things that just come with doing these kinds of projects. One of the biggest challenges we faced when doing this project was implementing a functional GUI that is easy to use for users. The line cursor was one of the ones that had me stumped for a bit and required me to think a little outside the box. Another problem I ran into was trying to figure out how to get each letter to show up in real time but also the line cursor still moves over after it's typed . On top of these the main problem we ran into was the Trie implementation. It was extremely difficult to try to understand how to properly and efficiently implement the word collection after the prefix was given but after doing some research we were eventually able to figure it out. If we were to start this project all over again there are a couple things we'd do much differently. To begin, we would start much earlier to allow enough time to go through and test every outcome and make sure everything is covered. Also if we started earlier the GUI could have been even better and more streamlined. Also with more time we could have tested more and maybe made the search engine even faster. On top of starting earlier we would have also had more of an emphasis on the planning stage and spent more time designing the project in the planning stage. A more clear breakdown of each person's responsibilities would have also helped. Nathan has gained understanding of creating a GUI and also gained knowledge and experience in loading data in from files like a CSV. Alex gained a surface level understanding of a wide variety of data structures from the research that went into trying to find which one would be best to search a vector of strings. Additionally, he also learned a lot about how tries work and why they can be so efficient for searching a dataset for specific substrings.