DOI: https://doi.org/10.59294/HIUJS2026081

An automated assessment system for data structures and algorithms with diverse question types and knowledge-driven evaluation

Hoang Ngoc Long*, Do Van Nhon
HongBang International University

ABSTRACT

This paper proposes an automated assessment system for Data Structures and Algorithms (DSA) that supports eight diverse question types-Multiple-Choice Questions (MCQ), Fill-in-the-Blank Number (FN), Fill-in-the-Blank Number Sequence (FNS), Fill-in-the-Blank Programming Syntax (FPS), Fill-in-the-Blank Short String (FSS), Fill-in-the-Blank Expression/Formula (FE), Matching Pairs (MP), and Programming Exercises (PE)-across seven difficulty levels (Easy to Expert). Leveraging Bloom's Taxonomy and psychometric parameters (difficulty: p, discrimination: d), the system uses knowledge matrices to integrate Course Learning Outcomes (CLOs), six content chapters, and cognitive levels, enabling adaptive test generation for quizzes, midterms, and exams. Automated scoring employs exact matching for MCQ/MP/FN/FNS, pattern matching for FPS/FSS, symbolic evaluation for FE, and test-case evaluation for PE. The system tracks knowledge progression across tests, per topic, and per CLO, providing comprehensive feedback. Initial implementations have shown the system's ability to improve assessment quality, decrease grading efforts, and facilitate data-driven teaching approaches in technical fields such as DSA.

Keywords: *automated assessment, data structures and algorithms, e-learning, intelligent tutoring system, knowledge representation*

1. INTRODUCTION

Intelligent Tutoring Systems (ITS) leverage AI for personalized instruction and assessment, integrating domain knowledge with student activities. In DSA, ITS aid mastery of complex concepts like algorithmic design [1, 2], but comprehensive DSA assessment systems are limited. Essential to ITS, assessment modules evaluate proficiency and track progress, helping students identify strengths and weaknesses. While ITS excel in STEM [1, 2, 3], DSA-specific frameworks lack support for diverse cognitive and practical skills. Bloom's Taxonomy organizes assessments across cognitive levels, with studies [4] showing its value in balancing theory and practice. However, existing systems often use basic formats like Multiple-Choice Questions (MCQs) [4, 5] and manual grading, limiting scalability. Research in [5] proposes an intelligent system for course content access and MCQ testing but lacks testing theories for question design. Commercial tools like the International English Test [6], VioEdu [7] support general education but not IT disciplines like DSA. Learning Management Systems (LMS) like Elearning [8] enable testing but lack deep performance analysis. Recent advancements have introduced automated test generation and scoring, yet many systems remain constrained by rigid question formats and insufficient alignment with CLOs. The study [9] has offered new insights on how an automated assessment can contribute to both curricula and pedagogical practice. The curriculum in this study corresponds to the use of formative exercises in a C++ programming course and pedagogic representation shows meaningful integration of an e-assessment tool to support learners and develop their knowledge competencies. Automated platforms like Moodle [8], CodeRunner [10], and LeetCode [11] have advanced assessment practices. Moodle [8] supports MCQs and short-answer questions, CodeRunner [10] excels in programming exercises (PE), and LeetCode [11] focuses on coding challenges. PrairieLearn, an open-source platform for STEM education, supports custom question variants with randomization and mastery-based adaptivity, allowing student retries until proficiency, and auto-grades code exercises via test cases as well as text-based explanations [12]. Yet, it relies on custom implementations for formats like fill-in-the-blank or matching pairs, with manual Bloom alignment, no integrated CLO tracking, and limited flexible difficulty for DSA assessments. However, these systems lack support for diverse

Corresponding author: Hoang Ngoc Long
Email: longhn@hiu.vn

question types (e.g., Fill-in-the-Blank Number Sequences (FNS) or Programming Syntax (FPS)) and flexible difficulty levels, hindering comprehensive evaluation. Moreover, alignment with CLOs and tracking knowledge progression by topic or across tests remain underdeveloped, resulting in no complete solution for assessing Information Technology subjects within ITS frameworks. To highlight the novelty, Table 1 compares our system with existing platforms on key aspects.

Table 1. Comparison with Existing Assessment Platforms

System	Question Types	Adaptivity	Auto PE	Bloom Alignment
Ours	8 types (MCQ–PE)	Matrix (7 levels)	Yes	Explicit
Moodle	MCQ, short answer	Limited	Partial	Manual
CodeRunner	Code exercises	Limited	Yes	Manual
PrairieLearn	Custom, MCQ, code	Mastery-based	Yes	Manual
LeetCode	Coding challenges	Difficulty rec.	Yes	Manual

Assessment systems should integrate diverse question types, support multiple cognitive levels, and offer automated scoring to reduce instructor workload. This study addresses these gaps by proposing an assessment framework with seven difficulty levels, eight question types, and knowledge matrices integrating CLOs, six content chapters, Bloom's Taxonomy, and psychometric parameters (difficulty: p, discrimination: d). The framework supports automated test generation, scoring, and evaluation per test, across tests, by topic, and per CLO, offering a scalable solution for DSA education. The remainder of this paper is structured as follows: Section 2 presents the knowledge representation model for assessment; Section 3 outlines the core assessment problems and details the algorithms for matrix generation, test creation, scoring, and progression tracking; Section 4 covers implementation, architecture, and evaluation; and Section 5 concludes with future directions.

2. KNOWLEDGE REPRESENTATION MODEL FOR ASSESSMENT

2.1. Knowledge base model for data structures and algorithms

A. Modeling the knowledge base for the content of DSA course

Definition 2.1: The knowledge organization model for representing the course content of DSA is defined as: *(T, Q, R, CLO)*. In which:

