

Some Topics in Elementary Computer Science

Nguyễn Quân Bá Hồng*

Ngày 17 tháng 5 năm 2023

Tóm tắt nội dung

Mục lục

1 Algorithm & Analysis of Algorithm – Thuật Toán & Phân Tích Thuật Toán	1
1.1 Algorithm – Thuật Toán	1
1.2 Analysis of Algorithm – Phân Tích Thuật Toán	1
2 Competitive Programming CP	3
3 Number Theory	3
Tài liệu	4

1 Algorithm & Analysis of Algorithm – Thuật Toán & Phân Tích Thuật Toán

See, e.g, [Dàm+09, Chuyên đề 1, pp. 5–12].

1.1 Algorithm – Thuật Toán

Định nghĩa 1 (Thuật toán). Thuật toán là dãy hữu hạn các bước, mỗi bước mô tả chính xác các phép toán hoặc hành động cần thực hiện, để giải quyết 1 vấn đề.

Definition 1 (Algorithm). “In mathematics & computer science, an algorithm is a finite sequence of *rigorous* instructions, typically used to solve a class of specific *computational problems* or to perform a *computation*.”

Algorithms are used as specifications for performing *calculations* & *data processing*. More advanced algorithms can use *conditionals* to divert the code execution through various routes (referred to as *automated decision-making*) & deduce valid *inferences* (referred to as *automated reasoning*), achieving *automation* eventually. Using human characteristics as descriptors of machines in metaphorical ways as already practiced by *Alan Turing* with terms e.g., “memory”, “search”, & “stimulus”.

In contrast, a *heuristic* is an approach to problem solving that may not be fully specified or may not guarantee correct or optimal results, especially in problem domains where there is no well-defined correct or optimal result.

As an *effective method*, an algorithm can be expressed within a finite amount of space & time, & in a well-defined formal language for calculating a function. Starting from an initial state & initial input (perhaps *empty*), the instructions describe a computation that, when *executed*, proceeds through a finite number of well-defined successive states, eventually producing “output” & terminating at a final ending state. The transition from 1 state to the next is not necessarily *deterministic*; some algorithms, known as *randomized algorithms*, incorporate random input.” – [Wikipedia/algorithm](https://en.wikipedia.org/wiki/Algorithm)

1.2 Analysis of Algorithm – Phân Tích Thuật Toán

Definition 2. “In computer science, the analysis of algorithms is the process of finding the *computational complexity* of algorithms – the amount of time, storage, or other resources needed to execute them.

Usually, this involves determining a function that relates the size of an algorithm’s input to the number of steps it takes (its *time complexity*) or the number of storage locations it uses (its *space complexity*). An algorithm is said to be efficient when this function’s values are small, or grow slowly compared to a growth in the size of the input. Different inputs of the same size may cause the algorithm to have different behavior, so *best*, *worst*, & *average case* descriptions might all be of practical interest. When not otherwise specified, the function describing the performance of an algorithm is usually an *upper bound*, determined from the worst case inputs to the algorithm.

*Independent Researcher, Ben Tre City, Vietnam
e-mail: nguyenquanbahong@gmail.com; website: <https://nqbh.github.io>.

The term “analysis of algorithms” was coined by **Donald Knuth**. Algorithm analysis is an important part of a broader **computational complexity theory**, which provides theoretical estimates for the resources needed by any algorithm which solves a given **computational problem**. These estimates provide an insight into reasonable directions of search for **efficient algorithms**.

In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input. **Big O notation**, **Big-omega notation** & **Big-theta notation** are used to this end. E.g., **binary search** is said to run in a number of steps proportional to the logarithm of the size n of the sorted list being searched, or in $O(\log n)$, colloquially “in **logarithmic time**”. Usually **asymptotic** estimates are used because different **implementations** of the same algorithm may differ in efficiency. However the efficiencies of any 2 “reasonable” implementations of a given algorithm are related by a constant multiplicative factor called a *hidden constant*.

