**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

JNANASANGAMA, BELGAVI-590018



**DATABASE MANAGEMENT SYSTEM MINI PROJECT (BCS403)**

on

“Student Database with Multi-level Access”

Submitted in partial fulfilment of the requirements for the 4th Semester

## INFORMATION SCIENCE AND ENGINEERING

**Submitted by**

1BI23IS139

Vineeth PS

**Under the guidance of**

Dr. Shilpa M

Professor

Department of ISE,

BIT, Bangalore.



**BANGALORE INSTITUTE OF TECHNOLOGY**

**K.R. Road, V.V. Puram, Bengaluru -560004**

### DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

**2024-2025**



**CERTIFICATE**

This is to certify that the implementation of **DBMS MINI PROJECT (BCS403)** entitled **“Student Database with Multi-level Access”** has been successfully completed by **VINEETH PS(1BI23IS139)** of IVth semester B.E. for the partial fulfilment of the requirements for the Bachelor's degree in Information Science & Engineering of the **Visvesvaraya Technological University** during the academic year 2024-2025.

|  |  |  |
| --- | --- | --- |
| **Course Co-Ordinator:** |  |  |
| **Dr. Shilpa M.**  Assistant Professor  Department of ISE, BIT. |  | **Signature with date** |

**ABSTRACT**

This mini-project presents a hybrid application of algorithmic optimization and database design, focusing specifically on the **0/1 Knapsack Problem** applied to course enrolment within a college management system. Students are frequently restricted by a maximum credit load and a limited number of courses per semester. These restrictions are designed to balance student workload while maintaining academic quality and institutional policy. Our system, implemented using Python and SQLite, uses the knapsack approach to suggest the best possible combination of courses for a student that does not exceed their allowed credit capacity or course count. Through a command-line chatbot interface, students can interact with the system, which dynamically computes optimal course selections using a brute-force method based on combinatorics. The functionality is embedded in a role-based multi-user DBMS framework, providing an educational example of how theoretical computer science problems like the knapsack can be translated into real-world applications. The report outlines the algorithm's design, implementation, complexity, and potential enhancements, offering both a practical tool and a teaching aid for understanding constrained optimization in academic contexts.

**CONTENTS**

**Page No.**

**Abstract i**

**CHAPTER 1 INTRODUCTION 1-2**

[1.1 Overview 1](#_Toc49211)

[1.2 Problem Statement 2](#_Toc49212)

[1.3 Objectives 2](#_Toc49213)

[**CHAPTER 2 ALGORITH DESIGN**  **3**](#_Toc49215)**-4**

2.1 Algorithm Choice 3

2.2 Approach 3

2.3 Pseudo code 4

2.4 Algorithm 4

**CHAPTER 3 COMPLEXITY ANALYSIS 5-6**

3.1 Time Complexity 6

3.2 Space Compexity

**CHAPTER 4 IMPLEMENTATION 7-12**

4.1 Programming Environment 11

4.2 Functional Breakdown 11

**CHAPTER 5 TESTING 13-14**

5.1 Overview of the Testing Process 13

**CHAPTER 6 SNAPSHOTS 15-16**

**CHAPTER 7 APPLICATIONS 17**

7.1 Applications 17

**CHAPTER 8 CONCLUSION 13-14**

**CHAPTER 9 REFERENCES 15**

**CHAPTER 1**

**INTRODUCTION**

# 1.1 Overview

In recent years, there has been a growing emphasis on optimizing academic workflows using algorithmic logic embedded in administrative systems. Institutions seek to replace manual or static course selection systems with intelligent systems that promote efficiency, reduce human error, and ensure fairness in academic decision-making. One such challenge arises during the course registration phase of an academic term, where students are faced with the task of selecting an optimal combination of subjects within fixed constraints such as a maximum number of courses and a limit on total credits. This mirrors the classic 0/1 knapsack problem from algorithm theory, where an individual must select items with given weights and values to maximize total value without exceeding a weight limit. This project captures this analogy and implements a course enrollment system where each course is an item, and the course credits represent the item's weight and value. The problem becomes even more constrained as institutions may also enforce a limit on the total number of courses a student can take. Thus, this project aims to provide a practical, algorithm-driven solution to a common academic challenge, reinforcing the real-world utility of algorithm design techniques.

