Đánh giá hiệu quả thuật toán

Bùi Việt Dũng

8th June 2022

Độ phức tạp thời gian của thuật toán

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:
  - Việc cài đặt chương trình rất tốn thời gian.

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:
  - Việc cài đặt chương trình rất tốn thời gian.
  - Tốc độ chạy chương trình trên các máy khác nhau là khác nhau

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:
  - Việc cài đặt chương trình rất tốn thời gian.
  - Tốc độ chạy chương trình trên các máy khác nhau là khác nhau
    - Máy của thí sinh vs. Máy của giám khảo

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:
    - Việc cài đặt chương trình rất tốn thời gian.
    - Tốc độ chạy chương trình trên các máy khác nhau là khác nhau
        - Máy của thí sinh vs. Máy của giám khảo
        - Không chương trình nào chạy hai lần trên cùng một máy.

Độ phức tạp thời gian của thuật toán

- Tại sao không viết hẳn chương trình ra để đánh giá hiệu quả thuật toán:
  - Việc cài đặt chương trình rất tốn thời gian.
  - Tốc độ chạy chương trình trên các máy khác nhau là khác nhau
    - Máy của thí sinh vs. Máy của giám khảo
    - Không chương trình nào chạy hai lần trên cùng một máy.
- Độ phức tạp tính toán là một cách biểu diễn số **phép tính** thuật toán cần thực hiện so với kích thước đầu vào.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.
Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.
- Một số hàm Toán học đơn giản như $\mathtt{sin}, \mathtt{cos}, \mathtt{tan}, \mathtt{cot}, \mathtt{log}, \mathtt{sqrt}$

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.
- Một số hàm Toán học đơn giản như $\mathtt{sin}, \mathtt{cos}, \mathtt{tan}, \mathtt{cot}, \mathtt{log}, \mathtt{sqrt}$
- Ta có thể ghép một cụm các câu lệnh trên nếu thời gian chạy vẫn không phụ thuộc vào độ lớn của dữ liệu vào.

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.
- Một số hàm Toán học đơn giản như $\mathtt{sin}, \mathtt{cos}, \mathtt{tan}, \mathtt{cot}, \mathtt{log}, \mathtt{sqrt}$
- Ta có thể ghép một cụm các câu lệnh trên nếu thời gian chạy vẫn không phụ thuộc vào độ lớn của dữ liệu vào.

Không phải ví dụ:

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.
- Một số hàm Toán học đơn giản như $\mathtt{sin}, \mathtt{cos}, \mathtt{tan}, \mathtt{cot}, \mathtt{log}, \mathtt{sqrt}$
- Ta có thể ghép một cụm các câu lệnh trên nếu thời gian chạy vẫn không phụ thuộc vào độ lớn của dữ liệu vào.

Không phải ví dụ:

- Hàm sắp xếp ($\mathtt{sort}$)

Thế nào là phép tính?

Phép tính là một hay nhiều câu lệnh có thời gian chạy không phụ thuộc vào độ lớn của dữ liệu vào.

Ví dụ (ở các trường hợp thông thường):

- Lệnh đọc một kí tự.
- Lệnh gán.
- Lệnh cộng, trừ, nhân, chia, so sánh hai số.
- Lệnh dịch bit.
- Lệnh khởi tạo một mảng biết số phần tử nhưng không đặt trước các giá trị trong mảng.
- Một số hàm Toán học đơn giản như $\mathtt{sin}, \mathtt{cos}, \mathtt{tan}, \mathtt{cot}, \mathtt{log}, \mathtt{sqrt}$
- Ta có thể ghép một cụm các câu lệnh trên nếu thời gian chạy vẫn không phụ thuộc vào độ lớn của dữ liệu vào.

Không phải ví dụ:

- Hàm sắp xếp (`sort`)
- Hàm lũy thừa (`pow`)

Ví dụ

Tính số phép tính của thuật toán sau theo n:

Ví dụ

Tính số phép tính của thuật toán sau theo n:

```
cin >> n;
int res = 0;
for (int i=1; i<=n; ++i) {
    for (int j=1; j<=n; ++j) {
        for (int k=1; k<=n; ++k) {
            res += i + 2*j + 3*k;
        }
    }
}

for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Ví dụ

Tính số phép tính của thuật toán sau theo n:

```
cin >> n;
int res = 0;
for (int i=1; i<=n; ++i) {
    for (int j=1; j<=n; ++j) {
        for (int k=1; k<=n; ++k) {
            res += i + 2*j + 3*k;
        }
    }
}

for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Một đáp án có thể: $f(n) = n^3 + \frac{1}{2}n(n+1)$

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

$$\boxed{f(n) \in O(g(n)) \Leftrightarrow \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

$$\boxed{f(n) \in O(g(n)) \Leftrightarrow \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$

Nói cách khác

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

$$\boxed{f(n) \in O(g(n)) \Leftrightarrow \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$

Nói cách khác

$$O(g(n)) = \{f(n) | \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

$$\boxed{f(n) \in O(g(n)) \Leftrightarrow \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$

Nói cách khác

$$O(g(n)) = \{f(n) | \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

(Để đơn giản, từ giờ cho đến hết bài trình bày này, chúng ta sẽ quy ước hàm f có tập xác định là tập số nguyên dương và tập giá trị là tập số thực dương)

Kí pháp chữ O lớn

Gọi $f(n)$ là số phép tính thuật toán cần thực hiện nếu độ lớn của dữ liệu vào là n (n có thể là độ dài mảng, độ lớn của số nguyên được cho, ...)

$$\boxed{f(n) \in O(g(n)) \Leftrightarrow \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$

Nói cách khác

$$O(g(n)) = \{f(n) | \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

(Để đơn giản, từ giờ cho đến hết bài trình bày này, chúng ta sẽ quy ước hàm f có tập xác định là tập số nguyên dương và tập giá trị là tập số thực dương)
Một số tập O hay gặp: $O(1), O(\log n), O(n^2), O(n^3), O(2^n), O(n!)$

Một số tính chất của O

Một số tính chất của O

Tính chất 1

$f(n) \in O(f(n))$

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$

Một số tính chất của O

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.