Exact (not asymptotic) measures of efficiency can sometimes be computed but they usually require certain assumptions concerning the particular implementation of the algorithm, called **model of computation**. A model of computation may be defined in terms of an **abstract computer**, e.g., **Turing machine**, &/or by postulating that certain operations are executed in unit time. E.g., if the sorted list to which we apply binary search has n elements, & we can guarantee that each lookup of an element in the list can be done in unit time, then at most $\log_2 n + 1$ time units are needed to return an answer.” – [Wikipedia/analysis of algorithms](#)

Bài toán 1 ([**Dàm+09**], Ví dụ 1, p. 9). *Phân tích thời gian thực hiện của chương trình sau:*

```
var i, j, n, s1, s2: longint;
begin
  readln(n);
  s1 := 0;
  for i := 1 to n do
    s1 := s1 + i;
  s2 := 0;
  for j := 1 to n do
    s2 := s2 + j*j;
  writeln('1 + 2 + ... + ', n, '= ', s1);
  writeln('1^2 + 2^2 + ... + ', n, '^2 = ', s2);
end.
```

Bài toán 2 ([**Dàm+09**], Ví dụ 2, p. 10). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
c := 0;
for i := 1 to 2*n do
  c := c + 1;
for i := 1 to n do
  for j := 1 to n do
    c := c + 1;
```

Bài toán 3 ([**Dàm+09**], Ví dụ 3, p. 10). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
for i := 1 to n do
  for j := 1 to i do
    c := c + 1;
```

Bài toán 4 ([**Dàm+09**], 1.1., p. 11). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
for i := 1 to n do
  if i mod 2 = 0 then c := c + 1;
```

Bài toán 5 ([**Dàm+09**], 1.2., p. 11). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
for i := 1 to n do
  if i mod 2 = 0 then c1 := c1 + 1
  else c2 := c2 + 1;
```

Bài toán 6 ([**Dàm+09**], 1.3., p. 11). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
for i := 1 to n do
  if i mod 2 = 0 then
    for j := 1 to n do c := c + 1
```

Bài toán 7 ([**Dàm+09**], 1.4., p. 11). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```
a := 0;
b := 0;
c := 0;
for i := 1 to n do
```

```

begin
  a := a + 1;
  b := b + i;
  c := c + i*i;
end;

```

Bài toán 8 ([Đàm+09], 1.5., p. 11). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```

i := n;
d := 0;
while i > 0 do
begin
  i := i - 1;
  d := d + i;
end;

```

Bài toán 9 ([Đàm+09], 1.6., p. 12). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```

i := 0;
d := 0;
repeat
  i := i + 1;
  if i mod 3 = 0 then d := d + i;
until i > n;

```

Bài toán 10 ([Đàm+09], 1.7., p. 12). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```

d := 0;
for i := 1 to n - 1 do
  for j := i + 1 to n do d := d + 1;

```

Bài toán 11 ([Đàm+09], 1.8., p. 12). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```

d := 0;
for i := 1 to n - 2 do
  for j := i + 1 to n - 1 do
    for k := j + 1 to n do d := d + 1;

```

Bài toán 12 ([Đàm+09], 1.9., p. 12). *Phân tích thời gian thực hiện của đoạn chương trình sau:*

```

d := 0;
while n > 0 do
begin
  n := n div 2;
  d := d + 1;
end;

```

Bài toán 13 ([Đàm+09], 1.10., p. 12). *Cho 1 dãy số gồm $n \in \mathbb{N}^*$ số nguyên dương, xác định xem có tồn tại 1 dãy con liên tiếp có tổng bằng k hay không? (a) Đưa ra thuật toán có thời gian thực hiện $O(n^3)$. (b) Đưa ra thuật toán có thời gian thực hiện $O(n^2)$. (c) Đưa ra thuật toán có thời gian thực hiện $O(n)$.*