- *T (Topics)* is the set of topics in the course, organized in a hierarchical tree structure based on content relationships. It includes six chapters: (C1) Overview, (C2) Searching & Sorting, (C3) Linked Lists, (C4) Stacks & Queues, (C5) Binary Search Trees, and (C6) Hash Tables).
- *Q (Questions)* is the set of diverse questions in the course, expanded from objective multiple-choice questions to eight main types: Multiple-Choice Questions (MCQ), Fill-in-the-Blank Number (FN), Fill-in-the-Blank Number Sequence (FNS), Fill-in-the-Blank Programming Syntax (FPS), Fill-in-the-Blank Short String (FSS), Fill-in-the-Blank Expression/Formula (FE), Matching Pairs (MP), and Programming Exercises (PE). For each question $q \in Q$, the structure is defined in detail as: *(question_type, question_content, answers, correct_answers, difficult_level, discrimination_level, bloom_level, tolerance_range, partial_credit, synonyms, syntax_variants, format_variants, test_cases),* where, *question_type* is the type of question, determining its format and scoring method, including MCQ, FN, FNS, FPS, FSS, FE, MP, and PE; *question_content* is the content of the question; *answers* is the set of possible answers; *correct_answers* is the set of correct answers; *difficult_level* is the difficulty level of the question; *discrimination_level* is the discri-mination level, reflecting the question's ability to differentiate students by ability; initial values for *difficult_level (p)* and *discri-mination_level (d)* are expert-estimated by IT lecturers; these parameters are refined post-test using student data: p = (number of correct responses / total attempts), d = (proportion correct in upper 27% - proportion correct in lower 27%) per classical test theory [4]; questions with low d (< 0.2) or extreme p (<= 0.1 or >= 0.8) are flagged for refinement or removal; *bloom_level* represents the cognitive level of the question according to the Bloom's Taxonomy scale of the course's learning outcomes; *tolerance_range* represents acceptable error range for numerical (FN) or formula-based (FE) questions to handle minor calculation errors; *partial_credit* is the mechanism for awarding partial points if the answer is partially correct; *syntax_variants* is the set of valid syntax variations for FPS questions; *synonyms* is the set of synonymous terms or phrases for FSS

questions; *format_variants* is the set of valid formats for FE questions; *test_cases* is the set of test cases for PE questions. This extended structure supports flexible automated scoring, with mechanisms tailored to each question type: exact matching for MCQ, FN, FNS, and MP; pattern matching for FPS, FSS; symbolic evaluation for FE; and test-case evaluation for PE, enhanced by attributes like *tolerance_range*, *partial_credit*.

- *R (Relationships)* is the set of relationships between the course topics (T) and questions (Q), represented through knowledge matrices. These relationships map each question to one or more chapters, CLOs. Formally, *R = {(question_id, topic, clo)}*. In which, *question_id* is the identifier of a question from Q; *topic* is the topic from *T* associated with the question; *clo* is the course learning outcome from associated with the question. A *question_id* can have multiple tuples in *R*, each with a unique *(topic, clo)*.
- *CLOs* includes five objectives: understand algorithm concepts, complexity, and representation *(CLO1, Chapter 1)*; analyze search problems, constraints, and solutions *(CLO2, Chapter 2)*; analyze sorting problems, constraints, and solutions *(CLO3, Chapter 2)*; master basic data structures for programming *(CLO4, Chapters 3-6)*; implement DSA for simple problems *(CLO5, Chapters 3-6)*.

This knowledge base model has been applied to organize a bank of questions for the DSA course. This question bank is used to generate a test suitable for the requirements and abilities of students, and the results of this test serve as the basis for diagnosing students' knowledge levels.

B. Modeling user of the system

Definition 2.2: The user model in the system is designed to represent and manage individual student profiles and their interactions with the assessment system, enabling personalized test generation, scoring, and progress tracking. The model is defined as: *(PROFILE, TEST_LIST)*; where:

- *PROFILE* contains the personal and academic information of each student.
- *TEST_LIST* is the set of tests that a student has attempted, with each *test* ∈ *TEST_LIST* structured as: *(test_id, question_scores, total_score)*. In which, *test_id*: a unique identifier linking the test to the test definition *(L, question_sets, maximum_scores, time, score)* from the assessment framework, where *L* represents the difficulty level, *question_sets* categorizes questions by Bloom's levels *(Remembering, Understanding, Applying, Analyzing)*, and *maximum_scores* defines the maximum points for each question in the specific test; *question_scores*: a set of pairs *(question_id, achieved_score)*, where *question_id* is the identifier of a question from the question bank *Q*, associated with the extended structure; *achieved_score* is the score earned by the student for the corresponding question, calculated based on the *maximum_score* defined in the test and the student's response; *total_score* represents cumulative score of the test, computed as the sum of all *achieved_scores* in *question_scores*.

2.2. Question bank model and knowledge linkages

The question bank model *(Q)*, a core component of the DSA assessment system, serves as a dynamic repository for tailored test creation and automated evaluation within the knowledge organization model *(T, Q, R, CLO)*. It includes eight question types-MCQ, FN, FNS, FPS, FSS, FE, MP, PE-spanning six chapters (C1-C6). Each question is structured with attributes as defined in Definition 2.1 to support scoring mechanisms (e.g., test-case evaluation for PE). The design of the question bank should support operations like removing or revising low-quality questions and incorporating high-quality ones to ensure continuous improvement in both the number and quality of questions [4]. Knowledge linkages via the relationship set *R* connect *Q* to topics *T* and *CLOs* through knowledge matrices, incorporating *bloom_level*, *difficult_level*, and *question_type*. These matrices, generated per test difficulty, guide question selection. Table 2 shows a Level 3 (Medium) mid-term matrix, distributing 30 questions across Bloom's levels and types. This two-dimensional matrix has question types as columns and Bloom's Taxonomy levels as rows, with each cell containing an array of tuples. Each tuple pairs a chapter (topic) with the number of questions allocated to that chapter.

Table 2. Knowledge Matrix for a Mid-Term Exam at Level 3 - Medium

Bloom Levels	Question Types								Total
	MCQ	FN	FNS	FPS	FSS	FE	MP	PE	
R	-	(C2, 3), (C3, 3), (C4, 3)	-	-	-	-	-	-	9
U	-	(C2, 4)	(C3, 4)	(C4, 4)	-	-	-	-	12
AP	-	(C2, 2)	(C3, 2)	(C4, 2)	-	-	-	-	6

Bloom Levels	Question Types								Total
	MCQ	FN	FNS	FPS	FSS	FE	MP	PE	
AN	-	-	-	(C3, 1)	-	-	-	(C2, 1), (C4, 1)	3
Total	-	15	6	7	-	-	-	2	**30**

A test is typically created by choosing questions from a question bank, which is a comprehensive collection of numerous questions. It has structure and defined in Definition 2.3.