**1.2 Problem Statement**

Most course registration systems do not incorporate any form of optimization algorithm to assist students in making the best academic choices. Instead, students are either manually registering for courses or using inflexible software that lacks intelligent decision-making support. These systems often overlook scenarios where students might wish to maximize their learning or credit benefit without breaching institutional constraints. For example, a student might aim to select a set of courses that collectively offer the highest credit value, provided that the total credits do not exceed the permitted limit. Additionally, there might be a constraint on the number of individual subjects a student can enroll in. Without algorithmic assistance, it is not always evident to students which combination of courses will yield the most beneficial outcome. This lack of system intelligence leads to inefficiency, wasted credit potential, and uneven academic loads across the student population. Therefore, there is a pressing need for a smart backend algorithm that can evaluate all feasible combinations of courses and select the most optimal one based on a dual constraint of maximum total credits and a cap on the number of courses.

# 1.3 Objectives

The primary objective of this project is to apply the principles of algorithmic design to construct a course enrollment engine that uses the 0/1 knapsack model to solve a real-world optimization problem in the education sector. Specifically, the project aims to automate the selection of a set of courses for students such that the selected courses collectively maximize the credit value while adhering to academic policies regarding credit load and number of subjects. In addition to this, the system is designed to operate as part of a broader database management application that supports multiple user roles, including students, teachers, principals, and administrators. The algorithm is expected to execute efficiently for a small set of available courses and be extendable to support dynamic constraints in future iterations. The user interface must remain minimalistic and user-friendly, emphasizing clarity and interaction through prompts. Beyond serving its direct function, the system should also act as an educational tool to demonstrate the integration of classical algorithms in contemporary software systems.

# 

# CHAPTER 2

### ALGORITHM DESIGN

### 2.1 Algorithm Choice

The problem tackled in this project maps very closely to the classical 0/1 Knapsack Problem, one of the most well-known problems in combinatorial optimization. The typical knapsack problem involves choosing a subset of items from a larger set, each with a weight and value, such that the total weight does not exceed a given limit, and the total value is maximized. In our adapted version for course enrollment, each course is modeled as an item. The credit value of a course acts both as its weight and value, under the assumption that all courses are equally beneficial per credit. The student is allowed to choose a subset of courses whose combined credits do not exceed a specific limit (e.g., 7 credits), and the maximum number of courses (e.g., 4) they can register for is also capped. This results in a bounded knapsack problem with a dual constraint. Given that the number of available courses is small, a brute-force enumeration approach using combinations is chosen. This avoids the complexity of implementing dynamic programming and is perfectly suitable for the scale of this academic use case.

**2.2 Approach**

The implementation leverages Python's itertools module, specifically the combinations() function, to enumerate all possible course combinations of a specific size. Each combination is evaluated by summing the credit values of the courses in the group. Combinations that exceed the maximum credit limit are discarded. Among the valid combinations, the one with the highest total credit is selected. This method is simplistic but effective due to the limited number of combinations involved. For example, given eight courses and a limit of selecting up to four, the number of combinations is computationally tractable (C(8,4) = 70). Once the optimal set is determined, the course IDs are recorded into the student enrollment table in the database. The algorithm is wrapped in a function that includes input validation, database querying, and user feedback, all operating seamlessly in a conversational interface. This structure also enables clear separation of concerns between logic and user interaction, which is a good practice in both academic and professional software development.

