COP3530 Project 3: Group 56 Report

Administrative

Team Name: DSA Defenders
Team Members: Lester Bonilla, Nathan Gilman, James Hu
Github Repository: https://github.com/npgilman/Project3_Repository
Link to Video: https://youtu.be/QILaKbB1ye8

Extended and Refined Proposal

Problem:

Americans throughout the country struggle to create well-balanced, nutritious diets. From a dietary intake study conducted from 1999 to 2012, 46% of American adults have poor quality diets [1]. These diets are characterized by having "too little fish, whole grains, fruits, vegetables, nuts and beans, and too much salt, sugar-sweetened beverages and processed meats". Furthermore, the CDC states that nearly 173 billion dollars is spent annually on healthcare for obesity [2]. This shows both the health and economic impacts that poor nutrition is having on America.

Motivation:

As clearly outlined by the problem, nearly half of American adults struggle to create a healthy balanced diet. Identifying the necessary macronutrients for consumption varies from person to person and can be overwhelming to properly balance. By creating an app that simplifies this process, we believe we can help out Americans who struggle to create healthy diets and find the proper foods to eat.

Features Implemented:

One feature we implemented was a graphical user interface (GUI). We wanted our program to be user friendly to help simplify the process of creating a healthy diet. Our GUI consisted of 3 text boxes that allowed the user to enter their desired amounts for the primary macronutrient categories (Carbohydrates, Fats, Proteins). There was also a button that would display the user's suggested foods to eat, depending on their specifications. Each time the button was clicked, the foods would be updated to reflect the current macronutrient values. The GUI was divided into two panels: the left being used to accept user inputs and the right for displaying the resulting foods selected once the algorithms were run.

Another feature we added were sorting algorithms. With a large dataset consisting of 266,000 values, we wanted to properly sort our data to efficiently select foods that would be most relevant for each macronutrient. The algorithms sort the data by macronutrient in descending order. We then use a function to determine from which sorted list to pull from. We prioritize filling the protein and fat requirements first, as carbs are usually the most abundant macro in a diet and easy to fulfill.

Description of Data:

Wanting to ensure that we had accurate data in terms of the macronutrients and categories of various foods, we decided to use a large dataset from the United States Department of Agriculture's Food Composition Database. The data includes a variety of information for each food in the dataset including the amounts of different vitamins and minerals found in the foods as well as macronutrient percentages. The foods listed in the dataset span a large variety of popular household foods. Our dataset consists of roughly 266,000 values with approximately 53 attributes per food. This results in around 5,000 foods.

Tools/Languages/APIs/Libraries used:

Our group primarily used C++ to create this application. We felt the most comfortable with it because we have spent lots of time building our familiarity with it throughout the semester by completing course assignments. We used 2 external C++ libraries: SFML and JSON. SFML is a popular C++ library for creating simple or complex graphical applications. It provides an interface to various components of the computer, but we primarily focused on its graphical applications for this project. Additionally, we used a C++ library for parsing our dataset. The JSON format of our dataset made this library a straightforward choice and this library greatly sped up our rate of development.

Another programming language that we briefly used was Python. We used python to create some basic scripts to set up the development environment for group members. Since we decided to download and store the SFML and JSON libraries in our github repository, we needed to have a script that would dynamically update the paths for the SFML library depending on the current working directory and create commands to compile our executable.

Beyond programming languages and libraries, the tool that was most helpful to use was VS Code. Serving as a lightweight code editor, we were able to quickly prototype various portions of our code and then expand on the necessary features. VS Code also allowed us to use a variety of custom plugins that allowed us to see recent commits, display CSV and JSON files, and other useful features.

Algorithms implemented:

One of our two sorting algorithms we chose to implement was merge sort. We chose merge sort because it is a popular sorting algorithm that runs in O(nlog(n)) time, where n is the number of elements being sorted. This is superior to other sorts such as bubble sort and insertion sort. Dealing with a large dataset, we felt it was important to prioritize the efficiency of our sorting algorithms. Merge sort recursively divides a given container in half and merges the divided arrays into a sorted fashion once a base case is reached.

The other sorting algorithm we used was pancake sort. Pancake sort is similar to the more popular selection sort in that maximum/minimum values are first selected and then placed into a sorted region at the end. The algorithm flips the array multiple times to place the minimum element in the sorted region of the end of the array, and iterates this process by decreasing the size of the array that's covered by 1 every time.

The average and worst case time complexity of pancake sort is O(n^2), where n is equal to the number of elements being sorted. The worst case occurs when the array alternates between its largest and smallest values (ex. [20,0,19,1,18,2]), because more array flips are required to place minimum values at the end. We chose pancake sort because we wanted to learn other sorting algorithms not covered in class and we found pancake sort to be our personal favorites.

Additional Data Structures/Algorithms used:

In addition to the sorting algorithms, we frequently used vectors and maps for convenient data storage. In our sorting class, we used a map for each of a food's macro. The key for each of these maps was the unique description of the food, and the value was the macro. We also used a map as a mapper, mapping an integer to each food. We used a vector of integers as a stand in for all of the foods to make sorting more readable. For each integer in the vector, we can find its name by using it as the key for the mapper. We then use that food name as the key for the macro maps to find the macro values.

Our data comes in the form of a JSON. We used a library that makes working with JSONs simple by treating them similar to a STL container. We had to spend some time reading through the documentation of this library to learn how to work with it and the capabilities it provided. At the start of

the program, we iterate the original data and draw out the macros for each food and map each food to an integer.

In our windows utilities class, maps were used to store various data relating to the buttons and textboxes such as the data held within and the clickable areas of the screen. Maps were essential to the correct functionality of the GUI application.

