ALGORITHM IS A FUNDAMENTAL CONCEPT IN COMPUTER SCIENCE

Shazadaev Farkhod Alimovich
Teacher at Tashkent branch of Samarkand state veterinary medical, animal husbandry and biotechnological university

https://doi.org/10.5281/zenodo.10022846

Abstract. *An algorithm is a clear sequence of actions aimed at achieving a goal or solving a problem.*

The history of the development of algorithms has come a long way from intuitive understanding and spontaneous application to awareness of patterns and practical use in modern computers and computer systems.

"Algorithm" is a fundamental concept in computer science. An understanding of it is necessary for the effective use of computer technology to solve practical problems. An algorithm is an instruction to a performer (human or automatic) to perform a precisely defined sequence of actions aimed at achieving a given goal. An algorithm is a rule formulated in a certain language, indicating actions, the sequential execution of which leads from the initial data to the desired result. The meaning of the word algorithm is very similar to the meaning of the words recipe, process, method, method. However, any algorithm, unlike a recipe or method, necessarily has the following properties

Keywords: *algorithm, history, use in modern computers, computer systems, application of computer technology.*

The word algorithm comes from *algorithmi*, the Latin spelling of the name of a 9th-century Arabic mathematician. Al-Khorezmi, who formulated the rules for performing four arithmetic operations on multi-digit numbers.

For many centuries before this, people were convinced that the rules of counting were very complex and accessible only to a select few. In the textbook, Al-Khorezmi cited methods of counting that even a child can master. With the development of science and technology, humanity realized that it is possible to learn to perform complex actions if they are broken down into a sequence of simple ones. The word "algorithm" has acquired a different meaning, not related to arithmetic

Algorithms in computer science are needed to effectively solve various problems, including those that are difficult to perform “head-on” or completely impossible. In practice, there are algorithms for almost anything: sorting, traversing data structures, searching for elements, filtering information, mathematical operations, and so on.

For example, you can sort an array using an exhaustive search - this is the most obvious solution. Or you can use the quick sort algorithm: it is more complicated and not so obvious, but it works much faster and does not load the computer’s power so much. Strictly speaking, exhaustive search is also an algorithm, but a very simple one.

There are algorithmically unsolvable problems for which there is no and cannot be an algorithm. But most problems in IT are solvable algorithmically, and algorithms are actively used in working with them.

Algorithmization is the process of developing and describing a sequence of steps that must be performed to solve a specific problem or achieve a specific goal. Algorithmization is a key step in programming and software development.

When a task is algorithmized, clear instructions are created that the computer can understand and follow. Algorithms can be written as a text description, flowchart, pseudocode, or other formalized representation. They serve as the basis for writing program code that allows the computer to automatically solve problems according to pre-designed instructions.

Algorithmization plays an important role in computer science and programming, as well-developed algorithms ensure efficient and correct execution of tasks, and also simplify the process of debugging and maintaining program code.

Basic properties of algorithms:

Discreteness. An algorithm is not a single indivisible structure; it consists of individual small steps, or actions. These actions occur in a certain order, one begins after the completion of the other.

Productivity. Execution of the algorithm must lead to some result and leave no uncertainty. The result may also be unsuccessful - for example, the algorithm may report that there is no solution - but there must be one.

Determinism. At each step there should be no discrepancies or disagreements, instructions should be clearly defined.

Mass character. The algorithm can usually be extrapolated to similar problems with different initial data - just change the initial conditions. For example, the standard algorithm for solving a quadratic equation will remain the same no matter what numbers are used in the equation.

Clarity. The algorithm should include only actions that are known and understandable to the performer.

Limb. Algorithms are finite; they must complete and produce a result, in some definitions, in a predetermined number of steps.

What are the types of algorithms?

Despite the word “sequence,” the algorithm does not always describe actions in a strictly defined order. This is especially true now, with the spread of asynchrony in programming. Algorithms have a place for conditions, loops, and other nonlinear constructs.

Linear. This is the simplest type of algorithm: actions follow each other, each beginning after the previous one ends. They are not rearranged, not repeated, and are performed under any conditions.

Branching. In this type of algorithm, branching appears: some actions are performed only if certain conditions are true. For example, if a number is less than zero, then it must be removed from the data structure. You can add a second branch: what to do if the condition is false - for example, the number is greater than zero or equal to it. There can be several conditions; they can be combined with each other.

