COPYRIGHT NOTICE
THÔNG BÁO BẢN QUYỀN

COPYRIGHT (English):

This document is not up to date and may contain several errors or outdated information.

Last revision date: 2023-03-28

BẢN QUYỀN (Tiếng Việt):

Tài liệu này không được cập nhật và có thể chứa nhiều lỗi hoặc thông tin cũ.

Ngày sửa đổi cuối cùng: 2023-03-28

VNU-HUS MAT3500: Toán rời rạc

Thuật toán II

Độ phức tạp tính toán, thuật toán tham lam, thuật toán đệ quy

Hoàng Anh Đức

Bộ môn Tin học, Khoa Toán-Cơ-Tin học
Đại học KHTN, ĐHQG Hà Nội
hoanganhduc@hus.edu.vn

Nội dung

Độ phức tạp tính toán
Giới thiệu

- Một thuật toán cần *luôn xuất ra kết quả đúng* và tốt nhất là *có hiệu suất cao*
- Chúng ta đã tìm hiểu một số *ví dụ* về các thuật toán tìm kiếm và sắp xếp trong bài giảng trước và phương pháp *chứng minh tính đúng đắn* của chúng dựa trên bất biến vòng lặp (loop invariant).
- *Độ phức tạp (complexity)* của một tính toán là một cách đo độ "khó" của việc thực hiện tính toán đó
  - *Độ phức tạp theo thời gian (time complexity):* Số các toán tử hoặc số bước cần thiết
  - *Độ phức tạp theo bộ nhớ (space complexity):* Số các bit trong bộ nhớ cần thiết
- Phần lớn các thuật toán có độ phức tạp khác nhau đối với các đầu vào có kích thước khác nhau
  - Tìm kiếm trong một dãy dài thường tốn nhiều thời gian hơn tìm kiếm trong một dãy ngắn
- Do đó, độ phức tạp tính toán thường được biểu diễn dưới dạng một *hàm (function) của kích thước đầu vào (input size)*

Độ phức tạp tính toán
Giới thiệu

- Khi xét độ phức tạp của một thuật toán, ta không quan tâm đến số lượng chính xác các toán tử/bit cần thiết mà chỉ cần một *đánh giá tiệm cận (asymptotic estimate)*
- Một công cụ hữu ích cho việc đánh giá độ phức tạp tính toán là các ký hiệu O-lớn, Ω-lớn, và Θ-lớn
- Chúng ta sẽ tập trung vào *độ phức tạp tính toán theo thời gian (time complexity)* (chủ yếu là *trong trường hợp xấu nhất*)
  - *Độ phức tạp trong trường hợp xấu nhất (worst-case complexity):* xấp xỉ thời gian nhiều nhất cần để giải quyết các trường hợp đầu vào với mỗi kích thước đầu vào
  - *Độ phức tạp trong trường hợp trung gian (average-case complexity):* xấp xỉ thời gian trung bình cần để giải quyết các trường hợp đầu vào với mỗi kích thước đầu vào
  - *Độ phức tạp trong trường hợp tốt nhất (best-case complexity):* xấp xỉ thời gian ít nhất cần để giải quyết các trường hợp đầu vào với mỗi kích thước đầu vào

Với các hàm f và g từ $\mathbb{R}$ đến $\mathbb{R}$

- f là $O(g)$ tương tự "$\leq$"

$$\exists C, k \; \forall x > k \quad |f(x)| \leq C|g(x)|$$

- f là $\Omega(g)$ tương tự "$\geq$"

$$\exists C > 0, k \; \forall x > k \quad |f(x)| \geq C|g(x)|$$

- f là $\Theta(g)$ tương tự "$=$"

$$\exists C_1, C_2, k \; \forall x > k \quad C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

Độ phức tạp tính toán
Chú ý

- Ký hiệu $f(n) = O(g(n))$ thường được sử dụng để chỉ $f(n)$ là $O(g(n))$.
  - Ký hiệu này không hoàn toàn chặt chẽ về mặt toán học, do $f(n)$ là một hàm còn $O(g(n))$ *là một tập hợp các hàm*
  - $f(n) = O(g(n))$ trên thực tế nghĩa là $f(n) \in O(g(n))$, do đó có thể viết $n = O(n^2)$ nhưng *không nên viết* $O(n^2) = n$
- Bạn có thể gặp biểu thức dạng "$f(n) + O(g(n)) = O(h(n))$"
  - Dấu "$=$" ở đây nghĩa là "$\subseteq$". Cụ thể, biểu thức trên cần được hiểu là tập hợp S gồm các hàm $f(n) + g_1(n)$ với $g_1(n) \in O(g(n))$ là tập con của tập $O(h(n))$
- Bạn có thể gặp biểu thức dạng "$f(n) \leq g(n) + O(h(n))$ với mọi $n \geq 0$" hoặc tương tự
  - Nghĩa là tồn tại $e(n)$ sao cho (a) $f(n) \leq g(n) + e(n)$ với mọi $n \geq 0$ và (b) $e(n) \in O(h(n))$
- Một số tác giả định nghĩa O-lớn bằng cách thay điều kiện $|f(x)| \leq C|g(x)|$ bằng $0 \leq f(x) \leq C(g(x))$. (Làm việc với giá trị tuyệt đối và khả năng các hàm $f(x)$ và $g(x)$ có thể nhận giá trị âm thường khó hơn là chỉ làm việc với các hàm nhân giá trị dương.) Định nghĩa theo cách này không hoàn toàn chặt chẽ. Ví dụ như hàm $\log n$ có thể nhận giá trị âm với n nhỏ

Độ phức tạp tính toán
Chú ý

- *Để đơn giản, ta thường mô tả thời gian chạy của thuật toán theo O-lớn*. Chú ý rằng với nhiều thuật toán, ta có thể thu được đánh giá tốt hơn với xấp xỉ theo Θ-lớn. Tuy nhiên không phải lúc nào ta cũng làm được điều này
- Nhiều tác giả viết "$f(x)$ là $O(g(x))$" trong khi điều họ thực sự muốn thể hiện là "$f(x)$ là $\Theta(g(x))$"

Độ phức tạp tính toán
Giới thiệu

Một số quy tắc tính độ phức tạp thuật toán theo thời gian

- Thời gian thực hiện các lệnh gán ($:=$), trả lại (**return**) là $O(1)$, và giả sử thời gian thực hiện các phép toán cơ bản (cộng, trừ, nhân, chia, so sánh, v.v...) cũng là $O(1)$
  - Chú ý rằng với phần cứng hữu hạn (64-bit CPU chỉ biểu diễn được tối đa các số nguyên có giá trị nhỏ hơn 2^{64} dưới dạng số nhị phân), việc cộng hai số nguyên độ dài n bit biểu diễn dưới dạng số nhị phân có độ phức tạp $O(n)$
- Độ phức tạp của một chuỗi tuần tự các bước là tổng của độ phức tạp của từng bước
- Thời gian thực hiện cấu trúc **if...then...else** là thời gian lớn nhất thực hiện các lệnh sau **then** hoặc sau **else** và thời gian kiểm tra điều kiện
- Thời gian thực hiện vòng lặp là tổng thời gian thực hiện các lần lặp và thời gian kiểm tra điều kiện lặp. Nếu thời gian thực hiện mỗi lần lặp là giống nhau, thì tổng thời gian thực hiện các lần lặp là tích của số lần lặp và thời gian thực hiện mỗi lần lặp

Độ phức tạp tính toán
Giới thiệu

Bảng: Một số thuật ngữ thường dùng

Độ phức tạp	Thuật ngữ
$O(1)$	Độ phức tạp hằng số (constant complexity)
$O(\log n)$	Độ phức tạp lôgarit (logarithmic complexity)
$O(n)$	Độ phức tạp tuyến tính (linear complexity)
$O(n \log n)$	Độ phức tạp $n \log n$ (linearithmic complexity)
$O(n^b)$	Độ phức tạp đa thức (polynomial complexity)
$O(b^n)$, với $b > 1$	Độ phức tạp hàm mũ (exponential complexity)
$O(n!)$	Độ phức tạp giai thừa (factorial complexity)

