Chuyên đề 1

THUẬT TOÁN VÀ PHÂN TÍCH THUẬT TOÁN

1. Thuật toán

Thuật toán là một trong những khái niệm quan trọng nhất trong tin học. Thuật ngữ thuật toán xuất phát từ nhà khoa học Arập Abu Ja'far Mohammed ibn Musa al Khowarizmi. Ta có thể hiểu *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 một vấn đề.* Để hiểu đầy đủ ý nghĩa của khái niệm thuật toán chúng ta xem xét 5 đặc trưng sau của thuật toán:

- Đầu vào (Input): Thuật toán nhận dữ liệu vào từ một tập nào đó.
- Đầu ra (Output): Với mỗi tập các dữ liệu đầu vào, thuật toán đưa ra các dữ liệu tương ứng với lời giải của bài toán.
- Chính xác: Các bước của thuật toán được mô tả chính xác.
- Hữu hạn: Thuật toán cần phải đưa được đầu ra sau một số hữu hạn (có thể rất lớn) bước với mọi đầu vào.
- Đơn trị: Các kết quả trung gian của từng bước thực hiện thuật toán được xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và các kết quả của các bước trước.
- Tổng quát: Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho.

Để biểu diễn thuật toán có thể biểu diễn bằng danh sách các bước, các bước được diễn đạt bằng ngôn ngữ thông thường và các kí hiệu toán học; hoặc có thể biểu diễn thuật toán bằng sơ đồ khối. Tuy nhiên, để đảm bảo tính xác định của thuật toán, thuật toán cần được viết bằng các ngôn ngữ lập trình. Một chương trình là sự biểu diễn của một thuật toán trong ngôn ngữ lập trình đã chọn. Trong tài liệu này, chúng ta sử dụng ngôn ngữ tựa Pascal để trình bày các thuật toán. Nói là tựa Pascal, bởi vì nhiều trường hợp, để cho ngắn gọn, chúng ta không hoàn toàn tuân

theo quy định của Pascal. Ngôn ngữ Pascal là ngôn ngữ đơn giản, khoa học, được giảng dạy trong nhà trường phổ thông.

Ví dụ: Thuật toán kiểm tra tính nguyên tố của một số nguyên dương n $(n \geq 2)$, viết trên ngôn ngữ lập trình Pascal.

```
function is_prime(n):boolean;
begin
   for k:=2 to n-1 do
    if (n mod k=0) then exit(false);
   exit(true);
end;
```

2. Phân tích thuật toán

2.1. Tính hiệu quả của thuật toán

Khi giải một bài toán, chúng ta cần chọn trong số các thuật toán một thuật toán mà chúng ta cho là "tốt" nhất. Vậy dựa trên cơ sở nào để đánh giá thuật toán này "tốt" hơn thuật toán kia? Thông thường ta dựa trên hai tiểu chuẩn sau:

1. Thuật toán đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình).
2. Thuật toán hiệu quả: Chúng ta thường đặc biệt quan tâm đến thời gian thực hiện của thuật toán (gọi là độ phức tạp tính toán), bên cạnh đó chúng ta cũng quan tâm tới dung lượng không gian nhớ cần thiết để lưu giữ các dữ liệu vào, ra và các kết quả trung gian trong quá trình tính toán.

Khi viết chương trình chỉ để sử dụng một số ít lần thì tiêu chuẩn (1) là quan trọng, nhưng nếu viết chương trình để sử dụng nhiều lần, cho nhiều người sử dụng thì tiêu chuẩn (2) lại quan trọng hơn. Trong trường hợp này, dù thuật toán có thể phải cài đặt phức tạp, nhưng ta vẫn sẽ lựa chọn để nhận được chương trình chạy nhanh hơn, hiệu quả hơn.

2.2. Tại sao cần thuật toán có tính hiệu quả?

Kĩ thuật máy tính tiến bộ rất nhanh, ngày nay các máy tính lớn có thể đạt tốc độ tính toán hàng nghìn tỉ phép tính trong một giây. Vậy có cần phải tìm thuật toán hiệu quả hay không? Chúng ta **xem lại ví dụ** bài toán kiểm tra tính nguyên tố của một số nguyên dương n $(n \geq 2)$.

```
function is_prime(n):boolean;
 begin
```

```
    for k:=2 to n-1 do
     if (n mod k=0) then exit(false);
    exit(true);
 end;
```

Dễ dàng nhận thấy rằng, nếu n là một số nguyên tố chúng ta phải mất $n-2$ phép toán mod. Giả sử một siêu máy tính có thể tính được trăm nghìn tỉ (10^{14}) phép mod trong một giây, như vậy để kiểm tra một số khoảng 25 chữ số mất khoảng $\frac{10^{25}}{10^{14}\times 60\times 60\times 24\times 365}\sim 3170$ năm. Trong khi đó, nếu ta có nhận xét việc thử k từ 2 đến $n-1$ là không cần thiết mà chỉ cần thử k từ 2 đến $\sqrt{n}$, ta có:

```
function is_prime(n):boolean;
 begin
    for k:=2 to trunc(sqrt(n)) do
     if (n mod k=0) then exit(false);
    exit(true);
 end;
{hàm sqrt(n) là hàm tính √n, trunc(x) là hàm làm tròn x }
```

Như vậy để kiểm tra một số khoảng 25 chữ số mất khoảng $\frac{\sqrt{10^{25}}}{10^{14}}\sim 0.03$ giây!

