COURSE DESCRIPTION CARD - SYLLABUS

Course name
Algorithms and Data Structures [S1Cybez1>AiSD]

Course

Field of study
Cybersecurity

Year/Semester
1/2

Area of study (specialization)
–

Profile of study
general academic

Level of study
first-cycle

Course offered in
Polish

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
30

Laboratory classes
0

Other
0

Tutorials
0

Projects/seminars
24

Number of credit points

4,00

Coordinators

prof. dr hab. inż. Marta Szachniuk
marta.szachniuk@put.poznan.pl

Lecturers

Prerequisites

A student beginning this course should have a basic understanding of implementing simple algorithms in high-level programming languages (Python, C, C++). They should be able to solve basic programming problems and to test and debug their own programs. Additionally, the student should be capable of gathering information from the indicated sources of knowledge and should recognize the need to expand their competencies in algorithms and programming. With regard to social competences, the student should demonstrate honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, and respect for other people.

Course objective

1. Providing students with fundamental knowledge of computational complexity, including the analysis of the complexity of combinatorial problems and algorithms, the operation of deterministic and nondeterministic Turing machines, RAM machines, the classification of problems and algorithms, and the complexity classes P and NP. 2. Providing students with fundamental knowledge of algorithmics in the scope of sorting numeric sequences of various computational complexities, greedy, exhaustive, and dynamic programming, backtracking, and basic graph algorithms such as graph traversal, topological sorting, finding Euler and Hamilton cycles, as well as the basics of probabilistic algorithms. 4. Providing students with fundamental knowledge about data structures, including operations on arrays, lists, queues, stacks, trees (including BSTs and heaps), and graphs, as well as the analysis of memory complexity and computational complexity of basic operations performed on these data structures. 5. Developing students' skills in implementing the studied algorithms and data structures. 6. Developing students' skills in selecting appropriate algorithms and data structures for a given problem, and in evaluating the computational and memory complexity of their implementations. 7. Developing students' skills in testing implemented algorithms and assessing their correctness, error tolerance, scalability, and efficiency.

Course-related learning outcomes

Knowledge:
- Has advanced and in-depth knowledge of operational research, useful for formulating and solving complex computational problems.
- Possesses structured, theoretically grounded general knowledge in the field of selected combinatorial algorithms.
- Has detailed knowledge of algorithmics, data structures, and the analysis of computational and memory complexity of algorithms.
- Is familiar with fundamental methods, techniques, and tools used in solving basic computational problems and analyzing the computational complexity of algorithms.

Skills:
- Is able to plan and conduct experiments, including measuring algorithm execution time, interpret the obtained results, and draw conclusions regarding the correctness of the algorithm selection and its complexity.
- Can apply analytical and experimental methods to formulate and solve computational problems, selecting appropriate algorithms and data structures. Is capable of evaluating the computational complexity of algorithms and problems.
- Has the ability to develop algorithms and implement them using at least one high-level programming language.

Social competences:
Is able to appropriately determine priorities for completing a task, either self-assigned or assigned by others, by resolving the dilemma of whether the implementation of more efficient algorithms is worth the increased effort required for their development.

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Formative Assessment:
a) Lectures: The verification of the intended learning outcomes is conducted through:
- Rewarding student engagement and participation during lectures.
b) Project Classes: The verification of the intended learning outcomes is conducted through:
- Evaluation of project reports that document the implementation and analysis of assigned algorithms.
- Assessment of implemented algorithms utilizing various data structures.
Summative Assessment:
The verification of the intended learning outcomes is conducted through:
- Evaluation of projects and project reports focused on the implementation and analysis of algorithms and data structures.
- Assessment of knowledge and skills demonstrated in the mid-term and final written exams, which include:
- Several closed-ended tasks requiring students to complete missing calculations and analyses, testing their ability to solve algorithmic problems.

- A minimum requirement of 50% of the total points from both the mid-term and final exams to pass the lecture component.
Active participation during classes is rewarded with additional points, which are considered when determining the final semester grade.
The relationship between the grade and the number of points is defined by the Study Regulations.
Additionally, the course completion rules and the exact passing thresholds will be communicated to students at the beginning of the semester through the university's electronic systems and during the first class meeting (in each form of classes).