Độ phức tạp tính toán

Thuật toán tìm giá trị lớn nhất

Thuật toán 1: Tìm giá trị của phần tử lớn nhất

Input: $a_1, a_2, \ldots, a_n$: dãy số nguyên

Output: Giá trị của phần tử lớn nhất trong dãy

```
1 v := a_1                                   t_1
2 for i := 2 to n do                         t_2
3     if a_i > v then                        t_4^i
4         v := a_i                           t_5
5 return v                                   t_3
```

$$T(n) = t_1 + t_2 + t_3$$

t_1, t_3, t_5 là $O(1)$

$T(n)$ là $O(n)$ (xấu nhất)

$T(n)$ là $O(n)$ (tốt nhất)

$$t_2 = \sum_{i=2}^{n} t_4^i + (\#\text{ lần lặp vòng } \textbf{for}) = \sum_{i=2}^{n} t_4^i + (n-1)$$

$$t_4^i = t_5 + (\text{thời gian kiểm tra } a_i > v)$$

Độ phức tạp tính toán

Tìm kiếm tuyến tính

Thuật toán 2: Tìm kiếm tuyến tính (Linear Search)

Input: $a_1, \ldots, a_n$: dãy số nguyên, x: số nguyên
Output: Chỉ số i thỏa mãn $x = a_i$ hoặc 0 nếu x không có trong dãy

```
1 i := 1                                              t1
2 while i ≤ n và x ≠ a_i do                           t2
3     i := i + 1                                      t5^i
4 if i ≤ n then                                       t3
5     location := i                                   t6
6 else                                                t3
7     location := 0                                   t7
8 return location                                     t4
```

$$T(n) = t_1 + t_2 + t_3 + t_4$$

$T(n)$ là $O(n)$ (xấu nhất)

$T(n)$ là $O(1)$ (tốt nhất)

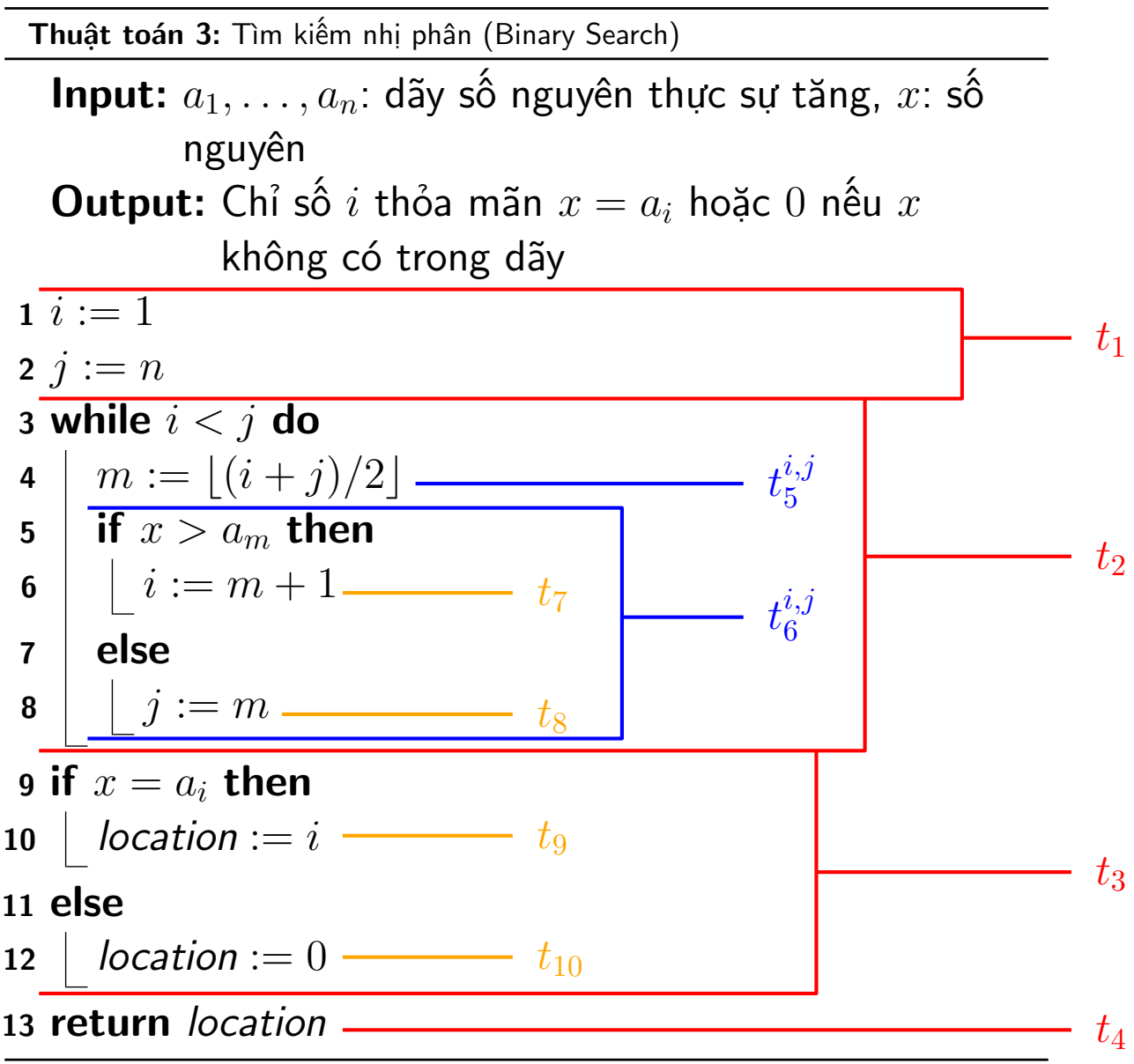
$t_1, t_4, t_5^i, t_6, t_7$ là $O(1)$

$$t_2 = \sum_{\{i \mid i \le n \wedge x \ne a_i\}} t_5^i + (\text{thời gian kiểm tra } i \le n \text{ và } x \ne a_i)$$

$$t_3 = \max\{t_6, t_7\} + (\text{thời gian kiểm tra } i \le n)$$

Độ phức tạp tính toán

Tìm kiếm nhị phân



Độ phức tạp tính toán

Tìm kiếm nhị phân

$$T(n) = t_1 + t_2 + t_3 + t_4$$

$t_1, t_4, t_5^{i,j}, t_7, \ldots, t_{10}$ là $O(1)$

$$t_2 = \sum_{\text{cặp } i,j \text{ mỗi lần lặp } \textbf{while}} (t_5^{i,j} + t_6^{i,j})$$

$+$ (thời gian kiểm tra $i < j$)

$$t_6^{i,j} = \max\{t_7, t_8\}$$

$+$ (thời gian kiểm tra $x > a_m$)

$$t_3 = \max\{t_9, t_{10}\}$$

$+$ (thời gian kiểm tra $x = a_i$)

cặp i, j mỗi lần lặp **while** là $O(\log n)$ (ví dụ $n = 2^k$)

$T(n)$ là $O(\log n)$ (xấu nhất)

