Administrative:

Team Name: B Tree Farmers

Team Members: Jackson Culbreth, Ernesto Perez Garcia, Muhimin Majlis

GitHub URL:

Video URL:

Extended and Refined Proposal:

Problem: The problem we are trying to solve is related to crime itself. The specific topic/data we chose to analyze was slavery. Slavery was a massive industry that had many different data points such as costs, interest rates, buyer names, seller names, seller origin, buyer origin, and etc. The problem is crime itself. Using data structures, we want to organize the data and see where the hot spots of crime are using parameters such as an id and location. Then we can see where crime is the most prevalent and then help law enforcement break down on crime.

Motivation: This is a problem because crime is prevalent everywhere in the United States. Not all locations have the same amount of crime and even if they do, there are different types of crimes different cities and or states deal with. Local Law enforcement must be able to pinpoint the crimes committed using organized locations. In reference to our example, slavery was an illegal operation and people who commit illegal operations are good at dodging law enforcement because they have experience. Individuals who commit crime flee to cities or states that have weak law enforcement. Identifying those locations are what we are trying to do by implementing data structure to help organize the large data sets.

Features: COMPLETE AFTER GUI

Description of Data: The data set we analyzed are a distribution of statistics about slavery and its widespread seller/buyer distributions. The excel sheet has about 9600 rows and 22 columns. We will be using all 9600 rows and will be using 19 columns eliminating 3 redundant data. This leaves us with approximately 182,400 data points. We will be using the record\_id which is the row number to organize the nodes in our data structures. The data section of our nodes will include the 18 columns which is excluding the record\_id. The data section of each node/record\_id will include the BuyersCounty, BuyersFullName, BuyerCityofOrigin, BuyerStateofOrigin, SellerCounty, SellerFullName, SellerCityofOrigin, SellerStateofOrigin, SlaveAge, SlaveGender, SlaveName, SlaveColor, TransactionDate, TransactionTotalSlaves, TransactionSaleDiscount, TransactionPaymentMethod, TransactionPredictedIntrest, TransactionPrice. The link to the dataset: <https://think.cs.vt.edu/corgis/csv/slavery/>. We will organize and search using record\_id number and retrieve the information/data from that specific record\_id.

Tools/Languages/APIs/Libraries Used: For the language since all group members are the most familiar with C++ we used that language. We used C++ version 17. We used the standard C++ libraries. As for tools we used built in debuggers in our IDE’s [VsCode & Clion]. We used Git and GitHub to make project collaboration easier and more efficient. We used Discord and Messages to communicate with each other. The software we used are Excel, Corgis Datasets for Data, and Word. We used the standard C++ library along with the regex library for parsing.

Data Structures/Algorithms: The two data structures we are using for comparison are a Red black tree which is a self-balancing BST. The other data structure we are using a B+ tree that uses group and linked lists for organization. For the Red Black Tree and B+ Tree we used in order traversals to print the record\_id along with the packet/data related to it. The Red Black Tree has a search function that uses binary search using the record\_id as the only parameter. Then retrieves the whole packet with the data related to that specific record\_id. The Red black Tree is a BST so binary search is the only one that made sense. The B+ tree uses index-based searching and comparing the nodes in groups. This is done with multiple pointers. We are going to compare the algorithms/data structures by using a clock. We are navigating using the console.

Distribution of Responsibility and Roles: Ernesto oversaw creating the B+ tree implementation along with all the methods such as inserting, searching, and in order traversals. which he completed. Muhimin oversaw creating the Red Black Tree implementation along with all the methods such as inserting, searching, and in order traversals. Jackson oversaw the implementation of the GUI/Console/Webframe for the program/project and combining the two data structures. Jackson was also in charge of creating the testing of the performance of each data structure/algorithms. We all split the report into 3 equal parts. Everyone oversaw their own Big O analysis of their respective data structures. Jackson oversaw creating the GitHub and Video as well.

Analysis:

Group Changes: Instead of using SFML we are using a Console based visual. This is because we lacked knowledge about SFML, and it would take too much time to relearn it from Prog2. We also decided it was wise to keep the outer deliverables simpler, so we could truly focus on getting the code to work and the whole program together.