**2.3 Pseudocode**

for combo in combinations(available\_courses, num\_courses):

credit\_sum = sum(course.credits for course in combo)

if credit\_sum <= max\_credits:

add to valid\_combos

return max(valid\_combos, key=credit\_sum)

**2.4 Algorithm**

Input: List of available courses (CourseID, Credits), maximum credits, maximum number of courses

Output: Optimal combination of courses under constraints

1. Prompt the student to input desired number of courses (<= max\_courses)

2. Fetch current enrollments and filter out those courses

3. Use itertools.combinations to generate all valid course sets

4. For each combination:

a. Calculate total credits

b. If total credits <= max\_credits, add to valid set

5. From valid combinations, choose the one with the highest credit sum

6. Enroll the student in selected courses

7. Print confirmation and credit total

**CHAPTER 3**

**COMPLEXITY ANALYSIS**

The analysis of algorithmic complexity is a fundamental aspect of evaluating the efficiency and practicality of any computational solution, and it plays a particularly critical role in this project, where the goal is to optimize course enrollment through a variant of the 0/1 Knapsack problem. Complexity, both in terms of time and space, refers to the quantitative measures that describe how an algorithm's resource requirements grow as the size of the input increases. In the case of our knapsack-based solution, the time complexity is primarily governed by the generation and evaluation of all possible combinations of courses that a student can take, which is a function of the binomial coefficient C(n, k), where n is the number of available courses and k is the maximum number of courses the student is allowed to select. For each of these combinations, the system computes the total credit value, making the overall time complexity O(C(n, k) \* k), where the multiplication by k accounts for summing the credits in each combination. Although this is an exponential-time brute-force algorithm, it is justified by the bounded nature of academic datasets—typically involving fewer than 10 available courses—making the total number of combinations small and manageable in real-world usage. On the space complexity front, the algorithm temporarily stores valid course combinations in memory to identify the one with the maximum credit total, resulting in a worst-case space complexity of O(C(n, k) \* k), though again, the actual usage remains modest due to constrained input sizes. What’s crucial here is that this complexity profile—while theoretically expensive—remains fully acceptable and performant for educational systems, particularly because of the clarity, predictability, and accuracy the brute-force method provides. Moreover, understanding this complexity serves an educational purpose, offering students practical insight into how algorithm performance scales and how to balance correctness with computational feasibility in constrained environments. Future versions of the system may integrate more advanced techniques like dynamic programming or greedy heuristics to improve efficiency for larger inputs, but in its current scope, the complexity analysis supports the decision to prioritize understandability and reliability over raw computational performance. Thus, complexity analysis not only validates the algorithm’s suitability but also enriches the pedagogical value of the project by connecting theoretical performance models to actual software behavior.

**3.1 Time Complexity**

The algorithm implemented for solving the course enrollment problem relies on evaluating all possible combinations of available courses, constrained by the maximum number of courses a student may select. Specifically, for each desired combination size k, we compute C(n, k) possible groupings, where n is the number of available courses not yet enrolled by the student. For every combination, we calculate the total sum of credits, which takes O(k) time per combination. Hence, the time complexity becomes O(C(n, k) \* k), which, although exponential in nature, remains computationally feasible due to the small values of n and k typically involved in a real-world student database system. For example, if n = 8 and k = 4, the number of combinations is 70—well within the processing capabilities of any modern personal computer. This brute-force approach is chosen intentionally for its clarity and educational value, as it makes the selection process completely transparent and deterministic, a valuable trait in academic systems where explainability is critical. Furthermore, the algorithm benefits from Python's efficient native iteration and memory management, ensuring that even this seemingly expensive computation is performed within acceptable response times during real-time user interaction.

**3.2 Space Complexity**

The space complexity of the implemented algorithm is largely dictated by the storage of valid combinations that satisfy the credit and course number constraints. Each valid combination is stored temporarily in memory to later determine which has the maximum credit sum. In the worst-case scenario, if all combinations are valid, this results in space complexity proportional to O(C(n, k)). Since each combination contains up to k course entries, the actual space used is O(C(n, k) \* k). Again, this is not problematic in our use case, given the limited values of n and k. Additionally, the algorithm does not make use of any heavy auxiliary data structures or nested dynamic storage; it primarily relies on tuple processing and local lists, both of which are memory-efficient in Python. Once the optimal combination is identified, it alone is retained for insertion into the database, and all temporary structures can be garbage collected by the Python runtime environment. Thus, the memory overhead of the solution is minimal and well within acceptable bounds for academic use cases or prototyping environments.