Definition 2.3: The test structure in the system defines the organization and composition of assessments used to evaluate student performance in the DSA course. It is designed to ensure a balanced evaluation across topics, learning outcomes, and cognitive levels, while supporting automated scoring and personalized feedback. The test structure is formally defined as:

(L, question_sets, maximum_scores, time, score); where:

- *L (Test Difficulty Level)* represents the overall difficulty of the test, categorized into seven levels ranging from Easy to Expert. The difficulty level *L* guides the selection of questions and influences the assignment of *maximum_scores*, ensuring alignment with the target student population's ability.
- *question_sets* is the collection of questions including *(Q_R, Q_U, Q_{AP}, Q_{AN}, ordered_questions)*. In which, Q_R, Q_U, Q_{AP}, Q_{AN} are the set of questions at the *Remembering (R), Understanding (U), Applying (A), Analyzing (A)* Bloom's levels correspondingly; *ordered_questions* is the collection of questions in particular order of *[Non_PE_Questions, PE_Questions]*.
- *maximum_scores* is a set of pairs *(question_id, maximum_score)*, where *question_id* identifies a question in *question_sets*, and *maximum_score* is the maximum points assigned to that question in the specific test. The *maximum_score* can be assigned by the user or is determined based on the question's complexity, Bloom's level, and the test's difficulty level *(L)*, allowing for flexibility across different tests.
- *time* is the duration of the test, measured in minutes; *score* is the desired total score for the test in points. The parameters *time* and *score* will be given by user.

The test structure integrates with the knowledge organization model *(T, Q, R, CLO)* by using the knowledge matrix to select questions from Q that align with the topics (T) and CLOs targeted by the test. The distribution of questions across Q_R, Q_U, Q_{AP}, and Q_{AN} is determined by the difficulty test level (L) as in Table 3. For instance, a Level 3 (Medium) test might have 30% R, 40% U, 20% AP, and 10% AN, as seen in prior examples.

Table 3. Difficult level of the test

Difficulty of Test Level (L)	Bloom Proportions (R/U/AP/AN)	Difficult level (p)	Discrimination level (d)	Question Types	Course Content
Level 1 (Easy)	50 / 30 / 10 / 10	$p \geq 0.7$	$0.2 \leq d \leq 1$	MCQ, FSS, MP	C1, C2
Level 2 (Moderate)	40 / 30 / 20 / 10	$0.6 \leq p < 0.7$	$0.2 \leq d \leq 1$	MCQ, FN, MP	C1, C2
Level 3 (Medium)	30 / 40 / 20 / 10	$0.4 \leq p < 0.6$	$0.2 \leq d \leq 1$	FN, FPS, FNS, PE	C2–C4
Level 4 (Difficult)	20 / 30 / 30 / 20	$0.3 \leq p < 0.4$	$0.2 \leq d \leq 1$	FN, FPS, PE, FE	C2–C5
Level 5 (Very Diff.)	10 / 20 / 30 / 40	$p < 0.3$	$0.2 \leq d \leq 1$	FN, PE, MP, FE	C2–C5
Level 6 (Advance)	5 / 15 / 30 / 50	$p < 0.25$	$0.3 \leq d \leq 1$	FN, PE, FPS, MP	C2–C5
Level 7 (Expert)	0 / 10 / 30 / 60	$p < 0.2$	$0.4 \leq d \leq 1$	FN, PE, FPS, FE	C1–C6

When a student completes a test, their performance is recorded in the *TEST_LIST* of the user model as *(test_id, question_scores, total_score)*, where *question_scores* contains pairs *(question_id, achieved_score)*. The *achieved_score* is calculated using the question's scoring mechanism and adjusted by attributes like *tolerance_range* or *partial_credit*. This structure supports the system's goals of generating adaptive tests, providing detailed feedback, and tracking student progress across CLOs, with the flexibility to adjust difficulty and scoring based on educational needs.

3. ASSESSMENT PROBLEMS AND ALGORITHM DESIGN

This section outlines five core challenges in developing an automated assessment system for DSA, using the knowledge model *(T, Q, R, CLO)*, question bank *Q*, and test structure *(L, question_sets, maximum_scores, time, score)* to enable adaptive testing and evaluation:

1) Knowledge Matrix Generation: building a matrix balancing questions across topics *(T)*, *CLOs*, Bloom's levels *(R, U, AP, AN)*, and types *(MCQ, FN, FNS, FPS, FSS, FE, MP, PE)* per test's difficulty level *(L)*.

2) Test Generation: creating tests meeting difficulty *(L)*, chapter coverage, and Bloom's distribution, aligned with *CLOs* and psychometric parameters.

3) Evaluation and Scoring: designing scoring for eight

question types, considering attributes (e.g., tolerance_range for FN, test_cases for PE), for accurate assessment.

4) Knowledge Assessment per Test: analyzing performance on single tests to assess topic and CLO mastery, providing immediate feedback.

5) Knowledge Progression Across Tests: tracking progress over multiple tests to identify trends and personalize learning.

3.1. Knowledge matrix generation problem and algorithm

Definition 3.1: The knowledge matrix generation problem creates a balanced question distribution across Bloom's levels and question types, modeled as: *(L, Q, T, N)* → *KM*, where *L* is the test difficulty level (Table 3), *Q* is the question bank, *T* is the set of topics, *N* is the number of questions, and *KM* is a 2D knowledge matrix.

Algorithm 1 generates the knowledge matrix based on the proportions defined in Table 3. When the number of available questions is insufficient or cell counts do not match, the system prompts the instructor via the User Interface to adjust the matrix. Instructors can modify question counts per *(bloom_level, question_type, chapter)* while ensuring the total matches *N*. This semi-automated approach preserves the required Bloom proportions while allowing pedagogical adjustments.

```
Algorithm 1: GENERATE_KNOWLEDGE_MATRIX
Input: L, Q, T, N
Output: KM
Procedure:
1. EXTRACT Bloom_Proportions, Covered_Question_Types, Covered_Chapters FROM L
2. INITIALIZE KM AS empty matrix [Bloom_Levels][Covered_Question_Types]
     WHERE Bloom_Levels = {R, U, AP, AN}
3. COMPUTE question counts per Bloom level:
     N_R ← FLOOR(N * Bloom_Proportions[R])
     N_U ← FLOOR(N * Bloom_Proportions[U])
     N_AP ← FLOOR(N * Bloom_Proportions[AP])
     N_AN ← FLOOR(N * Bloom_Proportions[AN])
     Total_Allocated ← N_R + N_U + N_AP + N_AN
   IF Total_Allocated ≠ N THEN
     PROMPT user to adjust N
     RECALCULATE N_R, N_U, N_AP, N_AN
   ENDIF
4. FOR EACH Bloom_Level IN Bloom_Levels DO
     N_Bloom ← SELECT {N_R, N_U, N_AP, N_AN} FOR Bloom_Level
     FOR EACH Question_Type IN Covered_Question_Types DO
       Cell_Total ← 0
       WHILE Cell_Total < N_Bloom DO
         Random_Chapter ← RANDOM(Covered_Chapters)
         Default_Count ← RANDOM_INTEGER(0, N_Bloom - Cell_Total)
         APPEND (Random_Chapter, Default_Count) TO
                    KM[Bloom_Level][Question_Type]
         Cell_Total ← Cell_Total + Default_Count
       ENDWHILE
       IF Cell_Total > 0 THEN
         VALIDATE KM[Bloom_Level][Question_Type]
       ENDIF
       PROMPT user to adjust KM[Bloom_Level][Question_Type]
     ENDFOR
     Bloom_Total ← SUM(count FOR (chapter, count) IN KM[Bloom_Level])
     IF Bloom_Total ≠ N_Bloom THEN
       PROMPT user to adjust KM[Bloom_Level]
     ENDIF
   ENDFOR
5. Total_Questions ← SUM(count FOR row IN KM FOR cell IN row
                          FOR (chapter, count) IN cell)
   IF Total_Questions ≠ N THEN
     PROMPT user to adjust KM
   ENDIF
6. RETURN KM
```