Một số tính chất của O

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n)$

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n) \Rightarrow f(n) \leq f(n)$

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n) \Rightarrow f(n) \leq f(n) \Rightarrow f(n) \leq 1 \cdot f(n)$

Một số tính chất của O

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n) \Rightarrow f(n) \leq f(n) \Rightarrow f(n) \leq 1 \cdot f(n)$
$\Rightarrow f(n) \leq c \cdot f(n)$
Do cách ta chọn $f(n), n_0, c, n$, từ đây ta có thể kết luận

Một số tính chất của O

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n) \Rightarrow f(n) \leq f(n) \Rightarrow f(n) \leq 1 \cdot f(n)$
$\Rightarrow f(n) \leq c \cdot f(n)$
Do cách ta chọn $f(n), n_0, c, n$, từ đây ta có thể kết luận
$\forall f(n) : \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$

Tính chất 1

$f(n) \in O(f(n))$

Chọn hàm $f(n)$ bất kì
Để chứng minh $f(n) \in O(f(n))$, ta cần chứng minh
$\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
Chọn $n_0 = c = 1 > 0$
Chọn $n \geq n_0$ bất kì.
Ta có: $f(n) = f(n) \Rightarrow f(n) \leq f(n) \Rightarrow f(n) \leq 1 \cdot f(n)$
$\Rightarrow f(n) \leq c \cdot f(n)$
Do cách ta chọn $f(n), n_0, c, n$, từ đây ta có thể kết luận
$\forall f(n) : \exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot f(n)$
$\Rightarrow \forall f(n) : f(n) \in O(f(n))$

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$

Do $f(n) \in O(g(n))$ nên

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$

Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$

Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$

và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho

$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$

Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh

$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$

Chọn $n_2 = ... > 0, c_2 = ... > 0$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$
Chọn $n_2 = ... > 0, c_2 = ... > 0$
Chọn số $n \geq n_2$ bất kì

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$
Chọn $n_2 = ... > 0, c_2 = ... > 0$
Chọn số $n \geq n_2$ bất kì
Do $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$ và $n \geq n_2 \geq n_1$, $f(n) \leq c_1 \cdot g(n)$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$
Chọn $n_2 = ... > 0, c_2 = ... > 0$
Chọn số $n \geq n_2$ bất kì
Do $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$ và $n \geq n_2 \geq n_1$, $f(n) \leq c_1 \cdot g(n)$
$\Rightarrow cf(n) \leq cc_1 \cdot g(n)$

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$ và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh $\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$
Chọn $n_2 = ... > 0, c_2 = ... > 0$
Chọn số $n \geq n_2$ bất kì
Do $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$ và $n \geq n_2 \geq n_1$, $f(n) \leq c_1 \cdot g(n)$
$\Rightarrow cf(n) \leq cc_1 \cdot g(n)$
$\Rightarrow cf(n) \leq c_2 \cdot g(n)$

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$

Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$

và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho

$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$

Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh

$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$

Chọn $n_2 = ... > 0, c_2 = ... > 0$

Chọn số $n \geq n_2$ bất kì

Do $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$ và $n \geq n_2 \geq n_1$, $f(n) \leq c_1 \cdot g(n)$

$\Rightarrow cf(n) \leq cc_1 \cdot g(n)$

$\Rightarrow cf(n) \leq c_2 \cdot g(n)$

Một số tính chất của O

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Lấy hàm $f(n), g(n)$ và số c bất kì sao cho $f(n) \in O(g(n))$ và $c > 0$
Do $f(n) \in O(g(n))$ nên $\exists n_0, c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
và vì thế ta có thể chọn số $n_1, c_1 > 0$ sao cho
$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$
Để chứng minh $cf(n) \in O(g(n))$, ta cần chứng minh
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$
Chọn $n_2 = n_1 > 0, c_2 = cc_1 > 0$
Chọn số $n \geq n_2$ bất kì
Do $\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$ và $n \geq n_2 = n_1$, $f(n) \leq c_1 \cdot g(n)$
$\Rightarrow cf(n) \leq cc_1 \cdot g(n)$
$\Rightarrow cf(n) \leq c_2 \cdot g(n)$
Do cách ta chọn n_2, c_2, n, ta có thể kết luận:
$\exists n_2, c_2 > 0 : \forall n \geq n_2 : cf(n) \leq c_2 \cdot g(n)$

Ứng dụng quy tắc nhân hằng số

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Ứng dụng quy tắc nhân hằng số

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Xét thuật toán sau:

Ứng dụng quy tắc nhân hằng số

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Xét thuật toán sau:

```
    cin >> n;
    for (int i=1; i<=n; ++i) {
        a+=n;
        b+=a;
        c+=b;
        d+=c;
        e+=d;
        f+=e;
        g+=f;
    }
```

Ứng dụng quy tắc nhân hằng số

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Xét thuật toán sau:

```
    cin >> n;
    for (int i=1; i<=n; ++i) {
        a+=n;
        b+=a;
        c+=b;
        d+=c;
        e+=d;
        f+=e;
        g+=f;
    }
```

$$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 8 = 8n^2 \in O(n^2)$$

Ứng dụng quy tắc nhân hằng số

Quy tắc nhân hằng số

Cho hàm $f(n) \in O(g(n))$ và $c > 0$. Khi đó $cf(n) \in O(g(n))$

Xét thuật toán sau:

```
    cin >> n;
    for (int i=1; i<=n; ++i) {
        a+=n;
        b+=a;
        c+=b;
        d+=c;
        e+=d;
        f+=e;
        g+=f;
    }
```

$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 8 = 8n^2 \in O(n^2)$
$g(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = n^2 \in O(n^2)$

