BÀI 5

ĐÁNH GIÁ THUẬT TOÁN

Học xong bài này, em sẽ:

- Trình bày được sơ lược khái niệm độ phức tạp thời gian của thuật toán. Nêu được ví dụ minh hoạ.
- Biết được kí pháp O lớn và các bậc độ phức tạp thời gian.

Theo em, một thuật toán như thế nào thì được xem là chạy nhanh/chạy chậm?

1 Các khái niệm cơ bản

Một thuật toán ngắn gọn, dễ hiểu, dễ lập trình cũng là những điểm tốt, nhưng trong Tin học, các thuật toán được đánh giá và so sánh dựa trên một tiêu chuẩn quan trọng hơn – Đó là tính hiệu quả. Thuật toán được coi là hiệu quả hơn nếu thời gian thực hiện chương trình và lượng bộ nhớ mà máy tính cần dùng là ít hơn.

Ước lượng thời gian thực thi chương trình và hiệu quả thời gian của thuật toán

Python có lệnh `time()` cho phép bấm giờ tính thời gian chạy thực thi chương trình. Nhiều ngôn ngữ lập trình khác cũng có lệnh tương tự. Nhưng cách tính giờ chạy thực thi chương trình cụ thể không áp dụng được khi muốn so sánh hiệu quả để lựa chọn thuật toán vì nó dẫn đến các vấn đề sau đây:

– Phải lập trình và chạy thử chương trình của tất cả các thuật toán cần so sánh.

– Thời gian đo được phụ thuộc vào nhiều yếu tố không liên quan tới thuật toán bởi chương trình chạy nhanh hay chậm phụ thuộc vào phần cứng máy tính, ngôn ngữ lập trình, chương trình dịch và bản thân kĩ năng lập trình của người viết.

– Không khả thi nếu muốn chọn cách lập nhiều chương trình khác nhau (lập trình viên, ngôn ngữ lập trình) rồi tính thời gian thực thi trung bình.

Kích thước đầu vào

Thời gian chạy chương trình phụ thuộc kích thước dữ liệu đầu vào. Ví dụ, chương trình tìm kiếm một số x trong dãy số sẽ kết thúc sớm hơn khi dãy chỉ gồm 10 số và sẽ lâu hơn khi dãy gồm 10 000 số. Thông thường, kích thước dữ liệu đầu vào được đại diện bằng một số tự nhiên n. Ví dụ, trong trường hợp bài toán sắp xếp dãy số hay tìm kiếm một số x trong dãy, thì kích thước đầu vào n là độ dài dãy số.

2 Độ phức tạp thời gian của thuật toán

Thời gian chạy một chương trình với đầu vào có kích thước n sẽ là một hàm số $T(n)$ của biến số n. Độ phức tạp thời gian là một khái niệm trong khoa học máy tính, là kết quả ước lượng thời gian thực hiện các chương trình cài đặt thuật toán để xử lí một lượng dữ liệu đầu vào có độ lớn n. Ước lượng này thể hiện *số phép toán* cần thiết để thực hiện thuật toán khi đã biết dữ liệu đầu vào có kích thước n. Khó tính đếm chính xác con số này vì nhiều lí do:

– Bộ xử lí thực hiện các phép toán bit, khó có thể xác định tương ứng số các phép toán bit với mỗi phép toán mà chúng ta vẫn biết như các phép toán số học (cộng, trừ, nhân, chia), các phép so sánh,...

– Ngay cả khi tính đếm số phép toán theo nghĩa thông thường với con người thì thế nào là một phép toán cũng không dễ thống nhất. Ví dụ: Phép khai căn, phép luỹ thừa,... là một hay nhiều phép toán số học.

Phép toán sơ cấp

Khi phân tích một thuật toán để tính đếm số lượng các phép toán cần thực hiện, người ta phân biệt các phép toán sơ cấp với phần còn lại được coi là không sơ cấp. Một phép toán sơ cấp là phép toán có thời gian thực hiện không lớn hơn một hằng số nào đó, không phụ thuộc n (n là kích thước dữ liệu đầu vào).