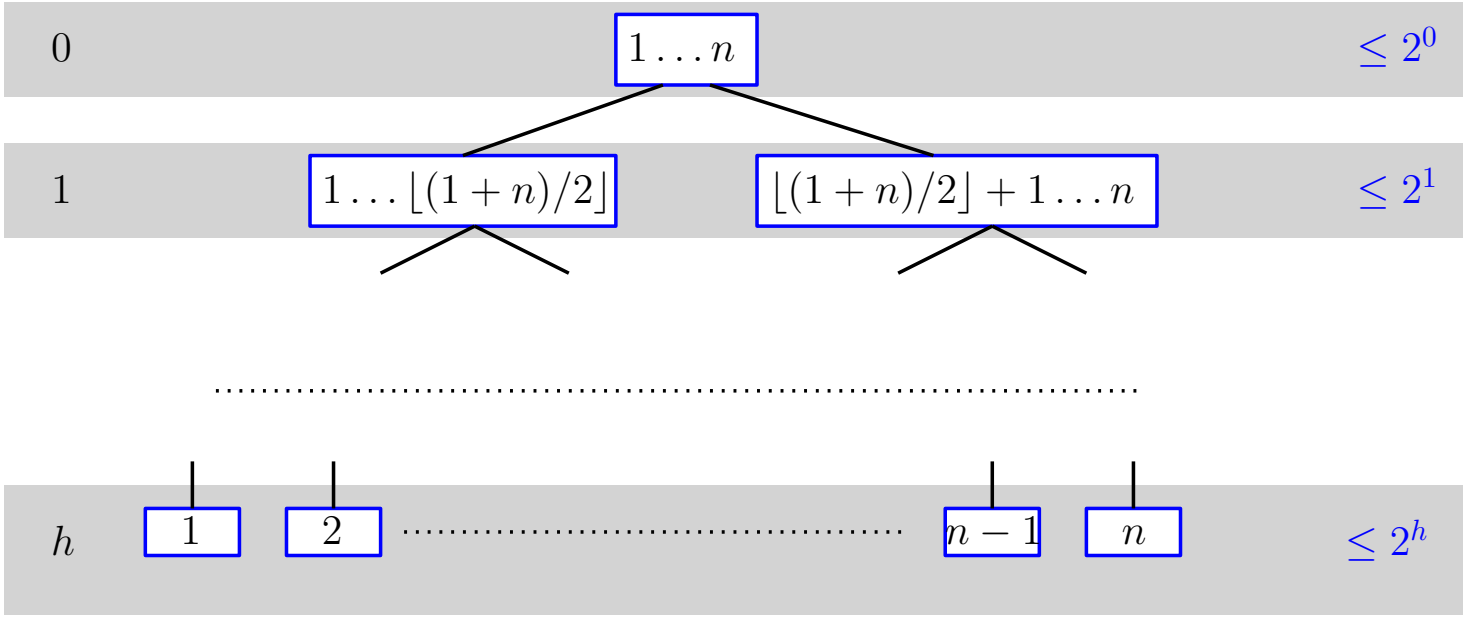
$T(n)$ là $O(\log n)$ (tốt nhất)

Độ phức tạp tính toán

Tìm kiếm nhị phân

cặp i, j mỗi lần lặp **while** là $O(\log n)$

- Đánh số các lần lặp **while** lần lượt bằng $0, 1, 2, \ldots, h$
- Mỗi hàng $0, 1, 2, \ldots, h$ liệt kê các khoảng $i \ldots j$ có khả năng xuất hiện ở lần lặp đánh số tương ứng.
- Vòng lặp **while** dừng ở lần lặp h khi mọi khoảng $i \ldots j$ chỉ có một phần tử $i = j$
- Ở mỗi hàng $k \in \{1, \ldots, h\}$, tổng số phần tử của tất cả các khoảng là $\leq n$, và có $\leq 2^k$ khoảng (Hình dưới mô tả trường hợp $n = 2^h$)



Độ phức tạp tính toán

Sắp xếp nổi bọt

Thuật toán 4: Sắp xếp nổi bọt (Bubble Sort)

Input: $a_1, a_2, \ldots, a_n$: dãy số thực $(n \geq 2)$

Output: Dãy đã cho được sắp xếp theo thứ tự tăng dần

```
1 for i := 1 to n − 1 do                                   — t_1
2     for j := 1 to n − i do                               — t_2^i
3         if a_j > a_{j+1} then                            — t_3^j
4             Hoán đổi giá trị của a_j và a_{j+1}          — t_4
```

$$T(n) = t_1 = \sum_{i=1}^{n-1} t_2^i + (\#\text{ lần lặp vòng } \textbf{for} \text{ ngoài}) = \sum_{i=1}^{n-1} t_2^i + (n-1)$$

$$t_2^i = \sum_{j=1}^{n-i} t_3^j + (\#\text{ lần lặp vòng } \textbf{for} \text{ trong}) = \sum_{j=1}^{n-i} t_3^j + (n-i)$$

t_4 là $O(1)$ $(v := a_j,\ a_j := a_{j+1},\ a_{j+1} := v)$

$t_3^j = t_4 + (\text{thời gian kiểm tra } a_j > a_{j+1})$

$T(n)$ là $O(n^2)$

(xấu nhất)

$T(n)$ là $O(n^2)$

(tốt nhất)

Độ phức tạp tính toán

Sắp xếp chèn

Thuật toán 5: Sắp xếp chèn (Insertion Sort)

Input: $a_1, a_2, \ldots, a_n$: dãy số thực $(n \geq 2)$

Output: Dãy đã cho được sắp xếp theo thứ tự tăng dần

```
1 for i = 2 to n do                                   t_1
2     m = a_i                                         t_2^i
3     j := i − 1                                      t_3^i
4     while j ≥ 1 và m < a_j do                       t_4^i
5         a_{j+1} := a_j                              t_6^{i,j}
6         j := j − 1                                  t_6^{i,j}
7     a_{j+1} := m                                    t_5^i
```

$$T(n) = t_1 = \sum_{i=2}^{n} (t_2^i + t_3^i + t_4^i + t_5^i) + (\#\text{ lặp vòng } \textbf{for})$$

$t_2^i, t_3^i, t_5^i, t_6^{i,j}$ là $O(1)$

$$t_4^i = \sum_{1 \leq j \leq i-1 \text{ và } m < a_j} t_6^{i,j}$$

$+$ (thời gian kiểm tra $j \geq 1$ và $m < a_j$)

$T(n)$ là $O(n^2)$ (xấu nhất)

$T(n)$ là $O(n)$ (tốt nhất)

Thuật toán tham lam
Giới thiệu

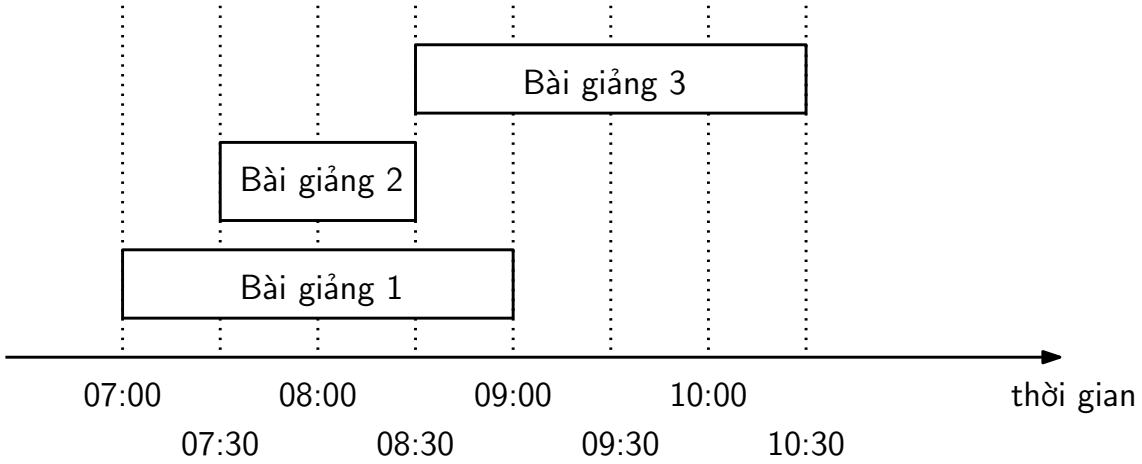
- *Các bài toán tối ưu (optimization problems)* yêu cầu cực đại hóa hoặc cực tiểu hóa một số tham số xét trên tập tất cả các đầu vào có thể
  - Tìm đường đi giữa hai thành phố với khoảng cách *nhỏ nhất*
  - Tìm cách mã hóa các thông điệp sử dụng số lượng bit *nhỏ nhất* có thể
- Một *thuật toán tham lam (greedy algorithm)* thường được sử dụng để giải bài toán tối ưu: luôn chọn biện pháp "tốt nhất" ở mỗi bước địa phương (theo một số tiêu chuẩn cục bộ nào đó) với hi vọng sẽ thu được một lời giải tối ưu trên toàn cục.
  - Giải thuật này không nhất thiết xuất ra một lời giải tối ưu cho toàn bộ bài toán, nhưng trong nhiều trường hợp cụ thể nó có thể xuất ra lời giải tối ưu
  - Sau khi mô tả cụ thể "lựa chọn tốt nhất ở từng bước", ta cố gắng chứng minh rằng giải thuật này luôn cho ta một lời giải tối ưu hoặc tìm một phản ví dụ để chỉ ra điều ngược lại.

