Chương 1
Tổng quan về phân tích giải thuật

Nội dung

Giải thuật

Phân tích thời gian thực hiện giải thuật

Đệ quy và giải thuật đệ quy

Giải thuật

Xác định bài toán

- Mô hình: Input → Process → Output
- Cần giải quyết vấn đề gì? Giả thiết nào đã cho? Cần đạt được kết quả gì?
- Cần quan tâm đến chi phí giải quyết vấn đề, có thể chấp nhận lời giải tốt tới mức nào đó, thậm chí là xấu ở mức chấp nhận được

Giải thuật

Xác định bài toán

- Bài toán: Sắp xếp một chuỗi số theo thứ tự không giảm dần
- Input: Một chuỗi n số $<a_1, a_2, \ldots, a_n>$
- Output: Một hoán vị $<a_1', a_2', \ldots, a_n'>$ của chuỗi input thoả điều kiện $a_1' <= a_2' <= \ldots <= a_n'$

Giải thuật

Xác định bài toán

- Bài toán: Một dự án có n người tham gia thảo luận, họ muốn chia thành các nhóm và mỗi nhóm thảo luận riêng về một phần của dự án. Nhóm có bao nhiêu người thì được trình bày bấy nhiêu ý kiến. Nếu lấy ở mỗi nhóm một ý kiến đem ghép lại thì được một bộ ý kiến triển khai dự án. Hãy tìm cách chia để số bộ ý kiến cuối cùng thu được là lớn nhất.

Giải thuật

Xác định bài toán

- Phát biểu lại: Cho một số nguyên dương n, tìm cách phân tích n thành tổng các số nguyên dương sao cho tích của các số đó là lớn nhất.
- Input: Số nguyên dương n
- Output: Một chuỗi k số <a_1, a_2, …, a_k> thoả mãn điều kiện $\sum_{i=1}^{k} a_i$ = n và $\prod_{i=1}^{k} a_i$ đạt max

Giải thuật

Khái niệm giải thuật

- Giải thuật là một hệ thống chặt chẽ và rõ ràng các quy tắc nhằm xác định một dãy thao tác sao cho: Với một bộ dữ liệu vào, sau một số hữu hạn bước thực hiện các thao tác đã chỉ ra, ta đạt được mục tiêu đã định.

Giải thuật

Các đặc trưng của giải thuật

- Tính đơn nghĩa: các thao tác ở mỗi bước của giải thuật phải rõ ràng, không nhập nhằng
- Tính dừng: giải thuật phải dừng lại và cho kết quả sau một số hữu hạn bước
- Tính đúng: các bước của giải thuật sau khi thực hiện phải cho ta phải được kết quả mong muốn với mọi bộ dữ liệu đầu vào

Giải thuật

Các đặc trưng của giải thuật

- Tính khả thi
  - Kích thước phải đủ nhỏ
  - Giải thuật phải chuyển được thành chương trình
  - Giải thuật phải được máy tính thực hiện trong thời gian cho phép, khác với lời giải toán (chỉ cần chứng minh là kết thúc sau hữu hạn bước).

Giải thuật

Các vấn đề liên quan đến cài đặt giải thuật

- Lập trình
  - Kỹ thuật lập trình tốt: nắm vững ngôn ngữ lập trình sử dụng, khả năng chuyển giải thuật thành chương trình hoàn chỉnh
- Kiểm thử
  - Xây dựng các bộ test để kiểm thử chương trình: đơn giản, đặc biệt, phức tạp
  - Tìm cách chứng minh tính đúng đắn của giải thuật và chương trình

Giải thuật

Các vấn đề liên quan đến cài đặt giải thuật

- Tối ưu chương trình
  - Sửa lại chương trình để chương trình ngắn hơn, chạy nhanh hơn
  - Các tiêu chuẩn tối ưu:
    - Tính tin cậy: chương trình mô tả đúng một giải thuật đúng
    - Tính uyển chuyển: chương trình dễ sửa đổi, giảm bớt công sức khi phát triển chương trình
    - Tính trong sáng: dễ đọc dễ hiểu, phụ thuộc công cụ và phong cách lập trình
    - Tính hữu hiệu: tiết kiệm không gian và thời gian