Ví dụ, những trường hợp dưới đây được coi là phép toán sơ cấp:

– Phép toán số học, phép so sánh,... với các toán hạng là giá trị cụ thể.

– Các hàm toán học với đầu vào là giá trị cụ thể không phụ thuộc n.

Khái niệm phép toán sơ cấp sẽ rõ dần qua một số ví dụ minh hoạ và thực hành.

Phép lặp (mô tả bằng cấu trúc lặp), *phép lựa chọn* (mô tả bằng cấu trúc rẽ nhánh) không phải là phép toán sơ cấp.

3 Ví dụ về độ phức tạp thời gian hằng số và độ phức tạp thời gian tuyến tính

Cho bài toán tính tổng dãy số: $S = 1 + 2 + ... + n$. Hãy cho biết cách giải nào tốt hơn trong hai cách giải sau đây:

Cách thứ nhất: Tính cộng dồn dần từng số.

Cách thứ hai: Vì dãy số là cấp số cộng nên có thể dùng công thức tính tổng cấp số cộng $S = \frac{n(n+1)}{2}$.

Độ phức tạp thời gian hằng số

Thuật toán có *độ phức tạp thời gian hằng số* khi mà số phép toán cần thực hiện không phụ thuộc kích thước n của dữ liệu đầu vào.

Cách giải thứ hai cho bài toán trong Hoạt động trên là một thuật toán có độ phức tạp thời gian hằng số vì chỉ cần 3 phép toán để tính tổng S, $T(n) = 3$. Chuyện kể rằng, đây là cách giải mà khi còn là học sinh phổ thông, nhà toán học thiên tài người Đức – Friedrich Gauss đã nêu ra khi làm bài tập trong giờ học môn Toán.

Độ phức tạp thời gian tuyến tính

Thuật toán có *độ phức tạp thời gian tuyến tính* nếu số phép toán cần thực hiện là hàm tuyến tính của n (n là kích thước dữ liệu đầu vào). Cách giải thứ nhất trong hoạt động trên là một thuật toán có độ phức tạp thời gian tuyến tính vì $T(n) = n - 1$.

4 Kí pháp và các bậc độ phức tạp thời gian

Cách ước lượng làm già thêm

Số phép toán cần thiết để thực hiện thuật toán không chỉ phụ thuộc kích thước n của dữ liệu đầu vào mà còn phụ thuộc vào việc ta may mắn gặp trường hợp dễ, ít việc phải làm hay không may gặp trường hợp khó, nhiều việc phải làm hơn.

Xét thuật toán tìm số lớn nhất trong dãy số:

– Đầu tiên, tạm gán $max = a_0$; đọc giá trị tiếp theo, so sánh với max và gán lại nếu cần.

– Khi phần tử đầu dãy a_0 có giá trị lớn nhất thì số lần phải gán lại giá trị $max = a_i$ là bằng 0. Số phép toán là ít nhất.

– Khi dãy số ban đầu là dãy tăng chặt, mọi số đều khác nhau, thì số lần phải gán $max = a_i$ là bằng n. Số phép toán là nhiều nhất.

Một cách tổng quát, có thể xét ba trường hợp: trường hợp thuận lợi nhất (số phép toán cần thực hiện ít nhất); trường hợp bất lợi nhất (số phép toán cần thực hiện nhiều nhất) và trường hợp ngẫu nhiên (số phép toán cần thực hiện ở mức trung bình).

Nói chung, ta muốn có một *ước lượng trung bình* cho tất cả các trường hợp ngẫu nhiên xảy ra. Tuy nhiên, không dễ tìm được ước lượng trung bình này. Người ta chọn cách dễ làm hơn, đó là *ước lượng làm già thêm*. Cách ước lượng đảm bảo rằng trong thực tế sẽ không có trường hợp nào vượt quá ước lượng đã đưa ra.