Một số tính chất của O

Quy tắc nhân

Nếu $f_1(n) \in O(g_1(n))$ và $f_2(n) \in O(g_2(n))$ thì
$f_1(n)f_2(n) \in O(g_1(n)g_2(n))$

Ứng dụng

```
for (int i=1; i<=n; ++i) sort(a[i], a[i]+n);
```

Do hàm `sort(a[i], a[i]+n)` hoạt động trong $O(n \log n)$ và hàm được gọi $n \in O(n)$ lần, theo tính chất 3, đoạn code này hoạt động trong thời gian $O(n^2 \log n)$

Một số tính chất của O

Quy tắc cộng

Nếu $f_1(n) \in O(g_1(n))$ và $f_2(n) \in O(g_2(n))$ thì
$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Một số tính chất của O

Quy tắc cộng

Nếu $f_1(n) \in O(g_1(n))$ và $f_2(n) \in O(g_2(n))$ thì $f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Ứng dụng

Khi phân tích độ phức tạp tính toán, ta chỉ cần nhìn vào số lần **phép tính tích cực** (phép tính hoạt động nhiều lần nhất) hoạt động.

Một số tính chất của O

Quy tắc cộng

Nếu $f_1(n) \in O(g_1(n))$ và $f_2(n) \in O(g_2(n))$ thì
$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Ứng dụng

Khi phân tích độ phức tạp tính toán, ta chỉ cần nhìn vào số lần **phép tính tích cực** (phép tính hoạt động nhiều lần nhất) hoạt động.

```
cin >> n;
int res = 0;
for (int i=1; i<=n; ++i) {
    for (int j=1; j<=n; ++j) {
        for (int k=1; k<=n; ++k) {
            res += i + 2*j + 3*k;
        }
    }
}

for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Một số tính chất của O

Quy tắc cộng

Nếu $f_1(n) \in O(g_1(n))$ và $f_2(n) \in O(g_2(n))$ thì
$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Ứng dụng

Khi tối ưu thuật toán, ta luôn ưu tiên tối ưu phần có độ phức tạp lớn nhất trước.

```
cin >> n;
int res = 0;
for (int i=1; i<=n; ++i) {
    for (int j=1; j<=n; ++j) {
        for (int k=1; k<=n; ++k) {
            res += i + 2*j + 3*k;
        }
    }
}

for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$
Dạng thường dùng

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Dạng thường dùng

Nếu $f(n) \leq g(n)$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$
Dạng thường dùng
Nếu $f(n) \leq g(n)$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Ứng dụng

```
for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$
Dạng thường dùng
Nếu $f(n) \leq g(n)$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Ứng dụng

```
for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Gọi $f(n)$ là số lần lặp của vòng lặp phía trong. Đặt $g(n) = n$ và $h(n) = n$, ta thấy $f(n) \leq g(n)$ và $g(n) \in O(h(n))$, nên $f(n) \in O(h(n)) \Rightarrow f(n) \in O(n)$

Một số tính chất của O

Tính chất bắc cầu

Nếu $f(n) \in O(g(n))$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$
Dạng thường dùng
Nếu $f(n) \leq g(n)$ và $g(n) \in O(h(n))$ thì $f(n) \in O(h(n))$

Ứng dụng

```
for (int i=1; i<=n; ++i) {
    for (int j=i+1; j<=n; ++j) {
        res += 2*(i+j);
    }
}
```

Gọi $f(n)$ là số lần lặp của vòng lặp phía trong. Đặt $g(n) = n$ và $h(n) = n$, ta thấy $f(n) \leq g(n)$ và $g(n) \in O(h(n))$, nên $f(n) \in O(h(n)) \Rightarrow f(n) \in O(n)$
Vòng lặp phía ngoài chạy $O(n)$ lần nên theo quy tắc nhân, độ phức tạp của vòng lặp này là $O(n^2)$

Mối quan hệ giữa O và giới hạn dãy số

Cho dãy số thực (u_n)

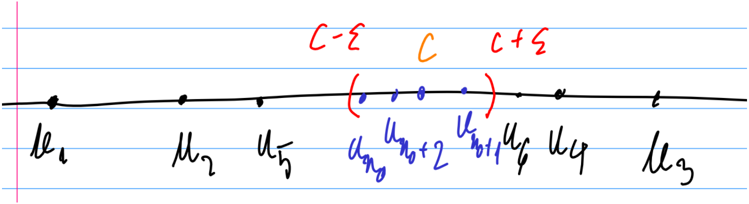
Mối quan hệ giữa O và giới hạn dãy số

Cho dãy số thực (u_n)
$\lim u_n = c \in \mathbb{R} \Leftrightarrow \forall \epsilon > 0 : \exists n_0 > 0 : \forall n \geq n_0 : |u_n - c| < \epsilon$

Mối quan hệ giữa O và giới hạn dãy số

Cho dãy số thực (u_n)

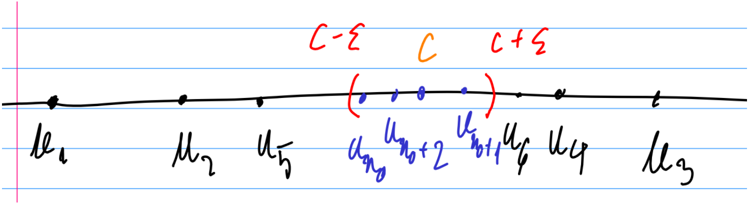
$\lim u_n = c \in \mathbb{R} \Leftrightarrow \forall \epsilon > 0 : \exists n_0 > 0 : \forall n \geq n_0 : |u_n - c| < \epsilon$



Mối quan hệ giữa O và giới hạn dãy số

Cho dãy số thực (u_n)

$\lim u_n = c \in \mathbb{R} \Leftrightarrow \forall \epsilon > 0 : \exists n_0 > 0 : \forall n \geq n_0 : |u_n - c| < \epsilon$



Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = ... > 0$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = ... > 0$

Lấy $\epsilon = ... > 0$, ta sẽ chọn được $n_0 > 0$ sao cho

$\forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n) \Rightarrow \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = c + 1 > 0$

