Federal State Autonomous Educational Institution of Higher Education "Moscow Institute of Physics and Technology (National Research University)"

Work program of the course (training module)

course:	Data Structures and Algorithms III/Структуры данных и алгоритмы III
major:	Information Science and Computer Engineering
specialization:	Computer Science/Информатика
	Phystech School of Applied Mathematics and Informatics
	Chair of Algorithms and Programming Technologies
term:	2
qualification:	Bachelor

Semester, form of interim assessment: 4 (spring) - Grading test

Academic hours: 60 AH in total, including:
  lectures: 30 AH.
  seminars: 0 AH.
  laboratory practical: 30 AH.

Independent work: 120 AH.

In total: 180 AH, credits in total: 4

Author of the program: V.V. Yakovlev, candidate of physics and mathematical sciences

The program was discussed at the Chair of Algorithms and Programming Technologies 21.05.2020

<div align="center">**Annotation**</div>

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language. The course studies classical algorithms on sorting, data access for various structures, and generic algorithms complexity.

1. Study objective

Purpose of the course
- Form an understanding of the various computational problems associated with streams in networks, tasks for finding strings with or without preliminary indexing, problems in the theory of pair games;
- provide theoretical and practical knowledge about algorithms and data structure, about algorithms and theoretical adjustments of their work, about parameters for evaluating the complexity of algorithms.

Tasks of the course
- Learn to interpret tasks in terms of theories studied, choose the appropriate algorithm for the task;
- development of algorithms for solving the tasks, assessment of the complexity of the algorithms, their modifications and combinations, the selection of suitable data structures for the tasks, the implementation of the algorithms in a generalized form in the C ++ programming language.

2. List of the planned results of the course (training module), correlated with the planned results of the mastering the educational program

Mastering the discipline is aimed at the formation of the following competencies:

Code and the name of the competence	Competency indicators
Gen.Pro.C-1 Apply fundamental knowledge acquired in the physical and mathematical fields and/or natural sciences and use it in professional settings	Gen.Pro.C-1.1 Analyze the task in hand, outline the ways to complete it

3. List of the planned results of the course (training module)
As a result of studying the course the student should:
know:
- Algorithms associated with processing streams in networks;
- string search algorithms and data structures related to indexing tasks,
- estimates of the complexity of standard algorithms.
be able to:
- Implement algorithms of varying complexity on graphs and indexing data structures in the C ++ programming language,
master:
- Methods of decomposition of tasks in the field of information technology and the construction of a single solution using the studied algorithms.

4. Content of the course (training module), structured by topics (sections), indicating the number of allocated academic hours and types of training sessions

4.1. The sections of the course (training module) and the complexity of the types of training sessions

№	Topic (section) of the course	Types of training sessions, including independent work			
		Lectures	Seminars	Laboratory practical	Independent work
1	String search	6		6	30
2	String search 2	8		8	30

3	Computational geometry	8		8	30
4	Combinator games	8		8	30
AH in total		30		30	120

Exam preparation	0 AH.
Total complexity	180 AH., credits in total 4

4.2. Content of the course (training module), structured by topics (sections)

Semester: 4 (Spring)

1. String search

• The concept of prefix, suffix, substring.
• The concept of their own prefix and suffix.
• The task of finding a substring in a string.
• A trivial algorithm for finding a substring in a string.
• Definition of a prefix function.
• Trivial finding algorithm.
• Linear finding algorithm.
• Proof of operating time by the potential method.
• Counting the prefix function for the string q $ t, where q is the pattern and t is the text.
• Knut-Morris-Pratt Algorithm. Stream processing of text without storing the prefix function for the entire line q $ t.
• Finding the maximum palindrome prefix.
• Finding the number of occurrences of each sample prefix in the text.
• The concept of a functional graph. Type of functional graph.
• Acyclic functional graph is a tree.
• The task "palindrome factory". It is required to find how many times a palindrome formed by the prefixes of a given sample enters. A palindrome is formed by string s if it has the form ss, where s is an inverted string. For example, the abaccaba palindrome is formed by the string abac.
• Reduction of the task to LCA search in the tree of the prefix function.

• Definition of the Z-function.
• Trivial search.
• Linear search of the Z-function. Invariants of the algorithm.
• Proof of operating time by the potential method.
• Application to search for substrings in a string.
• Storage of the Z-function only for the sample, and not for the entire line q $ t.
• Finding the maximum palindrome prefix.
• Finding the number of occurrences of each sample prefix in the text.
• Algorithm modification: search for the maximum substring of the palindrome.

