Syllabus for the Course «Algorithms and Data Structures»

Duration: 16 academic weeks.

Classroom load: 2 acad. hours per week
Independent work: 2 hours per week
Format: AI-Augmented Learning

1. General characteristics of the discipline

The discipline "Algorithms and data structures" is a mandatory part of the general scientific module of the Master's degree program "AI-Augmented Digital Systems Engineering" in the direction of 09.04.02 "Information systems and technologies".
The duration of the discipline is 2 ECTS credits (72 hours).
The intermediate certification form is an exam with a level defense.
The discipline is implemented in 1 semester and is fundamental for the following disciplines:

- Algorithmization and programming languages
- Data structures and databases
- Courses in IP architecture, ML, systems engineering, and project management.

The discipline uses in the practice of the educational process the knowledge, skills and abilities formed in the course

- "Fundamentals of AI and LLM: industrial and context engineering"

2. Place of the discipline in the program structure

The discipline serves as **an adaptive and cognitive leveling gateway** for students with diverse backgrounds (Computer Science, engineering, natural sciences, and humanities).
It generates:

- algorithmic thinking,
- structural representation of data,
- culture of formalization,
- basics of complexity analysis,
- ability to critically evaluate solutions, including those generated by LLM.

The discipline precedes courses in software engineering and AI and provides the necessary cognitive base for their development.

3. Objectives of mastering the discipline

The objectives of mastering the discipline are:

- formation of algorithmic autonomy of thinking;
- mastering data structures as architectural primitives of digital systems;
- develop skills in formalizing tasks and proving correctness;
- creating a culture of analyzing the complexity and scalability of solutions;
- mastering the model of conscious interaction with LLM in an engineering context.

4. Objectives of the discipline

Within the framework of the discipline, the following tasks are solved:

- formalization of engineering tasks without reference to a specific programming language.
- mastering abstract data types and their implementations.
- analysis of the relationship between the data structure and the algorithmic strategy;
- mastering methods for estimating temporal and spatial complexity;
- developing skills to prove the correctness of algorithms;
- develop the ability to critically evaluate solutions, including LLM solutions;
- developing an engineering understanding of time/memory tradeoffs.

5. Planned learning outcomes

As a result of mastering the discipline, the student must:

Know:

- the concept of an algorithm and its formal properties.
- abstract data types and their invariants.
- basic data structures and their asymptotics.
- basic algorithmic strategies.
- principles of complexity analysis.
- the role of the invariant in proving correctness.
- the nature of algorithmic trade-offs.

Be able to:

- formalize the problem into an algorithmic model.
- choose an adequate data structure.
- justify the choice of an algorithmic strategy.
- evaluate temporal and spatial complexity.
- prove the correctness of the decision.
- critically analyze alternative solutions, including LLM proposals.

Be proficient in:

- the method of algorithmic analysis.
- skills of well-reasoned engineering defense;
- culture of working with intelligent assistants;
- ability to move from formalization to architectural thinking.

6. Methodological concept of the discipline

6.1. Discipline as a cognitive adaptation module

The discipline "Algorithms and Data Structures" performs the function of an adaptive and cognitive leveling module in the master's program structure.

Taking into account the heterogeneity of the entrance level of students (graduates of Computer Science, engineering, natural science and humanities), the course:

- forms a single algorithmic conceptual field.
- calibrates the view of the data structure as architectural primitives.
- introduces a culture of formalization and proof.
- forms the discipline of complexity analysis.
- lays the foundation for engineering argumentation.

The course provides a foundation for the subsequent disciplines of software engineering and artificial intelligence, providing a transition from intuitive or language-oriented programming to structural engineering thinking.

6.2. The "structure before algorithm" principle

The methodological axis of the discipline is built around the relationship:

Issue formalization → Data Structure → Algorithm → Complexity → Scalability → Architectural Tradeoff

Data structures are considered not as elements of a specific programming language, but as:

- abstract data types.
- state carriers.
- architectural primitives of digital systems.

Algorithms are studied through the prism of operating structures, rather than as isolated procedures.

6.3. Invariant as an engineering correctness tool

Mastering the concept of invariant is considered as the master's minimum of the discipline.

The invariant is:

- a proof-of-correctness tool.
- a mechanism for detecting logical defects.
- a criterion for distinguishing between correct and plausible solutions.

This is especially important in the context of active use of LLM, as it allows the student to critically evaluate automatically generated solutions.

7. The role and contribution of LLM in the educational process

The use of large linguistic models (LLMs) within the discipline is systematic and methodologically meaningful and is not an auxiliary or optional element.

LLM is integrated into the educational process as a structural component of the pedagogical model.

