Portfolio of Algorithms

Introduction to Programming (IY499)

Student Name: Nathan Gideon
Student Number: 30306616

I confirm that this assignment is my own work. Where I have referred to academic sources or online sources, I have provided in-text citations and included the sources in the final reference list.

SECTION 1 - Filtering algorithm

The filtering algorithm is defined as a function accepting three parameters: data, filter_key, and filter_val. These act as placeholders that are populated with actual data only when the function is called. To ensure the original dataset is not mutated (modified), the function initializes an empty temporary list, fil_list.

The core logic is wrapped in a try-except block to handle potential errors gracefully. Inside this block, a for loop iterates through the dataset. In each iteration, the variable record temporarily represents a single item (dictionary) from the list. The algorithm checks if the value associated with filter_key in that record matches the filter_val. If this condition returns True, the record is appended to fil_list. Finally, the function returns the filtered list to the main program.

If an error occurs during this process (such as a missing key), the except block catches it. This prevents the program from crashing and allows it to display a custom error message instead.

I tested the algorithm’s flexibility using different datasets, including a food database where I successfully filtered items by type (e.g., fruit, vegetable, or processed). This confirmed the algorithm is adaptable to different data structures.

However, the program has limitations: the error handling is somewhat generic and does not always pinpoint exactly what interrupted the process. Additionally, the filtering is strictly case-sensitive (e.g., "Fruit" will not match "fruit") and requires the input data types to match exactly, which can lead to empty results if not managed carefully.

Section 2 - Aggregation Algorithm

The algorithm works by using a function to be easily called within the program; it uses three different variables within the function: dic, keygroup, value_key. The algorithm creates two internal dictionaries: sums (to store the running total of values for each group) and counts (to track the number of items in each group). It loops through the input list one record at a time. For each record, it extracts the group identifier (the group_key) and the numeric value (the value_key).

If the group is encountered for the first time, it is added to the dictionaries. The algorithm then adds the current number to that group's total in sums and increases the group's counter in counts by one. After processing all records, a final loop divides the total sum of each group by its count to determine the average.

I tested the program to search for the average of students in datasets, divided by the class they were in. By using the function aggregate(), I was able to read the averages of two classes that had been filled; proving the program's flexibility in logic and division.

However, the program was not able to understand the difference between different data types, such as inputting string in value_key instead of integer, which would need error handling. Another limitation is the case-sensitivity; when inputting the keygroup, one may type capital, which the program treats as a separate keygroup and turns up as an error. One might also find the lack of specific error messages to be a problem as well.

Section 3 - Sorting Algorithm

The sorting algorithm is implemented as a function that accepts two parameters: the dataset to be sorted and a specific key to sort by. The function begins by creating a deep copy of the original list. This prevents the sorting process from permanently altering the original dataset.

The core logic utilizes a Bubble Sort mechanism. This is achieved through a nested loop structure. The outer loop iterates through the entire list, ensuring the process repeats enough times to place every element correctly. The inner loop moves through the list comparing adjacent items. It checks if the value of the current item’s sorting key is greater than that of the next item, if this condition is true; the two items are swapped. This process effectively "bubbles" the largest values to the end of the list. The function is wrapped in a try-except block to catch potential errors, outputting a clean sorted list upon completion.

I tested this algorithm using a student dataset containing mixed data types, such as strings for names and integers for exam scores. By calling the function with the key "exam score," I verified that the output list was correctly reordered from the lowest score to the highest. I also tested with "class," which successfully sorted the records alphabetically by their class code.

Despite its functionality, the algorithm has distinct limitations. The Bubble Sort method is computationally inefficient, making it significantly slower than advanced algorithms. The current program supports ascending order; it cannot sort in descending order without certain modifications. Also, it is sensitive to missing keys, which can result in empty outputs.

Section 4 - Reflections

This assessment had challenged me, applying all that I had learnt from the first term. Such as understanding data structures and their commands, revisiting certain topics such as for loops or averages. With each task, it allowed me to understand some basic principles of a DBMS (database management system), and the higher complexities that must be addressed when dealing with higher forms of data.

Filtering was also a bit of an unexpected skill update. While I did know that it was a simple if-condition, what I needed was a reminder on how to get the records out of the database, which was a solution needing to use a for loop to access the database.

One of the hardest parts of the aggregate algorithm was keeping track as to where each value was being sent,and also deciding how to deal with grabbing the values from the table while ensuring each category of the dataset was properly placed with. While it took many trials and errors to work out the logic of the separator of key and value; I had eventually reached there, the rest of working out the average was easier done.

Using Bubble sort was not the most pleasant experience I had in coding. While I did have an idea of what needed to be done, the challenge was properly setting the for loops such that it wouldn’t crash the program. The second for loop was a difficult challenge, spending hours on properly figuring out how to find the end of the loop. Once that was done, I need to properly check if my values have not flipped anywhere.