• Statement of the problem of searching for several samples simultaneously. Examples from life.
• Estimation of operating time when using the ILC algorithm.
• The task of the algorithm that is being built is a one-time passage through the text.
• Description of boron data structure.
• Construction of boron, estimation of time and memory size.
• Different ways of storing wood children. Estimated amount of memory for each option.
• The concept of a suffix link. Example tree with suffix links.
• Analogy with the prefix function.
• Algorithm for constructing boron along with suffix links. Estimated work time.

• Algorithm Aho-Korasik. Estimated work time.

• Problems when one pattern is a suffix of another. An example of a problem.

• "Long" suffix links, that is, links going to the next terminal vertex, which is the suffix of the current one.

• Estimated runtime when using "long" suffix links.

• Counting the number of occurrences of each prefix in the text. Dynamic wood programming.

• Construction of automatic transitions. Construction of transitions by letters taking into account the movement of suffix links.

• Counting the number of lines over a given alphabet that contain exactly K occurrences of a given word.

• Generalization to counting the number of lines in a given alphabet that contain exactly K occurrences of given words.

• A set of forbidden words over a certain alphabet is given. Check for the existence of an infinite string over this alphabet that does not contain forbidden substrings.

2. String search 2

• Defining a suffix array.
• Construction in $O(N^2 \log N)$
• Optimization using digital sorting to $O(N^2)$
• Search for a substring in the text in an already constructed suffix array.
• Building a suffix array in $O(N \log^2 N)$ by doubling the prefix by which sorting occurs.
• Optimization to $O(N \log N)$ using digital sorting for pairs.
• Kasai algorithm. Proof of work time.
• Finding the number of occurrences of a string in the text.
• Finding the maximum length of the prefix that is included in the text at least K times. (Binary search by answer. Using a segment tree.)

• Definition of a suffix tree.
• The concept of a compressed suffix tree.
• Trivial construction of a compressed suffix tree.
• Constructive proof of the linearity of the number of vertices and edges.
• The linearity of the number of vertices and edges gives hope that the tree can be built faster than in $O(N^2)$.
• Storage of compressed suffix tree. Estimation of the total number of characters on the edges.
• A trivial tree update when adding one character to the end of a line. Two cases: the creation of a new sheet and the passage along the edge.
• Storing a list of vertices that are suffix.
• Heuristic sheet. Adding an infinite number of characters per edge when adding a sheet.
• The case of the existence of a position when adding a letter. Proof of the fact that all suffixes of a smaller size in this case are also contained in this tree.
• Refusal to store the list of suffix vertices.
• Suffix link. Support for the invariant that a suffix reference is calculated for all internal vertices.
• Proof of the fact that a suffix link always leads to the top.
• Move to a smaller suffix and count the suffix link for the newly created vertex.
• Fast descent, justification of its admissibility.
• Formulation of potentials to prove the operating time. The potential along the length of the word corresponding to the top. Potential by the number of intermediate vertices from the root.

• Proof of the total running time of fast descents.
• Proof of the fact that when following a suffix link, the potential of intermediate vertices decreases by no more than one.
• Proof of the running time of other operations.
• The task of finding the largest common substring of two strings $O(N)$.

• A generalization to search for a common substring of K lines O (NK), where N is the total length of all lines.

• Algorithm for finding the number of different colors of sheets in all subtrees of a given tree.

• Search for the largest common substring K of strings O (N + K).

• Search for the largest substring palindrome.

• Search for the number of occurrences of a string in the text.

• Search for the number of disjoint occurrences of the string in the text.

• Search for the largest line that goes without intersections in the text at least K times. O (N ^ 2 logN)

• Definition of a suffix automaton.

• The suffix tree is a suffix automaton, but with a very large number of vertices and edges.

• The concept of the right context. Physical meaning. Suffix numbering.

• Two representations of the right context: as many lines, as many suffix numbers.

• An example of constructing all right contexts for a string.

• Properties of strings that have the same right context.

• Machine states are right contexts.

• Constructing a transition from one state (right context) by a given letter.

• An example of constructing a suffix automaton. (Save the example on the board)

• Definition of terminal states.

• Rebuild the machine when adding a letter to the end. An analogy with the Ukkonen algorithm.