2 Competitive Programming CP

3 Number Theory

Definition 3. *An integer $a \in \mathbb{Z}$ is called a factor or a divisor of an integer $b \in \mathbb{Z}$ if a divides b (i.e., b is divisible by a). If a is a factor of b , we write $a \mid b$, or $b : a$, & otherwise we write $a \nmid b$, or $b \nmid a$.*

Bài toán 14 (Factor/Divisor – Ước số). *Với $n \in \mathbb{Z}$ được nhập từ bàn phím, viết chương trình Pascal, Python, C/C++ xuất ra tất cả: (a) các ước nguyên dương của n . (b) các ước nguyên của n .*

Bài toán 15 (Prime factorization – Phân tích ra thừa số nguyên tố). *Với $n \in \mathbb{Z}$ được nhập từ bàn phím, viết chương trình Pascal, Python, C/C++ xuất ra phân tích ra thừa số nguyên tố của n . E.g., với $n = 72$, xuất ra $72 = 2^3 \cdot 3^2$, với $n = 12$, xuất ra $12 = 2^2 \cdot 3$.*

Let $\tau(n)$ denote the number of (positive) divisors of an integer $n \in \mathbb{Z}$. E.g., $\tau(12) = 6$ since the divisors of 12 are 1, 2, 3, 4, 6, & 12. To calculate the value of $\tau(n)$, we can use the following formula:

$$n = \prod_{i=1}^k p_i^{\alpha_i} = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} \Rightarrow \tau(n) = \prod_{i=1}^k (\alpha_i + 1) = (\alpha_1 + 1)(\alpha_2 + 1) \cdots (\alpha_k + 1), \forall n \in \mathbb{Z},$$

because for each prime p_i , there are $\alpha_i + 1$ ways to choose how many times it appears in the factor.

Example 1. $12 = 2^2 \cdot 3 \Rightarrow \tau(12) = (2 + 1)(1 + 1) = 3 \cdot 2 = 6$.

Bài toán 16 ($\tau(n)$). Với $n \in \mathbb{Z}$ được nhập từ bàn phím, viết chương trình Pascal, Python, C/C++ xuất ra giá trị của hàm $\tau(n)$ số ước số của n .

Let $\sigma(n)$ denote the sum of divisors of an integer $n \in \mathbb{Z}$.

Example 2. $\mathcal{U}(12) \cap \mathbb{N} = \{1, 2, 3, 4, 6, 12\} \Rightarrow \sigma(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28$.

To calculate the value of $\sigma(n)$, we can use the following formula:

$$\begin{aligned} n = \prod_{i=1}^k p_i^{\alpha_i} = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} \Rightarrow \sigma(n) &= \prod_{i=1}^k \sum_{j=0}^{\alpha_i} p_i^j = \prod_{i=1}^k (1 + p_i + p_i^2 + \cdots + p_i^{\alpha_i}) = \prod_{i=1}^k \frac{p_i^{\alpha_i+1} - 1}{p_i - 1} \\ &= \frac{p_1^{\alpha_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{\alpha_2+1} - 1}{p_2 - 1} \cdots \frac{p_k^{\alpha_k+1} - 1}{p_k - 1}, \forall n \in \mathbb{Z}, \end{aligned}$$

where the latter form is based on the *geometric progression formula*.

Example 3. $12 = 2^2 \cdot 3 \Rightarrow \sigma(12) = \frac{2^3-1}{2-1} \cdot \frac{3^2-1}{3-1} = 28$.

Bài toán 17 ($\sigma(n)$). Với $n \in \mathbb{Z}$ được nhập từ bàn phím, viết chương trình Pascal, Python, C/C++ xuất ra giá trị của hàm $\sigma(n)$ tổng tất cả các ước số của n .

Tài liệu

[Đàm+09] Hồ Sĩ Đàm, Đỗ Đức Đông, Lê Minh Hoàng, and Nguyễn Thanh Hùng. *Tài Liệu Giáo Khoa Chuyên Tin, quyển 1*. Nhà Xuất Bản Giáo Dục Việt Nam, 2009, p. 219.