Thuật toán

Khoa CNTT

Trường Đại học Phenikaa

Thuật toán (4 tiết)

1. Định nghĩa và các tính chất

2. Độ phức tạp của thuật toán

1. Định nghĩa và các tính chất

- **Định nghĩa:**
  - Trong toán học và khoa học máy tính, một thuật toán, còn gọi là giải thuật, là một tập hợp hữu hạn các hướng dẫn được xác định rõ ràng, có thể thực hiện được bằng máy tính, thường để giải quyết một lớp vấn đề hoặc để thực hiện một phép tính.
  - Là một phương pháp hiệu quả, một thuật toán có thể được biểu diễn trong một khoảng không gian và thời gian hữu hạn, và bằng một ngôn ngữ hình thức được xác định rõ ràng.

Ví dụ:

Algorithm 1 Tìm phần tử lớn nhất trong một tập hữu hạn phần tử

Input: Tập số nguyên $(a_1, a_2, ..., a_n)$

1: max $:= a_1$
2: Xét lần lượt $j = 1$ đến n:
3: Nếu max $< a_j$ thì max $:= a_j$

Output: max (số nguyên lớn nhất trong tập số nguyên đã cho)

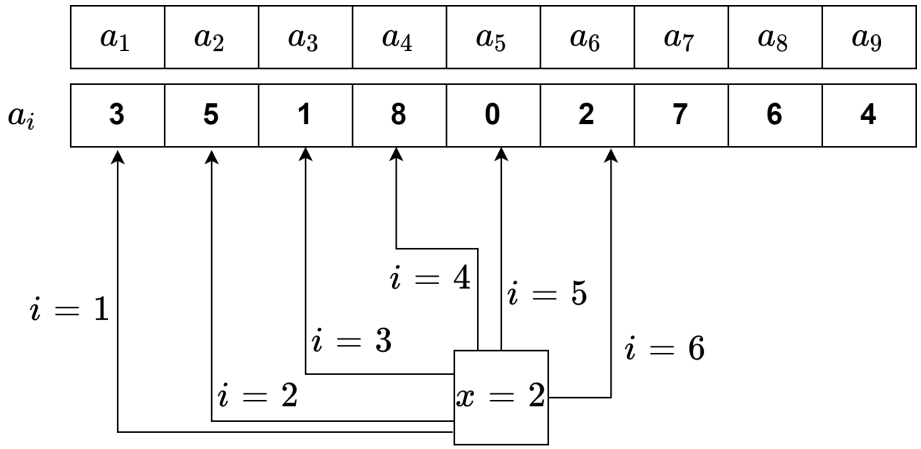
---

1. Định nghĩa và các tính chất

- **Tính chất:**
  - **Đầu vào (input):** Một thuật toán phải có đầu vào là các giá trị hoặc các tập xác đinh.
  - **Đầu ra (output):** Với mỗi input xác định, thuật toán sẽ đưa ra một kết quả tương ứng với input đó. Output còn được gọi là lời giải của bài toán.
  - **Tính xác định:** Số bước của thuật toán phải được xác định rõ.
  - **Tính đúng đắn:** Đầu ra của thuật toán phải đúng với đầu vào tương ứng.
  - **Tính hữu hạn:** Thuật toán phải cho ra output sau một khoảng thời gian.
  - **Tính hiệu quả:** Mỗi bước của thuật toán nên được thực hiện chính xác trong một khoảng thời gian hữu hạn.
  - **Tính tổng quát:** Một thuật toán tốt phải giải quyết được tất cả các vấn đề cùng khuôn mẫu với tất cả các input khác nhau, không phải chỉ giải quyết tốt một vài trường hợp input cụ thể.

Ví dụ:

Tìm vị trí của một số nguyên trong một dãy số nguyên.



Ví dụ:
Tìm vị trí của một số nguyên trong một dãy số nguyên.