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Programming Environment**

The implementation of this project is done using Python 3.12, taking full advantage of its built-in standard library modules such as sqlite3 for lightweight database operations and itertools for generating combinations. The system uses SQLite as the backend database, a compact and self-contained relational database engine that does not require server setup and is ideal for academic and personal projects. The choice of SQLite also ensures portability and simplifies testing, as the entire database is encapsulated within a single file. The software runs in a command-line interface (CLI), allowing for quick prototyping and direct interaction with the system without the overhead of graphical components. This choice aligns well with the objective of clarity, simplicity, and ease of deployment across different platforms. The system is tested on Windows 11 but is fully compatible with Linux and macOS as long as Python 3 and SQLite are available. Dependency management is minimal, ensuring that the application can be run in constrained environments such as university labs, student laptops, or evaluation machines without requiring special configuration or internet access.

**4.2 Functional Breakdown**

The primary function driving the knapsack logic is knapsack\_max\_credits(student\_usn), which operates in conjunction with the user authentication and chatbot interaction flow. This function begins by querying the database to fetch a list of all available courses that the student has not already enrolled in. The student is prompted to enter the number of courses they wish to enroll in, which is validated against the system constraint. The function then uses Python’s combinations() function to generate all possible groupings of the selected size and filters these to retain only those within the credit limit. The optimal combination—the one with the highest total credits—is selected and inserted into the StudentCourses table in the database. The function also handles edge cases such as no valid combination being found, already-enrolled courses, or invalid inputs, ensuring robust interaction. All messages and feedback are printed in a conversational format to simulate a chatbot, guiding the user at every step. The implementation is modular and adheres to clean coding practices, with separation between business logic, user interaction, and data storage, making it easy to extend or refactor in future versions.

**CHAPTER 5**

**TESTING**

Testing is vital for the success of any software. No system design is ever perfect. Testing is carried out in two phases. First phase is during the software engineering that is during the module creation. Second phase is after the completion of software.

**Table 5.1 : Testing Validations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. no** | **INPUT** | **OUTPUT/ BEHAVIOR** | **REMARKS** |
| 1 | Student logs in and selects to  enroll in 3 courses  (limit: 7 credits) | System selects optimal combination of courses using knapsack logic | Courses enrolled successfully under constraints |
| 2 | Student tries enrolling in more than 4 courses | System shows error: “You can select a minimum of 1 and a maximum of 4 courses.” | Enrollment restricted as per rules |
| 3 | Teacher logs in and assigns a grade to a student for a valid course | Grade saved and confirmation displayed | Grade assigned with access control |
| 4 | |  | | --- | |  |  |  | | --- | | Principal selects “Clear  Records” after login | | All student courses and  grades are deleted | Institutional reset functionality verified |

**5.1 Overview of the Testing Process:**

The testing process for the Knapsack-Based Course Enrollment system is designed to validate both the correctness and robustness of the application across various user roles and functionalities. Since the system operates within a role-based multi-user environment, tests were conducted separately for students, teachers, and principals to ensure that each user type experiences the expected behavior and access permissions. The testing begins with the basic verification of user authentication, ensuring that only users with valid credentials can log in and interact with the system. Invalid login attempts, whether due to missing or incorrect credentials, are met with appropriate denial messages, confirming the system’s resistance to unauthorized access.