Phân tích thời gian thực hiện giải thuật

Khái niệm độ phức tạp tính toán của giải thuật

- Căn cứ nào để so sánh tốc độ thực thi giải thuật?
- Nếu gọi n là kích thước dữ liệu nhập thì thời gian thực hiện của một giải thuật có thể biểu diễn một cách tương đối như một hàm của n: T(n).
- Phần cứng máy tính, ngôn ngữ viết chương trình và chương trình dịch thay đổi trên các loại máy → không thể dựa vào để tính T(n)

Phân tích thời gian thực hiện giải thuật

Khái niệm độ phức tạp tính toán của giải thuật

- Nếu thời gian thực hiện hai giải thuật là $T_1(n) = n^2$ và $T_2(n) = 100n$ thì khi n đủ lớn, thời gian thực hiện của giải thuật T_2 nhanh hơn giải thuật T_1.
- Cách đánh giá thời gian thực hiện giải thuật độc lập với các yếu tố liên quan tới máy tính như vậy sẽ dẫn tới khái niệm gọi là độ phức tạp tính toán của giải thuật.

Phân tích thời gian thực hiện giải thuật

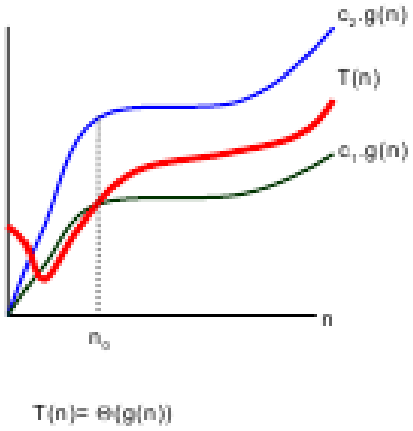
Các ký hiệu đánh giá độ phức tạp tính toán

- Cho một giải thuật thực hiện trên dữ liệu với kích thước n. Giả sử T(n) là thời gian thực hiện một giải thuật đó, g(n) là một hàm xác định dương với mọi n. Độ phức tạp tính toán của giải thuật là:

Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

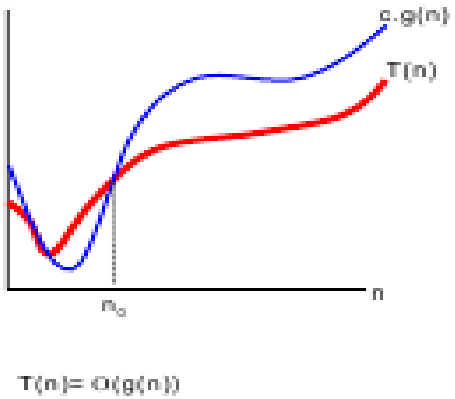
- ...
  - Θ(g(n)): tồn tại các hằng số dương c_1, c_2 và n_0 sao cho $c_1.g(n) \leq T(n) \leq c_2.g(n)$ với mọi $n \geq n_0$.



Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

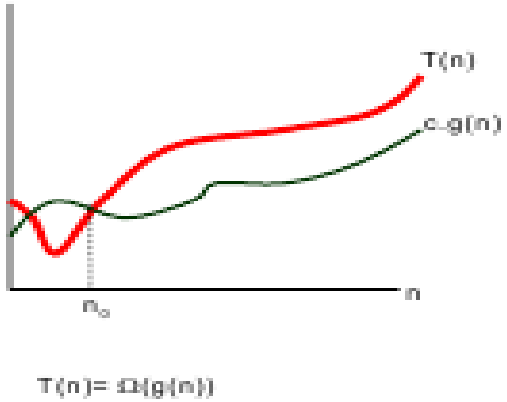
- ...
  - O(g(n)): tồn tại các hằng số dương c và n_0 sao cho T(n) ≤ c.g(n) với mọi n ≥ n_0.



Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- ...
  - $\Omega(g(n))$: tồn tại các hằng số dương c và n_0 sao cho $c.g(n) \leq T(n)$ với mọi $n \geq n_0$.



Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- …
  - o(g(n)): với mọi hằng số dương c, tồn tại một hằng số dương n_0 sao cho $0 \leq T(n) < c.g(n)$ với mọi $n \geq n_0$.
  - ω(g(n)): với mọi hằng số dương c, tồn tại một hằng số dương n_0 sao cho $0 \leq c.g(n) < T(n)$ với mọi $n \geq n_0$.

Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- Ví dụ $T(n) = n^2 + 1$, thì:
  - $T(n) = O(n^2)$. Chọn $c = 2$ và $n_0 = 1$ thì với mọi $n \geq 1$, ta có: $T(n) = n^2 + 1 <= 2n^2 = c.g(n)$
  - $T(n) \neq o(n^2)$. Chọn $c = 1$ thì không tồn tại n để: $n^2+1< n^2$, tức là không tồn tại n_0 thoả mãn định nghĩa của ký pháp chữ o nhỏ.

Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- Một số tính chất
  - Tính bắc cầu (transitivity): Tất cả các ký hiệu trên đều có tính bắc cầu
    - Nếu $f(n) = \Theta(g(n))$ và $g(n) = \Theta(h(n))$ thì $f(n) = \Theta(h(n))$
    - Nếu $f(n) = O(g(n))$ và $g(n) = O(h(n))$ thì $f(n) = O(h(n))$
    - Nếu $f(n) = \Omega(g(n))$ và $g(n) = \Omega(h(n))$ thì $f(n) = \Omega(h(n))$
    - Nếu $f(n) = o(g(n))$ và $g(n) = o(h(n))$ thì $f(n) = o(h(n))$
    - Nếu $f(n) = \omega(g(n))$ và $g(n) = \omega(h(n))$ thì $f(n) = \omega(h(n))$

Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- Một số tính chất
  - Tính phản xạ (reflexivity): Chỉ có các ký pháp "lớn" mới có tính phản xạ
    - $f(n) = \Theta(f(n))$
    - $f(n) = O(f(n))$
    - $f(n) = \Omega(f(n))$

Phân tích thời gian thực hiện giải thuật

Các ký hiệu đánh giá độ phức tạp tính toán

- Một số tính chất
  - Tính đối xứng (symmetry): Chỉ có ký pháp Θ lớn có tính đối xứng

$$f(n) = \Theta(g(n)) \text{ nếu và chỉ nếu } g(n) = \Theta(f(n))$$

  - Tính chuyển vị đối xứng (transpose symmetry)

$$f(n) = O(g(n)) \text{ nếu và chỉ nếu } g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \text{ nếu và chỉ nếu } g(n) = \omega(f(n))$$

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Quy tắc bỏ hằng số
  - Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(c_1.f(n))$ với c_1 là một hằng số dương thì có thể coi đoạn chương trình đó có độ phức tạp tính toán là $O(f(n))$.
  - Qui tắc này cũng đúng với các ký pháp Ω, Θ, o và ω.

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Quy tắc lấy max
  - Nếu đoạn chương trình P có thời gian thực hiện T(n) = O(f(n) + g(n)) thì có thể coi đoạn chương trình đó có độ phức tạp tính toán O(max(f(n), g(n))).
  - Qui tắc này cũng đúng với các ký pháp Ω, Θ, o và ω.

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Quy tắc cộng
  - Nếu đoạn chương trình P_1 có thời gian thực hiện $T_1(n)$ =O(f(n)) và đoạn chương trình P_2 có thời gian thực hiện là $T_2(n)$ = O(g(n)) thì thời gian thực hiện P_1 rồi đến P_2 tiếp theo sẽ là $T_1(n) + T_2(n) = O(f(n) + g(n))$
  - Qui tắc này cũng đúng với các ký pháp Ω, Θ, o và ω.

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Quy tắc nhân
  - Nếu đoạn chương trình P có thời gian thực hiện là T(n) = O(f(n)). Khi đó, nếu thực hiện k(n) lần đoạn chương trình P với k(n) = O(g(n)) thì độ phức tạp tính toán sẽ là O(g(n).f(n))
  - Qui tắc này cũng đúng với các ký pháp Ω, Θ, o và ω.

