ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

Strategies for Improving the Teaching Efficiency of Computer Related Courses in Big Data Majors: A Case Study of *Data Structures*

Jingwen Ye, Xiangqing Zhao*, Zhuangqi Ding, Shiyin Zhao

School of Mathematics and Physics, Suqian University, Suqian, 223800, China
**Corresponding author: zhao-xiangqing@163.com*

Abstract: *In the cultivation of computer science-related talent, data structures serve as a fundamental cornerstone course. They directly influence students' mastery of subsequent specialized subjects and their ability to solve engineering problems. At present, students commonly encounter difficulties such as conceptual abstractness, a disconnection between theory and practice, and weak knowledge-transfer skills. These challenges make it difficult for them to flexibly apply foundational structures to complex scenarios, thereby limiting the development of programming and algorithmic thinking. This study explores strategies to enhance the efficiency of data-structure learning, including strengthening prior knowledge reserves, deepening theoretical comprehension, reinforcing hierarchical practical training, and expanding the boundaries of knowledge application. These strategies operate across four interconnected levels: foundational support, cognitive deepening, ability transformation, and vision expansion. Specifically, early preparation lays the groundwork for efficient learning; theoretical study builds a systematic knowledge framework; practical training enables the transition from theory to skill; and knowledge expansion aligns learning with emerging disciplinary demands. Overall, these strategies help bridge the gap between theoretical learning and real-world application, equipping computer science students with both structural understanding and applied competence needed to address complex challenges in operating systems, databases, artificial intelligence, and related fields.*

Keywords: *Data Structure Learning; Computer Science Education; Theoretical-Practical Integration; Knowledge Transfer; Learning Strategies*

1. Introduction

Against the backdrop of rapid advancements in computer science and technology, data structures—as the foundation of software system design and optimization—play an increasingly vital role. Yan Weimin and others, in *Data Structures* (C Language Edition), point out that the study of data structures focuses on how data are organized, stored, and manipulated, forming the basis for system software such as operating systems and compilers, as well as for various application software[1]. This viewpoint highlights the pivotal role of data structures within the knowledge system of computer science and illustrates how students' mastery of data structures is closely tied to the depth of their subsequent learning and engineering practice.

However, nowadays computer science students commonly face challenges in learning data structures. Learners frequently report difficulty in understanding abstract concepts and grasping the adaptation between algorithms and structures. Consequently, they struggle to build systematic knowledge frameworks and often fall into a fragmented learning state of "knowing theory but not application."

Relevant studies provide valuable guidance in addressing these issues. Cormen et al., in *Introduction to Algorithms*, emphasize that the integration of data structures and algorithms forms the core of complex problem-solving[2]. Similarly, Deng Junhui, in *Data Structures and Algorithms* (Python Language Description), focuses on practical case-based approaches to make theory more accessible[3]. Building on this foundation, Wang et al. discuss how a "theory + practice" dual-track model stimulates students' learning initiative[4]. Hattie and Yates's research on Visible Learning further demonstrates that targeted practice and challenging tasks significantly enhance learning outcomes[5].

Taking these insights as a basis, this study centers on the entire learning process of data structures, aiming to establish a scientific methodology that transforms students from passive recipients into active

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

knowledge users. The paper systematically examines learning preparation, theoretical understanding, hands-on practice, and knowledge expansion to offer practical guidance for improving learning outcomes in data structures.

The remainder of this paper is organized as follows: Section 2 analyzes the limitations students face in learning *data structures*; Section 3 proposes core strategies to enhance learning effectiveness; Section 4 discusses curriculum improvement through practical examples; and Section 5 summarizes conclusions and future prospects.

2. Limitations in Students' Learning of *Data Structures*

This section analyzes the core issues that students commonly encounter when learning *data structures*, focusing on three aspects: conceptual understanding, theoretical–practical integration, and knowledge transfer. Most students can grasp basic definitions and simple operations, but they lack a rational understanding of structure selection and algorithmic efficiency in complex contexts[2].Consequently, they tend to "mechanically apply" knowledge without flexible adaptation. Furthermore, many students study structures in isolation, neglecting the interrelationships among them, which makes it difficult to identify logical flaws or performance bottlenecks when handling composite structures.

