Group Name: Gekko

Created by: Claudio Sciotto & John Torres

GitHub: https://github.com/fisizion/GameCAAD

Video: https://youtu.be/CtIiKUHlEuY

Video Game Finder

Extended proposal

Problem: When looking for the next video game one wants to play it is difficult to choose between hundreds of thousands of games out there, so we wanted to come up with a program where the user could input the genres of video games, they are interested in playing and the algorithm would output the best 5 rated games that has those genres. Specifically, the program searches for games in the IGDB.com database and through an API it returns the information needed. Then the algorithm stores the games in a HashMap and B+ tree data structure so later we can search for the necessary information.

Motivation: We believe that this is a problem for the gaming community since from personal experience we can say that every now and then when people are looking for new a video game to play it becomes a tedious process where one can spend hours searching online getting plenty of recommendations without any good answer. Sometimes those recommendations are biased or very broad, making the process more complicated than it should be. We want to simplify that process and give a concise list of the most compatible games based on the criteria provided by the user.

Features: The program will feature a basic command line interface menu that prompts the user to enter the number of genres in a video game they are looking for, followed by a prompt to enter the genres with a list showing all of them, and lastly it will output an ordered list one games based on the best 5 rated games that has those genres.

Description of Data: For the data we used the IGDB.com API to retrieve all the available video games on the website. The total amount of points that we retrieved was around 230.000 games. We also retrieved through the API their corresponding genres and ratings. Then we stored them all in a text file and through a regex algorithm sorted them into another text file that was easier for us to use with delimiters. Each genre has its own text file for easier organization.

Tools/Languages/APIs/Libraries: We used CLion for all the C++ code and IntelliJ IDEA for the java code. C++ is used for the data structure and regex algorithms and java is used for the IGDB API since it wasn't available for C++. The Maven Framework and okhttp Library were also use in API algorithm were Maven itself was used to download and implement okhttp and okhttp was used as the required HTTP client to connect to API we were using. Lastly, we used GitHub to store the project and simultaneously work on it.

Algorithms/Data Structure implemented: For the project we created two data structures and compared their efficiency for the retrieval of the data. One data structure was a HashMap and the other one was a B+ tree. Both structures contain the fundamental properties created from scratch. For the HashMap, each video game genre has its own HashMap that stores each title to their respective slot. Separate chaining was used to store titles that would have repeating hashing keys. Similarly for the B+ tree each genre has its own tree that stores the titles in respect to their corresponding ID's.

Distribution of Responsibility and Roles: Who did what?: The overall idea was 50/50 Both creators worked together to form the concept of what we wanted to build. As for individual responsibilities, Claudio implemented the API and took care of all the Java code, he also created the HashMap data structure with all of its implementations. John created the filtering algorithm that stored the retrieved data from the API to an easy-to-read text file by the use of regex, he also implemented the entirety of the B+ tree data structure.

Analysis

Any changes the group made after the proposal? The rationale behind the changes.

The biggest change made for the project was the shift from using the originally planned Steam Website API to the IGDB API. There were several reasons for having this change. The first reason was that Steam itself, although it contains a large database of video game titles, also lacks a large part of the total video game titles to date, including games from exclusive companies like Nintendo and Sony for example. Also, a large portion of independent or old school games that aren't available in the Steam Store would have been missing from our project. After some research we found that IGDB would be a more accurate database to pull our data from since it would be more complete for our purposes.

The second reason for this shift was that Steam's website API was very complex and had many requirements and features that were unnecessary for our project. The amount of effort that it would have required to make it work would have been much greater and it would have still been incomplete since their API wasn't made for what we were trying to achieve. On the other hand, IGDB's API was exactly what we were looking for, without any extra features or complex requirements. Simple retrieval of video game's data with their appropriate rating and genres.

Another change made to the project after the proposal was the data structures that we used. Originally we had planned to implement a B+ tree and compare it to some graph, but after thinking about the logistics of the algorithms, we realized that for our purposes there wasn't much to compare. They were two completely different data structures that worked in very different ways, also for this project creating a graph would have been very inefficient since the data simply doesn't fit well in that scenario. After some thinking we realized that a HashMap was perfect for storing video game titles with their respective ID's, genres, and ratings. We could put their unique IDs through a hashing function that would then store the video game's node to a respective slot. If we were to implement a graph there wouldn't have been any way of connecting the nodes (making them unweighted) and it would have been very similar to just looking for a game in a linear data structure.

