The Expense Tracker Bot

Retake project:
I1 - Computational Thinking and Programming - BC - (2024-2025)

By Neel Sabharwal
March 2025

Table of Contents

1. Background

1.1 Rationale

Managing personal finances is a critical yet often challenging task, particularly in today's fast-paced world, where expenses occur frequently and across multiple channels. Many individuals struggle to track their spending effectively due to a lack of accessible, user-friendly tools that provide both systematic expense tracking and meaningful insights. Without such tools, categorizing expenses, identifying spending trends, and making informed financial decisions becomes difficult, leading to potential overspending and financial instability.

Effective expense management is essential for achieving financial stability and meeting budgeting goals. By gaining a clear understanding of their spending patterns, individuals can recognize unnecessary expenditures and make proactive decisions to save money. This project addresses a significant gap in personal finance management by offering an intuitive expense tracker that empowers users to take control of their finances and develop responsible spending habits.

The project operates within the domain of personal finance, focusing on budgeting, expense tracking, and financial analysis. By integrating computational methods—such as divide-and-conquer algorithms for identifying peak spending periods and greedy algorithms for optimizing expenses—it not only provides practical financial management tools but also showcases the application of algorithmic thinking to real-world problems.

1.2 Functionality

The bot operates as an interactive command-line application that guides users step-by-step through various financial management tasks. Below as an overview of the interactive process:

Main Menu Presentation: When the application starts, the bot displays a main menu with several numbered options. Each option corresponds to a different functionality, such as adding an expense, viewing recorded expenses, summarizing spending, analyzing spending periods, or suggesting expense reductions.

User Input and Navigation: The user is then prompted to enter a number that corresponds to their desired action. The bot reads this input and then directs the flow of execution to the appropriate function. For example:

- Selecting **"1"** launches the process to add a new expense.
- Selecting **"4"** opens a submenu where the user can choose to analyze the entire expense history or a custom date range.

Interactive Data Entry: For tasks like adding an expense, the bot engages in a dialogue by prompting the user for details such as the date (defaulting to the current date if left blank), expense category, description, and amount. It validates these inputs (e.g., ensuring the amount is a valid number) before recording the information.

Feedback and Display of Results: After processing user input, the bot provides immediate feedback. For example, after adding an expense, it confirms the addition. When displaying summaries or analyses, it prints out the calculated results—such as total spending or the period with the highest expenditure—directly on the console.

Submenu for Advanced Analysis: When the user selects an option like analyzing spending periods, the bot presents an additional submenu. This allows users to choose between analyzing the complete expense history or a custom date range. Based on the selection, the bot further interacts by prompting for specific dates if needed, then processes and displays the result of the analysis.

Continuous Loop Until Exit: The interactive process is maintained within a loop. After completing any action, the bot returns to the main menu, allowing the user to perform additional tasks until they choose the exit option. This loop ensures a seamless, conversational experience where users can navigate through different features without restarting the application.

---

2. Computational Approach

2.1 Data Collection

2.1.1 CSV File for Data Persistence

The application utilizes a CSV file (expenses.csv) to store all expense records. This choice enables a simple yet effective way to handle data persistence without requiring complex databases. The file is automatically created and initialized with a header if it does not already exist.

2.1.2 Interactive Command-Line Interface

Users interact with the application via terminal inputs. Functions are provided to add, view, and summarize expenses, ensuring that the tool remains accessible and easy to use.

2.2 Divide-and-Conquer for Spending Period Analysis

To identify the contiguous period with the highest spending, a divide-and-conquer algorithm inspired by the maximum subarray problem was included. This algorithm recursively splits the list of daily expense totals into smaller segments, computes the maximum subarray for each segment, and then determines the maximum subarray that crosses the midpoint. This approach efficiently narrows down the period of peak spending, even for long spans of data, by breaking down the problem into manageable subproblems.

2.3 Greedy Algorithm for Expense Reduction Suggestions

For providing recommendations on which expenses could be reduced to meet a specified savings target, the application implements a greedy algorithm. The expenses are sorted in descending order by their amounts, and the algorithm iteratively selects the largest expenses until the cumulative sum meets or exceeds the target savings. While the greedy approach may not always produce the absolute optimal solution, it offers a straightforward heuristic that is easy to understand and implement. This method quickly identifies significant expenditures that, if reduced, could have a large impact on overall spending, thus offering actionable advice to the user.

2.4 Modular and Documented Code Structure

The application is organized into individual functions, each handling a specific aspect of the expense management process (e.g., adding expenses, viewing summaries, analyzing spending periods). This modular design promotes clarity, maintainability, and ease of extension. Each line of code is accompanied by detailed comments that explain its purpose, and each function is documented with a clear description of its role and parameters.

---

3. Code Analysis

3.1 Key Functions

3.1.1 File Initialization and Data Persistence Functions

initialize_file(): checks whether CSV file (expenses.csv) exists. If not, it creates the file and writes a header row (Date, Category, Description, Amount). This ensures all expense data is stored persistently using a simple file-based system.

3.1.2 Expense Management Functions

add_expense(): prompts users to input details for new expenses (date, category, description, and amount). If no date is provided, it defaults to the current date. The expense is then appended to the CSV file. This function manages the core data entry for the application.

view_expenses(): reads all rows from CSV file and prints each expense record. This allows users to review their recorded expenses in a straightforward manner.

view_summary(): calculates and displays total spending along with a breakdown by category. It processes each record from the CSV, sums up the amounts, and aggregates the expenses by their categories, providing an overview of where the money is being spent.

3.1.3 Spending Period Analysis Functions

find_max_crossing_subarray(arr, low, mid, high): helper function that, given a list of daily expense totals, finds maximum subarray that crosses the midpoint. This is a core component of the divide-and-conquer approach used in analyzing spending periods.

find_max_subarray(arr, low, high): implements the recursive divide-and-conquer algorithm to find the contiguous subarray (i.e., the period) with the maximum sum in a list. This function breaks down the problem into smaller parts, evaluates the left half, the right half, and the subarray crossing the midpoint, and then selects the best candidate.

max_spending_period(): groups expenses by date and calculates daily totals from the CSV file. It then applies the divide-and-conquer algorithm (via find_max_subarray) to determine the contiguous period with the highest overall spending across the entire expense history.

custom_spending_period(): similar to max_spending_period(), this function prompts the user to enter a custom start and end date to filter the expense data. It then analyzes the selected date range using the divide-and-conquer approach to identify the period with the highest spending within that custom range.

3.1.4 User-Driven Analysis and Suggestions Functions

analyze_spending_period(): provides a submenu that lets the user choose between analyzing the entire expense history or a custom date range. Depending on the user's selection, it calls either max_spending_period() or custom_spending_period().

suggest_expense_reductions(): helps users identify potential cost-cutting opportunities by asking for a target savings amount. It sorts the expenses in descending order (largest first) and selects the top expenses iteratively (using a greedy algorithm) until the target is met. The suggestions offer practical insight into which expenses could be reduced or eliminated.

3.1.5 Application Control Flow Functions

main(): entry point of the application. It first ensures that the CSV file is initialized. Then, it continuously displays a main menu with options for adding expenses, viewing records and summaries, analyzing spending periods (with a submenu for custom or complete history), suggesting expense reductions, and exiting the application. Based on the user's input, the appropriate function is called, ensuring an interactive and user-friendly experience.

The code's modular design, combined with extensive inline documentation, not only makes it easy to follow and maintain but also serves as a valuable educational resource. Each function is dedicated to a specific task, and the use of classic algorithms (divide-and-conquer and greedy methods) is clearly integrated into the application to provide practical financial analysis. This structure ensures that the application is both robust and scalable, while remaining accessible to beginner programmers.

3.2 Time Complexity

initialize_file(), add_expense(), view_expenses(), and view_summary(): Each of these functions perform operations that are linear with respect to the number of expense records. For example, reading through the CSV file to view or summarize expenses involves iterating over every record, thus yielding a time complexity of **O(n)**, where *n* is the number of expense entries.

find_max_crossing_subarray(arr, low, mid, high): This helper function performs two linear scans (one from the midpoint to the left and another from the midpoint to the right), so its complexity is **O(m)**, where *m* is the number of elements in the current subarray.

find_max_subarray(arr, low, high): Implemented using a divide-and-conquer strategy, this recursive function divides the list of daily totals into halves, computes the maximum subarray for each half, and then finds the crossing subarray. The recurrence relation for this algorithm is:

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

Solving this recurrence yields a complexity of **O(n log n)**, where *n* is the number of days (or unique dates) in the expense data. In the worst-case scenario—if every expense is on a separate day—*n* is proportional to the total number of expense records.

suggest_expense_reductions(): This function first sorts the expense records in descending order based on amount. The sorting operation has a time complexity of **O(n log n)**. After sorting, it iterates through the sorted list once to accumulate expenses until the target is met, which is an **O(n)** operation. Hence, the overall complexity for this function is dominated by the sorting step, resulting in **O(n log n)**.

***Note** this is the complexity analysis of the most computationally significant parts of the code—the functions that operate over the entire dataset or involve sorting and recursion. Many of the smaller helper functions or simple file operations have constant or linear complexity and don't dominate the overall runtime.

---

4. Evaluation & Reflection

4.1 Final Outcome

The code delivers a functional expense tracker that not only records and categorizes daily expenses but also provides deeper insights into spending habits through algorithmic analysis. Users can view detailed summaries of their expenditures, identify the contiguous period with the highest spending across their entire history or within a custom date range, and receive suggestions on potential expense reductions to meet a specified savings target.

4.2 Strengths

User-Friendly Design
The command-line interface is straightforward, providing clear prompts and feedback. The application automatically handles file initialization and manages data through a simple CSV file, ensuring that users can start tracking their expenses immediately without any additional setup.

Insightful Financial Analysis
By leveraging algorithmic techniques, the project goes beyond basic data entry. It helps users identify spending trends, pinpoint periods of peak expenditure, and even suggests actionable ways to reduce costs—all of which are valuable for improving personal financial management.

4.3 Weaknesses & Limitations

Data Persistence
The current implementation uses a local CSV file for data storage, which may not be robust for handling large datasets or multi-user scenarios.

User Interface Constraints
The application operates through a command-line interface, which might be less accessible or visually appealing for users accustomed to graphical user interfaces (GUIs). Future enhancements could include a web-based or desktop GUI to improve user interaction.

Expense Reduction Feature
The expense reduction feature solely focuses on the amount spent in each category without considering the essential nature of the expense. The algorithm identifies the category where you spend the most money and suggests cutting expenses from that category to meet your savings target. However, this approach does not differentiate between necessities and discretionary spending. For example, if you spend a large portion of your budget on food—a necessity—the bot might recommend reducing food expenses, even though it might be more practical to cut back on a non-essential subscription or leisure expense. This could lead to suggestions that are impractical or even harmful to your quality of life, as it overlooks the critical distinction between essential and non-essential spending.

Algorithmic Simplifications
While the greedy algorithm for expense reduction offers quick, practical suggestions, it might not always provide the most optimal solution in complex financial scenarios. Additionally, more sophisticated error handling and data validation could be implemented to manage diverse user inputs and edge cases more gracefully.

4.4 Conclusion

Overall, the project successfully demonstrates how computational thinking and algorithmic approaches can be applied to solve everyday problems in personal finance management. It provides a practical tool that helps users track expenses, analyze spending patterns, and make informed decisions about cost reduction. While there are limitations—particularly regarding data persistence, UI design, the expense reduction feature, and the optimality of the heuristic algorithms—the project lays a solid foundation for future enhancements. These could include more advanced analytics, improved error handling, and a more user-friendly interface, making the tool even more robust and scalable for broader use.