Programme content

The course lectures cover fundamental topics in algorithmics:
- Basic concepts: problem and algorithm, data and data operations, instance
- Algorithm correctness and verification
- Classification of problems into decision and optimization problems
- Deterministic and nondeterministic Turing machines and the RAM machine as examples of abstract computational models used for executing algorithms
- Definition of decision problem complexity classes P and NP, including subclasses such as NP-complete and strongly NP-complete problems, along with methods for proving problem membership in these classes
- Computational complexity of problems, including time and memory complexity of algorithms, methods for determining complexity, and expressing it using Big-O notation
- Algorithm construction techniques, such as the top-down approach, divide and conquer, and backtracking
- Comparison of the greedy method and dynamic programming, along with a discussion of pseudo-polynomial complexity using the knapsack problem as an example
- Graph representations in computing, including adjacency matrices, incidence lists, successor lists, and graph matrices

Course topics

Lectures:
The course begins by introducing fundamental algorithmic concepts, such as problems and algorithms, data and data operations, instances, and the concept of type. The topic of algorithm correctness, including its definition and verification, is also covered.The classification of problems into decision and optimization problems is presented, along with their characteristics and examples. The deterministic and nondeterministic Turing machines and the RAM machine are discussed as abstract models of computation used to execute algorithms. Based on this foundation, the concept and definition of complexity classes P and NP, along with their subclasses, are introduced. The computational and memory complexity of algorithms is discussed, including methods for determining and expressing complexity using Big-O notation. Using different sorting algorithms as examples, the course examines best-case, worst-case, and average-case complexity. General algorithm construction techniques such as top-down design, divide and conquer, and backtracking are presented in detail. Greedy, exhaustive search, and dynamic programming approaches are compared using the knapsack problem as an example. Various graph representations in computing are explored, including adjacency matrices, incidence matrices and lists, successor lists, edge lists, and graph matrices. Several graph-related problems (e.g., finding Eulerian and Hamiltonian paths and cycles, and topological sorting) are introduced along with example solutions. The course also covers tree-based data structures, including binary trees, binary search trees (BSTs), and heaps, along with fundamental tree operations and their practical applications. Additionally, the lectures introduce probabilistic algorithms and hashing functions.
The project-based component of the course complements the lectures by focusing on the implementation and practical application of algorithms and data structures. Students complete five projects, each requiring the implementation of multiple algorithms to solve a given combinatorial problem, validation of the correctness of the algorithms, the development of computational testing scripts, execution of performance tests, and preparation of a final report analyzing the test results. The first project involves implementing various sorting algorithms with different computational complexities, including bubble sort, selection sort, insertion sort, quicksort, shell sort, heap sort, merge sort, bucket sort, and counting sort. The second project focuses on advanced data structures, where students implement basic operations such as searching, inserting, and deleting elements in singly and

doubly linked lists, binary trees, balanced trees, and heaps. The third project concerns graph representations and graph-related problems, requiring students to implement breadth-first search (BFS) and depth-first search (DFS) algorithms, as well as topological sorting algorithms for directed graphs. The fourth project deals with backtracking algorithms, focusing on solving problems related to Eulerian and Hamiltonian paths and cycles. The fifth project is dedicated to solving the knapsack problem, where students implement greedy, exhaustive search, and dynamic programming approaches to solve its binary version. All projects are carried out in two-person teams, encouraging collaboration and problem-solving skills.

Teaching methods

1. Lecture: Presentation illustrated with examples.
2. Project: Independent programming of selected algorithms, learning through experience, solving practical problems, verifying code correctness, preparing and conducting computational experiments, reporting and documentation, project presentation and defense, teamwork, and

Bibliography

Basic:
1. Elementy analizy algorytmów, L. Banachowski, A. Kreczmar, WNT, W-wa, 1982
2. Algorytmy + struktury danych = programy, N. Wirth, WNT, W-wa, 2004
3. Złożoność obliczeniowa problemów kombinatorycznych, J. Błażewicz, WNT, W-wa, 1988
4. Wprowadzenie do algorytmów, T.H. Cormen, Ch.E.Leiserson, R.L. Rivest, C. Stein, PWN, W -wa, 2012

Additional:
1. Algorytmika praktyczna nie tylko dla mistrzów, P. Stańczyk, PWN, 2009

Breakdown of average student's workload

	Hours	ECTS
Total workload	109	4,00
Classes requiring direct contact with the teacher	54	2,00
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	55	2,00