Lấy $\epsilon = 1 > 0$, ta sẽ chọn được $n_0 > 0$ sao cho

$\forall n \geq n_0 : f(n) \leq (c + 1)g(n) \Rightarrow \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = c + 1 > 0$

Lấy $\epsilon = 1 > 0$, ta sẽ chọn được $n_0 > 0$ sao cho

$\forall n \geq n_0 : f(n) \leq (c + 1)g(n) \Rightarrow \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Do cách ta chọn k và n_0, ta có thể kết luận

$\exists k, n_0 > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = c + 1 > 0$

Lấy $\epsilon = 1 > 0$, ta sẽ chọn được $n_0 > 0$ sao cho

$\forall n \geq n_0 : f(n) \leq (c + 1)g(n) \Rightarrow \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Do cách ta chọn k và n_0, ta có thể kết luận

$\exists k, n_0 > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Mối quan hệ giữa O và giới hạn dãy số

Quy tắc giới hạn

Nếu $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm thì $f(n) \in O(g(n))$

Giả sử $\lim \frac{f(n)}{g(n)} = c$ với c là một số thực không âm nào đó.

Khi đó $\forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |\frac{f(n)}{g(n)} - c| < \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) < (c + \epsilon)g(n)$

$\Rightarrow \forall \epsilon > 0 : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq (c + \epsilon)g(n)$

Ta cần chứng minh $f(n) \in O(g(n))$, tức

$\exists n_0, k > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Lấy $k = c + 1 > 0$

Lấy $\epsilon = 1 > 0$, ta sẽ chọn được $n_0 > 0$ sao cho

$\forall n \geq n_0 : f(n) \leq (c + 1)g(n) \Rightarrow \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Do cách ta chọn k và n_0, ta có thể kết luận

$\exists k, n_0 > 0 : \forall n \geq n_0 : f(n) \leq k \cdot g(n)$

Độ phức tạp của hàm đa thức

Độ phức tạp của hàm đa thức

Nếu $f(n) = a_1 n^{b_1} + a_2 n^{b_2} + \ldots + a_d n^{b_d}$ với $a_1, a_2, \ldots, a_d, b_1, b_2, \ldots, b_d > 0$, $b_1 > b_2 > \ldots > b_d$ thì $f(n) \in O(n^{b_1})$

Độ phức tạp của hàm đa thức

Nếu $f(n) = a_1 n^{b_1} + a_2 n^{b_2} + ... + a_d n^{b_d}$ với $a_1, a_2, ..., a_d, b_1, b_2, ..., b_d > 0$, $b_1 > b_2 > ... > b_d$ thì $f(n) \in O(n^{b_1})$

Ta có

Độ phức tạp của hàm đa thức

Nếu $f(n) = a_1 n^{b_1} + a_2 n^{b_2} + \ldots + a_d n^{b_d}$ với $a_1, a_2, \ldots, a_d, b_1, b_2, \ldots, b_d > 0,\ b_1 > b_2 > \ldots > b_d$ thì $f(n) \in O(n^{b_1})$

Ta có
$\lim \frac{a_1 n^{b_1} + a_2 n^{b_2} + \ldots + a_d n^{b_d}}{n^{b_1}} = \lim \frac{a_1 n^{b_1}}{n^{b_1}} + \lim \frac{a_2 n^{b_2}}{n^{b_1}} + \ldots = a_1 + 0 + 0 + \ldots = a_1$

Độ phức tạp của hàm đa thức

Độ phức tạp của hàm đa thức

Nếu $f(n) = a_1 n^{b_1} + a_2 n^{b_2} + ... + a_d n^{b_d}$ với $a_1, a_2, ..., a_d, b_1, b_2, ..., b_d > 0,\ b_1 > b_2 > ... > b_d$ thì $f(n) \in O(n^{b_1})$

Ta có
$\lim \frac{a_1 n^{b_1} + a_2 n^{b_2} + ... + a_d n^{b_d}}{n^{b_1}} = \lim \frac{a_1 n^{b_1}}{n^{b_1}} + \lim \frac{a_2 n^{b_2}}{n^{b_1}} + ... =$
$a_1 + 0 + 0 + ... = a_1$
nên theo quy tắc giới hạn, $f(n) \in O(n^{b_1})$

Bài tập

Các hàm nào sau đây thuộc tập $O(n^2)$?

- $f_1(n) = 5$
- $f_2(n) = 100n^2 + 78n + 4$
- $f_3(n) = 21n + 34$
- $f_4(n) = 5n\sqrt{n} + 24$

Các hàm nào sau đây thuộc tập $O(n^2)$?

- $f_1(n) = 5$
- $f_2(n) = 100n^2 + 78n + 4$
- $f_3(n) = 21n + 34$
- $f_4(n) = 5n\sqrt{n} + 24$

Đáp án:

- $f_1(n) \in O(1) \subset O(n^2)$
- $f_2(n) \in O(n^2)$
- $f_3(n) \in O(n) \subset O(n^2)$
- $f_4(n) \in O(n^{\frac{3}{2}}) \subset O(n^2)$

Bài tập

Các hàm nào sau đây thuộc tập $O(n^2)$?

- $f_1(n) = 5$
- $f_2(n) = 100n^2 + 78n + 4$
- $f_3(n) = 21n + 34$
- $f_4(n) = 5n\sqrt{n} + 24$

Đáp án:

- $f_1(n) \in O(1) \subset O(n^2)$
- $f_2(n) \in O(n^2)$
- $f_3(n) \in O(n) \subset O(n^2)$
- $f_4(n) \in O(n^{\frac{3}{2}}) \subset O(n^2)$

Lưu ý: Để đạt được lợi ích tối đa khi tính độ phức tạp tính toán thì ta chọn tập O nhỏ nhất.

Thuật toán có độ phức tạp đa thức

Thuật toán có độ phức tạp $O(n^d)$ với một số thực $d > 0$ được gọi là thuật toán có độ phức tạp đa thức.