• Adding a new state that matches the entire row.

• The concept of a suffix link. Adding suffix links to the drawn example.

• Suffix path. Properties of the lines corresponding to the vertices of the path. Examples of intersections of suffix paths.

• Proof of the fact that only one state can be separated, that is, there is no more than one vertex that needs to be divided into two on a new line.

• Adding transitions by the added letter from terminal vertices that did not contain transitions by this letter.

• Clone vertices if necessary. An example of when this happens.

• Changing transitions by the added letter from terminal vertices that previously contained transitions by this letter.

• Potential formulation: Suffix path length from top to start vertex.

• Proof of the fact that the potential increases by no more than 1 when going through the letter.

• Estimation of the working time of the "first cycle": that is, adding transitions from terminal vertices that did not contain this transition.

• Estimation of the operating time of the "second cycle": that is, the change of transitions from terminal vertices that already contained this transition.

• General estimation of the running time of the algorithm.

• Finding the number of occurrences of a given string in the text.

• Finding the largest common substring of two strings.

• The text T is specified. Required to respond to requests. The queries specify a pattern Q and an arbitrary automaton A. Find the number of occurrences of pattern Q such in text T such that the suffix of text T is the next automaton after the occurrence of Q

3. Computational geometry

• Introduction. Lines, segments, planes. Scalar product, vector product.

• Convex hull 2D.

• Jarvis algorithm.

• Graham Algorithm and Andrew Algorithm.

• Convex hull 3D.

• Full search for O (n4).

- Gift wrapping for O (n2).
- The Divide and Conquer method for O (n log n).
- 2D calculations.
- Search for a pair of nearest points on the plane. Algorithm Drugs. O (n log n).
- Search for a triangle with a minimum perimeter beyond O (n log n).
- Search for the diameter of a set on a plane beyond O (n log n).
- Search for the covering rectangle of the minimum perimeter in O (n log n).
- Search for a covering rectangle of minimum area beyond O (n log n).
- Search for faces of a planar graph in O (n log n).
- Minkowski sum for O (n).
- The Minkowski sum of two convex polygons in O (m + n).
- Check for the intersection of two convex polygons in O (m + n).
- Scanning line.
- Verification of the intersection of the segments.
- The area of the union of the rectangles in O (n log n).
- KD-tree.
- Search for points in a rectangle.
- Search for the nearest neighbor.
- Delaunay triangulation.
- Iterative construction algorithm. Flips. Using the KD tree as a localization structure.
- The algorithm of construction by the method of "Divide and conquer" for O (n log n).
- EMOD - Euclidean minimal spanning tree. Sufficiency of using Delaunay triangulation ribs. O (V log V).
- Voronoi diagram.
- Fortune's algorithm.
- Equivalence of Delaunay triangulation.

4. Combinator games

- Math games. Intuitive concept of winning and losing games.
- Game with stones. There are N stones, a player can take from 1 to K stones. The player who takes the last stone wins.
- Modification: the player who takes the last stone loses.
- Playing coins at the round table. Players take turns placing round coins on a round table so that they do not intersect. A player who cannot make a move loses. Solution by the symmetric strategy method.
- The concept of playing on a graph. Winning and losing peaks.
- Normal and abnormal games. Normal is a game in which the leaves are losing. Reducing an abnormal game to normal.
- Fair and unfair games. Fair, when each player from one position can make the same moves as the opponent. An example of an unfair game: chess, checkers. Reduction of unfair to fair.
- The criterion of winning and losing strategies.
- Determining the optimal strategy based on the number of moves.
- Search for the optimal strategy in acyclic graphs.
- An example of a draw in cyclic graphs.
- The solution to an acyclic abnormal game, where the terminal vertices can be either winning or drawn.
- An iterative algorithm for solving a cyclic game.
- Proof by induction of the correctness of the algorithm.
- Retro analysis. Finding the move in optimal strategy during retro analysis.

- Game with stones. There are N stones, the player can take from a1, a2, ..., aK stones. The player who takes the last stone wins. Reduction to the game on the graph. Dynamic programming without explicit graph storage.
- View along reverse edges. In this case, you only need to look at the edges included in the losing vertices.

• Modification: the player who takes the last stone loses.

• Modification: ai does not exceed 10. In this case, only the previous 10 values are important. The frequency of the game. Estimation of the period.

