Bài 2.4: Độ phức tạp thuật toán

- ✓ Khái niệm
- ✓ Ví dụ thực tế
- ✓ Các trường hợp đo lường
- ✓ Cách xác định độ phức tạp
- ✓ Tầm quan trọng của thuật toán

Khái niệm

- Độ phức tạp của thuật toán là thước đo lượng thời gian và/hoặc không gian bộ nhớ mà một thuật toán cần để hoàn thành một bài toán với đầu vào kích thước nhất định.
- Độ phức tạp của thuật toán còn được gọi là thời gian chạy.
- Nếu một thuật toán phải mở rộng quy mô, nó sẽ tính toán kết quả trong một khoảng thời gian hữu hạn ngay cả với các giá trị lớn của n.
- Vì lý do này độ phức tạp được tính theo tiệm cận khi n tiến đến vô cùng.
- Phân tích độ phức tạp của thuật toán rất hữu ích khi so sánh các thuật toán hoặc tìm kiếm các cải tiến mới hiệu quả hơn.

Các yếu tố ảnh hưởng đến thời gian chạy của chương trình

- Máy tính đang được sử dụng, nền tảng phần cứng của nó.
- Các kiểu dữ liệu trừu tượng đượng sử dụng.
- Hiệu quả của trình biên dịch.
- Kĩ năng của lập trình viên.
- Độ phức tạp của thuật toán.
- Kích thước của dữ liệu đầu vào.
- Trong các yếu tố trên, độ phức tạp thuật toán và kích thước dữ liệu đầu vào là quan trọng nhất.

Ví dụ thực tế

- Giả sử bạn tìm kiếm một giá trị cụ thể trong danh sách rất dài chưa được sắp xếp. Bạn sẽ so sánh từng phần tử của danh sách đến khi nào gặp giá trị cần tìm hoặc kết thúc danh sách mà không có kết quả. Thời gian tìm kiếm tỉ lệ với kích thước danh sách. Lúc này độ phức tạp của thuật toán là tuyến tính.
- Nếu ta lấy phần tử bất kì trong mảng qua chỉ số, lấy phần tử đầu stack, thêm phần tử vào cuối queue thì độ phức tạp sẽ là hằng số.
- Nếu ta tra từ trong từ điển, tốc độ tìm kiếm sẽ nhanh hơn nhiều vì các từ đã được sắp xếp, bạn được quyết định dừng lại ở trang nào để tìm kiếm. Đây là ví dụ của độ phức tạp logarit.
- Khi tìm tất cả các phần tử trùng lặp trong một danh sách chưa sắp xếp, độ phức tạp thuật toán là bình phương.

Kí hiệu của độ phức tạp của thuật toán

- Kí hiệu big-O được sử dụng biểu thị cho độ phức tạp của thuật toán. Nó đưa ra giới hạn trên về độ phức tạp do đó biểu thị hiệu suất trong trường hợp xấu nhất của thuật toán. Viết là O(f).
- Với big-O, ta dễ dàng so sánh các thuật toán với nhau về hiệu quả xử lý vấn đề.
- Độ phức tạp hằng số được kí hiệu là O(1).
- Độ phức tạp tuyến tính kí hiệu là O(n).
- Độ phức tạp logarit kí hiệu là O(logn).
- Độ phức tạp loga-tuyến tính là O(nlogn).
- Độ phức tạp bình phương là O(n^2).
- Độ phức tạp số mũ là O(2^n).

Kí hiệu của độ phức tạp của thuật toán

- Độ phức tạp giai thừa là O(n!).
- Độ phức tạp càng nhỏ thì thuật toán càng tối ưu, tức càng tốt.
- Độ phức tạp càng lớn, thuật toán cảng rởm và tốn nhiều chi phí khi thực hiện.
- Thứ tự từ tốt đến tồi của các độ phức tạp thuật toán: O(1) < O(logn) < O($\sqrt{n}$) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n) < O(10^n) < O(n!).
- Khi phân tích độ phức tạp của thuật toán, chỉ xét số mũ cao nhất của biểu thức, bỏ qua các hằng số, hệ số và các số mũ nhỏ hơn.
- Ví dụ một thuật toán yêu cầu $2n^2$ + logn + 100 phép tính toán để hoàn thành, ta gọi độ phức tạp của thuật toán này là O(n^2).

