Administrative

Team Name: The Buddies
Team Members: Sean Kullmann, Turner Shultz, Nicholas Verdugo
Github Link: https://github.com/Kullmann/Advanced-Baseball-Statistics
Link to Video: https://youtu.be/l3MrbP2kbno

Extended and Refined Proposal

Problem: What problem are we trying to solve?

We are trying to make advanced baseball statistics more readily available.

Motivation: Why is this a problem?

This is a problem because at the high end of baseball many of the players are similar in skill and it is hard to discern who is the better player. This can be done by using advanced statistics to figure out which player is the best for the team.

Features implemented

We implemented a Hash Map, and an Alphabetized Binary Search Tree to store data. Also, we created a gui to display all of our data cleanly. This gui contains a section with user input, and all of the input boxes have error checking to make sure bad data such as non-digits do not get entered where they are not supposed to be. We also have a timer to see how fast each data structure implementation takes to run to find a player. In regards to the binary search tree, we implemented a method to store data from a .csv file, with a looping function to store player names, and their associated statistics. Also, functions are included that can generate statistics about players based on their inputted data, in order to help better compare individual players.

HashMap Functions
insertPlayer() - Inserts 1 player into the data structure with the specified name and data
searchPlayer() - Searches through data structure to find player with specified name
searchTable() - Searches through data structures with to find play with specified name to calculate the advanced statistics for that player
readData() - Read the input csv file into the data structure

BST Functions
insert() - takes in a player node and a tree object, then inserts the player and their corresponding stats into the BST.
calc() - takes in a players stats, then calculates that player's advanced stats and returns a vector of them
search() - takes in a string and a tree object, then locates the player in the tree and gets the stats for that player and calls the calc function

readFile() - reads the csv file and calls the insert function to place each element in the tree

Description of data

For the data that we used in this project, we needed to produce 100,000 names along with 9 stats to go along with each name. We used a random name generator to get the names, and then we randomly generated 9 numbers for each stat category, each within a specific and realistic range.

Tools/Languages/APIs/Libraries used

C++, SFML, SFML-Widgets

Data Structures/Algorithms implemented

Hash Map, Alphabetized Binary Search Tree

Data Structures/Algorithms used

Hash Map, Alphabetized Binary Search Tree

Distribution of Responsibility and Roles

Sean created and implemented the GUI and the Hash Map for the application, Turner created the logic for the search and data retrieval functions, and Nicholas created the binary search tree and functions used to add to it.

Analysis

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

The one change we made was to scrap the ability to search for a statistic and see which players had the number corresponding to that statistic. We decided to scrap because we felt it could not be properly implemented and did not align with the problem we were trying to solve. Instead, we added a section that allows the user to add players to the database permanently, and it makes calculation of the player added. The user input was recommended to us by Professor Kapoor.

Complexity analysis of the major functions/features you implemented in terms of Big O for the worst case

Worst Case for function in the Hash Map:

searchPlayer: O(n^2)

searchTable: O(n^2)

insertPlayer: O(1)

Worst Case for the BST:

calc: O(1)

search: O(logn)

Reflection

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

Overall, the project was a good experience because we learned how to work as a group to program something and each work on one module. This can be quite difficult in the end when trying to combine all of the modules into one. We enjoyed working in a group because we shared the weight of the project and everyone wrote a decent sized piece.

Did you have any challenges? If so, describe.

The one big challenge Sean had was creating the GUI, he had very little experience working with them before, and making it in C++ made it significantly more difficult. This was made easier through SFML and the sfml-widgets library. Once he figured out how to use the library making the gui became quite easy. A challenge that Nicholas had was deciding how to create the hierarchy for the binary search tree. Thankfully, C++ includes a function that can compare strings, making storing the player name as the key in a node much simpler and more logical than other solutions.

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

The only change we probably should have made was to definitely start earlier on the Binary Search Tree, so we could work through each problem we encountered much smoother and without as much stress.

Comment on what each of the members learned through this process.

Sean learned a lot more about how to code a gui in C++. He also thinks his Hash Map might be able to be optimized by storing a pointer to an array of integers as the value instead of a vector of integers.

Turner learned a lot more about reading and writing to files and how to manipulate data. He also learned a lot more about how to use a BST and how useful they can be when it comes to storing data.

Nicholas learned about pointer manipulation and gained a better understanding of binary search trees, and how to implement them. Also, he learned about file I/O and how to more effectively use C++ functions to better solve problems involving file I/O.

References

https://www.sfml-dev.org/
https://github.com/abodelot/sfml-widgets