• Given two piles of stones. The player can take any number of stones from one pile or take the same number of stones from both piles. The game is normal. Finding all losing positions with a total number of stones no more than N for O (N).

• Direct amount of games on the graph.

• Reduction to the usual game on the graph.

• An accurate estimate of the number of vertices and edges in a new graph.

• Inability to analyze such big games.

• Associativity and commutativity of the direct sum (within the meaning).

• A theorem on the possibility of replacing a game in direct sum with an equivalent one.

• The symmetric strategy theorem for acyclic games.

• The theorem that adding to the direct amount of a losing game does not affect the result.

• The theorem on the equivalence of all losing games to neem from zero stones.

• Neem equivalence criterion.

• A theorem on the equivalence of neem games, of which moves are only nym.

• The direct sum of two nims.

• The nonequivalence theorem of two different nims.

• Examples of games with cycles that are not equivalent to neem. Loop, Save.

• A generalization of the neem equivalence theorem to the case when there are moves to games that are not equivalent to neem.

• An iterative algorithm for finding all games equivalent to neem in a cyclic graph.

• Proof that at the end of the algorithm, all unmarked vertices are not equivalent to neem. Induction along the length of the optimal strategy.

• Proof of the fact that uncertain vertices are not losing.

• The direct sum of the two uncertainties of a draw.

• Classification of uncertainties, selection of representatives. Proof that the uncertainty is equivalent to the corresponding representative of the equivalence class.

• Completion of the algebra of direct sums of games.

• Game with stones. There are N stones, the player can take from a1, a2, ..., aK stones. The player who takes the last stone wins. Calculation of the Grandi function.

• Modification: the player who takes the last stone loses.

• Acceptance of graph modifications for abnormal games. The ability to analyze the direct amount only if the player loses when taking the last stone in any pile. Generalization to acyclic graph.

• Modification: ai does not exceed 10. In this case, only the previous 10 values are important. The frequency of the game. Estimation of the period depending on K.

• Game chords. There are N points on the circle, the player connects two points so that the chords do not have common points. The game is normal. This is an example of a game that comes down to a direct sum during the game.

• Oktal games. The decision of an arbitrary octal game.

• An example of a non-octal game. There are a bunch of stones, it is allowed to divide the pile into two not empty and not equal piles.

• Classification of vertices of a functional graph depending on the parity of the cycle.

5. Description of the material and technical facilities that are necessary for the implementation of the educational process of the course (training module)

Classroom equipped with computers for each student.

6. List of the main and additional literature, that is necessary for the course (training module) mastering

Main literature

Additional literature

7. List of web resources that are necessary for the course (training module) mastering

1.http://neerc.ifmo.ru/wiki/index.php?title=Дискретная_математика,_алгоритмы_и_структуры_данных – «Викиконспекты», сайт Санкт-Петербургского Института теоретической механики и оптики.
2. https://ejudge.lksh.ru/archive/2014/07/A/games.pdf – Станкевич А., Игры на графах.

8. List of information technologies used for implementation of the educational process, including a list of software and information reference systems (if necessary)

Software for developing and debugging programs in the C ++ programming language.

9. Guidelines for students to master the course

A student studying a discipline must, on the one hand, master the general conceptual apparatus, and on the other hand, must learn to put theoretical knowledge into practice.

As a result of studying the discipline, the student must know the basic definitions, concepts.

Successful development of the course requires intense independent work of the student. The course program provides the minimum necessary time for the student to work on the topic. Independent work includes:

- reading and taking notes of recommended literature;

- study of educational material (according to lecture notes, educational and scientific literature);

- preparation for differentiated classification.

The management and control of the student's independent work is carried out in the form of individual consultations.

It is important to gain an understanding of the material being studied, and not its mechanical memorization. If it is difficult to study certain topics, questions, you should consult a teacher.

Assessment funds for course (training module)

major:	Information Science and Computer Engineering
specialization:	Computer Science/Информатика
	Phystech School of Applied Mathematics and Informatics
	Chair of Algorithms and Programming Technologies
term:	2
qualification:	Bachelor

Semester, form of interim assessment: 4 (spring) - Grading test

Author: V.V. Yakovlev, candidate of physics and mathematical sciences

1. Competencies formed during the process of studying the course

Code and the name of the competence	Competency indicators
Gen.Pro.C-1 Apply fundamental knowledge acquired in the physical and mathematical fields and/or natural sciences and use it in professional settings	Gen.Pro.C-1.1 Analyze the task in hand, outline the ways to complete it

2. Competency assessment indicators

As a result of studying the course the student should:

know:
- Algorithms associated with processing streams in networks;
- string search algorithms and data structures related to indexing tasks,
- estimates of the complexity of standard algorithms.

be able to:
- Implement algorithms of varying complexity on graphs and indexing data structures in the C ++ programming language,

master:
- Methods of decomposition of tasks in the field of information technology and the construction of a single solution using the studied algorithms.