Algorithm 2 Tìm kiếm tuyến tính

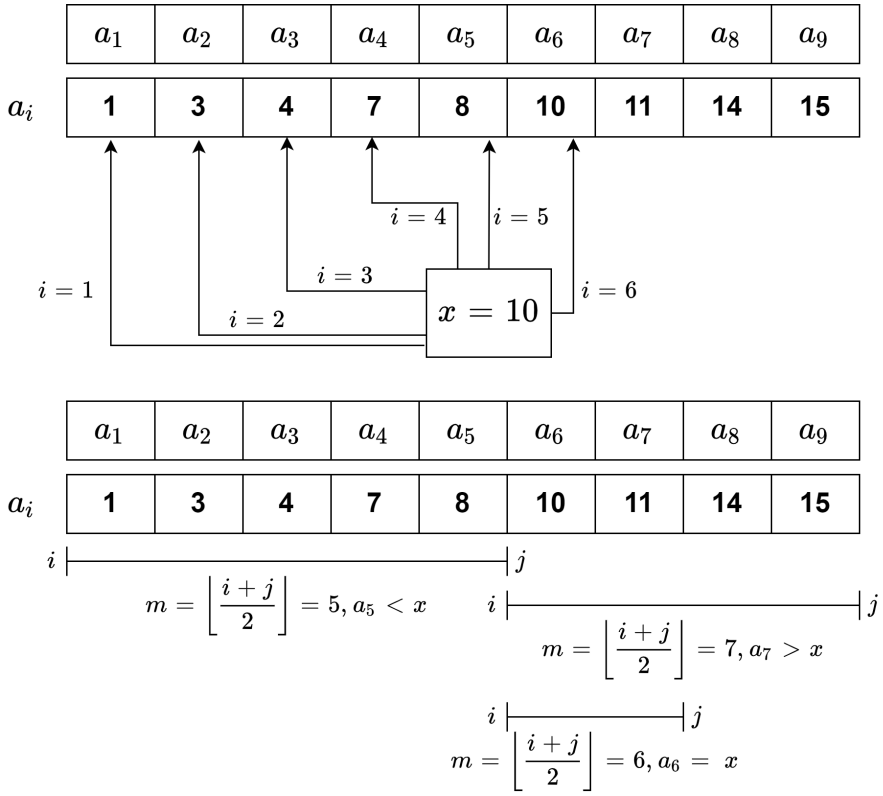
Input: Số nguyên x và tập số nguyên phân biệt $(a_1, a_2, ..., a_n)$

1: $i := 1$
2: Khi mệnh đề $(i \leq n) \wedge (x \neq a_i)$ vẫn đúng:
3: $\quad i := i + 1$ {Tăng dần i}
4: Nếu $i \leq n$ thì vị trí $:= i$
5: Trái lại thì vị trí $:= 0$

Output: vị trí

Ví dụ:

Tìm vị trí của một số nguyên trong một dãy số nguyên tăng dần.



Ví dụ:
Tìm vị trí của một số nguyên trong một dãy số nguyên tăng dần.

Algorithm 3 Tìm kiếm nhị phân

Input: Số nguyên x và dãy số nguyên tăng dần $(a_1, a_2, ..., a_n)$

1: $i := 1$ {i là cực trái của khoảng tìm kiếm}
2: $j := n$ {j là cực phải của khoảng tìm kiếm}
3: Khi mệnh đề $(i < j)$ vẫn đúng:
4: $\quad m := \lfloor (i + j)/2 \rfloor$
5: $\quad$ Nếu $x > a_m$ thì $i := m + 1$
6: $\quad$ Trái lại thì $j := m$
7: Nếu $x = a_i$ thì vị trí $:= i$
8: Trái lại thì vị trí $:= 0$

Output: vị trí

Ví dụ:
Sắp xếp một dãy số cho trước theo thứ tự tăng dần.

Algorithm 4 Sắp xếp sủi bọt

Input: Dãy số thực $(a_1, a_2, ..., a_n)$ với $n \geq 2$

1: Xét lần lượt $i = 1$ đến $n - 1$:
2: Xét lần lượt $j = 1$ đến $n - i$:
3: Nếu $a_j > a_{j+1}$ thì đổi vị trí a_j và a_{j+1}

Output: Dãy số đã được sắp xếp tăng dần

Link

Ví dụ:
Sắp xếp một dãy số cho trước theo thứ tự tăng dần.

Algorithm 5 Sắp xếp chèn

Input: Dãy số thực $(a_1, a_2, ..., a_n)$ với $n \geq 2$

1: Xét lần lượt $j = 2$ đến n:
2: $i := 1$
3: Khi điều kiện $(a_j > a_i)$ vẫn đúng:
4: $i := i + 1$
5: $m := a_j$
6: Xét lần lượt $k = 0$ đến $j - i - 1$:
7: $a_{j-k} := a_{j-k-1}$
8: $a_i := m$

Output: Dãy số đã được sắp xếp tăng dần

Link

1. Định nghĩa và các tính chất

Thuật toán tham lam

- Là một giải thuật được dùng nhiều trong các bài toán tối ưu.
- Lựa chọn tốt nhất sẽ được lấy ngay vào lời giải cuối cùng ở mỗi bước của thuật toán thay vì xem xét toàn bộ quá trình từ bước đầu đến bước cuối.
- Cách làm này vô cùng đơn giản và có thể đưa ra đáp án tối ưu cho một số bài toán.

Ví dụ:
Cài đặt thuật toán trả lại tiền thừa cho máy bán hàng tự động sao cho mỗi lần trả lại đúng số tiền thừa cho khách với số đồng tiền ít nhất. Giả sử rằng trong máy có sẵn 4 mệnh giá tiền 1k, 2k, 5k, 10k với số đồng tiền mỗi loại không giới hạn.

Algorithm 6 Trả lại tiền thừa bằng thuật toán tham lam

Input: Mệnh giá tiền $(c_1, c_2, ..., c_r \in \mathbb{Z}^+ : c_1 > c_2 > ... > c_r)$ và số tiền phải trả lại $n \in \mathbb{Z}^+$.