Thuật toán tham lam
Lập lịch

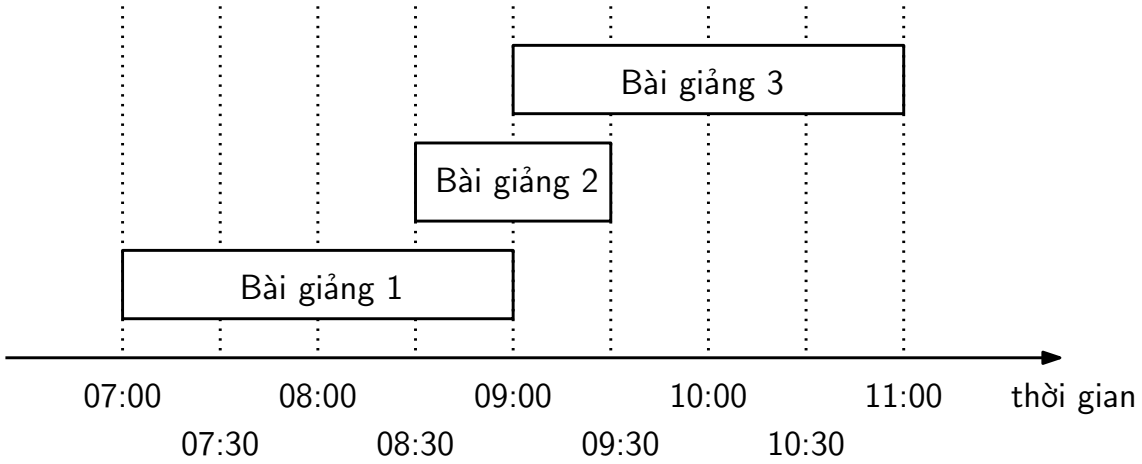
- **Bài toán:**
  - **Input:**
    - Một nhóm các bài giảng với thời gian bắt đầu và kết thúc
    - Chỉ có một giảng đường duy nhất
    - Khi một bài giảng bắt đầu, nó tiếp diễn cho đến khi kết thúc
    - Không có hai bài giảng nào được tiến hành ở cùng thời điểm.
    - Ngay sau khi một bài giảng kết thúc, một bài giảng khác có thể bắt đầu
  - **Output:** Một danh sách các bài giảng dài nhất có thể
- Ở đây, nếu ta muốn áp dụng giải thuật tham lam, làm thế nào để "lựa chọn tốt nhất" ở mỗi bước của thuật toán? Nói cách khác, ta sẽ chọn bài giảng như thế nào?
  - Chọn bài giảng bắt đầu sớm nhất trong số các bài giảng bắt đầu sau các bài giảng ta vừa chọn trước đó?
  - Chọn bài giảng ngắn nhất trong số các bài giảng bắt đầu sau các bài giảng ta vừa chọn trước đó?
  - Chọn bài giảng kết thúc sớm nhất trong số các bài giảng bắt đầu sau các bài giảng ta vừa chọn trước đó?

Thuật toán tham lam
Lập lịch

- Chọn bài giảng bắt đầu sớm nhất? *Không xuất ra lời giải tối ưu trong mọi trường hợp*



- Chọn bài giảng ngắn nhất? *Không xuất ra lời giải tối ưu trong mọi trường hợp*



Thuật toán tham lam

Lập lịch

Thuật toán 6: Lập lịch tham lam (Greedy Scheduling)

Input: $(s_1, e_1), (s_2, e_2), \ldots, (s_n, e_n)$: thời gian bắt đầu và kết thúc bài giảng $b_1, b_2, \ldots, b_n$

Output: Danh sách bài giảng S có số bài giảng lớn nhất trong đó không có hai bài giảng nào xung đột nhau

1 Sắp xếp các bài giảng theo thứ tự tăng dần theo thời gian kết thúc và gán lại nhãn bài giảng sao cho $e_1 \leq e_2 \leq \cdots \leq e_n$

2 $S = \emptyset$

3 **for** $j := 1$ **to** n **do**

4 **if** *Bài giảng* j *không xung đột với các phần tử của* S **then**

5 $S := S \cup \{j\}$

6 **return** S

Thuật toán tham lam
Lập lịch

Định lý 1

Thuật toán 6 xuất ra một danh sách các bài giảng tối ưu

Chứng minh.

- Giả sử $S^\star$ là một danh sách bài giảng tối ưu trong đó các bài giảng $x_1, x_2, \ldots, x_k$ được sắp xếp theo thứ tự tăng dần theo thời gian kết thúc
- Giả sử S là một danh sách bài giảng xuất ra từ Thuật toán 6 trong đó các bài giảng $y_1, y_2, \ldots, y'_k$ được sắp xếp theo thứ tự tăng dần theo thời gian kết thúc
- Do $S^\star$ là tối ưu, $k \geq k'$
- Nếu $S = S^\star$ thì ta có điều phải chứng minh. Ngược lại, nếu $S \neq S'$, gọi $i \geq 1$ là chỉ số đầu tiên thỏa mãn $x_i \neq y_i$, nghĩa là

$$S^\star = \langle x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_{k'}, \ldots, x_k \rangle$$
$$S = \langle x_1, x_2, \ldots, x_{i-1}, y_i, \ldots, y_{k'} \rangle$$

Chứng minh (tiếp).

- Do y_i được chọn bởi Thuật toán 6, y_i kết thúc trước khi x_i kết thúc, nghĩa là nó không xung đột với bất kỳ bài giảng nào sau x_i trong $S^\star$.
- Do đó, dãy $S^\star \setminus \{x_i\} \cup \{y_i\}$ cũng là một dãy tối ưu. Ta gán $S^\star := S^\star \setminus \{x_i\} \cup \{y_i\}$ và lặp lại lý luận trên cho $S^\star$ và S.
- Bằng cách liên tục sử dụng lý luận trên, ta thu được dãy $S^\star$ tối ưu có dạng

$$S^\star = \langle y_1, y_2, \ldots, y_{k'}, x_{k'+1}, \ldots, x_k \rangle$$

- Do $y_{k'}$ là bài giảng cuối cùng được chọn bởi Thuật toán 6, các bài giảng còn lại đều xung đột với $y_{k'}$ và có thời gian kết thúc sau khi $y_{k'}$ kết thúc. Nói cách khác, các phần tử $x_{k'+1}, \ldots, x_k$ trong S' đều phải bằng $y_{k'}$, nghĩa là S là một dãy tối ưu

Giải và ước lượng hệ thức truy hồi
Giới thiệu

- Một ứng dụng của việc Giải hệ thức truy hồi mà chúng ta sẽ đề cập ở phần sau là trong việc phân tích độ phức tạp của các thuật toán đệ quy
- Một số phương pháp giải hệ thức truy hồi
  (1) Đoán nghiệm và chứng minh bằng phương pháp quy nạp
  (2) Sử dụng đa thức đặc trưng
  (3) Sử dụng hàm sinh
- Một số phương pháp ước lượng hệ thức truy hồi
  (1) Sử dụng cây đệ quy
  (2) Sử dụng định lý thợ (Master Theorem)

Giải hệ thức truy hồi
Đa thức đặc trưng

- *Hệ thức truy hồi tuyến tính thuần nhất bậc k với hệ số hằng (Linear homogeneous recurrence relation of degree k with constant coefficients)* là hệ thức có dạng

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

  trong đó $c_1, c_2, \ldots, c_k$ là các số thực, $c_k \neq 0$, và $n \geq k$
- Một dãy $\{a_n\}$ $(n \geq 0)$ thỏa mãn hệ thức truy hồi trên được *xác định một cách duy nhất* bởi *hệ thức này* và k *điều kiện ban đầu* $a_0 = C_0, a_1 = C_1, \ldots, a_{k-1} = C_{k-1}$
  - Hệ thức truy hồi $f_n = f_{n-1} + f_{n-2}$ là một hệ thức truy hồi tuyến tính thuần nhất bậc hai với hệ số hằng. Dãy Fibonacci $\{f_n\}$ được xác định bởi hệ thức trên và điều kiện ban đầu $f_0 = 0, f_1 = 1$
  - Các hệ thức $g_n = ng_{n-1}$, $g_n = g_{n-1} + g_{n-2}^2$ không là hệ thức truy hồi tuyến tính thuần nhất với hệ số hằng

Giải hệ thức truy hồi
Đa thức đặc trưng

- Hệ thức $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$ $(c_k \neq 0)$ có hai tính chất quan trọng
  - Nghiệm của hệ thức có dạng $a_n = r^n$ *với* $r \neq 0$ là một hằng số nào đó. Từ đó, $r^n = c_1 r^{n-1} + \cdots + c_k a r^{n-k}$. Chia hai vế cho r^{n-k} và chuyển vế, ta có

