

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

Data Structure and Algorithm

(ES221)

**Report:** The Student Helper Program.

**Submitted to:** Dr. Zubair Ahmad Associate Professor FCSE

**Student Name:** Rana Muhammad Ayaan Azam

**Registration No:** 2023596

Okay, here is a further condensed version of the report, formatted using only alphanumeric characters and numbers for structure, aiming to fit within the 7-page limit.

REPORT: STUDENT HELPER PROGRAM

1. PROJECT OVERVIEW

The Student Helper Program is a console application designed to assist students in three key areas: financial management, academic integrity, and time management. It comprises three interconnected modules accessible through a central command-line interface.

1.1 Budget Manager: This module helps students manage finances by defining a budget, allocating funds to prioritized categories, and dividing budgets into smaller periods. It supports preset or custom percentage allocations and tracks priority access history.

1.2 Plagiarism Detector: This module checks for text similarity between a source and a target document. It uses n-grams and Jaccard similarity to calculate a plagiarism percentage and can generate reports with potential matches and text complexity analysis.

1.3 Time Manager: This module functions as a task scheduler. Users can add tasks with priorities and deadlines, define dependencies between tasks, and get a valid execution order using topological sort. It allows managing active, completed, and deferred tasks.

1. IMPLEMENTED DATA STRUCTURES AND ALGORITHMS

The program's functionality relies on specific data structures and algorithms.

2.1 Budget Manager

a. Data Structures:

i. Doubly Linked List: Used to store budget priorities in an ordered manner.

ii. Stack: Used for tracking history of accessed priorities (LIFO).

iii. Arrays: Used for quick validation of priority names and ranks.

b. Algorithms:

i. Linear Search: Finds specific priorities within the linked list.

ii. Arithmetic Calculations: Performs budget allocation and division calculations.

iii. Recursive Traversal: Used for displaying budget details.

2.2 Plagiarism Detector

a. Data Structures:

i. Hash Map (Custom Implementation): Stores source n-grams for efficient average O(1) lookup during comparison. Uses chaining for collision handling.

ii. Dynamic Arrays (Vectors): Store tokens and common words.

iii. Linked Lists (std::list): Temporarily store generated n-grams.

b. Algorithms:

i. String Tokenization: Breaks text into words.

ii. N-gram Generation: Creates sequences of words for comparison.

iii. Hashing Algorithm: Computes indices for the hash map.

iv. Jaccard Similarity Coefficient: Quantifies similarity between n-grams.

v. Flesch Reading Ease: Calculates text readability score.

vi. Common Word Filtering: Removes stop words to focus analysis.

2.3 Time Manager

a. Data Structures:

i. Priority Queue (Min-Heap): Manages active tasks based on priority, providing efficient O(log N) access to the highest-priority task.

ii. Doubly Linked List: Stores completed and deferred tasks.

iii. Graph (Implicit Adjacency List): Represents task dependencies using an array of nodes with dependency lists.

iv. Arrays: Underlying storage for the heap and graph nodes.

b. Algorithms:

i. Heapify (Up and Down): Maintains the min-heap property in the priority queue.

ii. Topological Sort (Kahn's variant): Determines a valid task execution order respecting dependencies and detects cycles.

iii. Graph Traversal: Used implicitly within topological sort and dependency addition.

1. PERFORMANCE ANALYSIS

Performance varies by module based on data structures and input size.

3.1 Budget Manager:

* Time Complexity: O(1) for insertions/stack ops, O(N) for search/display (N = number of priorities).
* Space Complexity: O(N) for priorities, O(H) for history stack.

3.2 Plagiarism Detector:

* Time Complexity: Dominated by file size (M) and number of tokens (T). HashMap operations are average O(1), but N-gram generation and comparison scale with T. Overall heavily dependent on text length and n-gram size.
* Space Complexity: O(M\_source + M\_target) for text/tokens, O(Ng\_source) for source n-grams in HashMap. Can be significant for large texts.

3.3 Time Manager:

* Time Complexity: O(log N) for priority queue ops (N = active tasks). O(V+E) for Topological Sort (V = tasks, E = dependencies). O(N) for removing tasks by name from lists/queue (if search is needed).
* Space Complexity: O(N) for priority queue, O(C) for completed, O(P) for pending, O(V+E) for graph representation.
* Note on Fixed-Size Arrays: Use of fixed-size arrays in the graph and priority queue limits scalability to a predefined maximum capacity.

1. CHALLENGES FACED AND SOLUTIONS

Development involved addressing several challenges:

1. Managing Diverse Needs: Solved by adopting a modular design, separating concerns into three distinct modules.
2. Efficient Text Comparison: Addressed using an n-gram approach with a custom Hash Map for fast lookups and Jaccard similarity for quantifying matches. Common word filtering also helped reduce noise.
3. Prioritization with Dependencies: Resolved by combining a Priority Queue (Min-Heap) for task execution order based on priority and a directed Graph with Topological Sort for managing and validating dependencies.
4. Designing Module Logic: Selected specific data structures (Doubly Linked List for Budget, Priority Queue/Graph for Time) tailored to the core requirements of each module.
5. Console Interface Usability: Implemented a clear, menu-driven system with numbered options and input validation.
6. Data Persistence: Partially addressed for plagiarism reports (file output); recognized as a limitation for other modules where data is in-memory.
7. Fixed-Size Structures: Managed within project scope by choosing a reasonable fixed size, noting dynamic structures as a future improvement.
8. FUTURE IMPROVEMENTS

Potential enhancements include:

1. GUI Development: Transition to a graphical interface for improved user experience.
2. Data Persistence: Implement saving/loading data for all modules using file I/O or a database.
3. Dynamic Structures: Replace fixed-size arrays with dynamic data structures to enhance scalability.
4. Advanced Plagiarism: Implement more robust algorithms (e.g., fingerprinting), semantic analysis, or comparison against external sources.
5. Budget Features: Add expense tracking, recurring entries, and visual reporting.
6. Time Features: Incorporate calendar views, reminders, sub-tasks, and categorization.
7. User Accounts: Add support for multiple users to store data separately.
8. Robust Handling: Improve error handling and input validation.
9. Code Structure: Refactor code using header/source files and a build system.
10. Module Integration: Explore linking features between modules.
11. COURSE CONCEPTS UTILIZATION

The project applied fundamental data structures and algorithms concepts:

1. ADTs and OOP: Used classes and structs to represent program entities (Task, Priority, Managers, Structures), demonstrating encapsulation and abstraction.
2. Linear Structures: Employed Arrays for storage and lookup, Linked Lists (Singly and Doubly) for ordered data and history, Stacks for LIFO history tracking, and Priority Queues (heap-based) for priority ordering.
3. Non-Linear Structures: Utilized Heaps (Min-Heap) as the underlying structure for the priority queue and Graphs (Implicit Adjacency List) to model and process task dependencies.
4. Algorithmic Techniques: Implemented Searching (Linear, Hash Lookup), Sorting (Topological Sort), String Processing (Tokenization, N-grams, Hashing, Similarity), Recursion (for display), and Greedy approach (task execution).
5. Algorithm Analysis: Considered Time and Space Complexity (Big O) when choosing and implementing data structures and algorithms.
6. File I/O: Implemented reading from and writing to files for plagiarism analysis and reporting.
7. Software Principles: Applied modularity, abstraction, and encapsulation in the design.

This report summarizes the design, implementation, and key aspects of the Student Helper Program, highlighting the data structures and algorithms employed and outlining areas for future development.