2.1 Difficulty in Understanding Abstract Concepts

At the cognitive level, the foremost limitation lies in the difficulty of understanding abstract concepts. In most teaching contexts, instruction is dominated by theoretical exposition—focusing on definitions and basic operations of linear and hierarchical structures—while neglecting conceptual deconstruction and visual presentation. Teachers rarely integrate logical structures (such as binary tree hierarchies) with physical storage models (like linked storage using pointers) or relate them to real-world analogies that illustrate why a particular structure is chosen and how it optimizes problem-solving.

Concept explanations often revolve around formal definitions rather than contextual comprehension. Without guided visualization, diagrammatic reasoning, or relatable analogies, students struggle to transform abstract logic into concrete mental models. They tend to memorize definitions mechanically without understanding structural applicability or core value. Ultimately, this hinders their ability to learn and apply more complex structures.

2.2 Disconnection Between Theory and Practice

At the level of ability transformation, the primary problem lies in the disjunction between theoretical knowledge and practical implementation, specifically, inadequate programming and problem-solving capabilities. Many students fail to bridge theory and practice, neither attempting to implement fundamental operations (such as linked list insertions or binary tree traversals) nor systematically organizing the process of "theoretical features → code logic → debugging and optimization."

This imbalance, emphasizing theory while neglecting practice, prevents students from translating abstract properties into concrete programming constructs. For instance, they cannot translate the "sequential storage" feature of linear lists into array-based implementation logic or accurately handle address conditions (like null nodes in linked lists)[3]. As a result, they struggle to operationalize data structure theory into executable solutions, severely limiting their practical competence.

2.3 Weak Knowledge Transfer Ability

In terms of application, students' learning limitations also manifest in poor knowledge transfer, particularly across different contexts or combined structural applications. Many learners view data structures as isolated knowledge units rather than a integrated toolkit for problem-solving. When faced with complex problems involving multiple structures, such as file systems combining trees and hash indices, they often apply single-structure solutions without engaging in problem decomposition, structure selection, or efficiency evaluation.

Students' limited understanding of how data structures are used in databases, operating systems, or

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

AI systems prevents them from connecting classroom learning with professional applications. This lack of transferability diminishes the value of data structure knowledge and constrains their ability to address complex engineering abilities (see Figure 1and 2 for the details).