Cyclic. Such algorithms are executed in a loop. When a block of actions ends, these actions begin again and are repeated a certain number of times. A loop can involve a single action or sequence, and the number of repetitions can be fixed or dependent on a condition: for example, repeat this block of code until there are no empty cells left in the data structure. In some cases the loop can be endless.

Recursive. Recursion is a phenomenon where an algorithm calls itself, but with different input data. This is not a cycle: the data is different, but there are several “instances” of running programs, not just one. A well-known example of a recursive algorithm is the calculation of Fibonacci numbers.

Recursion allows you to elegantly solve some problems, but you need to be careful with it: such algorithms can heavily load system resources and work slower than others.

Probabilistic. Such algorithms are mentioned less often, but they are quite an interesting type: the operation of the algorithm depends not only on the input data, but also on random variables. These include, for example, the well-known Las Vegas and Monte Carlo algorithms.

Basic and auxiliary. This is another type of classification. The main algorithm solves the immediate problem, the auxiliary one solves the subtask and can be used inside the main one - for this purpose its name and input data are simply indicated there. An example of an auxiliary algorithm is any software function.

Become a data scientist and solve ambitious problems using neural networks

More details

Become a data scientist and solve ambitious problems using neural networks

Graphic representation of algorithms

Algorithms can be written in text, code, pseudocode, or graphically in the form of flowcharts. These are special diagrams consisting of geometric shapes that describe certain actions. For example, the starting and ending points on the diagram are, respectively, the beginning and end of the algorithm, a parallelogram is the input or output of data, a rhombus is a condition. Simple actions are indicated by rectangles, and the figures are connected using arrows - they show sequences and cycles.

Example: block diagram of an algorithm

The diagrams contain specific actions, conditions, number of cycle repetitions and other details. This allows you to understand the algorithms more clearly.

The concept of “complexity” is one of the key ones in the study of algorithms. It does not mean how difficult it is to understand a particular method, but the resources spent on the calculation. If the complexity is high, the algorithm will run slower and possibly waste more hardware resources; it is advisable to avoid this.

Complexity is usually described with a capital O. After it, in parentheses, is a value that determines the execution time. This is a notation from mathematics that describes the behavior of different functions.

What is the difficulty? We will not fully analyze the mathematical O-notation, as it is called, but will simply list the main designations of complexity in the theory of algorithms.

O(1) means that the algorithm runs in a fixed constant time. These are the most efficient algorithms.

O(n) is the complexity of linear algorithms. n hereinafter denotes the size of the input data: the larger n, the longer the algorithm takes to execute.

$O(n^2)$ also means that the larger n, the higher the complexity. But the dependence here is not linear, but quadratic, that is, the speed increases much faster. These are inefficient algorithms, for example with nested loops.

O(log n) is a more efficient algorithm. The speed of its execution is calculated logarithmically, that is, it depends on the logarithm of n.

O(√n) is an algorithm whose speed depends on the square root of n. It is less efficient than logarithmic, but more effective than linear.

There are also O(n^3), O(nn) and other inefficient algorithms with high degrees. Their complexity increases very quickly, and it is better not to use them.

Graphic description of difficulty. A graph can help you better understand the complexity in O-notation. It shows how the execution time of the algorithm changes depending on the size of the input data. The flatter the line the graph gives, the more efficient the algorithm.

O-notation is used to evaluate whether a particular sequence of actions is efficient to use. If the data is large or there is a lot of it, they try to look for more efficient algorithms to speed up the program.

Use of algorithms in IT

We will give several examples of the use of different algorithms in programming industries. In fact, there are many more - we took only a part to help you understand the practical significance of algorithms.

Software and website development. Algorithms are used for parsing, that is, “parsing” data structures such as JSON. Parsing is one of the basic tasks, for example on the web. Algorithms are also needed when drawing dynamic structures, displaying alerts, customizing application behavior, and much more.

Working with data. Algorithms are very actively used when working with databases, files where information is stored, structures such as arrays or lists. There can be a lot of data, and choosing the right algorithm allows you to speed up working with it. Algorithms solve the problems of sorting, changing and deleting necessary elements, and adding new data. With their help, they fill and pass through structures such as trees and graphs.

Algorithms are of particular importance in Big Data and data analysis: there they allow you to process a huge amount of information, including raw information, without spending too many resources on it.