Thuật toán có độ phức tạp đa thức

Thuật toán có độ phức tạp $O(n^d)$ với một số thực $d > 0$ được gọi là thuật toán có độ phức tạp đa thức.

Thuật toán hiệu quả

Gọi $f(n)$ số phép tính của thuật toán T nếu độ lớn của dữ liệu vào là n. Thuật toán T được gọi là thuật toán hiệu quả khi và chỉ khi $\exists c > 0 : \forall n > 0 : f(2n) \leq cf(n)$

Thuật toán có độ phức tạp đa thức

Thuật toán có độ phức tạp $O(n^d)$ với một số thực $d > 0$ được gọi là thuật toán có độ phức tạp đa thức.

Thuật toán hiệu quả

Gọi $f(n)$ số phép tính của thuật toán T nếu độ lớn của dữ liệu vào là n. Thuật toán T được gọi là thuật toán hiệu quả khi và chỉ khi
$\exists c > 0 : \forall n > 0 : f(2n) \leq cf(n)$
(Nếu độ lớn dữ liệu vào tăng gấp 2 lần, thời gian chạy tăng gấp c lần với c không phụ thuộc vào độ lớn dữ liệu vào).

Thuật toán có độ phức tạp đa thức

Thuật toán có độ phức tạp $O(n^d)$ với một số thực $d > 0$ được gọi là thuật toán có độ phức tạp đa thức.

Thuật toán hiệu quả

Gọi $f(n)$ số phép tính của thuật toán T nếu độ lớn của dữ liệu vào là n. Thuật toán T được gọi là thuật toán hiệu quả khi và chỉ khi $\exists c > 0 : \forall n > 0 : f(2n) \leq cf(n)$
(Nếu độ lớn dữ liệu vào tăng gấp 2 lần, thời gian chạy tăng gấp c lần với c không phụ thuộc vào độ lớn dữ liệu vào).

Người ta chứng minh được một thuật toán hiệu quả khi và chỉ khi thuật toán đó có độ phức tạp đa thức.

Mở rộng cho trường hợp nhiều biến

Gọi $f(n, m)$ là số phép tính thuật toán cần thực hiện

Mở rộng cho trường hợp nhiều biến

Gọi $f(n, m)$ là số phép tính thuật toán cần thực hiện
$f(n, m) \in O(g(n, m)) \Leftrightarrow \exists n_0, m_0, c > 0 : \forall n \geq n_0, m \geq m_0 :$
$f(n, m) \leq c \cdot g(n, m)$

Mở rộng cho trường hợp nhiều biến

Gọi $f(n,m)$ là số phép tính thuật toán cần thực hiện
$f(n,m) \in O(g(n,m)) \Leftrightarrow \exists n_0, m_0, c > 0 : \forall n \geq n_0, m \geq m_0 :$
$f(n,m) \leq c \cdot g(n,m)$
Ví dụ: Độ phức tạp của thuật toán đọc một mảng hai chiều kích thước $n \times m$ là $O(nm)$

Mở rộng cho trường hợp nhiều biến

Gọi $f(n, m)$ là số phép tính thuật toán cần thực hiện
$f(n, m) \in O(g(n, m)) \Leftrightarrow \exists n_0, m_0, c > 0 : \forall n \geq n_0, m \geq m_0 :$
$f(n, m) \leq c \cdot g(n, m)$
Ví dụ: Độ phức tạp của thuật toán đọc một mảng hai chiều kích thước $n \times m$ là $O(nm)$
Các tính chất trình bày ở trên vẫn đúng trong trường hợp nhiều biến.

Một số độ phức tạp tính toán thường gặp

$O(1)$

Không phụ thuộc vào độ lớn của dữ liệu vào.

Một số độ phức tạp tính toán thường gặp

$O(1)$

Không phụ thuộc vào độ lớn của dữ liệu vào.

$O(\log n)$

Độ phức tạp của thuật toán tìm kiếm nhị phân

Một số độ phức tạp tính toán thường gặp

$O(1)$

Không phụ thuộc vào độ lớn của dữ liệu vào.

$O(\log n)$

Độ phức tạp của thuật toán tìm kiếm nhị phân

$O(\sqrt{n})$

Độ phức tạp của thuật toán kiểm tra số nguyên tố.

Một số độ phức tạp tính toán thường gặp

$O(1)$

Không phụ thuộc vào độ lớn của dữ liệu vào.

$O(\log n)$

Độ phức tạp của thuật toán tìm kiếm nhị phân

$O(\sqrt{n})$

Độ phức tạp của thuật toán kiểm tra số nguyên tố.

$O(n)$

Độ phức tạp của thuật toán tìm kiếm tuần tự.

Một số độ phức tạp tính toán thường gặp

$O(1)$

Không phụ thuộc vào độ lớn của dữ liệu vào.

$O(\log n)$

Độ phức tạp của thuật toán tìm kiếm nhị phân

$O(\sqrt{n})$

Độ phức tạp của thuật toán kiểm tra số nguyên tố.

$O(n)$

Độ phức tạp của thuật toán tìm kiếm tuần tự.

$O(n \log n)$

Độ phức tạp của hàm `sort` trong C++

Một số độ phức tạp tính toán thường gặp

$O(n^2)$

Độ phức tạp của thuật toán sắp xếp nổi bọt, thuật toán Quicksort và hàm `sort` trong Java

Một số độ phức tạp tính toán thường gặp

$O(n^2)$

Độ phức tạp của thuật toán sắp xếp nổi bọt, thuật toán Quicksort và hàm `sort` trong Java

$O(2^n)$

Duyệt hết các tập con của một tập có n phần tử.

Một số độ phức tạp tính toán thường gặp

$O(n^2)$

Độ phức tạp của thuật toán sắp xếp nổi bọt, thuật toán Quicksort và hàm `sort` trong Java

$O(2^n)$

Duyệt hết các tập con của một tập có n phần tử.