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_{k-1} r - c_k = 0.$$

    Đa thức cuối cùng gọi là *đa thức đặc trưng (characteristic equation)* của hệ thức, và nghiệm của nó gọi là *nghiệm đặc trưng (characteristic root)* của hệ thức
  - Nếu s_n và t_n thỏa mãn hệ thức truy hồi, thì *mọi tổ hợp tuyến tính của* s_n *và* t_n, nghĩa là mọi biểu thức có dạng $b_1 s_n + b_2 t_n$ với b_1, b_2 là các số thực nào đó, cũng thỏa mãn hệ thức truy hồi

Giải hệ thức truy hồi
Đa thức đặc trưng

Định lý 2

Cho các số thực c_1, c_2. Giả sử $r^2 - c_1 r - c_2$ có hai nghiệm phân biệt r_1 và r_2. Dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ $(n \geq 2)$ với điều kiện ban đầu $a_0 = C_0$ và $a_1 = C_1$ khi và chỉ khi $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ với mọi $n \geq 0$, trong đó α_1, α_2 là các hằng số nào đó

Chứng minh sơ lược.

Ta cần chứng minh hai điều

(1) Nếu r_1, r_2 là các nghiệm của đa thức đặc trưng, thì $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ với mọi $n \geq 0$, trong đó α_1, α_2 là các hằng số nào đó, thỏa mãn hệ thức truy hồi

(2) Nếu $\{a_n\}$ là nghiệm của hệ thức truy hồi với các điều kiện ban đầu $a_0 = C_0$ và $a_1 = C_1$, thì tồn tại các hằng số α_1, α_2 sao cho $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$

Giải hệ thức truy hồi
Đa thức đặc trưng

Tương tự, ta có

Định lý 3

Cho các số thực $c_1, c_2, \ldots, c_k$. Giả sử $r^k - c_1 r^{k-1} - \cdots - c_{k-1} r - c_k$ có k nghiệm phân biệt $r_1, r_2, \ldots, r_k$. Dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ $(n \geq k)$ với điều kiện ban đầu $a_0 = C_0, a_1 = C_1, \ldots, a_{k-1} = C_{k-1}$ khi và chỉ khi $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$ với mọi $n \geq 0$, trong đó $\alpha_1, \alpha_2, \ldots, \alpha_k$ là các hằng số nào đó

Giải hệ thức truy hồi
Đa thức đặc trưng

Tổng quát cho trường hợp đa thức đặc trưng có nghiệm bội

Định lý 4

Cho các số thực $c_1, c_2, \ldots, c_k$. *Giả sử* $r^k - c_1 r^{k-1} - \cdots - c_{k-1} r - c_k$ *có* t *nghiệm phân biệt* $r_1, r_2, \ldots, r_t$ *với các bội tương ứng* $m_1, m_2, \ldots, m_t$ *thỏa mãn* $m_i \geq 1$ *với* $1 \leq i \leq t$ *và* $m_1 + \cdots + m_t = k$ *(nghĩa là, có* m_i *nghiệm có giá trị* r_i*). Dãy* $\{a_n\}$ *là nghiệm của hệ thức truy hồi* $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ $(n \geq k)$ *với điều kiện ban đầu* $a_0 = C_0, a_1 = C_1, \ldots, a_{k-1} = C_{k-1}$ *khi và chỉ khi*

$$\begin{aligned} a_n = &(\alpha_{1,0} + \alpha_{1,1} n + \cdots + \alpha_{1,m_1-1} n^{m_1-1}) r_1^n \\ &+ (\alpha_{2,0} + \alpha_{2,1} n + \cdots + \alpha_{2,m_2-1} n^{m_2-1}) r_2^n \\ &+ \cdots + (\alpha_{t,0} + \alpha_{t,1} n + \cdots + \alpha_{t,m_t-1} n^{m_t-1}) r_t^n \end{aligned}$$

với $n \geq 0$, *trong đó* $\alpha_{i,j}$ *là các hằng số với* $1 \leq i \leq t$ *và* $0 \leq j \leq m_i - 1$

Ví dụ 1

Giải hệ thức truy hồi $f_n = f_{n-1} + f_{n-2}$ $(n \geq 2)$ với điều kiện ban đầu $f_0 = 0$ và $f_1 = 1$

- Đa thức đặc trưng của hệ thức truy hồi là $r^2 - 2r - 1$
- Đa thức đặc trưng có hai nghiệm phân biệt $r_1 = \frac{1+\sqrt{5}}{2}$ và $r_2 = \frac{1-\sqrt{5}}{2}$
- Do đó, nếu dãy $\{f_n\}$ là nghiệm của hệ thức truy hồi thì $f_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ với các hằng số α_1, α_2 nào đó
- $\{f_n\}$ cần thỏa mãn điều kiện ban đầu $f_0 = 0$ và $f_1 = 1$. Thay vào dạng tổng quát của f_n, ta có hệ phương trình

$$f_0 = \alpha_1 + \alpha_2 = 0$$

$$f_1 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right) + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)$$

Từ đó $\alpha_1 = 1/\sqrt{5}$ và $\alpha_2 = -1/\sqrt{5}$

Ví dụ 2

Giải hệ thức truy hồi $a_n = 6a_{n-1} - 9a_{n-2}$ với các điều kiện ban đầu $a_0 = 1$ và $a_1 = 6$

- Đa thức đặc trưng của hệ thức truy hồi là $r^2 - 6r + 9$
- Đa thức đặc trưng có một nghiệm $r_1 = 3$ với bội 2
- Do đó, nếu dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi thì $a_n = \alpha_{1,0} r_1^n + \alpha_{1,1} n r_1^n = \alpha_{1,0} 3^n + \alpha_{1,1} n 3^n$ với các hằng số $\alpha_{1,0}, \alpha_{1,1}$ nào đó
- Dãy $\{a_n\}$ cũng cần thỏa mãn điều kiện ban đầu $a_0 = 1$ và $a_1 = 6$. Thay vào dạng tổng quát của a_n, ta có hệ phương trình

$$a_0 = \alpha_{1,0} = 1$$
$$a_1 = \alpha_{1,0} \cdot 3 + \alpha_{1,1} \cdot 3 = 6$$

Do đó, ta có $\alpha_{1,0} = 1$ và $\alpha_{1,1} = 1$

Ví dụ 3

Giải hệ thức truy hồi $a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3}$ với các điều kiện ban đầu $a_0 = 1$, $a_1 = -2$, và $a_2 = -1$

- Đa thức đặc trưng của hệ thức là $r^3 + 3r^2 + 3r + 1$
- Đa thức đặc trưng có một nghiệm $r_1 = -1$ với bội 3
- Do đó, nếu $\{a_n\}$ là nghiệm của hệ thức truy hồi thì $a_n = \alpha_{1,0}r_1^n + \alpha_{1,1}nr_1^n + \alpha_{1,2}n^2r^n$
- Dãy $\{a_n\}$ cũng cần thỏa mãn điều kiện ban đầu $a_0 = 1$, $a_1 = -2$, và $a_2 = -1$. Thay vào dạng tổng quát của a_n, ta có hệ phương trình

$$
\begin{aligned}
a_0 &= \alpha_{1,0} = 1 \\
a_1 &= -\alpha_{1,0} - \alpha_{1,1} - \alpha_{1,2} = -2 \\
a_2 &= \alpha_{1,0} + 2\alpha_{1,1} + 4\alpha_{1,2} = -1
\end{aligned}
$$

Do đó ta có $\alpha_{1,0} = 1$, $\alpha_{1,1} = 3$, và $\alpha_{1,2} = -2$

Giải hệ thức truy hồi
Đa thức đặc trưng

- *Hệ thức truy hồi tuyến tính không thuần nhất bậc k với hệ số hằng (Linear nonhomogeneous recurrence relation of degree k with constant coefficients)* là hệ thức có dạng

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

  trong đó $c_1, c_2, \ldots, c_k$ là các số thực, $c_k \neq 0$, $F(n)$ là một hàm chỉ phụ thuộc vào n và không phải luôn bằng 0, và $n \geq k$
- Hệ thức $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ được gọi là *hệ thức truy hồi thuần nhất tương ứng (associated homogeneous recurrence relation)* của hệ thức trên