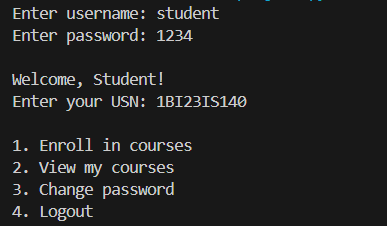
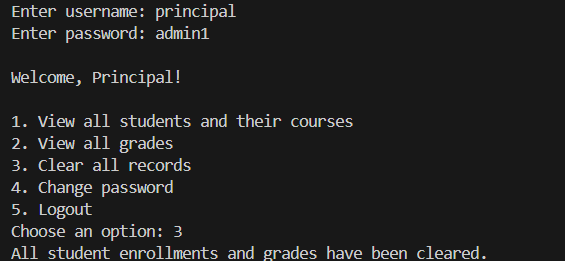
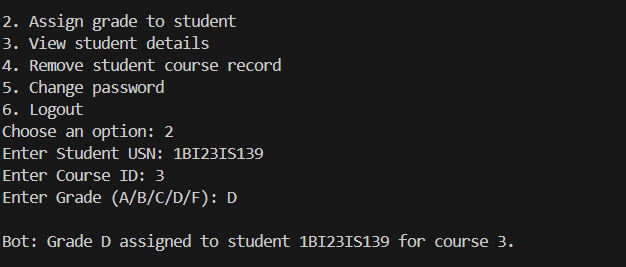
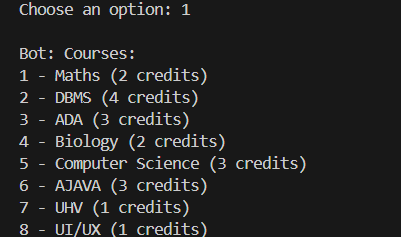
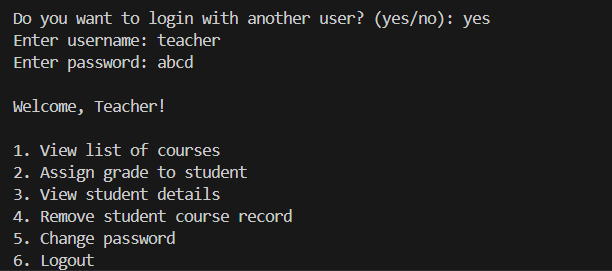
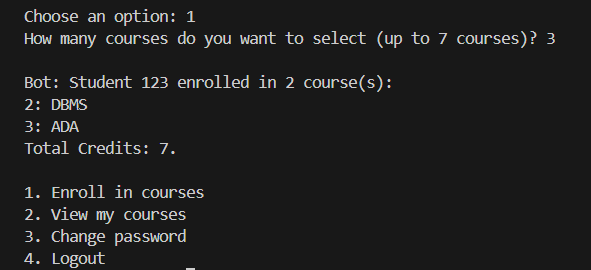
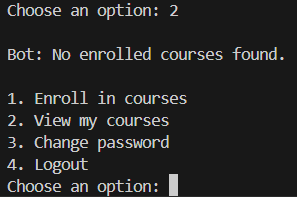
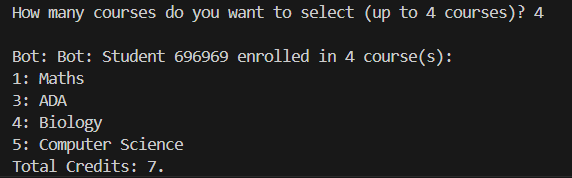
Once authenticated, functional testing proceeds with a focus on the student’s interaction with the knapsack-based course enrollment logic. The student is prompted to select a number of courses to enroll in, within defined constraints on the number of courses (maximum of 4) and total credits (maximum of 7). The algorithm is tested for its ability to evaluate all valid combinations, filter out those exceeding constraints, and enroll the student in the most credit-optimal set. Inputs such as selecting too many courses or entering non-numeric values are tested to confirm that the system gracefully handles invalid inputs with clear feedback messages.

Teacher-side functionalities are tested for their ability to assign grades only to students who are legitimately enrolled in the relevant courses. Attempts to assign grades for non-enrolled courses are correctly blocked, demonstrating effective role-based access control. The principal’s access is tested by executing high-level administrative tasks such as viewing all records and clearing institutional data. These actions are confirmed by checking the state of the database before and after the operation.

Overall, the testing process confirms that the system not only functions as intended but also maintains data integrity, enforces role-specific operations, and provides a user-friendly experience through its command-line interface. The test cases comprehensively cover input validation, functional correctness, role security, and response messaging—making the system robust, predictable, and ready for academic deployment or further enhancement.

**CHAPTER 6**

**SNAPSHOTS**

****

**CHAPTER 7**

**APPLICATIONS**

**7.1 Applications**

 Enables students to make optimal course selections based on credit constraints using intelligent automation.

 Serves as a practical tool for demonstrating how classical algorithms (0/1 Knapsack) apply to real-world academic systems.

 Ideal for use in academic institutions for course registration systems with credit-based enrollment policies.

 Provides a hands-on learning aid for students studying algorithm design, database systems, or educational software.