Search tasks. Search algorithms are a complex industry in their own right. They are separated into a separate group, which now includes dozens of different algorithms. Search is important in data science, artificial intelligence techniques, analytics, and more. The most obvious example is search engines like Google or Yandex. By the way, search engines usually keep details about the algorithms they use secret.

Machine learning. In machine learning and artificial intelligence, the approach to algorithms is slightly different. If a regular program operates according to a given order of actions, then a “smart machine”—a neural network or a trained model—generates an algorithm for itself during training. The developer describes the model and trains it: gives it initial data and shows examples of what the final result should look like. During training, the model itself comes up with an algorithm for achieving this result.

Such AI algorithms can be even more powerful than conventional ones and are used to solve problems that the developer cannot consciously break down into simple actions. For example, to recognize objects, you need to use a huge number of processes in the nervous system: a person is simply physically unable to describe them all in order to repeat them programmatically.

During the creation and training of the model, the developer can also use algorithms. For example, the error propagation algorithm allows you to train neural networks.

Speaking informally, an algorithm can be called any correctly defined computational procedure when some value or set of values is given as input, and the result of this procedure is an output value or set of values. We can say that an algorithm is a certain sequence of actions (computational steps), due to which the input data is converted into output data.

You also need to understand that an algorithm as a sequence of steps allows you to solve a specific problem and must:

1. Work in a finite amount of time. If an algorithm is unable to solve a problem in a finite amount of time, it can be said to be useless.

2. Have clearly defined instructions and order. Every step must be precisely defined. His instructions must be unambiguous for any sequence of steps.

3. Be fit for use. The algorithm must be able to solve the problem it was created to solve.

If you are interested in details, here are separate articles about the properties of the algorithm, and here are about ways to represent and write the algorithm.

Today, algorithms are used in data processing both in computer science and programming, and in mathematics. By the way, the earliest mathematical algorithms are factorization and square root extraction - they were used in ancient Babylon back in 1600 BC. e. But we will not go far into the past, but will consider, as promised, the main programming algorithms for today.

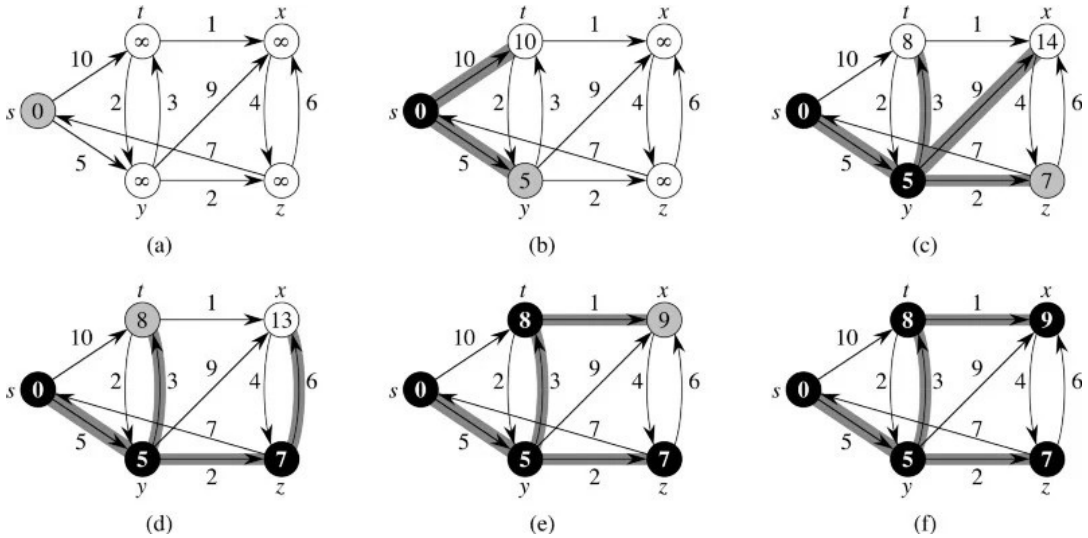
Which sorting algorithm is considered the best? There is no definite answer here, because it all depends on your preferences and the tasks assigned to you. Let's look at a few main ones:

1. Merge sort. The most important one today is based on the principle of comparing elements and uses a “divide and conquer” approach, allowing you to more efficiently solve problems that were once solved in O(n^2) time. Merge sort was invented by mathematician John von Neumann back in 1945.

2. Quick sort. This is a different approach and a slightly different procedure. Here the algorithm is based both on in-place division and on the “divide and conquer” principle. However, this sorting is unstable, which is its problem. But the algorithm is effective when sorting arrays in RAM.