Các trường hợp đo lường độ phức tạp thuật toán

- Khi phân tích một thuật toán, ta sẽ xem xét và dự đoán số lần chương trình cần thực hiện để giải quyết xong vấn đề nêu ra với bộ input đầu vào kích thước n.
- Thuật toán đó khi chạy có thể rơi vào một trong các trường hợp: tốt nhất, tệ nhất hoặc trung bình.
- Trường hợp tệ nhất: là thời gian tối đa cần thiết thuật toán cần để hoàn thành tác vụ.
- Trường hợp tốt nhất: ta gặp trường hợp cụ thể đặc biệt của bộ dữ liệu đầu vào, ví dụ sắp xếp tăng một mảng đã được sắp xếp tăng dần.
- Trường hợp trung bình: thường xảy ra nhất. Tuy nhiên để đo lường thì khó xác định vì yêu cầu nghiên cứu thực nghiệm và phân loại đầu vào kết hợp tính toán hiệu suất cao.

Cách xác định độ phức tạp thuật toán

- Trước hết phải phân loại thuật toán là lặp hay đệ quy.
- Nếu lặp: một hàm thực hiện lặp đi lặp lại đoạn chương trình cho đến khi điều kiện lặp false. Ta sẽ xem xét chương trình lặp bao nhiêu lần.
- Nếu đệ quy: một hàm gọi lại chính nó đến khi gặp điều kiện dừng. Ta xem xét số lần chương trình gọi lại chính nó.
- Nếu chương trình không lặp cũng không đệ quy thì ta kết luận rằng không có mối liên hệ giữa kích thước input và thời gian chạy, do đó độ phức tạp ở đây là hằng số, O(1).

Xác định độ phức tạp vòng lặp đơn

- Ví dụ: tìm độ phức tạp của đoạn chương trình sau:

```
int n = 100;
for (int i = 0; i < n; i++)
{
    cout << "Wellcome to Branium Academy!" << endl;
}
```

- Vì biến i chạy từ 0 đến n-1 nên chương trình lặp n lần việc in ra giá trị “Welcome …. My!”. Do đó độ phức tạp thuật toán là O(n).

Xác định độ phức tạp vòng lặp lồng nhau

- Ví dụ: tìm độ phức tạp của đoạn chương trình sau:

```
int m = 20;
int n = 20;
for (int i = 0; i < m; i++)
{
    for (int j = 0; j < n; j++)
    {
        cout << " * ";
    }
    cout << endl;
}
```

- Vòng lặp ngoài chạy m lần, mỗi lần đó vòng lặp trong chạy n lần. Suy ra tổng cộng chương trình chạy m * n lần. Giả sử m < n. Coi n = m + k. Kết quả: m * n = m * (m + k) = m^2 + k*m.
- Do đó độ phức tạp của đoạn chương trình trên là O(n^2).

Xác định độ phức tạp đệ quy fibonacci

- Ví dụ: tìm độ phức tạp của đoạn chương trình sau:

```
// tìm số fibonacci dùng đệ quy
unsigned long long fibonacci(int n) {
    if (n < 2) {
        return n;
    }
    else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

- Gọi T(n) đại diện cho độ phức tạp của F(n).
- F(n) = F(n-1) + F(n-2) => T(n) = T(n-1) + T(n-2) + 1.
- Khi n = 0 hoặc n = 1, T(n) = 1.
- Giả sử T(n-1) ~ T(n-2) => T(n) = T(n-1) + T(n-1) + 1 = 2T(n-1) + 1
- Tương tự, T(n-1) = 2T(n-2) + 1, thay vào biểu thức trên được
- T(n) = 2T(n-1) + 1 = 2(2T(n-2) + 1) + 1 = 4T(n-2) + 3.

Xác định độ phức tạp đệ quy fibonacci

- Tiếp tục thay công thức tương tự cho T(n-2):
- T(n) = 4T(n-2) + 3 = 4(2T(n-3) + 1) + 3 = 8T(n-3) + 7.
- Suy diễn tương tự tới F(n-k): T(n) = 2^kT(n-k) + (2^k - 1).
- Khi k = n, T(n) = 2^nT(n-n) + (2^n - 1) với T(0) = 1, suy ra:
  T(n) = 2^n + 2^n - 1 = 2*2^n - 1 => O(2^n)

Tầm quan trọng của thuật toán

- Hiệu năng của máy tính ngày càng tăng, tốc độ cao, bộ nhớ lớn, rẻ, vậy tại sao phải quan tâm đến vấn đề tối ưu thuật toán?
- Lý do là dữ liệu máy tính phải xử lý ngày càng lớn và phức tạp. Mặt khác ta lại muốn nhanh chóng có kết quả thực hiện chương trình. Mong muốn thiết bị chạy nhanh, mượt, không nóng máy…
- Do đó với các bộ dữ liệu lớn, phức tạp, các phần cứng tốt nhất cũng sẽ dần trở nên kém hiệu quả và làm người dùng mất kiên nhẫn.
- Việc nâng cấp, cải tiến, sử dụng và phát minh ra các thuật toán mới hiệu quả trở nên quan trọng hơn bao giờ hết.

Nội dung tiếp theo

Hàm đệ quy