VNU-HUS MAT3500: Toán rời rạc

Thuật toán II

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

Thuật toán tham lam
- Giới thiệu
- Ví dụ

Ước lượng hệ thức truy hồi
- Giới thiệu
- Định lý thợ
- Cây đệ quy

Thuật toán đệ quy
- Giới thiệu
- Ví dụ

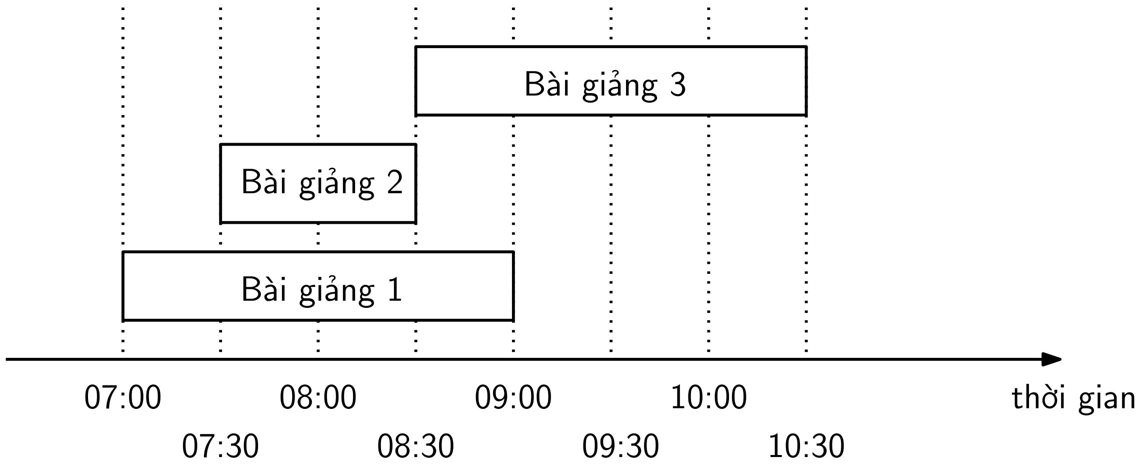
Thuật toán tham lam

Giới thiệu

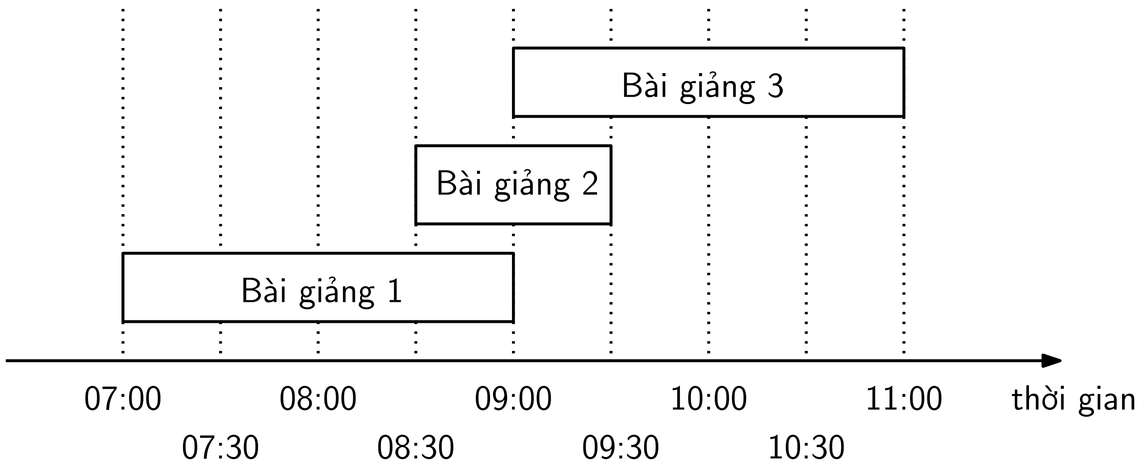
- *Các bài toán tối ưu (optimization problems)* yêu cầu cực đại hóa hoặc cực tiểu hóa một số tham số xét trên tập tất cả các đầu vào có thể
  - Tìm đường đi giữa hai thành phố với khoảng cách *nhỏ nhất*
  - Tìm cách mã hóa các thông điệp sử dụng số lượng bit *nhỏ nhất* có thể
- Một *thuật toán tham lam (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.