$O(n!)$

Duyệt hết các hoán vị của một tập có n phần tử.

Một số độ phức tạp tính toán thường gặp

$O(n^2)$

Độ phức tạp của thuật toán sắp xếp nổi bọt, thuật toán Quicksort và hàm `sort` trong Java

$O(2^n)$

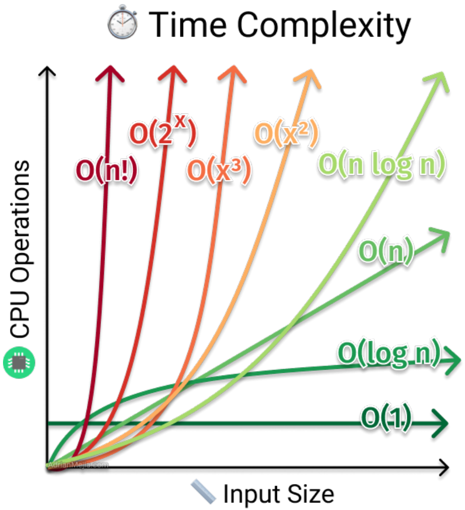
Duyệt hết các tập con của một tập có n phần tử.

$O(n!)$

Duyệt hết các hoán vị của một tập có n phần tử.

$O(1) \subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(2^n) \subset O(n!)$

Một số độ phức tạp tính toán thường gặp



Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

Ứng dụng trong lập trình thi đấu

Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

- Lấy n lớn nhất có thể có trong dữ liệu vào (được cho trong đề bài), tính $f(n)$

Ứng dụng trong lập trình thi đấu

Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

- Lấy n lớn nhất có thể có trong dữ liệu vào (được cho trong đề bài), tính $f(n)$
- Tính $\frac{f(n)}{10^8} = t$ (do máy tính thông thường tính được 10^8 phép tính một giây)

Ứng dụng trong lập trình thi đấu

Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

- Lấy n lớn nhất có thể có trong dữ liệu vào (được cho trong đề bài), tính $f(n)$
- Tính $\frac{f(n)}{10^8} = t$ (do máy tính thông thường tính được 10^8 phép tính một giây)
- Chương trình sẽ chạy trong thời gian ct với c là một hằng số nào đó.

Ứng dụng trong lập trình thi đấu

Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

- Lấy n lớn nhất có thể có trong dữ liệu vào (được cho trong đề bài), tính $f(n)$
- Tính $\frac{f(n)}{10^8} = t$ (do máy tính thông thường tính được 10^8 phép tính một giây)
- Chương trình sẽ chạy trong thời gian ct với c là một hằng số nào đó.

Nếu $t \leq 0.8$ thì thường chương trình sẽ chạy dưới 1 giây. Nếu $t > 0.8$ thì ta cố gắng tính tiếp hằng số c.

Ứng dụng trong lập trình thi đấu

Giả sử ta nhìn được độ phức tạp của chương trình là $O(f(n))$, ta có thể ước lượng độ phức tạp của thuật toán như sau:

- Lấy n lớn nhất có thể có trong dữ liệu vào (được cho trong đề bài), tính $f(n)$
- Tính $\frac{f(n)}{10^8} = t$ (do máy tính thông thường tính được 10^8 phép tính một giây)
- Chương trình sẽ chạy trong thời gian ct với c là một hằng số nào đó.

Nếu $t \leq 0.8$ thì thường chương trình sẽ chạy dưới 1 giây. Nếu $t > 0.8$ thì ta cố gắng tính tiếp hằng số c.Nếu $ct \leq 1$ thì chương trình nhiều khả năng sẽ chạy dưới 1 giây.

Ứng dụng trong lập trình thi đấu

Khi sử dụng bất cứ hàm nào trong một thư viện chuẩn, ta cần biết độ phức tạp của hàm đó trước khi sử dụng.

Ứng dụng trong lập trình thi đấu

Khi sử dụng bất cứ hàm nào trong một thư viện chuẩn, ta cần biết độ phức tạp của hàm đó trước khi sử dụng.
Đối với thư viện `C++ STL`, các bạn có thể tham khảo:

Ứng dụng trong lập trình thi đấu

Khi sử dụng bất cứ hàm nào trong một thư viện chuẩn, ta cần biết độ phức tạp của hàm đó trước khi sử dụng.
Đối với thư viện `C++ STL`, các bạn có thể tham khảo:

- "Tổng quan về thư viện chuẩn STL" - Điêu Xuân Mạnh (`https://vnoi.info/library/56/4958/`)

Ứng dụng trong lập trình thi đấu

Khi sử dụng bất cứ hàm nào trong một thư viện chuẩn, ta cần biết độ phức tạp của hàm đó trước khi sử dụng.
Đối với thư viện `C++ STL`, các bạn có thể tham khảo:

- "Tổng quan về thư viện chuẩn STL" - Điêu Xuân Mạnh (`https://vnoi.info/library/56/4958/`)
- Google tên thư viện rồi tìm trang của `cpluslus.com`. Sau đó tìm phần **Complexity**

Ứng dụng trong lập trình thi đấu

Khi sử dụng bất cứ hàm nào trong một thư viện chuẩn, ta cần biết độ phức tạp của hàm đó trước khi sử dụng.
Đối với thư viện `C++ STL`, các bạn có thể tham khảo:

- "Tổng quan về thư viện chuẩn STL" - Điêu Xuân Mạnh (`https://vnoi.info/library/56/4958/`)
- Google tên thư viện rồi tìm trang của `cpluslus.com`. Sau đó tìm phần **Complexity**
  - `constant` - $O(1)$
  - `logarithmic` - $O(\log n)$
  - `linear` - $O(n)$
  - `linearithmic` - $O(n \log n)$

Mẹo từ Competitive Programming 3

Về việc thừa $\log n$

Đối với các kì thi dành cho học sinh cấp 3 thì việc thừa $\log n$ trong độ phức tạp sẽ không gây hậu quả gì lớn.