Định lý 5

Nếu $\{a_n^{(p)}\}$ là một nghiệm riêng nào đó của hệ thức $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n)$ thì mọi nghiệm của hệ thức đó có dạng $\{a_n^{(p)} + a_n^{(h)}\}$, trong đó $a_n^{(h)}$ là nghiệm của hệ thức thuần nhất tương ứng $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$

Giải hệ thức truy hồi

Đa thức đặc trưng

Với một số dạng $F(n)$, nghiệm riêng có dạng đặc biệt

Định lý 6

Giả sử $\{a_n\}$ thỏa mãn hệ thức

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

trong đó $c_1, \ldots, c_k$ là các số thực, và

$$F(n) = (b_t n^t + b_{t-1} n^{t-1} + \cdots + b_1 n + b_0) s^n,$$

trong đó $b_0, \ldots, b_t$ và s là các số thực. Khi s không phải là nghiệm của đa thức đặc trưng của hệ thức thuần nhất tương ứng, tồn tại một nghiệm riêng có dạng $(p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n$. Khi s là một nghiệm với bội m của đa thức đặc trưng của hệ thức thuần nhất tương ứng, tồn tại một nghiệm riêng có dạng $n^m (p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n$.

Giải hệ thức truy hồi
Đa thức đặc trưng

Ví dụ 4

Giải hệ thức truy hồi $a_n = a_{n-1} + n$ $(n \geq 2)$ với điều kiện ban đầu $a_1 = 1$

- Hệ thức thuần nhất tương ứng là $a_n = a_{n-1}$ $(n \geq 2)$. Hệ thức này có nghiệm đặc trưng $r = 1$ và dãy $\{a_n^{(h)}\}$ với $a_n^{(h)} = c \cdot (1)^n$ là một dãy thỏa mãn hệ thức, trong đó c là hằng số nào đó
- Ta có $F(n) = n = (1 \cdot n + 0) \cdot 1^n$. Vì $s = 1$ là nghiệm đặc trưng của hệ thức thuần nhất tương ứng với bội 1, một nghiệm riêng của hệ thức không thuần nhất đã cho có dạng $a_n^{(p)} = n(p_1 n + p_0)1^n = p_1 n^2 + p_0 n$
- Thay dạng của nghiệm riêng vào hệ thức đã cho, ta có $n(2p_1 - 1) + (p_0 - p_1) = 0$, nghĩa là $2p_1 - 1 = 0$ và $p_0 - p_1 = 0$, do đó $p_0 = p_1 = 1/2$
- Cuối cùng, hệ thức ban đầu có nghiệm dạng $a_n = a_n^{(p)} + a_n^{(h)} = n(n+1)/2 + c$. Thay vào điều kiện ban đầu $a_1 = 1$ ta được $c = 0$

Bài tập 1

Giải các hệ thức truy hồi với điều kiện ban đầu sau

(1) $a_n = 2a_{n-1}$ *với* $n \geq 1, a_0 = 3$

(2) $a_n = 5a_{n-1} - 6a_{n-2}$ *với* $n \geq 2, a_0 = 1, a_1 = 0$

(3) $a_n = 4a_{n-1} - 4a_{n-2}$ *với* $n \geq 2, a_0 = 6, a_1 = 8$

(4) $a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3}$ *với* $n \geq 3, a_0 = 5, a_1 = -9, a_2 = 15$

(5) $a_n = 3a_{n-1} + 2^n$ *với* $n \geq 1, a_0 = 1$

(6) $a_n = 2a_{n-1} + 2n^2$ *với* $n \geq 2, a_1 = 5$

(7) $a_n = 5a_{n-1} - 6a_{n-2} + 2^n + 3n$ (***Gợi ý:*** *Tìm nghiệm riêng có dạng* $qn2^n + p_1 n + p_2$, *trong đó* q, p_1, p_2 *là các hằng số*)

Hàm sinh

Hàm sinh (generating function) $G_a(x)$ của một dãy vô hạn $\{a_n\}$ $(n \geq 0)$ được định nghĩa như sau

$$G_a(x) := \sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k + \ldots$$

Nói cách khác, a_n là hệ số của x^n trong $G_a(x)$

Ý tưởng: Giả sử dãy $\{a_n\}$ được định nghĩa bởi hệ thức truy hồi và điều kiện ban đầu nào đó. Trong nhiều trường hợp, ta có thể tìm được công thức tường minh cho $G_a(x)$, và từ đó suy ra nghiệm của hệ thức truy hồi tương ứng

Giải hệ thức truy hồi
Hàm sinh

Định lý 7

Cho $f(x) = \sum_{n=0}^{\infty} a_n x^n$ *và* $g(x) = \sum_{n=0}^{\infty} b_n x^n$. *Ta có*

$$f(x) + g(x) = \sum_{n=0}^{\infty} (a_n + b_n) x^n$$

$$f(x)g(x) = \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} a_k b_{n-k} \right) x^n$$

Chú ý: Định lý trên chỉ đúng cho các chuỗi lũy thừa hội tụ trong một khoảng nào đó, và tất cả các chuỗi chúng ta sẽ xét trong bài giảng này đều thỏa mãn điều kiện đó. Tuy nhiên, ngay cả khi các chuỗi không hội tụ, định lý trên có thể được sử dụng như là định nghĩa cho các phép cộng và nhân các hàm sinh

Ví dụ 5

Giải hệ thức truy hồi $a_n = 3a_{n-1}$ $(n \geq 1)$ với điều kiện ban đầu $a_0 = 2$

$$G_a(x) = \sum_{n=0}^{\infty} a_n x^n = a_0 + \sum_{n=1}^{\infty} (3a_{n-1})x^n$$

$$= 2 + 3x \sum_{n=1}^{\infty} a_{n-1}x^{n-1} = 2 + 3x \sum_{m=0}^{\infty} a_m x^m$$

$$= 2 + 3xG_a(x)$$

$$G_a(x) = \frac{2}{1-3x}$$

Nhắc lại rằng $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ với $-1 < x < 1$. Suy ra

$\sum_{n=0}^{\infty} c^n x^n = \frac{1}{1-cx}$ với $-1 < cx < 1$ trong đó $c \neq 0$ là hằng số nào đó

Sử dụng đẳng thức trên, ta có thể viết

$$G_a(x) = \frac{2}{1-3x} = 2\sum_{n=0}^{\infty} 3^n x^n = \sum_{n=0}^{\infty} (2 \cdot 3^n)x^n$$

Suy ra $a_n = 2 \cdot 3^n$

Ví dụ 6

Dãy Fibonacci $\{f_n\}$ cho bởi hệ thức $f_n = f_{n-1} + f_{n-2}$ $(n \geq 2)$ và điều kiện ban đầu $f_0 = 0$ và $f_1 = 1$

$$G_f(x) = \sum_{n=0}^{\infty} f_n x^n = f_0 + f_1 x + \sum_{n=2}^{\infty} (f_{n-1} + f_{n-2}) x^n$$

$$= x + x \sum_{n=2}^{\infty} f_{n-1} x^{n-1} + x^2 \sum_{n=2}^{\infty} x^{n-2}$$

$$= x + x \sum_{m=1}^{\infty} f_m x^m + x^2 \sum_{m=0}^{\infty} f_m x^m \qquad \text{đổi biến}$$

$$= x + x \sum_{m=0}^{\infty} f_m x^m + x^2 \sum_{m=0}^{\infty} f_m x^m \qquad f_0 x^0 = 0$$

$$= x + x G_f(x) + x^2 G_f(x)$$

$$G_f(x) = \frac{x}{1 - x - x^2}$$

Giải hệ thức truy hồi
Hàm sinh

Nhắc lại rằng $\sum_{n=0}^{\infty} c^n x^n = \frac{1}{1-cx}$ với $-1 < cx < 1$ trong đó $c \neq 0$ là hằng số nào đó

Bài tập 2

- *Viết* $G_f(x) = \frac{x}{1-x-x^2} = \frac{a}{1-Ax} + \frac{b}{1-Bx}$ *với các hằng số* a, b, A, B *nào đó*
- *Áp dụng công thức trên để đưa* $G_f(x)$ *về dạng* $a\sum_{n=0}^{\infty} A^n x^n + b\sum_{n=0}^{\infty} B^n x^n$ *với* $-1 < Ax < 1$ *và* $-1 < Bx < 1$
  (≡ Khai triển $G_f(x)$ *thành chuỗi lũy thừa)*
- *Từ định nghĩa hàm sinh, suy ra công thức cho dãy* $\{f_n\}$