2.3. Đánh giá thời gian thực hiện thuật toán

Có hai cách tiếp cận để đánh giá thời gian thực hiện của một thuật toán. Cách thứ nhất bằng thực nghiệm, chúng ta viết chương trình và cho chạy chương trình với các dữ liệu vào khác nhau trên một máy tính. Cách thứ hai bằng phương pháp lí thuyết, chúng ta coi thời gian thực hiện thuật toán như hàm số của cỡ dữ liệu vào (cỡ của dữ liệu vào là một tham số đặc trưng cho dữ liệu vào, nó có ảnh hưởng quyết định đến thời gian thực hiện chương trình. Ví dụ đối với bài toán kiểm tra số nguyên tố thì cỡ của dữ liệu vào là số n cần kiểm tra; hay với bài toán sắp xếp dãy số, cỡ của dữ liệu vào là số phần tử của dãy). Thông thường cỡ của dữ liệu vào là một số nguyên dương n, ta sử dụng hàm số $T(n)$ trong đó n là cỡ của dữ liệu vào để biểu diễn thời thực hiện của một thuật toán.

Xét ví dụ bài toán kiểm tra tính nguyên tố của một số nguyên dương n (cỡ dữ liệu vào là n), nếu n là một số chẵn $(n>2)$ thì chỉ cần một lần thử chia 2 để kết luận n không phải là số nguyên tố. Nếu n $(n>3)$ không chia hết cho 2 nhưng lại chia hết cho 3 thì cần 2 lần thử (chia 2 và chia 3) để kết luận n không nguyên tố. Còn nếu n là một số nguyên tố thì thuật toán phải thực hiện nhiều lần thử nhất.

Trong tài liệu này, chúng ta hiểu hàm số $T(n)$ là thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào cỡ n.

Sử dụng kí hiệu toán học ô lớn để mô tả độ lớn của hàm $T(n)$. Giả sử n là một số nguyên dương, $T(n)$ và $f(n)$ là hai hàm thực không âm. Ta viết $T(n) = O(f(n))$ nếu và chỉ nếu tồn tại các hằng số dương c và n_0, sao cho $T(n) \leq c \times f(n)$, với mọi $n \geq n_0$.

Nếu một thuật toán có thời gian thực hiện $T(n) = O(f(n))$ chúng ta nói rằng thuật toán có thời gian thực hiện cấp $f(n)$.

Ví dụ: Giả sử $T(n) = n^2 + 2n$, ta có $n^2 + 2n \leq n^2 + 2n^2 = 3n^2$ với mọi $n \geq 1$

Vậy $T(n) = O(n^2)$, trong trường hợp này ta nói thuật toán có thời gian thực hiện cấp n^2.

2.4. Các quy tắc đánh giá thời gian thực hiện thuật toán

Để đánh giá thời gian thực hiện thuật toán được trình bày bằng ngôn ngữ tựa Pascal, ta cần biết cách đánh giá thời gian thực hiện các câu lệnh của Pascal. Trước tiên, chúng ta hãy xem xét các câu lệnh chính trong Pascal. Các câu lệnh trong Pascal được định nghĩa đệ quy như sau:

1. Các phép gán, đọc, viết là các câu lệnh (được gọi là lệnh đơn).

2. Nếu S_1, S_2, ..., S_m là câu lệnh thì

```
Begin S1; S2; …; Sm; End;
```

là câu lệnh (được gọi là lệnh hợp thành hay khối lệnh).

3. Nếu S_1 và S_2 là các câu lệnh và E là biểu thức lôgic thì

```
If E then S1 else S2;
```

là câu lệnh (được gọi là lệnh rẽ nhánh hay lệnh If).

4. Nếu S là câu lệnh và E là biểu thức lôgic thì

```
While E do S;
```

là câu lệnh (được gọi là lệnh lặp điều kiện trước hay lệnh While).

5. Nếu S_1, S_2,…,S_m là các câu lệnh và E là biểu thức lôgic thì

```
Repeat
S1; S2; …; Sm;
Until E;
```

là câu lệnh (được gọi là lệnh lặp điều kiện sau hay lệnh Repeat)

6. Nếu S là lệnh, E_1 và E_2 là các biểu thức cùng một kiểu thứ tự đếm được thì

```
For i:=E1 to E2 do S;
```

là câu lệnh (được gọi là lệnh lặp với số lần xác định hay lệnh For).