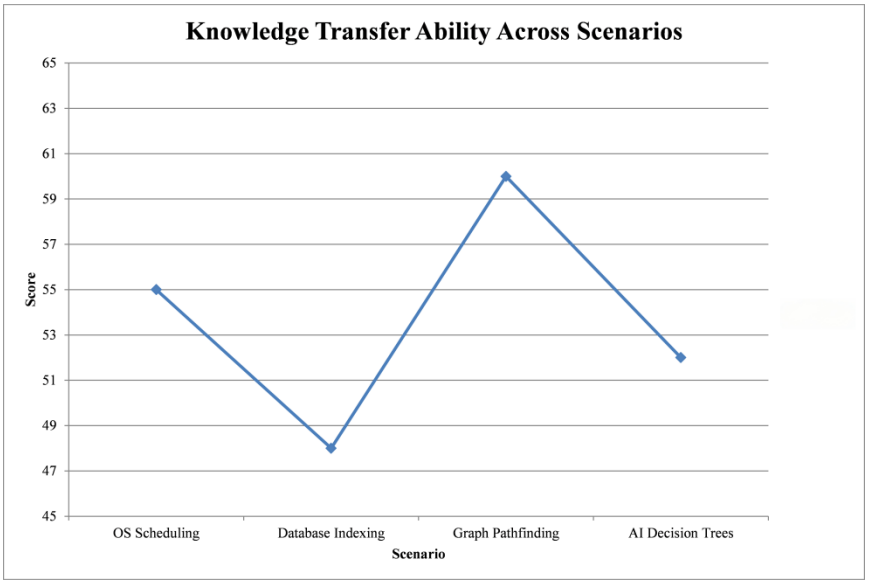

Figure 1. Knowledge transfer ability across scenarios

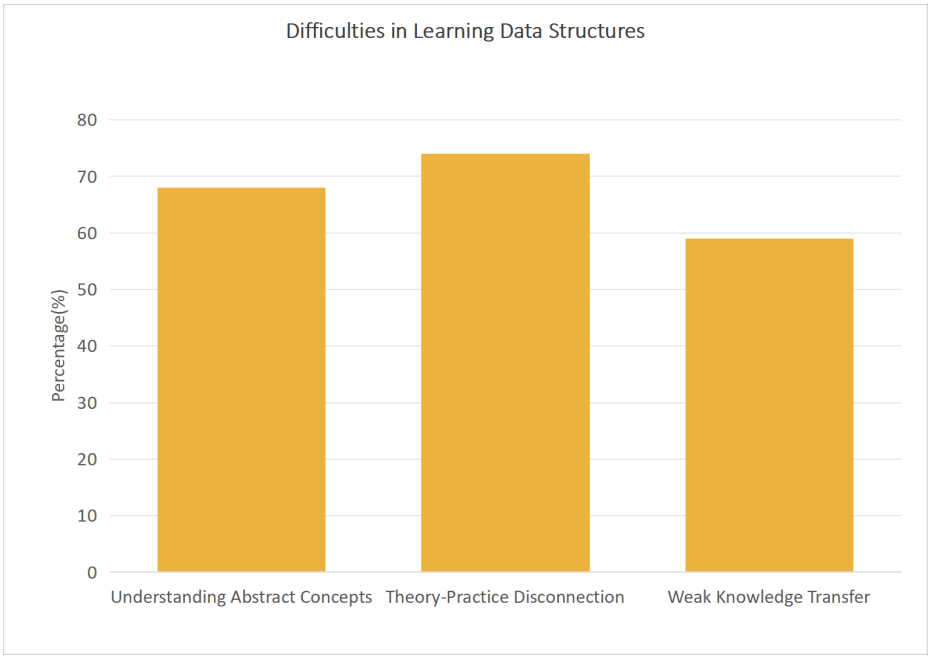

Figure 2. Difficulties in learning data structures

3. Core Strategies for Enhancing the Learning of *Data Structures*

This section proposes strategies to help students build a systematic understanding of *data structures* and develop strong problem-solving capabilities. These include strengthening foundational knowledge, deepening theoretical understanding, reinforcing layered practical training, and expanding the scope of knowledge application[6]. Together, these strategies aim to bridge the gap between theoretical mastery and engineering practice.

3.1 Strengthening Foundational Knowledge

Building a solid foundation is a prerequisite for efficient learning. Before formal study, students

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

should develop competence in two key areas: mathematics and programming. Mastery of discrete mathematics, particularly set theory and graph theory, facilitates understanding of data relationships and structural reasoning. Simultaneously, proficiency in at least one programming language (such as C or Java) is essential for translating abstract structures into implementable code.

For example, understanding pointers is fundamental to linked list manipulation, while grasping array behavior aids in implementing stacks or queues. Such preparatory knowledge minimizes learning barriers and facilitates cognitive linkage between mathematical logic → code implementation → structure application.

By systematically preparing in these areas, students can transition from a "zero-based" approach to a "foundation-supported" one, significantly lowering conceptual difficulty and paving the way for higher-level learning (see Table 1 for the details).

Table 1. Software development skill weakness, errors & time deviation metrics

Skill Area	Weakness Ratio (%)	Avg Error Count	Completion Time Deviation (%)
Code Implementation	72	12	38
Debugging Ability	65	19	45
Algorithm Reasoning	70	15	42
Real Case Application	76	22	53
Data Structure Optimization	68	14	40
Error Handling Strategy	63	18	46
Code Efficiency Evaluation	71	17	44

3.2 Deepening Theoretical Understanding

Deepening theoretical comprehension is central to improving learning outcomes. Teachers should guide students beyond rote memorization toward multi-dimensional exploration along the axes of structural properties, applicable scenarios, and efficiency analysis[5].

When teaching data structures like trees, a practical method is to first explain algorithmic logic, such as preorder and inorder traversals. After establishing this foundational understanding, instructors can extend the concept with real-world analogies. For example, one might compare a corporate hierarchy to a binary search tree or analyze the efficiency of a dictionary lookup using the principles of balanced versus unbalanced trees. This approach solidifies abstract concepts by connecting them to tangible, familiar scenarios.

Through such exercises, students learn not only what a structure is, but also why it is used and how it can be optimized. This fosters structured thinking and awareness of time–space trade-offs. By shifting from isolated memorization to systematic understanding, students cultivate the ability to select and optimize data structures for complex problems[1].