Giải hệ thức truy hồi
Hàm sinh

Chú ý rằng từ $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ với $-1 < x < 1$, bằng cách lấy đạo hàm hai vế và đổi chỉ số lấy tổng, ta có

$$\sum_{n=1}^{\infty} nx^{n-1} = \sum_{m=0}^{\infty} (m+1)x^m = \frac{1}{(1-x)^2}$$

Bài tập 3

Giải hệ thức truy hồi $a_n = 2a_{n-1} - a_{n-2}$ với $n \geq 2$ và các điều kiện ban đầu $a_0 = 1$, $a_1 = 1$ bằng cách sử dụng hàm sinh

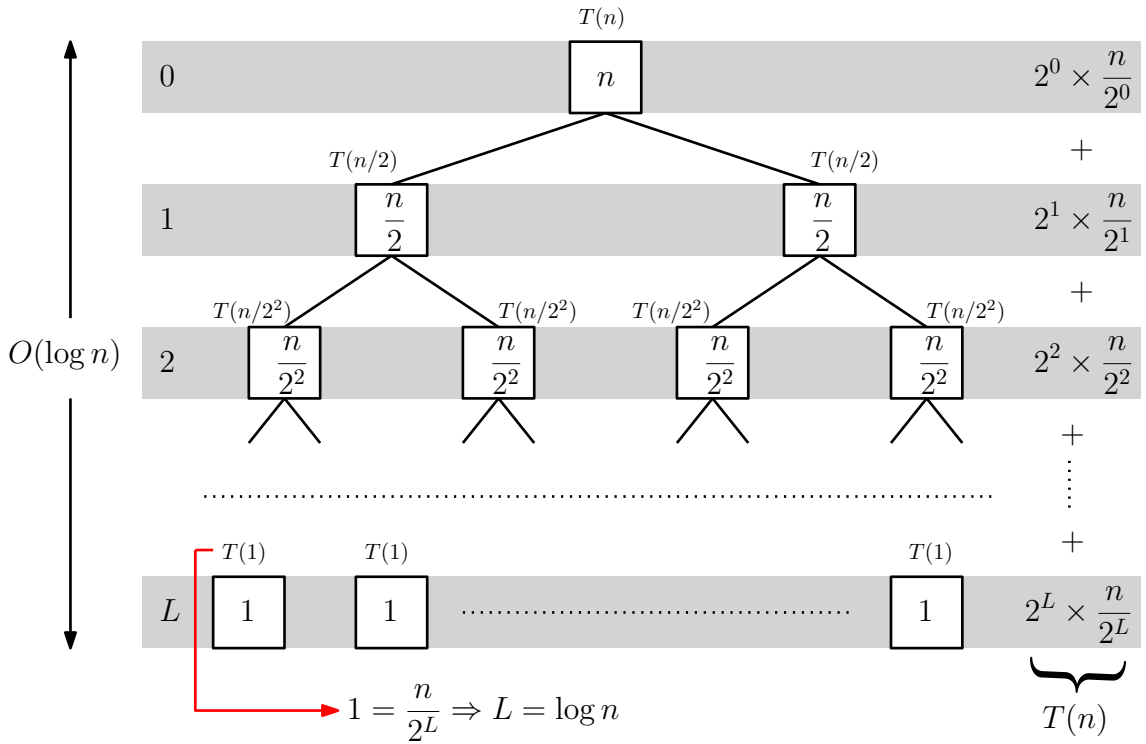
Bài tập 4 ($\star$)

Giải hệ thức truy hồi $a_n = 3a_{n-1} + n$ với $n \geq 1$ và điều kiện ban đầu $a_0 = 1$ bằng cách sử dụng hàm sinh

Ước lượng hệ thức truy hồi
Cây đệ quy

Ví dụ 7

Xét hệ thức truy hồi $T(n) = 2T(n/2) + n$ với điều kiện ban đầu $T(1) = 1$ và $n = 2^k$ với số nguyên $k \geq 1$ nào đó. Ta vẽ cây đệ quy cho hệ thức này như sau

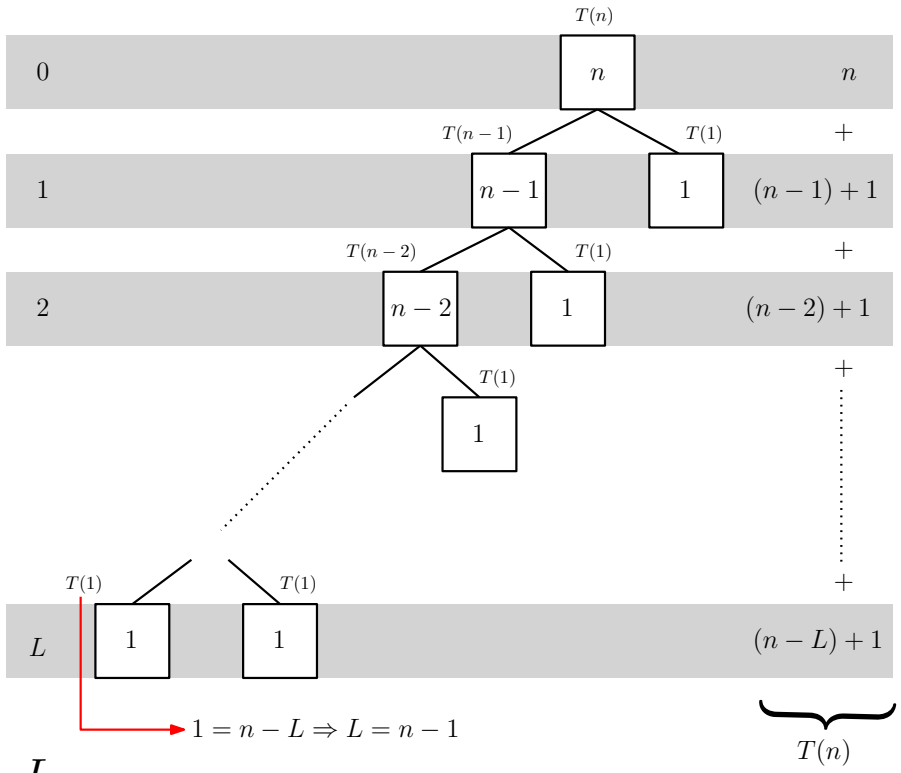


Ta có $T(n) = \sum_{i=0}^{L} 2^i \times \frac{n}{2^i} = n(\log n + 1) = O(n \log n)$

Ước lượng hệ thức truy hồi
Cây đệ quy

Ví dụ 8

Xét hệ thức truy hồi $T(n) = T(n-1) + T(1) + n$ với $n \geq 1$ và điều kiện ban đầu $T(1) = 1$. Ta vẽ cây đệ quy của hệ thức này như sau



Ta có $T(n) \leq \sum_{i=0}^{L} n = O(n^2)$

Ước lượng hệ thức truy hồi
Định lý thợ

Định lý 8: Định lý thợ (Master Theorem)

Gọi f là một hàm tăng thỏa mãn hệ thức truy hồi

$$f(n) = af(n/b) + cn^d$$

trong đó $n = b^k$ với k là số nguyên dương nào đó, $a \geq 1$, b là số nguyên dương lớn hơn 1, và c, d là các số thực với c dương và d không âm. Ta có

$$f(n) \text{ là } \begin{cases} O(n^d) & \text{nếu } a < b^d \\ O(n^d \log n) & \text{nếu } a = b^d \\ O(n^{\log_b a}) & \text{nếu } a > b^d \end{cases}$$

Ví dụ 9

- Với $T(n) = 2T(n/2) + n$, ta có $T(n) = O(n \log n)$
- Với $T(n) = T(n/2) + n$, ta có $T(n) = O(n)$
- Với $T(n) = 3T(n/2) + n$, ta có $T(n) = O(n^{\log 3})$