Figure 1. The algorithm of knowledge matrix generation for DSA assessment

3.2. Test generation problem and algorithm

Definition 3.2: The test generation problem is modeled as: *(knowledge_matrix, Q, time, score)* → *(L, question_sets, maximum_scores, time, score)*, where the knowledge matrix is obtained from Algorithm 1, and the output follows the test structure defined in Definition 2.3.

Algorithm 2 generates a test by selecting questions from the question bank according to the knowledge matrix. Questions are filtered based on Bloom's level, question type, chapter, difficulty *p*, and discrimination *d* to ensure balance across cognitive levels, topics, and psychometric parameters. In this algorithm, the *base_score* values are justified by teaching experience in DSA courses, scaled by *difficulty_factor* and *bloom_factor*. These weighting

parameters are currently fixed but can be made configurable in future versions.

Algorithm 2: GENERATE_TEST

Input:*knowledge_matrix* (*KM*), *Q, time, score*
Output: *L, question_sets, maximum_scores, time, score*

Procedure:

```
1. EXTRACT FROM KM: L ← KM.Difficulty_Level, Covered_Question_Types ← KM.Covered_Question_Types,
   Covered_Chapters ← KM.Covered_Chapters, Bloom_Proportions ← KM.Bloom_Proportions
2. INITIALIZE question_sets AS {QR ← [], QU ← [], QAP ← [], QAN ← [],
                    ordered_questions ← []}
3. FOR EACH Bloom_Level IN {R, U, AP, AN} DO
    Current_Set ← SELECT {QR, QU, QAP, QAN} FOR Bloom_Level
    FOR EACH Question_Type IN Covered_Question_Types DO
      FOR EACH (Chapter, Count) IN KM[Bloom_Level][Question_Type] DO
        Q_Filtered ← {q ∈ Q | q.bloom_level = Bloom_Level AND
                     q.question_type = Question_Type AND
                     q.chapter = Chapter AND
                     q.difficulty_level MATCHES L AND
                     q.discrimination_level MATCHES L}
        IF SIZE(Q_Filtered) < Count THEN
          PROMPT user to adjust Count
          UPDATE Count
        ENDIF
        Selected_Questions ← RANDOM_SUBSET(Q_Filtered, Count)
        APPEND Selected_Questions TO Current_Set
      ENDFOR
  ENDFOR
4. All_Questions ← CONCATENATE (QR, QU, QAP, QAN)
  PE_Questions ← {q ∈ All_Questions | q.question_type = PE}
  Non_PE_Questions ← SHUFFLE(All_Questions \ PE_Questions)
  question_sets.ordered_questions ← [Non_PE_Questions, PE_Questions]
  REDISTRIBUTE question_sets.ordered_questions TO {QR, QU, QAP, QAN} BY bloom_level
5. INITIALIZE maximum_scores AS empty set
  Total_Default_Score ← 0
  FOR EACH Question IN question_sets.ordered_questions DO
    IF Question.question_type = MCQ THEN
      base_score ← 1
    ELSE IF Question.question_type IN {FN, FNS, FPS, FSS, FE, MP} THEN
      base_score ← 3
    ELSE IF Question.question_type = PE THEN
      base_score ← 5
    ENDIF
    Difficulty_Factor ← MAP(L, {Level 1: 0.5, Level 2: 0.7, Level 3: 1.0,
                  Level 4: 1.2, Level 5: 1.5, Level 6: 1.8, Level 7: 2.0})
    Bloom_Factor ← MAP(Question.bloom_level, {R: 1.0, U: 1.2, AP: 1.5, AN: 2.0})
    Question.default_score ← base_score × Difficulty_Factor × Bloom_Factor
    Total_Default_Score ← Total_Default_Score + Question.default_score
  ENDFOR
  FOR EACH Question IN question_sets.ordered_questions DO
    Question.maximum_score ← ROUND((Question.default_score /
                    Total_Default_Score)× score)
    IF Question.maximum_score < 1 THEN
      Question.maximum_score ← 1
    ENDIF
    APPEND (Question.question_id, Question.maximum_score) TO maximum_scores
  ENDFOR
  Final_Score ← SUM(q.maximum_score FOR q IN question_sets.ordered_questions)
  IF Final_Score ≠ score THEN
    PROMPT user to adjust maximum_scores
  ENDIF
6. RETURN (L, question_sets, maximum_scores, time, score)
```

Figure 2. The algorithm of test generation for DSA assessment

3.3. Evaluation and scoring of question types

Four scoring mechanisms-exact matching, pattern recognition, symbolic evaluation, and test-case-based evaluation-are used to assess responses for different question types, aligned with Bloom's Taxonomy. These are detailed in Definition 3.3 and Algorithm 3.

- Exact matching is applied to MCQ, MP, FN, and FNS. This mechanism compares responses directly to correct answers. MCQ and MP award partial credit based on correct selections or pairs (e.g., 2/3 correct yields 2/3 score). FNS checks sequence elements with partial credit within a tolerance (e.g., ± 0.001). FN scores full points if within tolerance, else zero.
- Pattern recognition is applied to FPS and FSS. Student responses are normalized (e.g., removes spaces for FPS, lowercase for FSS) and matched

against predefined correct answers or valid variants. Full marks are awarded only if a match is found.

- Symbolic evaluation is used for FE questions. Responses are normalized and parsed into symbolic expressions. The system checks for mathematical equivalence with the correct answer using expression simplification. Equivalent expressions receive full marks.
- Test-case-based evaluation is used exclusively for PE. It combines static analysis (40%) via *cppcheck* and dynamic unit testing (60%) via *GoogleTest*. Compilation failure results in zero score. Successful compilations are evaluated for code style issues and functional correctness against test cases, with partial credit awarded for passed tests.