- **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?

- 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

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

$S := \emptyset$

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

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

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

return S

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 $x_1, x_2, \dots, x_k$ 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 $y_1, y_2, \dots, y_{k'}$ 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 là chỉ số đầu tiên trong $\{1, \dots, k'\}$ thỏa mãn $x_i \neq y_i$, nghĩa là

$$S^\star = \langle x_1 = y_1, x_2 = y_2, \dots, x_{i-1} = y_{i-1}, x_i, \dots, x_{k'}, \dots, x_k \rangle$$
$$S = \langle x_1 = y_1, x_2 = y_2, \dots, x_{i-1} = y_{i-1}, y_i, \dots, y_{k'} \rangle$$

Chứng minh (tiếp).

- Nếu i không tồn tại thì $S^\star = \langle y_1, y_2, \ldots, y_{k'}, x_{k'+1}, \ldots, x_k \rangle$
- Ngược lại, 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 $S^\star$ đều phải bằng $y_{k'}$, nghĩa là $S = \langle y_1, y_2, \ldots, y_{k'} \rangle$ là một dãy tối ưu

- Một ứng dụng của việc ước lượng hệ thức truy hồi là trong việc phân tích thời gian chạy của các thuật toán đệ quy
- Một số phương pháp ước lượng hệ thức truy hồi
  (1) Sử dụng cây đệ quy
  (2) Sử dụng định lý thợ (Master Theorem)

Định lý 2: 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 1

- 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 $T(n) = O(n^{\log 3})$

Bài tập 1

Sử dụng Định lý thợ, hãy ước lượng các hệ thức truy hồi sau theo O-lớn, giả sử $T(1) = 1$