Về việc thừa $\log n$

Đối với các kì thi dành cho học sinh cấp 3 thì việc thừa $\log n$ trong độ phức tạp sẽ không gây hậu quả gì lớn.

- Nếu có hai cách cài đặt, một cách $O(n \log n)$ nhưng dài và cách còn lại có độ phức tạp $O(n \log^2 n)$ nhưng ngắn hơn thì ta chọn cách thứ hai.

Mẹo từ Competitive Programming 3

Về việc thừa $\log n$

Đối với các kì thi dành cho học sinh cấp 3 thì việc thừa $\log n$ trong độ phức tạp sẽ không gây hậu quả gì lớn.

- Nếu có hai cách cài đặt, một cách $O(n \log n)$ nhưng dài và cách còn lại có độ phức tạp $O(n \log^2 n)$ nhưng ngắn hơn thì ta chọn cách thứ hai.
- Nếu chỉ nghĩ ra cách $O(n \log^2 n)$ mà không ra cách tối ưu xuống còn $O(n \log n)$ thì không nên bỏ tiếp thời gian để nghĩ mà cài thuật $O(n \log n)$

Về việc thừa $\log n$

Đối với các kì thi dành cho học sinh cấp 3 thì việc thừa $\log n$ trong độ phức tạp sẽ không gây hậu quả gì lớn.

- Nếu có hai cách cài đặt, một cách $O(n \log n)$ nhưng dài và cách còn lại có độ phức tạp $O(n \log^2 n)$ nhưng ngắn hơn thì ta chọn cách thứ hai.
- Nếu chỉ nghĩ ra cách $O(n \log^2 n)$ mà không ra cách tối ưu xuống còn $O(n \log n)$ thì không nên bỏ tiếp thời gian để nghĩ mà cài thuật $O(n \log n)$

Phản ví dụ: Con đường Tùng Trúc (VOI 2014), Street Lamps (APIO 2019)

Một số kĩ thuật tối ưu

- Lợi dụng hằng số nhỏ
- Sử dụng `bitset`
- Chia căn
- Hai con trỏ

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} 1 = \sum_{i=1}^{n} \frac{1}{2} i(i+1) = \frac{1}{6} n(n+1)(n+2)$

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} 1 = \sum_{i=1}^{n} \frac{1}{2} i(i+1) = \frac{1}{6} n(n+1)(n+2)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} \sum_{k=1}^{z} 1 = \sum_{i=1}^{n} \frac{1}{6} n(n+1)(n+2) = \frac{1}{24} n(n+1)(n+2)(n+3)$

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} 1 = \sum_{i=1}^{n} \frac{1}{2}i(i+1) = \frac{1}{6}n(n+1)(n+2)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} \sum_{k=1}^{z} 1 = \sum_{i=1}^{n} \frac{1}{6}n(n+1)(n+2) = \frac{1}{24}n(n+1)(n+2)(n+3)$

Nếu ta có k vòng lặp lồng nhau, biến lặp sau chỉ chạy đến biến lặp trước thì tổng số phép tính cần thực hiện là

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} 1 = \sum_{i=1}^{n} \frac{1}{2}i(i+1) = \frac{1}{6}n(n+1)(n+2)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} \sum_{k=1}^{z} 1 = \sum_{i=1}^{n} \frac{1}{6}n(n+1)(n+2) = \frac{1}{24}n(n+1)(n+2)(n+3)$

Nếu ta có k vòng lặp lồng nhau, biến lặp sau chỉ chạy đến biến lặp trước thì tổng số phép tính cần thực hiện là

$$\frac{1}{k!}n(n+1)(n+2)...(n+k)$$

(Chứng minh bằng quy nạp hoặc sử dụng kiến thức tổ hợp)

Kĩ thuật 1: Lợi dụng hằng số nhỏ

Nhận thấy:

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} 1 = \sum_{i=1}^{n} \frac{1}{2} i(i+1) = \frac{1}{6} n(n+1)(n+2)$
- $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{z=1}^{j} \sum_{k=1}^{z} 1 = \sum_{i=1}^{n} \frac{1}{6} n(n+1)(n+2) = \frac{1}{24} n(n+1)(n+2)(n+3)$

Nếu ta có k vòng lặp lồng nhau, biến lặp sau chỉ chạy đến biến lặp trước thì tổng số phép tính cần thực hiện là

$$\frac{1}{k!} n(n+1)(n+2)...(n+k)$$

(Chứng minh bằng quy nạp hoặc sử dụng kiến thức tổ hợp)

Ứng dụng: Bài VTRI - VNOI Marathon 2008

Bài toán

Cho n ($n \leq 10^5$) bóng đèn được đánh số từ 1 đến n. Ban đầu, các bóng đèn đều tắt. Thực hiện q truy vấn ($q \leq 10^5$) sau:

Bài toán

Cho n ($n \leq 10^5$) bóng đèn được đánh số từ 1 đến n. Ban đầu, các bóng đèn đều tắt. Thực hiện q truy vấn ($q \leq 10^5$) sau:

1. Cho hai số nguyên l, r ($1 \leq l \leq r \leq n$). Bật các bóng đèn $l, l+1, l+2, ..., r$

Bài toán

Cho n ($n \leq 10^5$) bóng đèn được đánh số từ 1 đến n. Ban đầu, các bóng đèn đều tắt. Thực hiện q truy vấn ($q \leq 10^5$) sau:

1. Cho hai số nguyên l, r ($1 \leq l \leq r \leq n$). Bật các bóng đèn $l, l+1, l+2, ..., r$
2. Cho hai số nguyên l, r ($1 \leq l \leq r \leq n$). Tắt các bóng đèn $l, l+1, l+2, ..., r$

Bài toán

Cho n ($n \leq 10^5$) bóng đèn được đánh số từ 1 đến n. Ban đầu, các bóng đèn đều tắt. Thực hiện q truy vấn ($q \leq 10^5$) sau:

1. Cho hai số nguyên l, r $(1 \leq l \leq r \leq n)$. Bật các bóng đèn $l, l+1, l+2, ..., r$
2. Cho hai số nguyên l, r $(1 \leq l \leq r \leq n)$. Tắt các bóng đèn $l, l+1, l+2, ..., r$
3. Tìm bóng đèn được bật có chỉ số nhỏ nhất.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.