Definition 3.3: The evaluation and scoring problem computes the achieved score of a student's response based on the question type and its corresponding scoring mechanism. It is modeled as: *(question, student_response, maximum_score) → (achieved_score)*, where the input *question* comes from the question bank (Definition 2.1) and the *maximum_score* is defined in the test structure (Definition 2.3). The detailed process is presented in Algorithm 3.

Algorithm 3: EVALUATE_AND_SCORE

Input: *question, student_response, maximum_*score
Output: *achieved_score*

```
Procedure:
1. INITIALIZE achieved_score ← 0
2. CASE question.question_type OF
    MCQ, MP:
      correct_count ← COUNT(student_response MATCHES question.correct_answers)
      achieved_score ← (correct_count / LENGTH(question.correct_answers)) ×
                       maximum_score × question.partial_credit
    FN:
      is_valid ← PARSE(student_response) WITHIN (question.correct_answers ±
                       question.tolerance_range)
      achieved_score ← is_valid ? maximum_score : 0
    FNS:
      correct_items ← COUNT(student_response[i] MATCHES question.correct_answers[i]
                      WITHIN question.tolerance_range FOR i IN
                      RANGE(LENGTH(question.correct_answers)))
      achieved_score ← (correct_items / LENGTH(question.correct_answers)) ×
                       maximum_score × question.partial_credit
    FPS, FSS:
      normalized_response ← NORMALIZE(student_response, question.question_type)
      variants ← question.question_type = FPS ?
                 question.syntax_variants : question.synonyms
      achieved_score ← (normalized_response MATCHES question.correct_answers
                       OR ANY v IN variants) ? maximum_score : 0
    FE:
      normalized_response ← STRIP(student_response)
      IF IS_VALID_EXPRESSION(normalized_response) THEN
        response_expr ← PARSE_TO_EXPRESSION(normalized_response)
        correct_expr ← PARSE_TO_EXPRESSION(question.correct_answers)
        achieved_score ← (SIMPLIFY(response_expr - correct_expr) = 0 OR
            normalized_response IN question.syntax_variants) ? maximum_score : 0
      ELSE
        achieved_score ← 0
      ENDIF
    PE:
      issues ← COUNT_ISSUES(cppcheck, student_response)
      static_score ← MAX(0, 1 - (issues / 10)) × 0.4 × maximum_score
      passed_tests ← COUNT_PASSED_TESTS(GoogleTest, question.test_cases,
                     student_response)
      unit_score ← (passed_tests / LENGTH(question.test_cases)) × 0.6 ×
                   maximum_score × question.partial_credit
      achieved_score ← static_score + unit_score
3. RETURN achieved_score
```

Figure 3. The algorithm of evaluation and scoring for diverse question types

3.4. Knowledge assessment per test

Definition 3.4: The per-test knowledge assessment problem evaluates a student's performance on a single test and generates feedback. It is modeled as: *(test, student_profile, question_scores, student_responses, R) → (performance_metrics, feedback)*, where the inputs follow the test structure (Definition 2.3) and user model (Definition 2.2), and *R* maps questions to topics and CLOs (Definition 2.1).

Algorithm 4 aggregates scores, computes performance metrics across topics, CLOs, and Bloom's levels, and generates detailed feedback, including PE-specific insights.

```
Algorithm 4: ASSESS_PER_TEST
Input: test, student_profile, question_scores, student_responses, R
Output: performance_metrics, feedback
Procedure:
1. INITIALIZE performance_metrics ← {topic_scores ← {}, clo_scores ← {},
        bloom_scores ← {}, bloom_max_scores ← {}, total_score ← 0}, feedback ← ""
2. FOR EACH (question_id, achieved_score) IN question_scores DO
    question ← test.question_sets[question_id]
    max_score ← test.maximum_scores[question_id]
    total_score ← total_score + achieved_score
    response ← student_responses[question_id]
    status ← achieved_score = max_score ? "Correct" : "Incorrect"
    APPEND TO feedback: "Question {question_id}: {status}
    ({achieved_score}/{max_score})\nCorrect Answer: {question.correct_answers}\n"
    IF status = "Incorrect" THEN
      IF question.question_type = PE THEN
        issues ← COUNT_ISSUES(cppcheck, response)
        failed ← COUNT_FAILED_TESTS(GoogleTest, question.test_cases, response)
        APPEND TO feedback: "Static Issues: {issues}\nFailed Tests:
                    {failed}/{LENGTH(question.test_cases)}\n"
      ELSE
        APPEND TO feedback: "Your Answer: {response}\n"
      ENDIF
    ENDIF
    relations ← {(topic, clo) | (q_id, topic, clo) IN R WHERE q_id = question_id}
    FOR EACH (topic, clo) IN relations DO
      normalized_score ← (achieved_score / max_score) / LENGTH(relations)
      topic_scores[topic] ← (topic_scores[topic, 0] + normalized_score) /
                    (COUNT(question_scores[topic, []]) + 1)
      clo_scores[clo] ← (clo_scores[clo, 0] + normalized_score) /
                    (COUNT(question_scores[clo, []]) + 1)
    ENDFOR
    bloom_level ← question.bloom_level
    bloom_scores[bloom_level] ← bloom_scores[bloom_level, 0] + achieved_score
    bloom_max_scores[bloom_level] ← bloom_max_scores[bloom_level, 0] + max_score
  ENDFOR
  FOR EACH bloom_level IN bloom_scores DO
    IF bloom_max_scores[bloom_level] > 0 THEN
      bloom_scores[bloom_level] ← bloom_scores[bloom_level] /
                    bloom_max_scores[bloom_level]
    ENDIF
  ENDFOR
3. PREPEND TO feedback: "Student: {student_profile.name}, Score:
            {performance_metrics.total_score}/{test.score}\n"
  FOR EACH topic IN topic_scores DO
    Pt ← topic_scores[topic]
    strength ← Pt > 0.75 ? "Strong" : Pt ≥ 0.5 ? "Moderate" : "Weak"
    advice ← Pt ≥ 0.5 AND Pt ≤ 0.75 ? " review concepts" : Pt < 0.5 ? "
                    focus on fundamentals" : ""
    APPEND TO feedback:"Topic {topic} ({Pt:.2f}): {strength} at
                    {test.L} level{advice}\n"
  ENDFOR
  FOR EACH clo IN clo_scores DO
    Pclo ← clo_scores[clo]
    status ← Pclo ≥ 0.5 ? "Meets" : "Below"
    advice ← Pclo < 0.5 ? " focus on related chapters" : ""
    APPEND TO feedback:"CLO{clo} ({Pclo:.2f}):{status} at {test.L} level{advice}\n"
  ENDFOR
  IF ANY(question.question_type = PE IN test.question_sets) THEN
    APPEND TO feedback: "Programming Recommendations:\n"
    IF issues > 0 THEN APPEND TO feedback: "- Review coding style\n"
    IF failed > 0 THEN APPEND TO feedback: "- Debug failed test cases\n"
  ENDIF
4. RETURN (performance_metrics, feedback)
```