(a) $T(n) = 4T(n/3) + n^2$

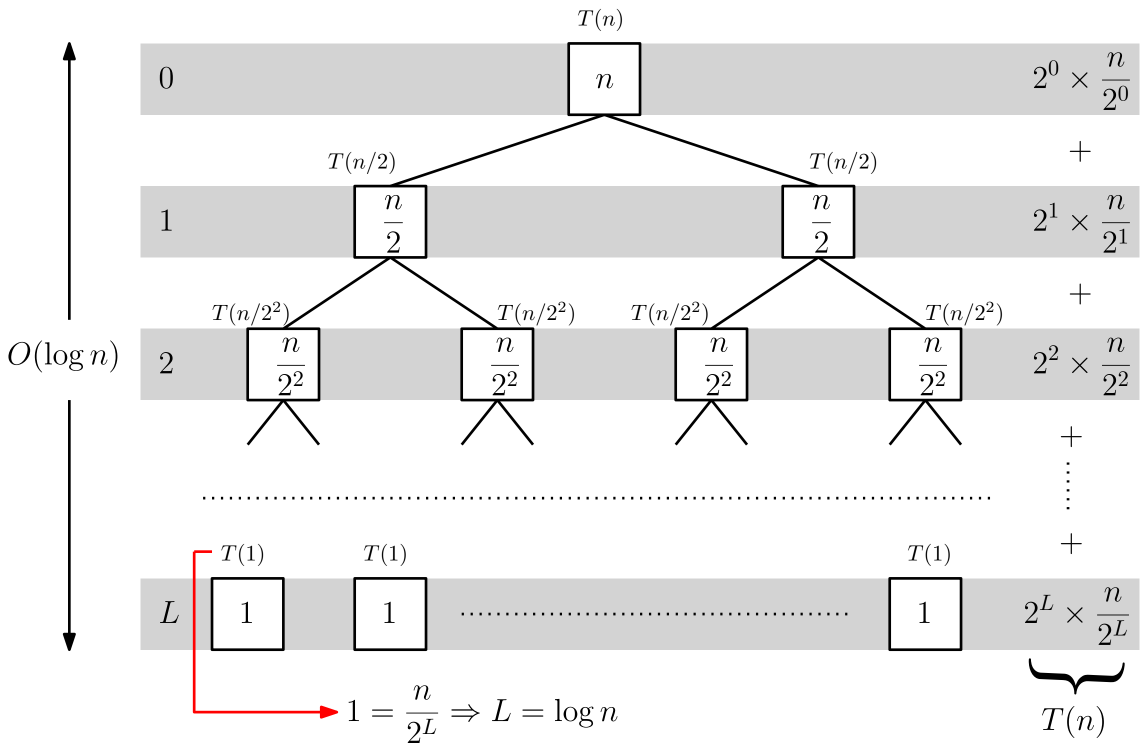
(b) $T(n) = 4T(n/2) + n^2$

(c) $T(n) = 3T(n/3) + n$

(d) $T(n) = 3T(n/3) + 1$

Ví dụ 2

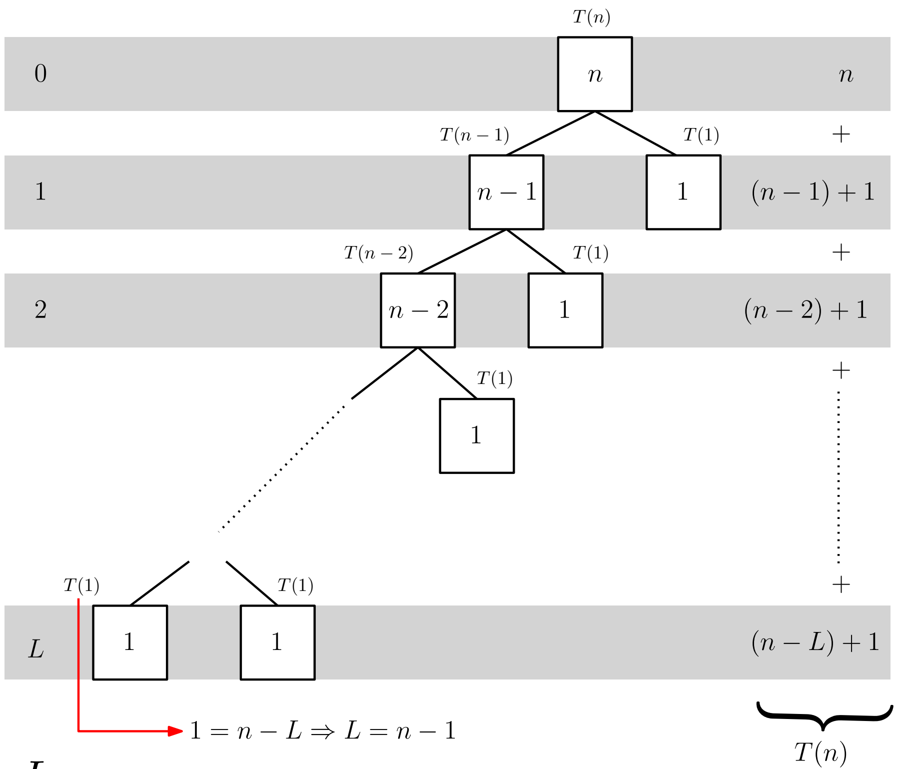
Xét hệ thức truy hồi $T(n) = 2T(n/2) + n$ với điều kiện ban đầu $T(1) = 1$ và $n = 2^k$ với số nguyên $k \geq 1$ nào đó. Ta vẽ cây đệ quy cho hệ thức này như sau