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Đọc định lý Master (Master Theorem) [1,2]
- Áp dụng tính độ phức tạp cho giải thuật có công thức truy hồi sau
  - $T(n) = 9T(n/3) + n$
  - $T(n) = T(2n/3) + 1$
  - $T(n) = 3T(n/4) + n\lg n$

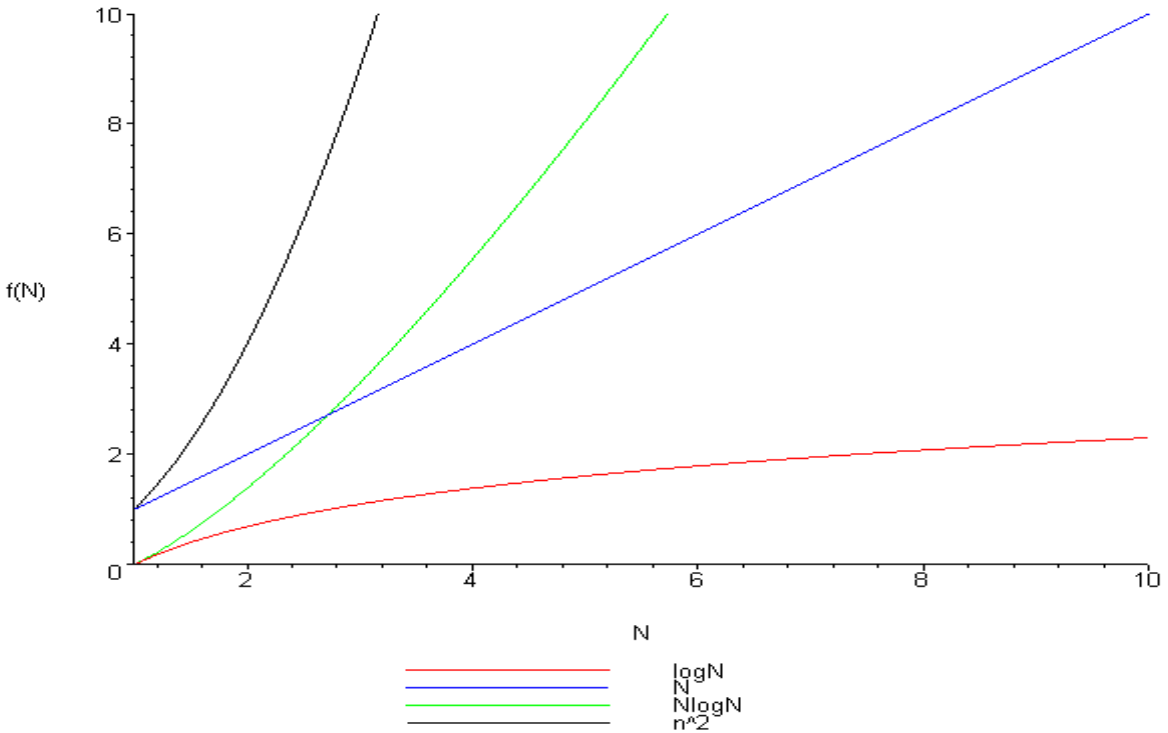
Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

lgn	n	nlgn	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	2147483648

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật



Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Độ phức tạp tính toán với tình trạng dữ liệu vào
  - Thời gian thực hiện giải thuật không phải chỉ phụ thuộc vào kích thước dữ liệu mà còn phụ thuộc vào tình trạng của dữ liệu
  - Khi phân tích giải thuật, ta sẽ phải xét tới trường hợp tốt nhất, trường hợp trung bình và trường hợp xấu nhất.

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Độ phức tạp tính toán với tình trạng dữ liệu vào
  - Phân tích thời gian thực hiện giải thuật trong trường hợp xấu nhất: Với kích thước dữ liệu n, tìm T(n) là thời gian lớn nhất khi thực hiện giải thuật trên mọi bộ dữ liệu kích thước n
  - Phân tích thời gian thực hiện giải thuật trong trường hợp tốt nhất: Với kích thước dữ liệu n, tìm T(n) là thời gian ít nhất khi thực hiện giải thuật trên mọi bộ dữ liệu kích thước n

