What is algorithm?

Algorithms (also known as Algorithms - English is Algorithms) is a finite set of instructions to be executed in a certain order to get the desired result. In general, the algorithm is independent of programming languages, ie an algorithm can be deployed in many different programming languages.

What is algorithm?

Algorithms (also known as Algorithms - English is Algorithms) is a finite set of instructions to be executed in a certain order to get the desired result. In general, the algorithm is independent of programming languages, ie an algorithm can be deployed in many different programming languages.

From the perspective of data structure, here are some important algorithms:

Algorithm Search : Algorithm for finding an element in a data structure.

Algorithm Alignment : Algorithms to arrange elements in some order.

Algorithm Insert : Algorithm to insert a word into a data structure.

Algorithm Update : Algorithm to update (or update) an element that already exists in a data structure.

Algorithm Delete : Algorithm to delete an existing element from a data structure.

Characteristics of the algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics:

Determination : Algorithms should be clear and not ambiguous. Each stage (or step) should be clear and carry only a certain purpose.

Defined input data : An algorithm should have 0 or more specified input data.

Output : An algorithm should have one or more defined output data, and should connect to the type of result you want.

Stop : Algorithms must end after a finite number of steps.

Efficiency : An algorithm should be enforceable with available sources, ie being able to effectively solve the problem in terms of time and resources allowed.

Popularity : An algorithm is popular if this algorithm can solve a class of similar problems.

Independence : An algorithm should have directives independent of any programming code.

How to write an algorithm?

Do not look, because there will not be any given criteria for writing algorithms. As you know, programming languages ??have loops (do, for, while) and flow control commands (if-else), . You can use these commands to write an algorithm.

We write algorithms in a way that is step by step. Writing algorithms is a process and is executed after you have clearly addressed the problem. From locating the problem, we will design a solution to solve that problem and then write the algorithm.

For example writing algorithms

You follow the example below to see the steps and how to write an algorithm. Of course, the following example is quite simple because this is just an illustrative example of how to write algorithms, so I think the simpler the better.

Problem : Design an algorithm to add two numbers and display the results.

```
 B??c 1 : B?t ??u B??c 2 : Khai báo ba s? a , b & c B??c 3 : ??nh ngh?
a các giá tr? c?a a & b B??c 4 : C?ng các giá tr? c?a a & b B??c 5 : L?u tr
? k?t qu? c?a B??c 4 vào bi?n c B??c 6 : In bi?n c B??c 7 : K?t thúc
```

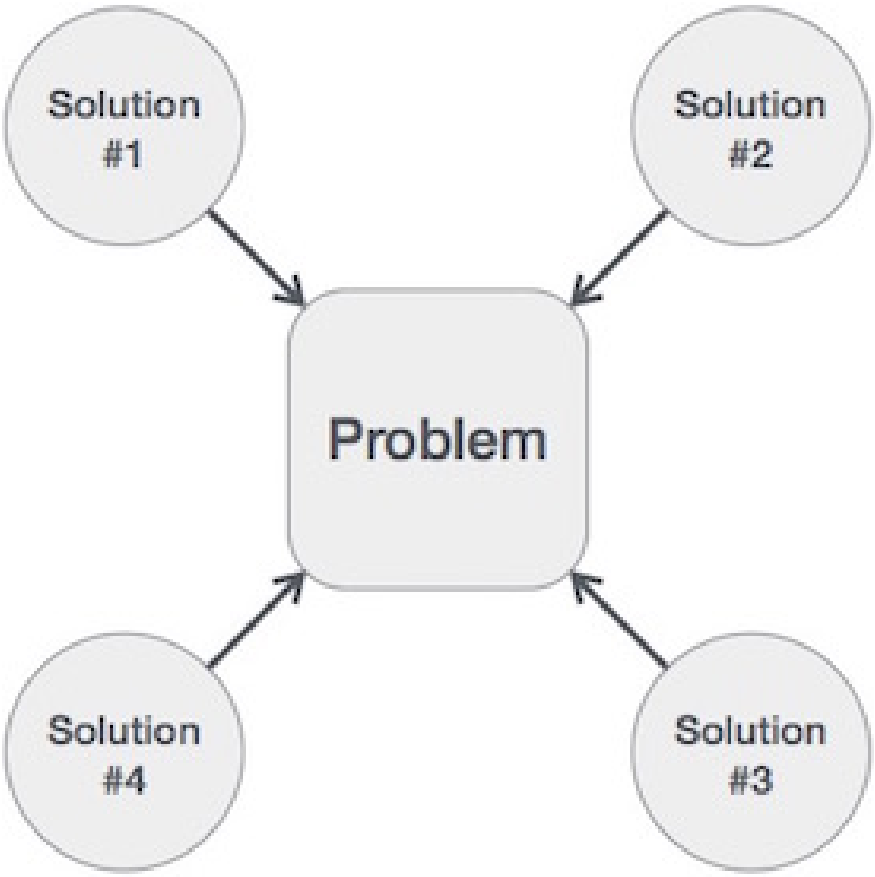
Algorithms tell developers how to write code. In addition, you can also write an algorithm for the above problem as follows:

```
B??c 1 : B?t ??u B??c 2 : L?y giá tr? c?a a & b B??c 3 : c ? a + b B??c 4
: Hi?n th? c B??c 5 : K?t thúc
```