7.1. LLM as a cognitive differentiation tool

Taking into account the heterogeneity of the input level of students, LLM is used for:

- adaptation of the pace of material development;
- individualization of explanations.
- generating additional examples.
- creating personalized trajectories for working through the topic.

At the same time, the course content remains the same for all students, and differentiation is achieved through the depth of analysis and the degree of independence.

7.2. LLM as an intelligent tutor

At the early stages of mastering the discipline, LLM performs the following functions:

- explanations of concepts;
- demonstration of alternative formulations;
- step-by-step simulation of the algorithm.
- identifying logical gaps in formalization.

In this mode, the model acts as a learning assistant that supports the process of understanding the material.

7.3. LLM as an analytical assistant

At the stage of studying data structures and algorithmic strategies, LLM is used for:

- comparative analysis of alternative structures;
- modeling of algorithm degradation.
- generating boundary cases.
- constructing counterexamples.

The model allows you to speed up the analytical part of the work and expand the space of the considered scenarios.

7.4. LLM as an opponent

At the advanced stage, LLM acts as an intellectual opponent:

- asks clarifying questions.
- identifies weak points in the proof.
- generates alternative solutions.
- provokes checking of invariants and boundary cases.

Thus, a culture of engineering argumentation is formed.

7.5. LLM as an examiner

During the session period, LLM can be used in the following mode:

- generating a complex task.
- forming clarifying questions.
- simulations of a role-playing exam dialog.

At the same time, the final assessment is formed by the teacher based on the analysis of the depth of argumentation, the correctness of the decision and the level of independence of the student.

7.6. The principle of mandatory reflection

Using LLM does not replace the student's algorithmic thinking.
Each solution obtained using the model must be accompanied by:

- independent justification;
- complexity analysis.
- formulation of the invariant.
- comparison with alternatives.

Thus, LLM acts not as a substitute for thinking, but as a means of strengthening and testing it.

7.7. Interaction and review format

The following scheme is used in the educational model of the discipline:
Student → LLM → Teacher → Student

1. The student formulates a solution.
2. LLM performs an initial analytical review.
3. The teacher corrects and deepens the review.
4. The student receives feedback and iterates if necessary.

This approach:

- reduces the entry barrier.
- accelerates feedback.
- retains academic responsibility of the teacher.

7.8. Formation of professional autonomy

The key contribution of LLM within the discipline is not to accelerate problem solving, but to develop the ability to:

- distinguish correctness from plausibility.
- check automatically received solutions.
- identify logical gaps.
- defend your position in a reasoned manner.

This creates the basis for further development of AI and software engineering disciplines, where LLM is used as a tool, and not as a source of uncritically accepted answers.

7.9. Critical position as a prerequisite for working with LLM

The use of LLM in the educational process implies a conscious attitude of the student to the nature and limitations of language models. LLM generates statistically plausible responses that may be superficially convincing, but contain factual, logical, or algorithmic errors.
In the context of the discipline "Algorithms and Data Structures", typical model errors are:

- incorrect asymptotic estimates for nonstandard input conditions.
- erroneous or incomplete loop invariants.
- pseudo-proof of correctness that contains logical breaks.
- incorrect data structure selection for a poorly specified task.
- confidently formulated but erroneous statements about the lower limits of complexity.

In this regard, critical review of LLM responses is not an optional practice, but a mandatory component of every interaction with the model within the discipline.
Required elements for verifying an LLM result:

1. Checking the invariant. The student independently formulates the invariant of the proposed algorithm and checks its preservation on boundary cases.
2. Independent difficulty assessment. The asymptotics proposed by the model are verified by independent analysis-without relying on the model statement.
3. Building a counterexample. The student purposefully searches for input data on which the proposed solution can give an incorrect result.
4. Comparison with an alternative. Any LLM decision is accompanied by consideration of at least one alternative strategy with an explicit justification for the preference.
5. Fixation of reflection. The result of interaction with the LLM is written by the student in the form of a brief reflective note: what the model suggested, what was checked, what was corrected or rejected, and why.

Verification format and control. A reflexive note is a mandatory artifact of the implementation of the Self- Directed Study (SDS) and is presented to the teacher together with the main decision. The lack of verification is considered an incomplete task, regardless of the quality of the final result.
Thus, interaction with LLM within the discipline does not form a dependence on the tool, but a professional habit: trust the argument, not the source.

Calendar and thematic plan (1 semester)

Key:

Lecture (Lect.) — theoretical material delivery.
Practical / Lab. — hands- on activities, simulations and discussions.
Self- directed work (SDS) — independent study on project tasks and analysis.
LLM use — applying Large Language Models for analysis, generation and decision support (integrated into independent study tasks).