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Độ phức tạp tính toán với tình trạng dữ liệu vào
  - Phân tích thời gian trung bình thực hiện giải thuật: Giả sử rằng dữ liệu vào tuân theo một phân phối xác suất nào đó và tính toán giá trị kỳ vọng (trung bình) của thời gian chạy cho mỗi kích thước dữ liệu n (T(n)), sau đó phân tích thời gian thực hiện giải thuật dựa trên hàm T(n).
  - Trên phương diện lý thuyết, đánh giá bằng ký pháp Θ(.) là tốt nhất, tuy vậy việc đánh giá bằng ký pháp Θ(.) sẽ phức tạp vì đòi hỏi phải đánh giá bằng cả ký pháp O(.) lẫn Ω(.). Vì vậy, ta sẽ dùng ký pháp T(n) = O(f(n))

Phân tích thời gian thực hiện giải thuật

Xác định độ phức tạp tính toán của giải thuật

- Chi phí thực hiện giải thuật
  - Nếu ta đánh giá được độ phức tạp tính toán của một giải thuật qua ký pháp Θ, không cần đánh giá qua những ký pháp khác nữa.
  - Nếu không
    - Để nhấn mạnh đến tính "tốt" của một giải thuật, các ký pháp O, o thường được sử dụng. Ý nói: Chi phí thực hiện giải thuật tối đa là…, ít hơn…
    - Để đề cập đến tính "xấu" của một giải thuật, các ký pháp Ω, ω thường được sử dụng. Ý nói: Chi phí thực hiện thuật toán tối thiểu là…, cao hơn…

Phân tích thời gian thực hiện giải thuật

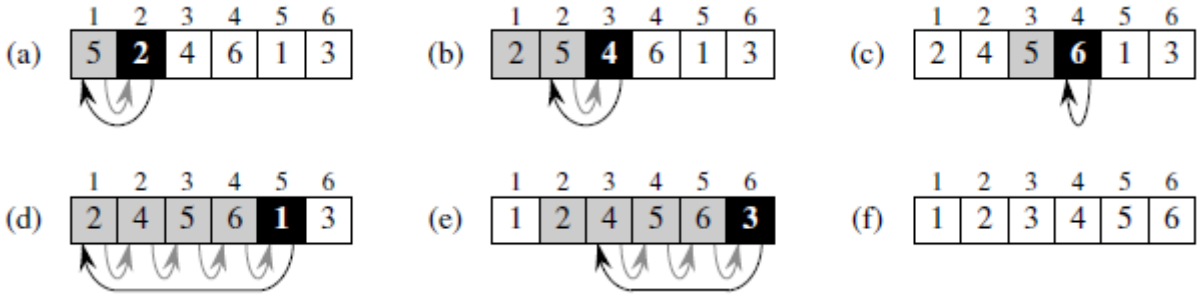
Phân tích một số giải thuật tìm kiếm và sắp xếp

- Insertion Sort
  - Ý tưởng: Lần lượt chèn từng phần tử vào vị trí đúng trong một dãy con đã được sắp xếp của dãy cần sắp cho tới khi toàn bộ dãy đều được sắp xếp

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Insertion Sort
  - Ví dụ: Sắp xếp dãy <5, 2, 4, 6, 1, 3>



Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Insertion Sort
  - Giải thuật

```
INSERTION-SORT (A)
    for i = 1 to n-1
        temp = A[i]
        // Insert A[i] into the sorted sequence A[0..i-1]
        j = i – 1
        while j >= 0 and A[j] > temp
            A[j+1] = A[j]
            j = j – 1
        A[j+1] = temp
```

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Insertion Sort
  - Trường hợp tốt nhất ứng với dãy có thứ tự, mỗi lượt chỉ cần 1 phép so sánh, như vậy tổng số phép so sánh được thực hiện là n - 1. Độ phức tạp tính toán tốt nhất của Insertion Sort là O(n).
  - Trường hợp xấu nhất ứng với dãy có thứ tự ngược thì ở lượt thứ i, cần có i phép so sánh và tổng số phép so sánh là: (n - 1) + (n - 2) + … + 1 = n * (n - 1) / 2. Độ phức tạp tính toán xấu nhất của Insertion Sort là $O(n^2)$.

