Tripoli University

Faculty of Engineering

Electrical and Electronic Engineering Department

EE432

	TEAM	ID
NAME:	Mohamed Salah Idhbea	2210255513
NAME:	Mohamed Khaled Alosta	2190203481
NAME:	Ibrahim Ahmed Ahtash	2220202620

SUPERVISED BY Dr. Nouri Ben Barka

FALL 2024

Introduction :

The following report outlines the design and implementation of a Python-based desktop application for managing to-do lists, emphasizing the practical application of core data structures: **linked lists**, **hash tables**, and **binary search trees (BST)**. This project aims to deliver a robust task management system that enables users to add, delete, search, display, and prioritize tasks efficiently. By integrating these data structures, the application ensures optimal performance for key operations, such as **O(1) average-case search complexity** using hash tables and **O(log n) priority-based management** via BSTs. The implementation leverages Python's **Tkinter** framework to provide a user-friendly interface while maintaining clarity and adherence to Pythonic coding principles. This report details the technical architecture, team responsibilities, and individual contributions, emphasizing the structured collaboration required to achieve a cohesive and scalable solution.

---

1. Mohamed Idhbea's Experience (Hero Team: Core Data Structure Implementation)

As part of the **Hero Team**, my primary responsibility was to implement the **doubly linked list**, which served as the backbone for storing and managing tasks in the to-do list application. Here's a detailed account of my contributions and experiences:

Implementing the Doubly Linked List

The doubly linked list was crucial for maintaining the order of tasks as they were added to the application. My tasks included:

- **Node Design**: I designed a TaskNode class to represent each task, which included attributes like description, priority, completion_status, and pointers to the next and previous nodes.
- **Insertion Logic**: I implemented methods to add tasks to the linked list, ensuring that new tasks were appended to the end of the list while maintaining the correct pointers.
- **Deletion Logic**: I wrote functions to remove tasks from the list, updating the next and previous pointers of neighboring nodes to maintain list integrity.
- **Traversal**: I implemented a method to traverse the list and retrieve all tasks in the order they were added. This was used by the team to display tasks in the GUI.

Collaborative Work

- **Core Task Management Logic**: Together with **Mohamed Alosta** and **Ibrahim Ahtash**, I contributed to developing the core logic for task management. This included ensuring that tasks were added, deleted, and updated consistently across all data structures (linked list, hash table, and BST).
- **Tkinter-Based GUI**: I collaborated with the team to design and build the **Tkinter-based GUI**. My role involved integrating the linked list traversal logic with the GUI to display tasks in a clear and organized manner.

Challenges and Collaboration

- **Synchronization**: One of the challenges I faced was ensuring that the linked list stayed synchronized with the hash table and BST whenever a task was added or deleted. This required close collaboration with **Mohamed Alosta** (hash table) and **Ibrahim Ahtash** (BST).
- **Edge Cases**: Handling edge cases, such as deleting the first or last node in the list, required careful testing and debugging.

Personal Contribution

- Designed and implemented the **doubly linked list** for task storage.
- Ensured seamless integration with the hash table and BST.
- Conducted unit tests to validate the linked list's functionality.

Reflection

This project was a great opportunity to deepen my understanding of linked lists and their practical applications. Working with **Mohamed Alosta** and **Ibrahim Ahtash** taught me the importance of collaboration and clear communication in achieving a common goal. I'm proud of the role I played in building the foundation of the application.

---

2. Mohamed Alosta's Experience (Hero Team: Core Data Structure Implementation)

As a member of the **Hero Team**, my role was to design and implement the **hash table** for efficient task searching. Here's a detailed account of my contributions:

Implementing the Hash Table

The hash table was essential for enabling fast searches by task description. My tasks included:

- **Hash Function**: I designed a hash function to map task descriptions to indices in the hash table. The function ensured an even distribution of keys to minimize collisions.
- **Collision Handling**: I implemented **chaining** to handle collisions, where each index in the hash table stored a linked list of tasks with the same hash value.
- **Search Functionality**: I wrote a method to search for tasks by description, which traversed the linked list at the appropriate index to find a match.

Collaborative Work

- **Core Task Management Logic**: Together with **Mohamed Idhbea** and **Ibrahim Ahtash**, I contributed to developing the core logic for task management. This included ensuring that tasks were added, deleted, and updated consistently across all data structures (linked list, hash table, and BST).
- **Tkinter-Based GUI**: I collaborated with the team to design and build the **Tkinter-based GUI**. My role involved integrating the hash table search functionality with the GUI to enable users to quickly find tasks by description.

Challenges and Collaboration

- **Integration**: Integrating the hash table with the linked list and BST required careful coordination with **Mohamed Idhbea** and **Ibrahim Ahtash**. For example, when a task was deleted, it had to be removed from all three data structures.
- **Performance Optimization**: Ensuring that the hash table performed efficiently, even with a large number of tasks, was a key focus. I conducted stress tests to validate its performance.

Personal Contribution

- Designed and implemented the **hash table** with collision handling.
- Enabled **O(1) average-case search complexity** for task descriptions.
- Collaborated with the team to integrate the search functionality into the GUI.

Reflection

This project allowed me to apply my knowledge of hash tables to a real-world problem. Working with my teammates was a rewarding experience, and I learned a lot about teamwork and problem-solving. I'm proud of the efficient search functionality we delivered.