While designing and analyzing algorithms, often the second method is used to describe an algorithm. This second way makes it easy to analyze algorithms when omitting unnecessary definitions. Looking at this second way, one can know which operations are being used and how the execution process works.

Of course, writing step names is **optional** .

We write an algorithm to find a solution to a certain problem. A problem can be solved in many different ways.



Therefore, a problem may have many solutions. So which solution would be most appropriate for that problem. Please continue to follow.

Algorithm analysis

The effect of an algorithm can be analyzed based on two angles: before deployment and after deployment:

Theory analysis : It can be considered as an analysis based on theory only. The effectiveness of the algorithm is assessed by assuming that all other factors (eg, processor speed, .) are constant and do not affect algorithm deployment.

Proximity analysis : This algorithm analysis is conducted after it has been carried out in a certain programming language. After running and checking the relevant parameters, the effect of the algorithm is based on parameters such as runtime, execution time, amount of memory needed, .

In this chapter we will explore theoretical analysis. As for asymptotic analysis, we will learn in the next chapter.

Algorithm complexity (Algorithm Complexity)

In essence, the algorithm complexity is an estimation function (which may be inaccurate) the number of operations that the algorithm needs to perform (thereby making it easier to deduce the algorithm's execution time) for the data set Input (Input) has size n. Where, n can be the number of elements of the array in case of

sorting or searching problem, or it can be the number of numbers in the problem of primes, .

Suppose X is an algorithm and n is the size of the input data. The time and amount of memory used by algorithm X are the two main factors that determine the effectiveness of algorithm X:

Time factor : Time is evaluated by calculating the number of main operations (such as comparisons in sorting algorithms).

Memory factor : The amount of memory is evaluated by calculating the maximum amount of memory the algorithm needs to use.

The complexity of an algorithm (a function f (n)) provides the relationship between runtime and / or the amount of memory that needs to be used by the algorithm.

Memory complexity (Space complexity) in algorithm analysis

The memory factor of an algorithm represents the amount of memory an algorithm needs to use in the algorithm's life cycle. The amount of memory (assumed to be **S (P))** that an algorithm needs to use is the sum of the following two components:

The fixed part (assuming called **C)** is the amount of memory needed to store certain data and variables (this part is independent of the size of the problem). For example: simple variables and constants, .

The variable part (assuming called **SP (I))** is the amount of memory needed by variables, whose size depends on the size of the problem. For example: dynamic memory allocation, recursive recursive memory, .

From above, we will have **S (P) = C + SP (I)** . You follow the following simple example:

Algorithm: find the sum of two numbers A and B
Step 1 - Start
Step 2 - C ? A + B + 10
Step 3 - Finish

Here we have three variables A, B and C and a constant. Therefore: **S (P) = 1 + 3** .

Now, the amount of memory will depend on the data type of the given variables and constants and will equal the product of the total with the memory for the corresponding data type.

Time complexity (Time Complexity) in algorithm analysis

The time factor of an algorithm represents the amount of run time required from the beginning until the end of an algorithm. The required time can be represented by a function **T (n)** , where **T (n)** can be evaluated as the number of steps.

For example, the addition of two n-bit integers will have n steps. Therefore, the total calculation time will be **T (n) = c * n** , where c is the time to perform two-bit addition. Here, we consider the function **T (n) to** increase linearly when the input data size increases.

According to Tutorialspoint

Previous article: What is SQL

Next lesson: Asymptotic analysis in Data Structure and Algorithm