3. Pyramid sort. An in-place algorithm that uses a priority queue (due to this queue, the data search time is reduced).

Dijkstra's algorithm



Without this algorithm, again, the Internet will not be able to work effectively. With its help, problems are solved in which the problem is represented in the form of a graph that provides

a search for the shortest path between 2 nodes. Even today, when we have better solutions, programmers still use shortest path search when it comes to systems that require increased stability.

Algorithms in science and technology

The role of algorithmic sequences in modern science and technology is very great. Without exaggeration, they are considered the wealth of science and technology that has accumulated over decades. You can recall the notorious Turing machine and more.

However, in fact, the algorithms that have been accumulated in mathematics are of greatest importance, because it is mathematics that permeates other technical sciences and makes them more structured. It has been noticed that if it is possible to obtain an algorithmic sequence of data that allows solving a specific problem, then it is possible to create a corresponding machine, and that if it is possible to obtain an algorithm for solving a problem, then it is possible to create a machine, and therefore automate the solution of this problem. And automation is important not only in the IT sector, but also in industry, technology, science, medicine and many other sectors.

From this point of view, the essence of the algorithms can be expressed differently, that is, they are:

- form of presentation of scientific data, research results and analysis;
- a real guide to action to solve previously studied problems;
- a means of saving time and mental work;
- an important and necessary stage in automation;
- a tool for studying new problems (especially in mathematics).

Since we remembered mathematics, let's say that Soviet (Russian) scientists made a great contribution to the development of algorithms here. For example, the so-called four Russian algorithms are well known - a method of algorithmic acceleration using Boolean matrices. It was created by four Russian scientists V.L. Arlazarov, E.A. Dinits, M.A. Kronrod and I.A. Faradzhev in 1970 in Moscow. Also worthy of mention is one well-known method of the Russian scientist Anatoly Karatsuba - the algorithm he created is used for fast multiplication.

Algorithms are an important part of both the entire science and the local processing of initial data, but this part does not exhaust the content of the science itself. No less important are the concepts and definitions that are included in this science, established facts (proven theorems), and developed approaches to the phenomena and objects being studied.

REFERENCES

1. Bondarenko, S. A. Computer and laptop for children / [S. A. Bondarenko]. – Moscow: Eksmo, 2016. - 79 p.
2. Goryachev, A. V. Informatics. Computer science in games and tasks: workbook: 2nd grade: at 2 o'clock / [Goryachev A.V., Gorina K.I., Volkova T.O.]. – Moscow: Balass, Part 1. - 2016. - 64 p.
3. Goryachev, A. V. Informatics. Computer science in games and tasks: workbook: 2nd grade: at 2 o'clock / [Goryachev A.V., Gorina K.I., Volkova T.O.]. – Moscow: Balass, Part 2. - 2016. - 96 p.
4. Doctor Beat. Computer science for beginners: theory, practice, tests: 1st stage: for primary school students. – Moscow: Dragonfly, 2009. - 72, [3] p.: color. ill.

5. Doctor Beat. Computer science for beginners: theory, practice, tests: Level 2: for primary school students. – Moscow: Dragonfly, 2009. - 78, [3] p.: color. ill.
6. Zlatopolsky, D.M. Entertaining computer science: textbook / D.M. Zlatopolsky. - 4th ed. – Moscow: Knowledge Laboratory Laboratory, 2017. - 424 p.: ill.
7. Dzhumaev M.Use of Computer Software in Education of Students in Primary School and Preschool. www.biogenericpublishers.com.28.09.2021 AngilY.
8. Djumaev M. Mathematical regularity and development of creative thinking of students. Deutsche internationale Zeitschrift für zeitgenössische Wissenschaft /German International Journal of Modern Science. Edition: № 28/2022 (February) – 28th Passed in press in February 2022 №28 2022. 26-28 st.
9. Computer science for elementary school: in tables and diagrams: what a computer consists of. Working with files and algorithms. Windows programs and the Internet: [textbook] / author - comp.: V. V. Moskalenko; resp. ed. Oksana Morozova. – Rostov-on-Don: Phoenix, 2012. - 64 p.
10. Computer science: textbook for grade 2: at 2 hours / N. V. Matveeva, E. N. Chelak, N. K. Konopatova [etc.] - Moscow: Binom. Knowledge Laboratory, Part 1. - 2017. – 80 p.