Kí pháp O lớn

Theo định nghĩa, nếu số phép toán sơ cấp cần thực hiện không vượt quá một hằng số C, không phụ thuộc n thì thuật toán có độ phức tạp thời gian là hằng số. Kí hiệu $T(n) = O(1)$.

Nếu số phép toán sơ cấp cần thực hiện không vượt quá một hàm tuyến tính của n, $T(n) \leq C_1 n + C_2$ (với C_1, C_2 là hằng số) thì độ phức tạp thời gian của thuật toán là tuyến tính. Viết ngắn gọn $T(n) = O(n)$ nghĩa là độ phức tạp thời gian của thuật toán là tuyến tính. *Bảng 1* dưới đây là một số kí hiệu O lớn về thời gian thực hiện thuật toán thường gặp:

Bảng 1. Một số kí hiệu O lớn về thời gian thực hiện thuật toán

Kí hiệu O lớn	Tên gọi độ phức tạp thời gian thuật toán
$O(1)$	Hằng số
$O(\log_2 n)$	Logarit
$O(n)$	Tuyến tính (linear)
$O(n^2)$	Bậc hai (quadratic)
$O(C^n)$	Hàm mũ (exponential) ($C > 1$)
$O(n!)$	Giai thừa

Một số công thức liên quan:

Công thức 1:

Nếu $f_1(n) = O(g_1(n))$ và $f_2(n) = O(g_2(n))$ thì $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$. Công thức áp dụng cho hai cấu trúc điều khiển được thực hiện tuần tự.

Công thức 2:

Nếu $f_1(n) = O(g_1(n))$ và $f_2(n) = O(g_2(n))$ thì $f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n))$. Công thức áp dụng cho hai cấu trúc điều khiển lồng nhau.

5 Các quy tắc khi ước lượng thời gian thực hiện thuật toán

Quy tắc chung

Khi tính đếm số phép toán cần thực hiện, các quy tắc ước lượng cho phép bỏ bớt những phần có bậc lớn thấp hơn, chỉ giữ lại những phần có bậc lớn cao nhất và các hằng số nhân C đều coi là 1.

Mô tả thuật toán chỉ sử dụng ba cấu trúc: cấu trúc tuần tự, cấu trúc rẽ nhánh, cấu trúc lặp. Cấu trúc tuần tự thực hiện *dãy phép toán* nối tiếp nhau. Dãy phép toán của một cấu trúc tuần tự gồm các phép toán sơ cấp và có thể có các phép toán không sơ cấp như *phép lựa chọn* (thực hiện bằng cấu trúc rẽ nhánh) hay *phép lặp* (thực hiện bằng cấu trúc lặp).

Lồng bên trong các cấu trúc rẽ nhánh và cấu trúc lặp lại là các dãy phép toán tuần tự khác. Cần ước lượng số phép toán từ bên trong trở ra ngoài.

Lời gọi hàm

Hàm trong chương trình thực chất là một chương trình con, thực hiện một thuật toán cụ thể. Ước lượng độ phức tạp thời gian một lời gọi hàm chia làm hai trường hợp:

– Lời gọi các hàm toán học sơ cấp, các hàm thư viện,... với đầu vào là giá trị cụ thể không phụ thuộc n. Trường hợp này có độ phức tạp thời gian là $T(n) = O(1)$.

– Lời gọi hàm trong các trường hợp còn lại sẽ được ước lượng độ phức tạp như với một thuật toán.

Cấu trúc tuần tự và quy tắc lấy max

Cấu trúc tuần tự là một dãy gồm C phép toán; C là số xác định, không phụ thuộc n.

– Nếu tất cả C phép toán là sơ cấp, độ phức tạp thời gian là $T(n) = O(1)$.

– Trái lại, thời gian thực hiện bằng *ước lượng lớn nhất* trong số các ước lượng của các phép toán có trong dãy.

