Administrative Details

Team Name : CitySafe

Team Members : Jaylom Pairol (/jaylomp), Farid Mussulman (/FMuss), Aparna Sai Nimmagadda (/AparnaSai1)

GitHub Repository Link : https://github.com/jaylomp/DSA-Project-3.git

Video Demo Link: https://youtu.be/32LvHpeqRDA?si=wfu4pF2R8cLxHoJZ

Extended & Refined Proposal

What problem are we trying to solve? We are addressing the challenge of efficiently predicting and preventing being victims of crimes in urban environments by analyzing trends and patterns in historical crime data. Crime is a significant concern, and having awareness of which areas are safer than others can help users make informed decisions about relocation, traveling arrangements, and various other situations.

There are numerous reasons why someone may want to know how safe or dangerous a certain area is. Whether they are relocating and looking to find housing or planning a trip, crime can be more rampant in some parts of a city compared to others. This tool would be helpful in decision-making in many scenarios. For instance, one might decide if a place is suitable for solo travel or research apartments before signing a lease. This tool aims to provide users with the necessary information to make these decisions safely.

The solution is a menu-driven tool where users can enter the proper name of a location (like a county in Florida) and receive information of crime prevalence in the area. This functionality enables users to gauge the crime prevalence of the area they are interested in and make informed decisions accordingly.

We will use a public dataset to provide accurate and up-to-date crime statistics. For example, we will leverage data from sources like the Florida Department of Law Enforcement which covers detailed crime statistics by county over different time periods. This dataset includes various columns, such as crime type, location, date, and the number of incidents, which are essential for our analysis.

For the backend development, we will use C++, incorporating both a Hash Tree and a B Tree. The frontend is intended to be developed using ReactJS to create a responsive and interactive user interface.

We will implement both B Tree and Hash Tree data structures to store and manage the crime data. These data structures will enable efficient search operations, allowing users to quickly retrieve crime statistics for specific locations.

In addition to the primary algorithms, we will use various search algorithms to optimize the retrieval of crime data. When a location is chosen, each data structure's search algorithm will be called to find the corresponding crime rate. The time each search takes will be recorded and displayed along with the visual representation of the crime rate, providing users with insight into the performance of our system.

In terms of how the team members contributed to the project, Jaylom and Faris handled the backend development, focusing on data structures. Aparna conducted research for the initial problem statement and publicly available data sets. She contributed to the front-end development and completed documentation (report).

Analysis

We made a few changes to the project after the initial proposal as the primary solution included an interactive map of the state of Florida divided by counties where users can click on cities or specific areas within a city to reveal the prevalence of crime in that area. When hovering over an area, it turns red, yellow, or green, depending on the crime prevalence compared to other places. The rationale behind this change was the difficulty of integrating JavaScript into the project due to the amount of data fetching and rendering, especially with such a large data set. After much experimentation, there was not a feasible way to effectively integrate this UI using JavaScript given time constraints.

For the hash function in the HashTable, the worst case time complexity is O(n), where n is the number of characters in the county name. This is because std::hash traverses through each character in a string in order to create a hashed int, which is then modded by the size of the hash table. For the insert function, the worst case time complexity is O(n), where n is the number of elements in the table at the time of insertion. This is because in the worst case, the load factor is equal to the maximum load factor, resulting in rehashing, which is O(n). The rehashing function is O(n), for the number of items in the table because in order to rehash every element, it must first traverse through every bucket and list to rehash every element into a resized table. The search method in the worst case is O(n), where n is the number of elements in the list. This is because in the worst case scenario, all elements are conflicting, resulting in a single bucket of all the elements chained together, resulting in the need to traverse every element.

For the insertNotFull function in the B-Tree the worst-case time complexity is O(log n) where n is the total number of keys in the B-Tree. This is because the function needs to recursively traverse the height of the B-Tree which is logarithmic. For the splitChild function the worst-case time complexity is O(1) which is constant time complexity. This is because the

operation involves moving a fixed number of keys and children pointers. For the insert function the worst-case time complexity is O(log n) where n is the total number of keys in the tree. This is because the function needs to traverse down the appropriate leaf node, since the height of the tree is O(log n) that is what the time complexity comes out to. For the traverse function the time complexity is O(n) where n is the total number of keys in the B-Tree. This is because it has to visit every node in the tree. For the countFeloniesInCounty function the worst case time complexity is O(n) where n is the total number of keys in the tree. This is because the function has to potentially visit every key in the tree to count the felonies. For the buildFromDataset function the worst case time complexity is O(m log n) where m is the number of elements in the data set, and n is the total number of keys after the insertions. This is because each insert operation takes O(log n) and this is done for each of the m elements.

Reflection

As a group, the overall experience for the project was both challenging and rewarding. We navigated through the complexities of integrating various technologies. Each team member brought unique strengths and previous knowledge that made for a collaborative environment.

We encountered several challenges during the project. On the frontend side, one significant challenge was ensuring the interactive map accurately reflected real-time data from the backend. For example, there was a lot of data fetching that had to be done and overall caused issues with representing the data. Additionally, implementing the dynamic color-coding system for the map (red, yellow, green) based on crime turned out to be more complex than expected. There was oversight in terms of the statistical analysis that would have to be done as the county being a certain color would depend on the average.

If we were to start the project again, we would implement more robust project management practices from the beginning. This includes setting clearer milestones and deadlines, which would help with our time management and help with mitigating unexpected issues. For instance, if we ran into front-end issues earlier, we would have been able to come up with a more effective visual alternative compared to the menu-based system.

References

Data Set – https://www.fdle.state.fl.us/CJAB/UCR/Annual-Reports

SVG Image – https://commons.wikimedia.org/wiki/File:Forgotten_Coast.svg