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Insertion Sort
  - Trường hợp các giá trị xuất hiện một cách ngẫu nhiên, ta có thể coi xác suất xuất hiện mỗi giá trị là đồng khả năng, thì có thể coi lượt thứ i cần trung bình i/2 phép so sánh và tổng số phép so sánh là: (1/2) + (2/2) + … + ((n-1)/2) = (n - 1) * n / 4.
  - Độ phức tạp tính toán trung bình của Insertion Sort là $O(n^2)$

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Selection Sort
  - Ý tưởng: Tìm phần tử nhỏ nhất trong dãy con cần sắp xếp, đặt nó vào đúng vị trí trong dãy đã sắp xếp cho tới khi xét hết toàn bộ dãy
  - Giải thuật

```
SELECTION-SORT (A)
    for i = 0 to n-2
        min = i
        for j = i+1 to n-1
            if A[j] < A[min] then min = j
        if min ≠ i then swap(A[min], A[i])
```

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Selection Sort
  - Phân tích: Ở lượt thứ i, để chọn ra phần tử nhỏ nhất bao giờ cũng cần n - i - 1 phép so sánh, số lượng phép so sánh này không phụ thuộc vào tình trạng ban đầu của dãy. Tổng số phép so sánh sẽ phải thực hiện là:

$$(n - 1) + (n - 2) + \ldots + 1 = n * (n - 1) / 2$$

Độ phức tạp tính toán của Selection Sort là $O(n^2)$

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Bubble Sort
  - Ý tưởng: Đổi chỗ hai phần tử đứng kế nhau trong dãy con chưa sắp xếp sao cho phần tử nhỏ hơn phải đứng trước. Thực hiện từ cuối về đầu dãy sẽ chuyển phần tử nhỏ nhất về đúng vị trí. Tiếp tục sắp xếp những phần tử còn lại của dãy.
  - Giải thuật

```
BUBBLE-SORT (A)
    for i = 1 to n-1
        for j = n-1 downto i
            if A[j] < A[j-1] then swap(A[j],A[j-1])
```

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Bubble Sort
  - Phân tích: Tổng số phép so sánh sẽ phải thực hiện là:

$$(n - 1) + (n - 2) + \ldots + 1 = n * (n - 1) / 2$$

Độ phức tạp tính toán của Bubble Sort là $O(n^2)$

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Sequential Search
  - Ý tưởng: So sánh lần lượt từng phần tử trong dãy với giá trị cần tìm cho đến khi tìm thấy hoặc duyệt hết dãy
  - Giải thuật

```
SEQUENTIAL-SEARCH (A,x)
    i = 0
    while i <= n-1 and A[i] ≠ x
        i = i + 1
    if i == n then return -1
    return i
```

Phân tích thời gian thực hiện giải thuật

Phân tích một số giải thuật tìm kiếm và sắp xếp

- Sequential Search
  - Độ phức tạp của giải thuật trong trường hợp tốt nhất là O(1), trong trường hợp xấu nhất là O(n) và trong trường hợp trung bình là O(n).

Đệ quy và giải thuật đệ quy

Khái niệm đệ quy

- Một đối tượng là đệ quy nếu nó được định nghĩa qua chính nó hoặc một đối tượng khác cùng dạng với chính nó bằng quy nạp
- Ví dụ
  - Giai thừa của n (n!)
    - Nếu n = 0 thì n! = 1
    - Nếu n > 0 thì n! = n.(n-1)!
  - Ký hiệu số phần tử của một tập hợp hữu hạn S là |S|
    - Nếu S = ∅ thì |S| = 0
    - Nếu S ≠ ∅ thì có một phần tử x ∈ S, khi đó |S| = |S\{x}| + 1.

Đệ quy và giải thuật đệ quy

Giải thuật đệ quy

- Nếu lời giải của một bài toán P được thực hiện bằng lời giải của bài toán P' có dạng giống như P thì đó là một lời giải đệ quy
- P’ phải nhỏ hơn và dễ giải hơn P

Đệ quy và giải thuật đệ quy

Giải thuật đệ quy

- Một hàm hay thủ tục đệ quy gồm hai phần
  - Phần cơ sở: Phần này được thực hiện khi mà công việc đơn giản, có thể giải trực tiếp
  - Phần đệ quy: Trong trường hợp bài toán chưa thể giải được bằng phần cơ sở, ta xác định những bài toán con và gọi đệ quy giải những bài toán con đó. Khi đã có lời giải của những bài toán con thì phối hợp chúng lại để giải bài toán ban đầu

Đệ quy và giải thuật đệ quy

Một số ví dụ

- Hàm tính giai thừa

```
FACTORIAL (n)
    if i == 0 then return 1
    else return n * FACTORIAL(n-1)
```

- Ví dụ: Tính 3!