Distribution of Responsibilities and Roles:

- Lester Bonilla: Using C++ and JSON library to parse and store dataset into usable format, Create the merge sort function to organize the foods based on macros
- Nathan Gilman: Setting up github repository and creating python script for easy setup of development environments, Creating graphical user interface elements and functionality using C++ and SFML library
- James Hu: Create the pancake sort algorithm to organize the foods based on macros, created wireframe design for graphical user interface

---

Analysis

Any changes the group made after the proposal?

We did not make any changes to the direction of our project after the professor. Initially, we believed that we might face issues without, but after talking with TAs and Professor Kapoor, we were able to find ways to circumvent the issue. We slightly deviated from the initial wireframe because of a change in our desires for application appearance.

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

For each function, N is the number of foods in our dataset.

Function Signature: Sorting();

Worst Case: O(Nlog(N)) - The function iterates through the initial JSON database to separate the data into different maps.

Function Signature: void mergeSort(vector<int> &data, int start, int end, const map<string, float>& macro);

Worst Case: O(Nlog(N)) - This function is a traditional mergeSort. It recursively divides the array in half until it reaches 1 unit, and builds it back by sorting the smaller subarrays.

Function Signature: json calculateFoods(bool algorithm, float protein, float carb, float fat);

Worst Case: O(N^2) - This function calls the pancakeSort function, which is itself O(N^2) and described below. It includes the helper functions *chooseFoods* and *createFoodJSON* which are described below. *chooseFoods* is O(N) and *createFoodJSON* is O(Nlog(N)), so they are dropped from this function's complexity.

Function Signature: vector<int> chooseFoods(float protein, float carb, float fat);

Worst Case: O(N). Where P, C, and F are the wanted values of protein, carbs, and fats, respectively. This function calls another function - *choose* - which loops the data set until an appropriate food is found - O(N). The macros of that food are subtracted from the P, C, and F values. *chooseFoods* calls *choose* until P, C, and F are 0. This results in O((P+C+F)N). Because we know P, C, and F are limited by a constant input value, this simplifies to O(N).

Function Signature: json createFoodJSON(const vector<int>& foods);

Worst Case: O(Nlog(N)) - This function iterates the resulting vector from *chooseFoods*, and accesses foods from a map - log(N). In the worst case, N foods would be inserted into the json.

Function Signature: void pancakeSort(vector<int> &data, int size, const map<string, float>& macro);
Worst case: O(N^2) - This function iterates for N times, and calls the findMax and pancakeFlip functions which are O(N) complexity for each iteration, so the worst case time complexity is O(N*N) = O(N^2).
Function Signature: void WindowUtils::drawResults(sf::RenderWindow &window);
Worst case: O(N) - This function has an O(N) runtime, because in the worst case, it would take every food in the dataset to reach the desired amount of macronutrients.
Function Signature: void WindowUtils::handleText(sf::RenderWindow &window, sf::Event event);
Worst case: O(1) - This function takes an event parameter and adds the input text to the currently selected textbox. This is O(1) because it adds a single character to a string.
Function Signature: void WindowUtils::handleClick(sf::RenderWindow &window, sf::Event event);
Worst case: O(1) - This function takes an event parameter and calculates the location of the mouse. It then loops through a map of the button bounds and determines if a button was clicked or not and handles it appropriately. This function is O(1) because, despite looping through the existing buttons, there is a fixed number of buttons.
Function Signature: void drawBackground(sf::RenderWindow &window);
Worst case: O(1) - This function runs in O(1) time because it is responsible for rendering a fixed number of objects on the screen.

Reflection

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

This experience was a good introduction on what it takes to make a product as a team, starting from creating an idea and developing it over a given period of time. We worked to delegate roles and responsibilities, as well as help each other out when needed. We view this as a successful experience because our application closely models the proposal we submitted for assignment “Project 3a”. We also learned a variety of key principles for project design and management.

Did you have any challenges? If so, describe.

One challenge that we encountered was setting up SFML in a way that would allow each of our group members to use it with minimal setup. We needed to have an absolute path for the includePath on the libraries. To get around this, we created a python file that would get the path of the current working directory and append it to the C++ properties file that VS Code uses.

We faced a challenge in learning how to access and work with JSONs. We searched for a library that added JSON functionality and had to learn how to use it by reading through its documentation.

A major challenge involved figuring out the function to figure out which food from the sorted list to select. We searched for the dietary guidelines published by the US government and found that the recommended diet has about 45g protein, 130g carb, and 30g fat. Using this, we decided to prioritize foods that filled the protein and fat requirements as they would be harder to source.

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

If we were to start this project again, one change to our workflow would be the design stage. I think if we spent a longer amount of time designing our project and each component, the integration would have been extremely easy. Not only would integration time between components have been reduced, but the potential for future features would have been increased by clearly outlining what each component should be responsible for.

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

- Lester Bonilla:I learned the usefulness of libraries in helping us not to recreate the wheel. I initially planned to make a class that would parse the JSON, but finding a library and reading its documentation helped me see how important the coding community is.
- Nathan Gilman: I learned how to create a python script that can help ease the setup of a project's development environment and how libraries can be managed within a project. I also learned basic features of GUI and user interaction.
- James Hu: I learned how to understand other people's code and integrate my code within a collaborative codebase, and also learned the basics of designing graphical user interfaces.

References

[1] https://jamanetwork.com/journals/jama/fullarticle/2529628
[2] https://www.cdc.gov/chronicdisease/resources/publications/factsheets/nutrition.htm
[3] https://www.sfml-dev.org/index.php
[4] https://github.com/nlohmann/json#projects-using-json-for-modern-c