Figure 4. The algorithm of Per-test knowledge assessment and feedback

3.5. Knowledge progression across tests

Definition 3.5: The knowledge progression across tests problem evaluates a student's performance trends over multiple tests and provides feedback on their learning trajectory. It is modeled as: *(student_profile, test_list, R) → (progression_metrics, progression_feedback)*, where the inputs follow the user model (Definition 2.2) and *R* maps questions to topics and CLOs (Definition 2.1).

Algorithm 5 aggregates results from Algorithm 4 across tests, computes average performance, classifies levels, and identifies trends using a slope-based method to generate actionable recommendations.

```
Algorithm 5: ASSESS_ACROSS_TESTS
Input: student_profile, test_list, R
Output: progression_metrics, progression_feedback
Procedure:
1. INITIALIZE progression_metrics ← {topic_progression ← {},clo_progression ← {},
        bloom_progression ← {},overall_progression ← []},progression_feedback ← ""
2. FOR EACH test_entry IN SORT(test_list BY timestamp) DO
    metrics ← ASSESS_PER_TEST(test_entry.test, student_profile,
          test_entry.question_scores, test_entry.student_responses, R)
    APPEND (metrics.total_score / test_entry.test.score) TO
                    progression_metrics.overall_progression
    FOR EACH (dim, score) IN metrics.topic_scores DO
      APPEND score TO topic_progression[dim, []]
    ENDFOR
    FOR EACH (dim, score) IN metrics.clo_scores DO
      APPEND score TO clo_progression[dim, []]
    ENDFOR
    FOR EACH (dim, score) IN metrics.bloom_scores DO
      APPEND score TO bloom_progression[dim, []]
    ENDFOR
  ENDFOR
3. FOR EACH (key, data_name) IN [(topic_progression, "Topic"), (clo_progression,
                    "CLO"), (bloom_progression, "Bloom")] DO
    FOR EACH (dim, s_list) IN key DO
      mu ← MEAN(s_list)
      level ← mu < 0.3 ? "Failed" : mu < 0.5 ? "Weak" : mu < 0.7 ?
              "Average" : mu < 0.9 ? "Good" : "Excellent"
      slope ← LINEAR_SLOPE(s_list) IF LENGTH(s_list) > 1 ELSE 0
      trend ← slope > 0 ? "Improving" : slope < 0 ? "Declining" : "Stable"
      key[dim] ← (s_list, level, trend)
      APPEND TO progression_feedback: "{data_name} {dim}: {level}
                    ({[s:.2f FOR s IN s_list]}, {trend})\n"
    ENDFOR
  ENDFOR
4. mu ← MEAN(progression_metrics.overall_progression)
  level ← mu < 0.3 ? "Failed" : mu < 0.5 ? "Weak" : mu < 0.7 ?
          "Average" : mu < 0.9 ? "Good" : "Excellent"
  slope ← LINEAR_SLOPE(progression_metrics.overall_progression) IF
          LENGTH(progression_metrics.overall_progression) > 1 ELSE 0
  trend ← slope > 1 ? "Improving" : slope < -1 ? "Declining" : "Stable"
  progression_metrics.overall_progression ← (progression_metrics.overall_progression,
                    level, trend)
  PREPEND TO progression_feedback: "Overall: {level} ({[s:.2f FOR s IN
        progression_metrics.overall_progression[0]]}, {trend})\n"
5. IF ANY(l IN ["Failed", "Weak"] FOR _, (_, l, _) IN [topic_progression,
        clo_progression]) THEN
    APPEND TO progression_feedback: "Recommendation: Focus on topics/CLOs
                    with Failed or Weak performance.\n"
    IF progression_metrics.overall_progression[1] IN ["Failed", "Weak"] THEN
      APPEND TO progression_feedback: "Recommendation: Review foundational
                    concepts across all areas.\n"
    ENDIF
  ENDIF
6. RETURN (progression_metrics, progression_feedback)
```

Figure 5. The algorithm of progression tracking across multiple tests

4. IMPLEMENTATION AND EVALUATION

4.1. System architecture and knowledge base

The architecture of the knowledge assessment system for DSA, as illustrated in Figure 6, supports two user types: students and knowledge engineers. Students engage in testing and progress tracking, while knowledge engineers manage the knowledge base via the User Interface (1). The system comprises a User Interface (1) and seven other interconnected modules: (2) Request Classification routes user requests; (3) Knowledge Management handles insert, update, and delete operations; (4) Knowledge Evaluation assesses general performance; (5) PE Evaluation processes Programming Exercises using *cppcheck*, and unit test cases; (6) Test Generation creates adaptive tests via Algorithm 2; (7) Knowledge Base stores data in an SQL database; and (8) Explanation generates detailed feedback.

The Knowledge Base (7) comprises 1250 questions spanning six chapters, categorized into eight question types and four cognitive levels based on Bloom's Taxonomy. These questions are systematically mapped to specific topics and CLOs, as defined in Definition 2.1 and detailed in Table 4. Data flows from the User Interface (1) through Request

Classification (2) to specialized modules (3-6), with PE Evaluation (5) validating accuracy for coding tasks. Explanation (8) integrates results, ensuring comprehensive DSA assessment and feedback.