Để đánh giá, chúng ta phân tích chương trình xuất phát từ các lệnh đơn, rồi đánh giá các lệnh phức tạp hơn, cuối cùng đánh giá được thời gian thực hiện của chương trình, cụ thể:

1. Thời gian thực hiện các lệnh đơn: gán, đọc, viết là $O(1)$

2. Lệnh hợp thành: giả sử thời gian thực hiện của S_1, S_2,…,S_m tương ứng là $O(f_1(n)), O(f_2(n)), \ldots, O(f_m(n))$. Khi đó thời gian thực hiện của lệnh hợp thành là: $O(max(f_1(n), f_2(n), \ldots, f_m(n)))$.

3. Lệnh If: giả sử thời gian thực hiện của S_1, S_2 tương ứng là $O(f_1(n)), O(f_2(n)))$. Khi đó thời gian thực hiện của lệnh If là: $O(max(f_1(n), f_2(n)))$.

4. Lệnh lặp While: giả sử thời gian thực hiện lệnh S (thân của lệnh While) là $O(f(n))$ và $g(n)$ là số lần lặp tối đa thực hiện lệnh S. Khi đó thời gian thực hiện lệnh While là $O(f(n)g(n))$.

5. Lệnh lặp Repeat: giả sử thời gian thực hiện khối lệnh

```
Begin S1; S2;…; Sm; End;
```

là $O(f(n))$ và $g(n)$ là số lần lặp tối đa. Khi đó thời gian thực hiện lệnh Repeat là $O(f(n)g(n))$.

6. Lệnh lặp For: giả sử thời gian thực hiện lệnh S là $O(f(n))$ và $g(n)$ là số lần lặp tối đa. Khi đó thời gian thực hiện lệnh For là $O(f(n)g(n))$.

2.5. Một số ví dụ

Ví dụ 1: Phân tích thời gian thực hiện của chương trình sau:

```
var i, j, n     :longint;
    s1, s2      :longint;
BEGIN
{1}    readln(n);
{2}    s1:=0;
{3}    for i:=1 to n do
{4}      s1:=s1 + i;
{5}    s2:=0;
```

```
{6}     for j:=1 to n do
{7}       s2:=s2 + j*j;
{8}     writeln('1+2+..+',n,'=',s1);
{9}     writeln('1^2+2^2+..+',n,'^2=',s2);
END.
```

Thời gian thực hiện chương trình phụ thuộc vào số $\boldsymbol{n}$.

Các lệnh {1}, {2}, {4}, {5}, {7}, {8}, {9} có thời gian thực hiện là $O(1)$.

Lệnh lặp For {3} có số lần lặp là n, như vậy lệnh {3} có thời gian thực hiện là $O(n)$. Tương tự lệnh lặp For {6} cũng có thời gian thực hiện là $O(n)$.

Vậy thời gian thực hiện của chương trình là:

$$\max(O(1), O(1), O(n), O(1), O(n), O(1), O(1)) = O(n)$$

Ví dụ 2: Phân tích thời gian thực hiện của đoạn chương trình sau:

```
{1}     c:=0;
{2}     for i:=1 to 2*n do
{3}       c:=c+1;
{4}     for i:=1 to n do
{5}       for j:=1 to n do
{6}       c:=c+1;
```

Thời gian thực hiện chương trình phụ thuộc vào số n.

Các lệnh {1}, {3}, {6} có thời gian thực hiện là $O(1)$.

Lệnh lặp For {2} có số lần lặp là $2n$, như vậy lệnh {2} có thời gian thực hiện là $O(n)$.

Lệnh lặp For {5} có số lần lặp là n, như vậy lệnh {5} có thời gian thực hiện là $O(n)$. Lệnh lặp For {4} có số lần lặp là n, như vậy lệnh {4} có thời gian thực hiện là $O(n^2)$.

Vậy thời gian thực hiện của đoạn chương trình trên là:

$$\max(O(1), O(n), O(n^2)) = O(n^2)$$

Ví dụ 3: Phân tích thời gian thực hiện của đoạn chương trình sau:

```
{1}     for i:=1 to n do
{2}       for j:=1 to i do
{3}       c:=c+1;
```

Thời gian thực hiện chương trình phụ thuộc vào số n.

Các lệnh {3} có thời gian thực hiện là $O(1)$.

Khi i = 1, j chạy từ 1 đến 1 → lệnh lặp For {2} lặp 1 lần

Khi i = 2, j chạy từ 1 đến 2 → lệnh lặp For {2} lặp 2 lần

…

Khi i = n, j chạy từ 1 đến n → lệnh lặp For {2} lặp n lần

Như vậy lệnh {3} được lặp: $1 + 2 + .. + n = \frac{n(n+1)}{2}$ lần, do đó lệnh {1} có thời gian thực hiện là $O(n^2)$

Vậy thời gian thực hiện của đoạn chương trình trên là: $O(n^2)$

Bài tập

1.1. 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;
```

1.2. 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;
```

1.3. 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
```

1.4. 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;
```

1.5. 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;
```

1.6. 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;
```

1.7. 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;
```

1.8. 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;
```

1.9. 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;
```

1.10. Cho một dãy số gồm n số nguyên dương, xác định xem có tồn tại một 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 $O(n)$.