3.3 Reinforcing Layered Practical Training

Reinforcing practical training through a tiered system—basic implementation → classical problem-solving → project application—is key to transforming knowledge into competence.Basic implementation: Students code essential operations (e.g., linked list insertion/deletion, tree traversals) to master the handling of edge cases[7].Classical problem-solving: Students tackle targeted exercises on online platforms like LeetCode or NowCoder, starting from simple array searches and progressing to complex problems like shortest paths or balanced tree adjustments. Project application: Students apply multiple structures to mini-projects such as a file directory management system (tree-based) or a campus navigation system (graph-based).

This progressive model encourages active exploration. By managing projects independently, students conduct requirement analysis, structure selection, and performance optimization. This process fosters a strong sense of ownership and intrinsic motivation, effectively transforming them from passive learners into engaged, autonomous problem-solvers who are prepared for real-world challenges (see Table 2 for the details).

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

Table 2. Computer science concept learning metrics

Concept Type	Difficulty Level (%)	Misunderstanding Frequency (%)	Average Mastery Time (hrs)
Pointer Memory Model	78	55	6.2
Dynamic Memory Allocation	71	48	5.5
Tree Recursive Thinking	69	52	7.4
Balanced Tree Rotation Logic	74	58	8.1
Graph Adjacency Structures	63	42	5.0
Weighted Graph Representation	67	46	5.7
Algorithm-Structure Mapping	72	53	6.9
Time-Space Trade-off Reasoning	70	51	7.2

3.4 Expanding the Boundaries of Knowledge Application

Expanding knowledge application involves connecting classroom learning with cross-disciplinary and cutting-edge contexts. Teachers can illustrate the central role of data structures in fields such as Databases (e.g., B+ tree indexing for efficient retrieval), Operating systems (e.g., process scheduling via queues), Artificial intelligence (e.g., decision trees for classification)[9].

Advanced data structures such as skip lists, union–find sets, and segment trees can be introduced with real-world applications—e.g., skip lists in Redis sorted sets and union–find sets in network connectivity detection.

Students are encouraged to build personalized knowledge maps linking *Data Structures* with subsequent courses such as *Algorithm Design* and *Database Principles*. Peer discussions and sharing sessions, such as "Structure Selection Case Seminars," can further enhance understanding and knowledge integration.

By bridging classroom theory with real-world relevance, this strategy fosters motivation, broadens technical horizons, and strengthens knowledge transfer capability.

4. Curriculum Improvement Strategies

Building on the preceding analysis, this section presents two targeted strategies for improving practical learning, encouraging multiple-structure problem solving and introducing challenging practical tasks, to help students transition from passive learning to active application[4].

4.1 Encouraging Multiple-Structure Solutions for the Same Problem

To strengthen students' ability to evaluate trade-offs between data structures, instructors can require them to solve the same problem using multiple structures. For example, in data-retrieval tasks, students may compare the characteristics and performance of arrays for sequential storage, linked lists for linked storage, and hash tables for fast key-based access.

Arrays allow fast random access (O(1)) but have inefficient insertion and deletion; linked lists enable flexible insertions but require O(n) traversal for search; hash tables provide near O(1) retrieval and insertion but require resolving collisions.

By implementing and comparing these structures, such as in the 2020 Blue Bridge Cup Competition"*Data Statistics*" problem, students can observe how structure choice directly affects performance. This approach strengthens their analytical thinking and ability to match data structures to problem requirements.

4.2 Introducing Challenging Practical Tasks

Incorporating challenging tasks helps students step outside their comfort zones and apply multiple structures collaboratively. For example, the 2018 Blue Bridge Cup Competition Maze Pathfinding

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

problem requires integrating graph structures and queues (via BFS algorithms)[8].

Students must construct adjacency lists or matrices to represent the maze, apply queue-based BFS traversal, and track visited nodes using two-dimensional arrays. During implementation, they must complete the full workflow, problem decomposition, structure selection, coding, debugging, and optimization, while reasoning about algorithm choice (e.g., why BFS suits shortest path problems better than DFS).

Through such tasks, students gain a deeper understanding of collaborative structure use and transition from single-structure operation to multi-structure integration. Teaching experience shows that such tasks foster initiative, collaboration, and analytical depth, effectively strengthening students' problem-solving ability (see Table 3 for the details).