3. List of typical control tasks used to evaluate knowledge and skills

Not provided.

4. Evaluation criteria

The list of control questions for passing the test:

1) The relationship of hardware, operating system and programming system in a computer system.

2) Types of objects in computing systems (types implemented by hardware, operating system, user programs).

3) Methods for implementing types (type control): dynamic control, control at compile time. Hardware requirements for efficient type implementation.

4) Pointers to objects, virtual memory and the problem of stuck pointers.

5) Exceptions and abnormal methods for completing procedures.

6) The main types of objects sold by operating systems

7) The main properties of the address space and its relationship with other main objects of the operating system.

8) The main properties of the executable program and its relationship with other main objects of the operating system.

9) The main properties of the library of functions and its relationship with other main objects of the operating system.

10) The main properties of the computational flow (procedure call stack) and its relationship with other main objects of the operating system.

11) The main properties of the file and file system and their relationship with other main objects of the operating system

12) What is a user, task, session. By what means does the OS implement these concepts.

13) The approach of programming languages to data protection. Contextual naming of data arrays.

14) Organization of a file system for the contextual naming of external objects (files).

15) Types of data exchanges between files and arrays in operational (virtual) memory.

16) The Unix process as a connection in one design of the address space, the executable program and the stack of procedure calls. Address space expansion by file open table.

17) Signals, process groups, means of working with signals in Unix-type OS.

18) The main attributes of the process, generation and termination of the process, signals and work with them, synchronization of processes through signals in Unix-type OS.

19) Sessions, tasks (work) in OS type Unix. Teams of work with tasks.

20) File system, directories, file naming. Managing file lifetimes on an Unix-type OS.

21) Files and address space in a Unix-type OS.

22) Data exchange between files and RAM: basic buffered and streamed buffered; direct exchange; asynchronous exchange. Inter-process communication via pipe files in a Unix-type OS

23) Users and user groups. Data protection through file attributes on a Unix-type OS.

24) Structured use of non-local transitions in C.

25) Command language, options and arguments, assigning file names, handling symbolic links. Basic utilities like Unix.

26) Flags of the compiler and flags of the link editor. How to create a dynamically loaded library and how to load such a library with your program. What is dlopen ().

excellent

10 comprehensive, systematized, deep knowledge of the curriculum of the discipline and the ability to confidently apply them in practice when solving specific problems, free and correct justification of decisions made;

9 systematic, deep knowledge of the curriculum of the discipline and the ability to confidently apply them in practice when solving specific problems, the correct justification of decisions made;

8 deep knowledge of the curriculum of the discipline and the ability to apply them in practice when solving specific problems, the correct justification of decisions made;

good

7 firmly knows the material, correctly and essentially sets out it, knows how to apply the knowledge gained in practice, but admits some inaccuracies in the answer or in solving problems;

6 knows the material, correctly presents it, knows how to apply the acquired knowledge in practice, but admits some inaccuracies in the answer or in solving problems;

5 knows the basic material, correctly presents it, knows how to apply the knowledge gained in practice, but admits inaccuracy in the answer or in solving problems;

satisfactorily

4 fragmented, fragmented nature of knowledge, insufficiently correct wording of basic concepts, violation of logical sequence in the presentation of program material, but at the same time he owns the main sections of the curriculum necessary for further training and can apply the acquired knowledge in the standard situation;

3 the nature of knowledge is sufficient for further training and can apply the acquired knowledge on the model in a standard situation;

unsatisfactory

2 does not know most of the main content of the curriculum of the discipline, makes gross errors in the wording of the basic concepts of the discipline and does not know how to correctly use the knowledge gained in solving typical practical problems.

1 does not know the wording of the basic concepts of the discipline and does not know how to use the knowledge gained in solving typical practical problems.

5. Methodological materials defining the procedures for the assessment of knowledge, skills, abilities and/or experience

The test time is 2 academic hours. During the test, students can use the discipline program and the source code.