The last change we made was moving from a GUI to a CLI. This change was made mostly in the interest of time since we had other projects/exams for other classes that were close to the deadline of this project. However, we believe that this doesn't affect the quality of the algorithms at all, just the presentation. We created a command line menu that prompts the user with several questions asking first the number of genres that they would like to input followed by a list of all the genres available and a prompt to select the ones they would like.

Big O worst case time complexity analysis of the major functions/features you implemented.

The Data Collection portion of the project is web-based and goes through the current number of games the database has. The Filter Data portion of the project, where we go through the entire collected data and organize it in text files to use as inputs for our data structures, so that would have an **O(n**) where n is the total numbers of games collected.

<u>HashMap Time Complexities</u>:

- The overall **Insertion** functions have a time complexity of **O(G * TN * CN)** where G is the number of genres in the node that is being inserted, TN is the total number of nodes in the input files and CN is the number of chained nodes at the current key values, since the function uses separate chaining to resolve collisions. The nodes are being inserted simultaneously as the input files are being read. Note: Since there are several indented classes the different insertion functions work together since one calls the other one, so the time complexity corresponds to the overall sum of the insertion functions.
- The **searching** functions have an **O(IG + TGN)** where IG corresponds to the number of input genres the user wants to look through and TGN corresponds to the total number of nodes (or games) inside a single genre.

<u>B+ Tree Time Complexities</u>:

- **Insert O(log(n * m))**: where n is the amount of games saved on a node, m is the amount of leaf nodes there are. This is the case because for any given node, if the node is not a leaf, the code has to branch to the correct branch from a given list of nodes with a maximum size of 3 keys and 4 children, and then, when the correct leaf node is found, look through the keys.
- **findMatch O(n*log(m * n))**: this is the case because although it is implemented as a b+ tree, the algorithm still needs to go down the tree to find the first leaf node, and from there keep going to the other ones.

<u>Reflection</u>

As a group, how was the overall experience for the project?

Overall, the project was very challenging since many of the concepts and tools we ended up working with and using were never taught to us. How to implement APIs and set up connections to online databases were all new. Also, things like settings up GitHub were barely explained in programming 1 but never really taught in dept, so figuring out GitHub without losing our entire project was scary. However, the project was also very rewarding, after many hours of work it was satisfying to see everything working together. It feels good to know that we did something we think is useful and that we will be able to put on our resume.

The project was also very liberating knowing that we were able to choose what we wanted to do, of course, as long as it met the requirements, the idea was fully our own. Compared to the other projects where we had to do a very specific task implementing specific functions among other things.

Did you have any challenges? If so, describe.

The biggest challenge was figuring things out on our own. As mentioned before, many of the things we ended up doing were never or barely explained in the past so it required a lot of discipline and work learning how to do those, however we believe that this is good since it reflects how to real world is going to be. After we graduate there will be no Professors or TAs that we can go to, to ask for help or debugging. It required us to concentrate on the topics and fully understand what is happening in order to make something work.

If you were to start once again as a group, any changes you would make to the project and/or workflow?

The first thing we would have done differently is to have a clear idea of what the step-by-step process is going to look like. We always had a clear idea of what the goal was going to be and what we wanted our program to do but we didn’t think to much about the entire process and implementations, we had general concepts and ideas on our minds, but nothing concrete. This brought us many issues when thinking about what we had to do and what we had to avoid in order for everything to work. We spent a lot of time after starting to work on the project figuring out what data structure we were going to do or even if we should just scratch everything and change to a sorting/searching project. If we had had a clear workflow from the beginning, it would have made the entire process and workflow much smoother.

We also believe that assigning roles since day one would have been another good thing to have for our project since it makes everything clear giving responsibilities to everyone equally based on their strengths or interests.

Comments

Claudio Sciotto: I think the project was very useful for me to learn the basics of how APIs and data collection work. I learned a lot about downloading libraries and setting up environments. I also learned more about HashMaps and what their strengths are and why they can be a very efficient data structure for certain scenarios. Overall, it was a huge learning experience.

John Torres: For me, the project was very good at improving my understanding of Regex. I had to implement a complex regex-based reading algorithm that needed to be capable of understanding what should and shouldn't keep. I also learned a lot about the fundamentals of B+ Tree, and how complex they can be to implement. While also learning more about it's pros and cons when dealing with large amounts of data.

References

- IGDB database: https://www.igdb.com/
- IDGB API: https://api-docs.igdb.com/#getting-started
- Maven Documentation: https://maven.apache.org/guides/
- OkHttp Reference guide: https://www.baeldung.com/guide-to-okhttp