Table 3. Applied algorithm performance metrics

Application Scenario	Mastery Score (0-100)	Error Rate (%)	Cross-Scenario Transfer Score
OS Process Scheduling	57	22	41
Memory Page Replacement	52	26	37
DB Indexing (B+ Tree)	49	31	33
Graph Shortest Path (Dijkstra)	63	19	47
AI Decision Tree Pruning	55	23	42
Distributed Hash Storage	48	34	30
Cache Replacement Strategy	51	29	35

5. Conclusion

This study investigates how scientific learning methodologies can enhance the learning of data structures among computer science students. Addressing issues such as difficulty in conceptual understanding, the disconnection between theory and practice, and weak knowledge transfer, four key strategies were proposed: strengthening foundational preparation, deepening theoretical comprehension, reinforcing layered practice, and expanding knowledge boundaries.

Empirical application shows that strategies such as solving the same problem with multiple structures and engaging in challenging tasks (e.g., Blue Bridge Cup Competition problems) effectively enhance students' analytical and structural thinking. They foster the ability to construct a coherent cognitive chain: Problem → Structure → Algorithm → Efficiency.

This research provides theoretical and practical reference for teaching data structures. Future work may incorporate AI-assisted learning tools (e.g., code-debugging aids and structure visualization software) and strengthen interdisciplinary integration with courses like databases and operating systems. As technologies such as big data and distributed systems continue to evolve, ongoing exploration of adaptive and practice-oriented learning methods will be essential for the cultivation of innovative computer science professionals[10].

Acknowledgements

(1) College Students' Innovation and Entrepreneurship Projects for Suqian University (X2025141600358)

(2) 2025 Special Project for Teaching Reform Research of Suqian University (2025ZYRZ02)

(3) Key Project of the China Association of Higher Education's (25LK0203).

(4) The interdisciplinary Integration and Innovation Project of Suqian university (2025XKTD02)

(5) Suqian Sci & Tech Program (M202206)

References

[1] Y.Song, S.Jin. "Research on Teaching Content Reform of Data Structure for Data Science and Big Data Technology Major," in Proc. 13th Int. Conf. Inf. Technol. Med. Educ. (ITME), Wuyishan, China,

ISSN 2522-6398 Vol. 9, Issue 5: 231-237, DOI: 10.25236/FER.2026.090531

2023, pp.55-57.
[2] X Li, X.P. Fan, X.L. Qu, et al. Curriculum Reform in Big Data Education at Applied Technical Colleges and Universities in China[J]. IEEE Access, 2019, 7: 125511-125521.
[3] C.Q Wang, X.Q. Zhao, Z.W.Lv, et al. Curriculum Optimization for the Big Data Major through Industry-Education Integration Oriented toward Enhancing College Students' Professional Competencies[J]. Curriculum and Teaching Methodology, 2024, 7(8): 200-207.
[4] Z.S. Seidametova. Some Methods for Improving Data Structure Teaching Efficiency[J]. Educational Dimension, 2022, 6: 164-175.
[5] C.Q Wang, X.Q. Zhao, Z.W.Lv, et al. Exploration on the Development of the Big Data Major in Local Colleges and Universities with a Focus on Enhancing Regional Service Capability[J]. Transactions on Comparative Education, 2025, 7(1): 152-159.
[6] J.M. Wing. Computational thinking[J]. Communications of the ACM, 2006, 49(3): 33-35.
[7] L Porter, M Guzdial, C McDowell, et al. Success in introductory programming: what works?[J]. Communications of the ACM, 2013, 56(8): 34-36.
[8] C.G. Pérez, P.M. García, J.S. López. "Project-Based Learning Experience on Data Structures Course," in Proc. 2011 IEEE Global Eng. Educ. Conf. (EDUCON), 2010, pp.561-566.
[9] H Lv, J Liu. New teaching model of professional big data courses in universities based on an outcome-oriented educational concept[J]. International Journal of Emerging Technologies in Learning (iJET), 2023, 18(22): 214-227.
[10] P.Ihantola, T. Ahoniemi, V. Karavirta, et al. "Review of Recent Systems for Automatic Assessment of Programming Assignments," in Proc. 10th Koli Calling Int. Conf. Comput. Educ. Res., Koli, Finland, 2010, pp.86-93.