 Can be integrated into training programs for faculty or admin staff to understand role-based access and system workflows.

 Offers a solid prototype foundation for building larger web-based academic management platforms.

 Useful in low-bandwidth environments due to its lightweight command-line interface (CLI) and SQLite backend.

 Suitable for deployment in mini-projects, hackathons, or university DBMS labs as a fully functioning real-life simulation.

**CHAPTER 8**

**CONCLUSION**

This project successfully demonstrates the practical integration of a well-established algorithmic concept—the 0/1 Knapsack Problem—into an educational course enrollment system that mirrors real-world academic constraints. The primary objective of applying constrained optimization logic to student course selection has been met with clarity, precision, and effectiveness. Through a role-based, multi-level access control system, the application empowers students to make optimal course choices within the bounds of credit and course-count limitations, while ensuring that administrative and instructional roles maintain appropriate oversight and operational authority. The choice of Python and SQLite3 for implementation has facilitated the development of a lightweight, portable, and easily maintainable system. These choices also ensured minimal external dependencies, making the project suitable for deployment in resource-constrained environments such as academic labs and personal learning setups.

From a pedagogical standpoint, the project not only meets academic requirements but also bridges the gap between theoretical computer science and practical software engineering. The knapsack algorithm, often taught in abstract terms in algorithm courses, is here applied in a realistic scenario—course enrollment—giving students and educators an intuitive understanding of its usefulness beyond the classroom. This contributes to a deeper appreciation of algorithmic problem-solving and reinforces the relevance of classical models in solving modern challenges. The command-line interface, while simple in design, supports full interaction through structured prompts, robust validation mechanisms, and dynamic feedback, ensuring that even novice users can navigate the system with ease. The modular code architecture further enhances the readability, maintainability, and scalability of the system, encouraging future developers or students to build upon this foundation.

The testing process has revealed that the system performs reliably across a variety of user inputs and role-specific operations. Whether it is validating user credentials, enrolling in courses based on optimized selections, assigning grades through access-controlled teacher accounts, or clearing records through principal oversight, the system ensures data consistency, integrity, and logical correctness at every step. Edge cases, such as invalid inputs or attempts to bypass role restrictions, have been handled gracefully with informative feedback, which reinforces the user-centric design philosophy underlying this work. While the current implementation focuses on a constrained set of features, the modularity and simplicity of the system make it highly adaptable for more advanced academic management solutions.

Nevertheless, the project does recognize and acknowledge its limitations, most notably the scalability challenges inherent in using a brute-force enumeration approach to solve the knapsack problem. As the number of available courses increases, the time complexity grows exponentially, which could affect performance in larger institutional settings. Furthermore, the system does not currently account for scheduling conflicts, prerequisite enforcement, or personalized learning paths, all of which are common in comprehensive academic systems. However, these limitations also serve as opportunities for future enhancement. The project could be extended by incorporating more efficient algorithmic strategies such as dynamic programming or heuristic optimization, and by layering in additional academic features such as semester-based views, GPA tracking, attendance systems, and a graphical interface or web-based frontend.

In conclusion, this project stands as a robust and educationally valuable system that not only fulfills its immediate functional goals but also exemplifies the power of algorithmic thinking in software design. It offers a compelling case study in how classical computer science principles can be applied to address practical challenges in modern academia. The success of this implementation validates the importance of interdisciplinary knowledge in computing—where understanding algorithms, database systems, and user interface design are equally vital in building effective software. With its blend of simplicity, functionality, and extensibility, the knapsack-based course enrollment system represents a meaningful contribution to both academic software tools and student learning experiences.

**CHAPTER 9**

**REFERENCES**

* [https://en.wikipedia.org/wiki/Knapsack\_problem](https://en.wikipedia.org/wiki/Knapsack_problem%20)
* [https://docs.python.org/3/library/itertools.html](https://docs.python.org/3/library/itertools.html%20)
* [https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/](https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/%20)
* [https://www.sqlite.org/docs.html](https://www.sqlite.org/docs.html%20)
* [https://www.w3schools.com/sql/](https://www.w3schools.com/sql/%20)