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

Ví dụ 3

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

Bài tập 2

Sử dụng cây đệ quy để ước lượng $T(n)$ cho bởi các hệ thức truy hồi sau

(a) $T(n) = 2T(n/2) + n^2$

(b) $T(n) = T(n/2) + 1$

(c) $T(n) = 2T(n-1) + 1$

Bài tập 3

Hệ thức $T(n) = 2T(n/2) + n \log n$ không thỏa mãn các điều kiện của Định lý thợ nên ta không thể ước lượng nó thông qua Định lý. Tuy nhiên, ta vẫn có thể sử dụng cây đệ quy để ước lượng $T(n)$ trong trường hợp này. Hãy sử dụng cây đệ quy để chỉ ra $T(n) = O(n \log^2 n)$

Bài tập 4

Ta có thể chứng minh Định lý thợ bằng cách sử dụng cây đệ quy

(a) *Vẽ cây đệ quy cho $T(n) = aT(n/b) + cn^d$ 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*

(b) *Tính tổng từng hàng và viết công thức của $T(n)$ dưới dạng tổng của các hàng trong cây.*

(c) *Xét các trường hợp $a < b^d$, $a = b^d$, và $a > b^d$*

- 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

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

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: $n!$

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

Ví dụ 4 (Thuật toán đệ quy tính giai thừa là đúng)

Ta chứng minh tính đúng đắn của Thuật toán 7 bằng quy nạp. Gọi `factorial`(n) là giá trị trả lại bởi Thuật toán 7

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

Ví dụ 5 (Thời gian chạy của thuật toán đệ quy tính giai thừa)

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

nghĩa là tồn tại hằng số C thỏa mãn $T(n) = T(n-1) + C$. Suy ra $T(n) = O(n)$

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

procedure $\texttt{power}(a, n)$:
    if $n = 0$ **then**
        return 1
    else
        return $a \times \texttt{power}(a, n-1)$

Bài tập 5

(a) *Chứng minh tính đúng đắn của Thuật toán 8*

(b) *Xây dựng hệ thức truy hồi để đánh giá thời gian chạy của Thuật toán 8. Giải hoặc ước lượng hệ thức bạn tìm được*

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

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

Bài tập 6

Xây dựng hệ thức truy hồi để đánh giá thời gian chạy của Thuật toán 9. Giải hoặc ước lượng hệ thức bạn tìm được

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

procedure `BinarySearch`(i, j, x):
- $m := \lfloor (i+j)/2 \rfloor$ `// Đi đến giữa dãy`
- **if** $x = a_m$ **then** `// Đúng vị trí?`
  - **return** m
- **else**
  - **if** $x < a_m$ và $i < m$ **then** `// ` x ` ở nửa bên trái?`
    - **return** `BinarySearch`$(i, m-1, x)$
  - **else**
    - **if** $x > a_m$ và $j > m$ **then** `// ` x ` ở nửa bên phải?`
      - **return** `BinarySearch`$(m+1, j, x)$
    - **else** `// Không tìm thấy`
      - **return** 0

Bài tập 7

Xây dựng hệ thức truy hồi để đánh giá thời gian chạy của Thuật toán 10. Giải hoặc ước lượng hệ thức bạn tìm được

Thuật toán 11: Trộn hai dãy sắp thứ tự (Merge)

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

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

```
procedure Merge(A, B):
    if A = ∅ then
        return B
    if B = ∅ then
        return A
    if a_1 < b_1 then
        return (a_1, Merge(a_2, ..., a_|A|, B))
    else
        return (b_1, Merge(A, b_2, ..., b_|B|))
```

Thuật toán 12: 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

procedure `MergeSort`(L)**:**
    if $n > 1$ **then**
        $m := \lfloor n/2 \rfloor$
        $L_1 := a_1, \ldots, a_m$
        $L_2 := a_{m+1}, \ldots, a_n$
        $L :=$ `Merge(MergeSort(`L_1`), MergeSort(`L_2`))`