---

3. Ibrahim Ahtash's Experience (Hero Team: Core Data Structure Implementation)

As part of the **Hero Team**, my responsibility was to implement the **binary search tree (BST)** for managing tasks by priority. Here's a detailed account of my contributions:

Implementing the Binary Search Tree

The BST was used to organize tasks by priority (High, Medium, Low) for efficient filtering and retrieval. My tasks included:

- **Node Design**: I designed a PriorityNode class to represent tasks in the BST, with attributes like priority, task_reference, and pointers to left and right children.
- **Insertion Logic**: I implemented methods to insert tasks into the BST based on their priority, ensuring the tree remained balanced for optimal performance.
- **Traversal**: I wrote functions for in-order traversal to retrieve tasks sorted by priority, which was used by the team for priority-based filtering.

Collaborative Work

- **Core Task Management Logic**: Together with **Mohamed Idhbea** and **Mohamed Alosta**, I contributed to developing the core logic for task management. This included ensuring that tasks were added, deleted, and updated consistently across all data structures (linked list, hash table, and BST).
- **Tkinter-Based GUI**: I collaborated with the team to design and build the **Tkinter-based GUI**. My role involved integrating the BST traversal logic with the GUI to enable users to filter and sort tasks by priority.

Challenges and Collaboration

- **Balancing the Tree**: Ensuring the BST remained balanced was a challenge, especially when tasks were added or deleted frequently. I explored techniques like AVL trees but ultimately implemented a basic BST due to time constraints.
- **Integration**: Coordinating with **Mohamed Idhbea** and **Mohamed Alosta** to ensure that tasks were added to and removed from the BST in sync with the linked list and hash table was critical.

Personal Contribution

- Designed and implemented the **binary search tree** for priority-based task management.
- Enabled efficient filtering and retrieval of tasks by priority.
- Conducted tests to validate the BST's functionality and performance.

Reflection

This project was a great learning experience, allowing me to apply my knowledge of BSTs to a practical problem. Collaborating with **Mohamed Idhbea** and **Mohamed Alosta** was both challenging and rewarding. I'm proud of the role I played in delivering a robust and efficient task management system.

Summary of Team Contributions

Team Member	**Role**	**Key Contributions**
Mohamed Idhbea	Doubly Linked List	Implemented task storage and traversal logic.
Mohamed Alosta	Hash Table	Enabled O(1) search complexity and handled collisions.
Ibrahim Ahtash	Binary Search Tree (BST)	Managed tasks by priority and enabled efficient filtering.

Discussion

The development of the to-do list application provided a comprehensive opportunity to apply theoretical knowledge of data structures—**linked lists**, **hash tables**, and **binary search trees (BST)**—in a practical, real-world scenario. The project successfully demonstrated how these data structures can be integrated to create an efficient and user-friendly task management system. Below are the key takeaways and insights from the project:

1. **Efficiency and Scalability**:
    - The use of a **hash table** enabled O(1) average-case search complexity, ensuring that users could quickly find tasks by description, even as the number of tasks grew.
    - The **BST** allowed for efficient sorting and filtering of tasks by priority, with O(log n) complexity for insertion, deletion, and retrieval operations.
    - The **doubly linked list** ensured that tasks could be traversed sequentially, providing a straightforward way to display tasks in the order they were added.
2. **Integration of Data Structures**:
    - One of the most challenging aspects of the project was ensuring that all three data structures (linked list, hash table, and BST) remained synchronized during task additions and deletions. This required careful coordination and testing to avoid inconsistencies.
    - The modular design of the application allowed each data structure to handle a specific aspect of task management, ensuring clarity and maintainability in the codebase.
3. **User Experience**:
    - The **Tkinter-based GUI** provided a simple yet effective interface for users to interact with the application. Features like task addition, deletion, searching, and priority-based filtering were implemented in a way that prioritized usability.
    - The ability to sort and filter tasks by priority and completion status added significant value to the application, making it more versatile for users with diverse needs.
4. **Challenges and Lessons Learned**:
    - Balancing the BST was a notable challenge, especially when dealing with frequent additions and deletions. While a basic BST was implemented, future iterations could explore self-balancing trees like AVL or Red-Black trees for improved performance.
    - Debugging and testing were critical to ensuring the application's reliability. Edge cases, such as empty task lists or invalid user inputs, were handled to provide a robust user experience.
    - Collaboration and communication within the team were essential for aligning individual contributions and ensuring the project's success.

Conclusion

The to-do list application project successfully achieved its objectives by leveraging the strengths of **linked lists**, **hash tables**, and **binary search trees** to deliver a functional and efficient task management system. The application not only met the technical requirements but also provided a user-friendly interface that enhanced its practicality.

This project underscored the importance of selecting the right data structures for specific tasks and highlighted the value of teamwork in solving complex problems. Each team member—**Mohamed Idhbea**, **Mohamed Alosta**, and **Ibrahim Ahtash**—played a critical role in implementing and integrating the core components of the application, demonstrating the power of collaboration in software development.

Moving forward, the application could be further enhanced by incorporating additional features, such as task deadlines, reminders, or cloud-based synchronization. Moreover, exploring more advanced data structures or frameworks could improve performance and scalability. Overall, this project was a valuable learning experience, providing hands-on exposure to data structures, Python programming, and team-based software development.