Week	Topic and Content (including LLM and SDS)	Lect. (h)	Prac./ Lab. (h)	SDS (h)
1-4	**Formalization and state model**			
1	**Introduction. An algorithm outside the language. Formalizing the task.** Concepts: task, input, output, state, constraint. Algorithm as a sequence of state transitions. **LLM:** diagnostics of the level of formalization; generation of variable tasks; primary review of formalization. **SDS:** formalize 3 tasks of different nature; get an LLM review; correct the formalization; brief reflection (what was changed and why).	1	1-2	2
2	**Abstract Data Type (ADT). Structure invariant.** The difference between abstraction and implementation. Operations as a contract. **LLM:** checking the completeness of the ADT description; detecting missing operations; generating counterexamples. **SDS:** describe the ADT for a given subject model; formulate an invariant; get an LLM review and correct it.	1	1-2	2
3	**Array and linked list. Cost of operations.** The concept of access, insertion, deletion, and indexing. **LLM:** comparative analysis of operations; modeling of degradation with increasing n. **SDS:** solve the problem with two structures; build a table of operations and complexity; justify the choice.	1	1-2	2
4	**Stack and queue as models of order.** LIFO/FIFO. Relationship between structure and state transitions. **LLM:** modeling algorithm execution step-by-step; generating boundary cases. **SDS:** design an algorithm using a stack or queue; describe state transitions.	1	1-2	2

Week	Topic and Content (including LLM and SDS)	Lect. (h)	Prac./ Lab. (h)	SDS (h)
5-9	**Data structures as architectural primitives**			
5	**Set and Hash table.** Fast search, collisions, memory/time tradeoff. **LLM:** comparison of iteration and hashing; worst-case modeling. **SDS:** solve the problem through iteration and through a set; analyze scalability.	1	1-2	2
6	**Tree. Binary search tree. BST.** **LLM invariant:** search for violations of the invariant; generate counterexamples. **SDS:** check the correctness of a given tree; formulate an invariant and prove its conservation.	1	1-2	2
7	**Heap and priority queue.** Minimum extraction, priority tasks. **LLM:** modeling of insert/delete operations; cost analysis. **SDS:** design an algorithm with priorities; estimate complexity.	1	1-2	2
8	**Graph. Graph representations.** Adjacency lists and adjacency matrix. **LLM:** Comparing memory and operations for different views. **SDS:** choose a representation for the problem; justify the choice.	1	1-2	2
9	**Comparative analysis of structures.** Table of operations and complexities. **LLM:** generating an alternative structure; provocative questions. **SDS:** prepare an analytical note on the choice of structure for a complex task.	1	1-2	2
10-13	**Algorithms through the prism of structures**			
10	**Search. Linear and binary options. Binary search invariant.** **LLM:** checking the invariant; modeling boundary cases. **SDS:** formulate and prove an invariant; find a counterexample when conditions are violated.	1	1-2	2
11	**Sorting and comparison model limits.** O(n2) vs O(n log n). **LLM:** comparative analysis of sorts; discussion of the lower bound.	1	1-2	2

Week	Topic and Content (including LLM and SDS)	Lect. (h)	Prac./ Lab. (h)	SDS (h)
	SDS: build a comparative sorting table; evaluate its applicability.			
12	**Graph traversal (BFS, DFS).** The relationship between strategy and structure. **LLM:** traversal modeling; generation of complex graph configurations. **SDS:** implement pseudo-code traversal; estimate complexity.	1	1-2	2
13	**Reduction of iteration through the structure.** Hashing and preprocessing. **LLM:** generating an alternative approach; comparing solutions. **SDS:** suggest an alternative strategy for solving the problem; justify it.	1	1-2	2
14 - 16	**Complexity and engineering reasoning**			
14	**Asymptotics. O, Ω, Θ. Scalability.** **LLM:** modeling the growth of functions; comparative analysis. **SDS:** estimate the complexity of several algorithms; draw a conclusion about scalability.	1	1-2	2
15	**Loop invariant and proof of correctness.** A counterexample. **LLM:** opposing the proof; identifying logical breaks. **SDS:** prove the correctness of the algorithm; conduct саморецензиюself-review through LLM.	1	1-2	2
16	**Time/memory compromise. Architectural choice.** **LLM:** generate alternative solutions with different trade-offs. **SDS:** analytical essay: choosing the structure and strategy for a scalable system.	1	1-2	2
17-18	**Session block**			
17	Complex task pre-defense. Formalization → structure → algorithm → complexity → alternative. LLM examiner.	–	2	2
18	Final defense. Individual interview. Level assessment (basic / advanced / research).	–	2	2