3! = 3 * 2!
↓
2! = 2 * 1!
↓
1! = 1 * 0!
↓
0! = 1

Đệ quy và giải thuật đệ quy

Một số ví dụ

- Bài toán tháp Hà Nội
  - Mô tả

    Chuyển n đĩa xếp theo thứ tự nhỏ trên, lớn dưới từ cọc 1 sang cọc 2, dùng cọc 3 làm trung gian với yêu cầu nhỏ trên, lớn dưới vẫn phải đảm bảo.

Đệ quy và giải thuật đệ quy

Một số ví dụ

- Bài toán tháp Hà Nội
  - Phân tích
    - Phần cơ sở: Nếu n = 1 thì ta chuyển đĩa đó từ cọc 1 sang cọc 2 là xong.
    - Phần đệ quy: Giả sử rằng ta có phương pháp chuyển được n - 1 đĩa từ cọc 1 sang cọc 3. Khi đó cọc 1 chỉ còn 1 đĩa lớn nhất, ta sẽ chuyển đĩa này sang cọc 2. Sau đó, ta lại chuyển n - 1 đĩa từ cọc 3 sang cọc 2.

Đệ quy và giải thuật đệ quy

Một số ví dụ

- Bài toán tháp Hà Nội
  - Giải thuật

```
MOVE(n, x, y)
    if n == 1 then writeln("Move one disk from "  + x +  " to " + y)
    else
        MOVE(n-1, x, 6-x-y)
        MOVE(1, x, y)
        MOVE(n-1, 6-x-y, y)
```

Đệ quy và giải thuật đệ quy

Phân tích giải thuật đệ quy

- Với một giải thuật đệ quy, thời gian chạy đối với bộ dữ liệu nhập kích thước n tùy thuộc vào thời gian chạy của những bộ dữ liệu nhập nhỏ hơn
- Tính chất này được mô tả bằng một công thức toán học được gọi là hệ thức truy hồi (*recurrence relation*)
- Để dẫn xuất ra độ phức tạp của một giải thuật đệ quy, ta phải giải hệ thức truy hồi này

Đệ quy và giải thuật đệ quy

Phân tích giải thuật đệ quy

- Ví dụ 1
  - Bài toán: Một chương trình đệ quy mà lặp qua bộ dữ liệu nhập để loại đi một phần tử. Hệ thức truy hồi của nó như sau:

$$C_N = C_{N-1} + N \text{ với } N \geq 2$$
$$C_1 = 1$$

Đệ quy và giải thuật đệ quy

Phân tích giải thuật đệ quy

- Ví dụ 1
  - Phân tích độ phức tạp

$$
\begin{aligned}
C_N &= C_{N-1} + N \\
&= C_{N-2} + (N-1) + N \\
&= C_{N-3} + (N-2) + (N-1) + N \\
&\cdots \\
&= C_1 + 2 + \ldots + (N-2) + (N-1) + N \\
&= 1 + 2 + \ldots + (N-1) + N \\
&= N(N+1)/2
\end{aligned}
$$

Vậy $C_N = O(N^2)$

Đệ quy và giải thuật đệ quy

Phân tích giải thuật đệ quy

- Ví dụ 2
  - Bài toán: Một chương trình đệ quy giảm một nửa bộ dữ liệu nhập trong một bước làm việc. Hệ thức truy hồi là:

$$C_N = C_{N/2} + 1 \text{ với } N \geq 2$$
$$C_1 = 0$$

Đệ quy và giải thuật đệ quy

Phân tích giải thuật đệ quy

- Ví dụ 2
  - Phân tích độ phức tạp

$$\text{Đặt } N = 2^n$$

$$C(2^n) = C(2^{n-1}) + 1$$

$$= C(2^{n-2}) + 1 + 1$$

$$= C(2^{n-3}) + 3$$

.

.

$$= C(2^0) + n$$

$$= C_1 + n = n$$

$$C_N = n = \lg N$$

Vậy $C_N = O(\lg N)$

- THANKS YOU