Big O Analysis:

Red-Black Tree: The constructor and destructor of the RedBlackTree has a time complexity of O(1). The destructor helper function destroyTree() has a time complexity of O(n) where n is the number of nodes in the tree. The left and right rotation functions have a worst time complexity of O(1). The insert function has a time complexity of O(logn) where n is the number of nodes in the tree. The insert operation involves traversing the height of the tree, and since the tree is balanced, the height is logarithmic. The time complexity of the insert fixup function is also O(logn). Where n is the number of nodes in the tree. Like the insert operation, the fixup operation involves traversing the height of the tree. The search function has a time complexity of O(logn) where n is the number of nodes. The search will traverse the height of the tree. The inorder traversal function is O(n) where n is the number of nodes. This is because every node needs to be visited in the worst case. The time complexity of the minimum red, black tree function is O(logn). Where n is the number of nodes in the tree. The minimum operation involves traversing the left child pointers until the leftmost node is reached, which is at depth log n.

B Tree: Time complexity for the B Tree varies by method. There were 3 main methods, them being insert(), search(), and inorderTraversal(), with insert() having a call to splitNode(). The worst case time complexity of inorderTraversal() is O(n), where n is the number of keys in the Tree. When I am traversing through the tree, I am accessing every key, to print their ID and Buyer State. Since I am accessing every key exactly once, the worst case is O(n). The worst case time complexity for search() is O(log5n). Where n is the number of keys in the Tree. As a search through the tree, I am making a choice, either go left, middle-left, middle, middle-right, or right (this is the case because my N = 5 for this B Tree). Since I am diving the choice every time by 5, there’s a logarithmic time of O(log5n) in the worst case. In the case of insert(), in the worst case, it is O(log5n) as well. It involves descending from the root to a leaf, which takes time, as each level in the tree can have at most m children. If a split occurs, it might go up to the root, causing the tree to grow in height. However, this is infrequent, and the overall amortized complexity remains O(log5n). All other functions and methods are constant in time.

Reflection:

Overall Experience: As a group the overall experience for this project was stressful. We learned a lot about collaboration and communication which we will need as future software engineers. We will have to work closely with senior engineers and have deliverables and be ready to help whenever we can or whenever we are needed.

Challenges: Making sure everyone schedules lined up so we can have group meetings were difficult. Making sure we held each other accountable was also a challenge. We all had multiple deliverables and one deliverable sometimes depended on another person’s deliverables. Planning this project around the work we had around our other classes was so difficult. Another big challenge was wrapping our heads around the idea of a B+ tree. Implementing a b+ tree was even worse of a challenge. We spent multiple days trying to debug the insert and search functions.

Workflow Changes: If we were to start all over again, we would not choose to implement a b+ tree because of the sheer amount of knowledge and debugging it takes. The B+ tree is one of the more difficult data structures and implementing algorithms using it was a challenge. So, in summary we would probably choose another data structure. We would also implement a better GUI using SFML if we still had the knowledge from Prog2. Also, if we were to start over, we would’ve started the day it was assigned because this project took a lot longer than any of us expected. The sheer number of options on how we can go about this project had us confused for a long time. If we were to start over, we should’ve also gotten help from the Professor or TA’s to better help us be guided in the right direction. If we were to start over, we would’ve better communicated our ideas and communicated better in general.

What each of us learned:

Muhimin(): What I learned during this project was the amount of collaboration and communication software engineering group projects take. I thought it was going to be light communicating but we were almost communicating every single day. We were talking about the progress on our deliverables and what we need to do as well as what changes we need to make. I also learned how to implement a red-black tree which I didn’t know how to do before. Jackson and Ernesto also taught me how to use Github to make collaborating on code way easier. I also learned how to draw out problems more clearly when it came to inserting nodes and re balancing/re coloring the nodes.

Ernesto(): This project was a valuable experience. In it, I gained experience in teamwork and GitHub, both of which are invaluable since future jobs will almost certainly require both. Communication and planning is key for these types of projects, and this experience has allowed me to get used to these environments a little bit more. I also familiarized myself with another ADT, in this case it was the B Tree. Up until now I knew how B Trees worked conceptually, but after implementing it I could truly see how different it was, with there being many keys per node and having to split nodes, which I had never done before with the AVL Tree. Both the coding experience and the project management experience constitute great experiences that will surely benefit me for the future.

Jackson():

References: