4-2 Milestone Three: Enhancement Two: Algorithms and Data Structure

Carl Allen

CS-499 Computer Science Capstone

**Introduction:**

For Enhancement Two: Algorithms and Data Structures, I chose to improve the search algorithm in my Python application that manages data for pets in animal shelters. I worked on this enhancement using the PyCharm development environment. The application is built with Python, a high-level programming language known for its rich library of algorithms and logical structures. I selected this project because of its strong foundation and the opportunity to refine existing functionality. This application was originally developed during the 22EW4 term for the CS-340 Client/Server Development course. I determined this was the best project to enhance stemmed from how popular Python’s programming language is, ranking as the 4th most popular language according to a 2021 Overflow survey. Python’s widespread use in web and game development, areas where I excel. Along with its prevalence in emerging fields like machine learning and artificial intelligence (AI) solidified my decision. With the recent influx of AI and machine learning happening around the world, joining a team in either domain could have potential contributions to groundbreaking innovations in the future.

**Overview:**

In this application, I demonstrate a wide range of skills using the Python programming language within the PyCharm development environment. Initially, the codebase was cluttered and inefficient. To improve it, I restructured the code in PyCharm with a focus on optimizing functions related to data structures. Since the application relies heavily on filtering data from a database, I addressed the clutter by moving the filter logic outside of the functions and placing it before the page layout—right after my library and document imports. This adjustment significantly improved readability and made the code easier to customize, which was a valuable new technique I learned during the process.

Another key part of the enhancement involved optimizing the search algorithm. I accomplished this by modifying the data structure and learning to use a dictionary to store filter\_criteria for each column in the database, such as Breed, Sex, and Name. Figure 1 shows a snippet of the filter criteria used for the default view under the "All" option in a set of radio buttons. When the application launches, this default filter displays all animals in the database whose ages range from 0 to 999 on the dashboard.

**Figure 1**

Filter criteria used in data table.

A screenshot of a computer

Description automatically generated

I placed my dictionary directly after the library imports to allow for easier customization and centralized management of the filter criteria. This setup lets me review and adjust the filters before writing the code that uses them, contributing to the overall optimization of my Python code. By consolidating the filter logic in one location, I eliminated the need for duplicate queries, which not only shortened the code but also made it easier to maintain and debug. Keeping the dictionary and filtering logic together also streamlines development by reducing the time spent searching through different layout sections and callback functions to manage filters. Lastly, I created regex patterns using, `**create\_regex\_pattern(keywords)**`. Figure 2, my regex implementation code snippet.

**Figure 2**

Regex function for example query.

A computer screen shot of a computer code

Description automatically generated

This block of code defines a function that generates a regular expression (regex) pattern from a list of keywords. The keywords parameter serves as the input for building the pattern. The function uses re-compile to compile the final regex pattern, and includes the IGNORECASE flag to ensure the search is case-insensitive, allowing it to match keywords regardless of their capitalization.

An example of how this function is used can be seen in the following MongoDB query: query = {"$or": [{"breed": {"$regex": create\_regex\_pattern(breed\_keywords)}}]}. This line constructs a MongoDB query using the $oroperator, specifying that the breed field must match the regex pattern generated by the create\_regex\_pattern function.

These enhancements transformed the original mediocre search algorithms and data structure into a well-optimized, modular, and readable application by utilizing regex patterns to enable more flexible matching for filters such as “breed” using `**query = {"$or": [{"breed": {"$regex": create\_regex\_pattern(breed\_keywords)}}]}`** which also constructs a pattern that shows partial and case-insensitive matches enhancing the flexibility of your search, this also provides a more inclusive and user-friendly search experience. The regex function further optimized my data structure and algorithms by reducing the need for redundant queries by allowing a more inclusive match and encapsulating logic of matching multiple keywords, simplifying the overall structure of my code. The overall structure of my code was simplified as using regex patterns allows you to express complex pattern matching in a concise and readable manner.

In summary, using regex functions in my code optimized the search algorithm by providing a more flexible, case-insensitive, and accurate approach to pattern matching. This cleared any redundancy and simplified code logic that contributed to streamlined data retrieval process. These optimizations collectively enhanced the readability and performance of my Python application for managing pet data in animal shelters.

For the algorithmic logic of time complexity involves examining operations that scale with the input size. My applications primary operations involve filtering and querying data from my MongoDB database and performing operations on pandas DataFrames. The time complexity of these operations is dependent on the underlying implementation of the database queries and DataFrame manipulations. In Figure 3, I use (`**shelter.getRecordCriteria(query)**`), the time complexity of these operations is regulated by the efficiency of the underlying database and pandas library.

**Figure 3**

Time complexity runs off efficiency in this example for the getRecordCriteria.

A screen shot of a computer program

Description automatically generated

The size of the dataset and the chosen indexing strategy can significantly impact time complexity. By using recompiling the create\_regex\_pattern function, the regex pattern is pre-compiled, which may improve performance. However, the actual time complexity depends largely on the underlying implementation of the regex engine.

Similarly, the time complexity of creating a DataFrame from records depends on both the number and structure of those records. Fortunately, the pandas library is optimized for efficient DataFrame operations, which supports my decision to use it in this application.

Additionally, I included descriptive headers outlining the purpose and functionality of each code section or node. This improves readability and makes it easier for other developers to understand and modify the code as needed.

This header is included in all files in the project for my application, each files header is updated with the intent and decision based on those files current code, Figure 4.

**Figure 4**

Inline commenting for easy readability and understanding for future programmers to use when modifying code.

A screenshot of a computer

Description automatically generated

I also improved my inline commenting of the code describing its purpose and functionality, further improving readability and aiding future modifications, see below image with a snippet of my code showing an example of my commenting, Figure 5.

**Figure 5**

Figure 10

A screen shot of a computer code

Description automatically generated

**Conclusion:**

I demonstrated my knowledge and skills in designing and evaluating computing solutions by applying algorithmic principles and established computer science practices to solve real-world problems—specifically, through enhancing the search functionality of my Python application for managing pet shelter data. This aligns with the course outcome focused on using appropriate data structures and algorithms while managing trade-offs in design. The create\_regex\_pattern function reflects my understanding of algorithmic principles by enabling flexible, case-insensitive searches. Meanwhile, my use of the filter\_criteria dictionary showcases my ability to organize and manage data systematically through effective use of data structures.

Centralizing the filter logic in a single dictionary reduced redundancy and improved maintainability—an intentional design trade-off between initial setup effort and long-term scalability. I managed this trade-off by consolidating logic, utilizing regex patterns for efficient querying, and creating a modular, readable codebase that addressed the shortcomings of the original version.

This enhancement also fulfills the course outcome related to professional communication. I clearly presented my rationale, approach, and improvements in a logical sequence, beginning with the purpose of the enhancement and followed by technical explanations. I incorporated code snippets and inline comments to clarify functionality and support reader understanding. Feedback from peers guided improvements in documentation and helped refine both visual and written communication. Reflecting on the process, I was able to identify optimization opportunities and effectively communicate the impact of my design decisions.

In conclusion, the enhancements I made—such as restructuring code in PyCharm, optimizing data structure functions, and improving search functionality using centralized filtering and regex—significantly increased the application's efficiency, flexibility, and clarity. These changes highlight my solid foundation in algorithmic thinking and computer science principles, and demonstrate my ability to make thoughtful design decisions that balance functionality with maintainability. As a result, I now have a more optimized, modular, and professional-quality Python application.