Thuật toán đệ quy
Giới thiệu

- Định nghĩa theo đệ quy không những có thể áp dụng cho các hàm và tập hợp mà còn cho cả các thuật toán
- Một *thủ tục đệ quy (recursive procedure)* là một thủ tục gọi chính nó
- Một *thuật toán đệ quy (recursive algorithm)* là một thuật toán giải một bài toán bằng cách chuyển về việc giải chính bài toán đó nhưng với đầu vào có kích thước nhỏ hơn
  - *Kỹ thuật chia để trị (divide-and-conquer technique):* giải một bài toán ban đầu thông qua việc chia nó thành các bài toán nhỏ hơn cùng loại và giải chúng

Thuật toán đệ quy
Giới thiệu

Một thuật toán đệ quy thường có dạng như sau

Trường hợp cơ sở Với một số đầu vào kích thước nhỏ hoặc một số trường hợp đặc biệt, thuật toán sẽ cho ra kết quả một cách trực tiếp

Định nghĩa bài toán con Trong trường hợp đầu vào khác với những đầu vào định nghĩa trong trường hợp cơ sở, thuật toán định nghĩa một hoặc nhiều "bài toán con" với các đầu vào nhỏ hơn được tính từ đầu vào ban đầu

Giải bài toán con Thuật toán gọi chính nó để giải các bài toán con và lưu trữ các kết quả tính toán

Xuất ra lời giải Sau khi giải quyết toàn bộ các bài toán con, thuật toán xuất ra lời giải dựa trên đầu vào ban đầu và các lời giải của các bài toán con

Thuật toán đệ quy
Giới thiệu

Chứng minh tính đúng đắn của thuật toán đệ quy bằng quy nạp mạnh

- **Phát biểu điều cần chứng minh:** Một điểm quan trọng là cần *chỉ rõ "thuật toán đúng" nghĩa là gì*
- **Bước cơ sở:** Các trường hợp khi *thuật toán cho ra kết quả một cách trực tiếp* mà không cần thông qua gọi đệ quy chính nó là các trường hợp cần xét trong bước cơ sở
  - Sử dụng mô tả của thuật toán để chỉ ra thuật toán sẽ trả lại gì trong trường hợp cơ sở
  - Chỉ ra rằng giá trị trả lại của thuật toán là đúng
- **Bước quy nạp:** Giả thiết rằng thuật toán đúng cho *mọi đầu vào kích thước nhỏ hơn*. Chỉ ra rằng thuật toán cũng đúng cho đầu vào hiện tại
  - Phát biểu giả thiết quy nạp: Giả sử thuật toán đúng với mọi đầu vào giữa trường hợp cơ sở và các đầu vào có kích thước nhỏ hơn một đơn vị so với đầu vào hiện tại
  - Mô tả cụ thể thuật toán trả lại gì với đầu vào hiện tại dựa trên các lần gọi đệ quy
  - Sử dụng giả thiết quy nạp để thay mỗi lần gọi đệ quy bằng đáp án chính xác. Chỉ ra rằng những điều này dẫn tới đáp án đúng cho trường hợp hiện tại
  - Nếu bạn xét nhiều trường hợp trong thuật toán thì cần thực hiện hai điều trên với từng trường hợp một

Thuật toán đệ quy
Tính giai thừa

Với mọi số nguyên không âm n

$$0! = 1$$
$$n! = n \times (n-1)! \quad \forall n \geq 1$$

Thuật toán 7: Tính $n!$

Input: n: số nguyên không âm
Output: $n!$

```
1 procedure factorial(n):
2     if n = 0 then
3         return 1
4     else
5         return n × factorial(n − 1)
```

Thuật toán đệ quy
Tính giai thừa

Ví dụ 10 (Thuật toán đệ quy tính giai thừa là đúng)

Ta chứng minh tính đúng đắn của Thuật toán 7 bằng quy nạp mạnh. Gọi $\texttt{factorial}(n)$ là giá trị trả lại bởi Thuật toán 7

- Ta chứng minh $\texttt{factorial}(n) = n!$ với mọi $n \geq 0$
- **Bước cơ sở:** Khi $n = 0$, $\texttt{factorial}(n) = 1 = n!$
- **Bước quy nạp:** Giả sử $\texttt{factorial}(j) = j!$ với mọi j thỏa mãn $0 \leq j \leq k$ với số nguyên $k \geq 0$ nào đó. Ta chứng minh $\texttt{factorial}(k+1) = (k+1)!$. Thật vậy, Thuật toán 7 trả lại $\texttt{factorial}(k+1) = (k+1) \times \texttt{factorial}(k)$. Theo giả thiết quy nạp, $\texttt{factorial}(k) = k!$. Do đó, $\texttt{factorial}(k+1) = (k+1) \times k! = (k+1)!$

Ví dụ 11 (Thời gian chạy của thuật toán đệ quy tính giai thừa)

$$T(n) = \max\{O(1), T(n-1) + O(1)\} + O(1) = T(n-1) + O(1)$$

Thuật toán đệ quy

Tính lũy thừa

Với mọi số thực $a \neq 0$ và số nguyên không âm n,

$$a^0 = 1$$
$$a^n = a \times a^{n-1} \quad \forall n \geq 1$$

Thuật toán 8: Tính a^n

Input: a: số thực khác 0, n: số nguyên không âm
Output: a^n

```
1 procedure power(a, n):
2     if n = 0 then
3         return 1
4     else
5         return a × power(a, n − 1)
```

Thuật toán đệ quy

Tìm kiếm tuyến tính

Thuật toán 9: Tìm kiếm tuyến tính (Linear Search)

Input: $a_1, a_2, \ldots, a_n$: dãy số nguyên, i, j, x: số nguyên, $1 \leq i \leq j \leq n$

Output: Nếu $x \in \{a_i, a_{i+1}, \ldots, a_j\}$ thì trả lại $k \in \{i, i+1, \ldots, j\}$ sao cho $x = a_k$. Ngược lại thì trả lại 0

```
1 procedure LinearSearch(i, j, x):
2     if a_i = x then  // Ở đúng vị trí? Trả lại kết quả
3         return i
4     else
5         if i = j then                    // Không tìm thấy
6             return 0
7         else                             // Tìm trong phần còn lại
8             return LinearSearch(i + 1, j, x)
```

Thuật toán đệ quy

Tìm kiếm nhị phân

Thuật toán 10: Tìm kiếm nhị phân (Binary Search)

Input: $a_1, a_2, \ldots, a_n$: dãy số nguyên thực sự tăng, i, j, x: số nguyên, $1 \leq i \leq j \leq n$

Output: Nếu $x \in \{a_i, a_{i+1}, \ldots, a_j\}$ thì trả lại $k \in \{i, i+1, \ldots, j\}$ sao cho $x = a_k$. Ngược lại thì trả lại 0

```
1  procedure BinarySearch(i, j, x):
2      m := ⌊(i + j)/2⌋                          // Đi đến giữa dãy
3      if x = a_m then                            // Đúng vị trí?
4          return m
5      else
6          if x < a_m và i < m then               // x ở nửa bên trái?
7              return BinarySearch(i, m − 1, x)
8          else
9              if x > a_m và j > m then           // x ở nửa bên phải?
10                 return BinarySearch(m + 1, j, x)
11             else                               // Không tìm thấy
12                 return 0
```

Thuật toán đệ quy

Sắp xếp trộn

Thuật toán 11: Sắp xếp trộn (Merge Sort)

Input: $L = a_1, a_2, \ldots, a_n$: dãy số nguyên
Output: Dãy số nguyên sắp thứ tự tăng dần

```
1 procedure MergeSort(L):
2     if n > 1 then
3         m := ⌊n/2⌋
4         L1 := a1, ..., am
5         L2 := am+1, ..., an
6         L := Merge(MergeSort(L1), MergeSort(L2))
```

Thuật toán đệ quy

Sắp xếp trộn

Thuật toán 12: Trộn hai dãy sắp thứ tự (Merge)

Input: $A = (a_1, \ldots, a_{|A|}), B = (b_1, \ldots, b_{|B|})$: dãy số nguyên sắp thứ tự

Output: Dãy các số nguyên trong cả A và B sắp thứ tự tăng dần

```
1 procedure Merge(A, B):
2     if A rỗng then
3         return B
4     if B rỗng then
5         return A
6     if a_1 < b_1 then
7         return (a_1, Merge(a_2, ..., a_|A|, B))
8     else
9         return (b_1, Merge(A, b_2, ..., b_|B|))
```