Cấu trúc rẽ nhánh và quy tắc lấy max

Máy tính thực thi một cấu trúc rẽ nhánh (hai nhánh hay nhiều nhánh) sẽ phải kiểm tra điều kiện và thực hiện một trong số các nhánh. Thường việc kiểm tra điều kiện là tính giá trị biểu thức logic gồm biểu thức số học và một phép so sánh, độ phức tạp thời gian là $T(n) = O(1)$.

Độ phức tạp thời gian của cấu trúc rẽ nhánh là độ phức tạp thời gian lớn nhất trong các độ phức tạp thời gian của các nhánh. Nếu trong biểu thức kiểm tra điều kiện có lời gọi hàm thì độ phức tạp thời gian của việc kiểm tra điều kiện trong cấu trúc rẽ nhánh có thể sẽ không còn là $O(1)$.

Ví dụ:

Câu lệnh mã giả	*Độ phức tạp thời gian*
if *dãy là cấp số cộng*:	$O(n)$
Tính tổng theo công thức Gauss	$O(1)$
else:	
Tính cộng dồn	$O(n)$

Hình 1. Mã giả và độ phức tạp thời gian để tính tổng dãy số

Nếu như chưa biết trước, việc kiểm tra dãy có phải là cấp số cộng hay không sẽ cần thời gian $O(n)$. Độ phức tạp thời gian của cấu trúc rẽ nhánh trong *Hình 1* là $O(n)$ trong mọi trường hợp.

Cấu trúc vòng lặp và quy tắc nhân

Máy tính thực hiện một cấu trúc vòng lặp sẽ phải kiểm tra điều kiện và thực hiện thân vòng lặp. Thân vòng lặp là một cấu trúc tuần tự các phép toán (có thể có phép lựa chọn hay phép lặp khác).

Thời gian thực hiện cấu trúc vòng lặp được tính bằng số lần lặp nhân với tổng thời gian kiểm tra điều kiện lặp và thời gian thực hiện thân vòng lặp.

Ví dụ: Trong *Hình 2*, độ phức tạp thời gian của thuật toán tìm giá trị cực tiểu một dãy số $a_1, a_2, a_3, ..., a_n$ là $O(n)$.

Câu lệnh mã giả	*Số phép toán sơ cấp*
i_min ← 1 # Tạm gán chỉ số đạt min *i_min*	1
for *k* **in** { *k* | 2 ≤ *k* ≤ *n* }:	*n* − 1
if *a* [*i_min*] > *a*[*k*]:	1
i_min ← *k* # Gán lại chỉ số đạt min	1
return *i_min*	

Hình 2. Mã giả và số phép toán sơ cấp để tìm cực tiểu của một dãy số

Em hãy cho ví dụ một lời gọi hàm được tính là phép toán sơ cấp và một lời gọi hàm không được tính là phép toán sơ cấp.

Câu 1. Xét bài toán sắp xếp dãy số. Hãy cho biết khi nào ta có trường hợp thuận lợi nhất, số phép toán cần làm là ít nhất?

Câu 2. Ước lượng số phép toán sơ cấp cần thực hiện để tìm số lớn nhất trong dãy số:

a) Đầu vào là dãy ngẫu nhiên.

b) Đầu vào là dãy giảm dần.

Câu 1. Tại sao không thể đánh giá thuật toán qua chương trình cài đặt thuật toán?

Câu 2. Khi nào thì áp dụng quy tắc lấy max?

Câu 3. Quy tắc nhân áp dụng cho cấu trúc vòng lặp là gì?

Tóm tắt bài học

- Độ phức tạp thời gian của thuật toán thể hiện tổng số phép toán sơ cấp cần thực hiện để hoàn thành thuật toán và được ước lượng xấp xỉ bằng một hàm số phụ thuộc *n* (*n* là kích thước dữ liệu đầu vào).
- Để ước lượng độ phức tạp thời gian của thuật toán, phải xác định đúng phép toán sơ cấp và tuân thủ các quy tắc tính số phép toán cho cấu trúc tuần tự, cấu trúc rẽ nhánh, cấu trúc lặp.