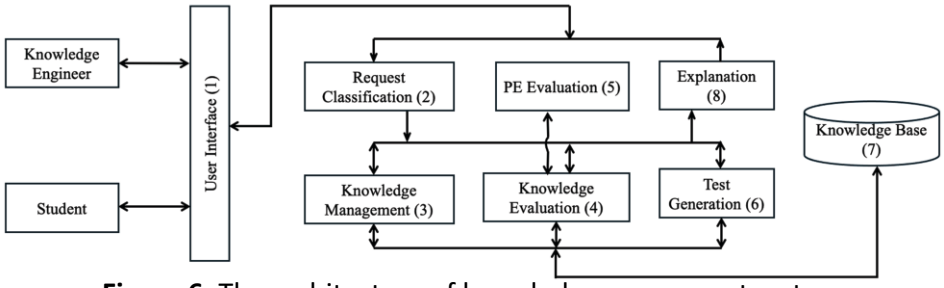

Figure 6. The architecture of knowledge assessment system

Table 4. Question Bank Distribution by Chapter and Bloom's Level

Chapter	Remembering (R)	Understanding (U)	Applying (AP)	Analyzing (AN)	Total
C1: Overview	60	55	45	45	205
C2: Searching & Sorting	80	75	65	60	280
C3: Linked Lists	60	55	45	40	200
C4: Stacks & Queues	55	45	45	45	190
C5: Tree	45	45	55	55	200
C6: Hash Table	45	45	45	40	175
Total	345	320	300	285	**1250**

4.2. Implementation and testing

The system was implemented using Spring Boot for backend logic and React for the frontend interface, with a MySQL database. It provides two main views: (1) a post-test feedback page showing individual question results, PE-specific analysis, and performance summaries aligned with CLOs and Bloom's levels; and (2) an overall progress dashboard displaying trend graphs, performance levels, and personalized recommendations. Pilot testing was conducted with 20 IT students at Hong Bang International University using the 1250 -question knowledge base. Three test types were evaluated: quizzes (10 questions, Level 1 - 2), midterms (30 questions, Level 3 - 4), and finals (50 questions, Level 5). Algorithm 3 achieved 90% scoring accuracy. Progression tracking (Algorithms 4 - 5) revealed that only 10% of students showed positive learning trends, indicating a need for stronger teacher support and reevaluation of question difficulty. User feedback highlighted the interface clarity but suggested optimizing PE evaluation speed.

4.3. Discussion of results

Pilot testing with 20 IT students demonstrated the system's practical value. The architecture-with eight modules including PE Evaluation (5) using *GoogleTest*, *cppcheck*-supports diverse question types and difficulty levels, leveraging classical test theory for adaptive test generation aligned with CLOs, utilizing a 1250-question knowledge base (Table 3). Algorithm 3 achieved 90% scoring accuracy, and the system effectively tracks knowledge across topics and CLOs. The post-test feedback page and progress dashboard (Figure 7–8) were well-received for providing clear, personalized feedback. It automatically assesses knowledge per test or across tests by topic and CLOs, using data mapping (e.g., *question_type*, *bloom_level*) for objective selection, with stored results enabling progress tracking and recommendations. However, only 10% of students showed positive learning trends, highlighting engagement challenges. The small sample size and single-institution context also limit the generalizability of the results. Future work should focus on optimizing PE evaluation performance and conducting larger-scale trials with statistical validation to strengthen the findings.


Kết Quả Kiểm Tra Giữa Kì - Lê Hiếu Nghiêm

Thời Gian: 60 phút (Hoàn thành lúc 2:45 PM, Thứ Ba, 15/04/2025)

Thông Tin Chung

Điểm Số: 63.95/100 (Dưới Trung Bình)

Phân Tích Theo Chủ Đề

Chủ Đề	Điểm Số	Số Câu Đúng/Tổng	Phần Trăm	Nhận Xét
Tìm Kiếm và Sắp Xếp (C2)	39.8/54	10/15	73.7%	Tốt
Danh Sách Liên Kết (C3)	12.75/22	4/7	58.0%	Trung Bình
Ngăn Xếp và Hàng Đợi (C4)	15/24	4/8	62.5%	Trung Bình

Phân Tích Theo Chuẩn Đầu Ra (CLO)

CLO	Mô Tả	Điểm Số	Số Câu Đúng/Tổng	Phần Trăm	Nhận Xét
CLO1	Nắm được khái niệm, vai trò của cấu trúc dữ liệu và giải thuật trong chương trình, các tiêu chuẩn đánh giá cấu trúc dữ liệu, các tiêu chuẩn đánh giá giải thuật. Khái niệm về độ phức tạp thuật giải, kỹ thuật tính toán cơ bản, biểu diễn thuật giải.	39.8/54	10/15	73.7%	Đạt
CLO2	Xác định và phát biểu bài toán tìm kiếm, các yếu tố có liên quan, các yếu tố ràng buộc trong bài toán, mô hình hóa bài toán, xác định các phương pháp giải quyết bài toán, phân tích các ưu điểm và hạn chế của các phương pháp giải quyết bài toán.	20/29	5/7	69.0%	Đạt
CLO3	Xác định và phát biểu bài toán sắp xếp, các yếu tố có liên quan, các yếu tố ràng buộc trong bài toán, mô hình hóa bài toán, xác định các phương pháp giải quyết bài toán, phân tích các ưu điểm và hạn chế của các phương pháp giải quyết bài toán.	19.8/25	5/8	79.2%	Đạt
CLO4	Nắm vững các cấu trúc dữ liệu cơ bản và có khả năng ứng dụng các cấu trúc dữ liệu cơ bản để viết các chương trình ứng dụng đơn giản.	22.75/36	6/11	63.2%	Đạt
CLO5	Nắm vững cách thức cài đặt các cấu trúc dữ liệu, thuật toán và có thể áp dụng để giải quyết các bài toán đơn giản.	6.2/20	2/6	31.0%	Không Đạt

Nhận Xét Chung

Hiệu suất tổng thể của bạn là 63.95/100 (Dưới Trung Bình). Bạn mạnh ở Tìm Kiếm và Sắp Xếp (Tốt), nhưng cần cải thiện ở Danh Sách Liên Kết và Ngăn Xếp/Hàng Đợi (Trung Bình). Kỹ năng Áp Dụng (42.9%) cần chú trọng; Nhớ và Hiểu cũng cần củng cố. Hãy tập trung vào CLO5 (Không Đạt) và thực hành thêm các bài tập cài đặt. Quản lý thời gian trong 60 phút khá tốt.