1: Xét lần lượt $i = 1$ đến r:
2: $d_i := 0$ {d_i dùng để đếm số đồng tiền mệnh giá c_i đã dùng}
3: Khi điều kiện $(n \geq c_i)$ vẫn đúng:
4: $d_i := d_i + 1$ {dùng thêm một đồng tiền mệnh giá c_i}
5: $n := n - c_i$:

Output: $(d_1, d_2, ..., d_r)$

2. Độ phức tạp của thuật toán

- Thời gian mà máy tính khi thực hiện một thuật toán không chỉ phụ thuộc vào bản thân thuật toán đó, ngoài ra còn tùy thuộc từng máy tính.

  $\Longrightarrow$ Để đánh giá hiệu quả của một thuật toán, có thể xét **số các phép tính** phải thực hiện khi thực hiện thuật toán này thay vì chỉ xét thời gian chạy.
- Thông thường số các phép tính được thực hiện phụ thuộc vào cỡ của bài toán, tức là độ lớn của đầu vào. Vì thế **độ phức tạp thuật toán** là một hàm phụ thuộc đầu vào.
- Trong những ứng dụng thực tiễn, chúng ta không cần biết chính xác hàm này mà chỉ cần biết một ước lượng đủ tốt của chúng.

2. Độ phức tạp của thuật toán

- Để ước lượng độ phức tạp của một thuật toán ta thường dùng khái niệm big-O, kí hiệu là $O()$.
- Độ phức tạp không phải là độ đo chính xác lượng tài nguyên máy cần dùng, mà đặc trưng cho động thái của hệ thống khi kích thước đầu vào tăng lên.
- Xét quan hệ giữa **độ lớn đầu vào** n và **tài nguyên cần dùng** $f(n)$, nếu như tìm được hằng số $C > 0$, C không phụ thuộc vào n, và hằng số k sao cho :

$$|f(n)| \leq C|g(n)|$$

với mọi $n > k$ thì ta nói thuật toán có độ phức tạp cỡ $O(g(n))$.

Ví dụ:
Chứng minh rằng: $f(x) = x^2 + 2x + 1$ là $O(x^2)$.

Ví dụ:
Chứng minh rằng: $f(x) = x^2 + 2x + 1$ là $O(x^2)$.
Giải:

- Khi $x > 1$ thì $1 < x^2$ và $x < x^2$.
  $\Longrightarrow 0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$
  $\Longrightarrow$ Chọn $C = 4, k = 1$.
- Khi $x > 2$ thì $2x \leq x^2$ và $1 \leq x^2$.
  $\Longrightarrow 0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$
  $\Longrightarrow$ Chọn $C = 3, k = 2$.

2. Độ phức tạp của thuật toán

Định lý 4.1

Nếu $f_1(x)$ là $O(g_1(x))$ và $f_2(x)$ là $O(g_2(x))$ thì:

$$(f_1 + f_2)(x) \text{ là } O(\max(|g_1(x)|, |g_2(x)|))$$

Định lý 4.2

Nếu $f_1(x)$ là $O(g_1(x))$ và $f_2(x)$ là $O(g_2(x))$ thì:

$$(f_1 f_2)(x) \text{ là } O(g_1(x)g_2(x))$$

2. Độ phức tạp của thuật toán

- Các độ phức tạp thường gặp đối với các thuật toán thông thường gồm có:
  - Độ phức tạp **hằng số**, $O(1)$. Số phép tính/thời gian chạy/dung lượng bộ nhớ không phụ thuộc vào n. Ví dụ: các thao tác hệ thống: đóng, mở tập tin.
  - Độ phức tạp **tuyến tính**, $O(n)$. Số phép tính/thời gian chạy/dung lượng bộ nhớ có xu hướng tỉ lệ thuận với n. Ví dụ: tính tổng các phần tử của một mảng một chiều.
  - Độ phức tạp **đa thức**, $O(P(n))$, với P là đa thức bậc cao (từ 2 trở lên). Ví dụ: các thao tác tính toán với mảng nhiều chiều (tính định thức ma trận).
  - Độ phức tạp **logarit**, $O(\log n)$ (chú ý: bậc của nó thấp hơn so với $O(n)$). Ví dụ: thuật toán Euclid để tìm ước số chung lớn nhất.
  - Độ phức tạp **hàm mũ**, $O(2^n)$. Trường hợp này bất lợi nhất và sẽ rất phi thực tế nếu thực hiện thuật toán với độ phức tạp này.

Ví dụ:

Giả sử n là kích thước của đầu vào.

- Với thuật toán có độ phức tạp tuyến tính $O(n)$, nếu n tăng gấp đôi thì có thể ước tính rằng tài nguyên cần dùng cũng tăng khoảng gấp đôi.
- Với thuật toán có độ phức tạp bình phương $O(n^2)$, nếu n tăng gấp đôi thì tài nguyên sẽ tăng gấp bốn.
- Với thuật toán có độ phức tạp hàm mũ $O(2^n)$ thì chỉ cần n tăng thêm 2 đơn vị cũng đã làm tài nguyên tăng gấp 4 lần (tức là theo cấp số nhân).