Figure 7. A part of test tesult user interface (summary of the test, topic and CLOs alignment, and recommendations)

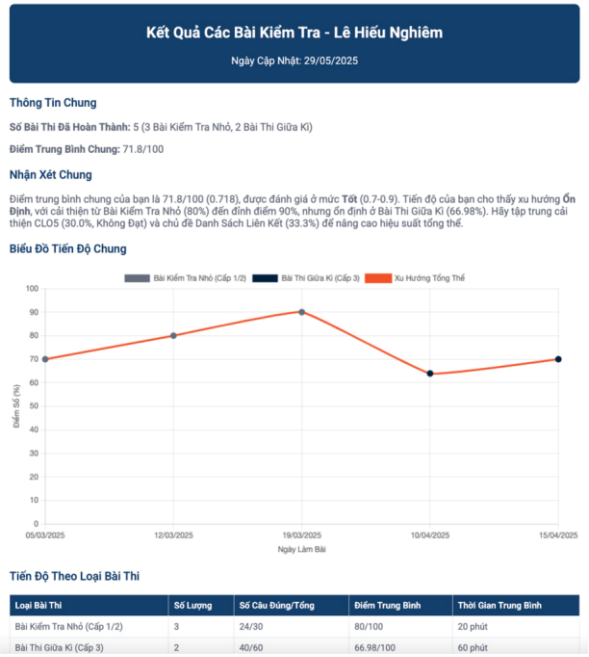

Figure 8. Overall progress dashboard (summary of tests, trend graph of scores accross tests, performance levels, and recommendations)

5. CONCLUSION

This study presents an automated assessment system for the DSA course that supports eight diverse question types and integrates knowledge matrices aligned with CLOs, Bloom's Taxonomy, and psychometric parameters. The system enables adaptive test generation, automated scoring through exact matching, pattern recognition, symbolic evaluation, and test-case evaluation, while providing detailed feedback and tracking student progress across topics and CLOs. Pilot deployments demonstrated its effectiveness in improving assessment quality, reducing grading workload, and supporting data-driven teaching. The architecture is scalable and suitable for larger classes or broader curricular integration. Future work will focus on two main directions: (1) developing intelligent question recommendation mechanisms using reinforcement learning or Hidden Markov Models, and (2) extending the system to other foundational computing courses with enhanced natural language processing capabilities. This research contributes to the development of intelligent, scalable assessment solutions in computer science education.

ACKNOWLEDGEMENT

This work is funded by Hong Bang International University under grant code GVTC18.02.

REFERENCES

[1]C.-C. Lin, A. Y. Q. Huang and O. H. T. Lu, "Artificial intelligence in intelligent tutoring systems toward sustainable education: a systematic review," *Smart Learning Environments,* vol. 10, no. 41, 2023.

[2]M. Kurni, M. S. Mohammed and S. K. G, A Beginner's Guide to Introduce Artificial Intelligence in Teaching and Learning, Springer Cham, 2023.

[3]E. Mousavinasab, N. Zarifsanaiey, S. R. N. Kalhori, M. Rakhshan, L. Keikha and M. G. Saeedi, "Intelligent tutoring systems: A systematic review of characteristics, applications, and evaluation methods," *Interactive Learning Environments,* vol. 29, no. 1, pp. 142-163, 2021.

[4]H. N. Long, M. T. Thanh and D. V. Nhon, "Designing a knowledge assessment system for the Data structures and Algorithms course," *Hong Bang International University Journal of Science,* vol. 6, pp. 65-74, 2024.

[5]H. D. Nguyen, "Intelligent System in Education: Requirements and Design Method". Journal of Electronic Voltage and Application," *Journal of Electronic Voltage and Application,* vol. 4, no. 2, pp. 12-19, 2023.

[6]International English Test, "International English Test," [Online]. Available: https://international-englishtest.com/. [Accessed 23 May 2025].

[7]FPT, "VioEdu," FPT, [Online]. Available: https://vio.edu.vn/. [Accessed 23 May 2025].

[8]Hong Bang International University, "Elearning," Hong Bang International University, [Online]. Available: https://elearning.hiu.vn/. [Accessed 23 May 2025].

[9]A. L. C. Barczak, A. Mathrani, B. Han and N. H. Reyes, "Automated assessment system for programming courses: a case study for teaching data structures and algorithms," *Education Tech Research Dev* , no. 71, p. 2365-2388, 2023.

[10]R. Lobb and J. Harlow, "Coderunner: a tool for assessing computer programming skills," *ACM Inroads,* vol. 7, no. 1, pp. 47-51, 2016.

[11]LeetCode, "LeetCode," LeetCode, [Online]. Available: https://leetcode.com/. [Accessed 23 May 2025].[12]M. West, N. Walters, M. Silva, T. Bretl and C. Zilles, "Integrating Diverse Learning Tools using the PrairieLearn Platform," in *Proceedings of SPLICE 2021 workshop CS Education Infrastructure for All III: From Ideas to Practice*, Virtual Event, 2021.

Hệ thống đánh giá kiến thức tự động cho cấu trúc dữ liệu và giải thuật với nhiều loại câu hỏi

Hoàng Ngọc Long, Đỗ Văn Nhơn
Trường Đại học Quốc tế Hồng Bàng

TÓM TẮT

Bài báo đề xuất một hệ thống đánh giá tự động cho môn Cấu trúc Dữ liệu và Giải Thuật (DSA) hỗ trợ tám loại câu hỏi đa dạng-Hỏi Đáp Trắc Nghiệm (MCQ), Điền Số (FN), Điền Dãy Số (FNS), Điền Cú Pháp Lập Trình (FPS), Điền Chuỗi Ngắn (FSS), Điền Biểu Thức/Công Thức (FE), Ghép Đôi (MP), và Bài Tập Lập Trình (PE)-trên bảy cấp độ độ khó (Dễ đến Chuyên Gia). Sử dụng Phân loại Bloom và các tham số tâm lý trắc nghiệm (độ khó: p, khả năng phân biệt: d), hệ thống áp dụng ma trận tri thức để tích hợp Kết Quả Học Tập Khóa Học (CLOs), sáu chương nội dung, và các cấp độ nhận thức, cho phép tạo bài kiểm tra thích ứng cho bài tập, kỳ thi giữa kỳ, và kỳ thi cuối kỳ. Đánh giá tự động sử dụng phương pháp khớp chính xác cho MCQ/MP/FN/FNS, khớp mẫu cho FPS/FSS, đánh giá biểu thức cho FE và đánh giá dựa trên bộ kiểm thử cho PE. Hệ thống theo dõi tiến trình tri thức qua các bài kiểm tra, theo chủ đề, và theo CLO, cung cấp phản hồi toàn diện. Các triển khai ban đầu đã cho thấy khả năng của hệ thống trong việc nâng cao chất lượng đánh giá, giảm bớt công sức chấm điểm và hỗ trợ các phương pháp giảng dạy trong các lĩnh vực kỹ thuật như Cấu trúc Dữ liệu và Giải Thuật.

Từ khoá: *đánh giá tự động, cấu trúc dữ liệu và giải thuật, học trực tuyến, hệ thống hướng dẫn thông minh, biểu diễn tri thức.*

Received: 29/5/2025
Revised: 09/10/2025
Accepted for publication: 16/10/2025