Kĩ thuật 2: Sử dụng `bitset`

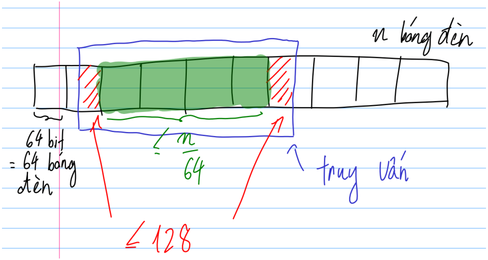
- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Khi thực hiện một truy vấn bật / tắt, ta sẽ chỉ cần chuyển nhiều nhất $\frac{n}{64}$ số nguyên thành $2^{64} - 1$ hoặc 0, và thay đổi giá trị của nhiều nhất 128 bit ở "phần đầu" và "phần cuối" của truy vấn.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Khi thực hiện một truy vấn bật / tắt, ta sẽ chỉ cần chuyển nhiều nhất $\frac{n}{64}$ số nguyên thành $2^{64} - 1$ hoặc 0, và thay đổi giá trị của nhiều nhất 128 bit ở "phần đầu" và "phần cuối" của truy vấn. Tổng cộng ta cần nhiều nhất $\frac{n}{64} + 128$ phép tính.

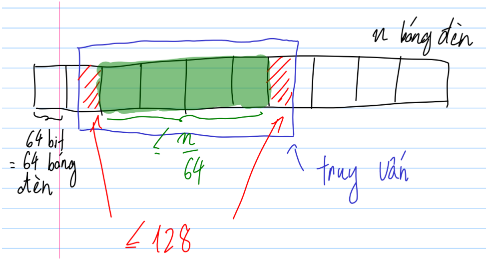
Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Khi thực hiện một truy vấn bật / tắt, ta sẽ chỉ cần chuyển nhiều nhất $\frac{n}{64}$ số nguyên thành $2^{64} - 1$ hoặc 0, và thay đổi giá trị của nhiều nhất 128 bit ở "phần đầu" và "phần cuối" của truy vấn. Tổng cộng ta cần nhiều nhất $\frac{n}{64} + 128$ phép tính.



Kĩ thuật 2: Sử dụng `bitset`

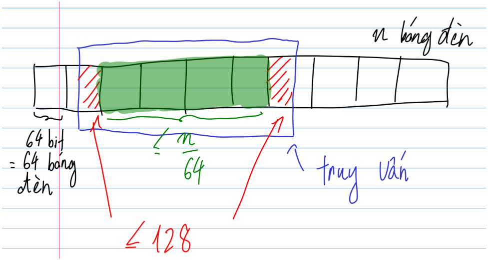
- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính



- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{64}$ số nguyên. Số nguyên đầu tiên khác 0 sẽ chứa bóng đèn bật.

Kĩ thuật 2: Sử dụng `bitset`

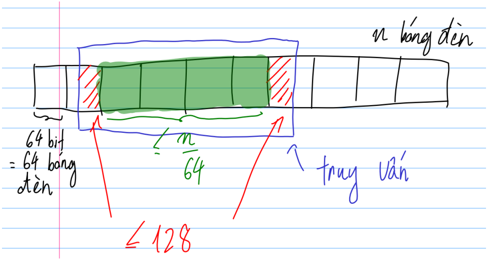
- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \le 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính



- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{64}$ số nguyên. Số nguyên đầu tiên khác 0 sẽ chứa bóng đèn bật. Ta duyệt 64 bit của số nguyên đầu tiên đó bằng nhiều nhất 64 phép tính để có bóng đèn cần tìm.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính



- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{64}$ số nguyên. Số nguyên đầu tiên khác 0 sẽ chứa bóng đèn bật. Ta duyệt 64 bit của số nguyên đầu tiên đó bằng nhiều nhất 64 phép tính để có bóng đèn cần tìm. Tổng cộng ta cần $\frac{n}{64} + 64$ phép tính.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong `C++`)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính
- Truy vấn tìm bóng đèn: $\frac{n}{64} + 64$ phép tính.
  $\Rightarrow$ Một truy vấn cần nhiều nhất $\frac{n}{64} + 128$ phép tính.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính
- Truy vấn tìm bóng đèn: $\frac{n}{64} + 64$ phép tính.
  $\Rightarrow$ Một truy vấn cần nhiều nhất $\frac{n}{64} + 128$ phép tính.

Vậy, ta cần nhiều nhất $\approx q(\frac{n}{64} + 128) \approx \frac{qn}{64}$ phép tính để giải bài toán.

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính
- Truy vấn tìm bóng đèn: $\frac{n}{64} + 64$ phép tính.
  $\Rightarrow$ Một truy vấn cần nhiều nhất $\frac{n}{64} + 128$ phép tính.

Vậy, ta cần nhiều nhất $\approx q(\frac{n}{64} + 128) \approx \frac{qn}{64}$ phép tính để giải bài toán. Với $q = n = 10^5$, $\frac{qn}{64} \approx 1.6 \times 10^8$, có khả năng chạy đủ nhanh trong 1 giây.

Kĩ thuật 2: Sử dụng `bitset`

- Ta có thể biểu diễn 64 bóng đèn bằng một số nguyên không âm 64-bit (kiểu `unsigned long long` trong C++)
  $\Rightarrow$ Ta có thể biểu diễn n bóng đèn bằng một dãy $\frac{n}{64} \leq 1600$ số nguyên 64-bit.
- Truy vấn bật/tắt: $\frac{n}{64} + 128$ phép tính
- Truy vấn tìm bóng đèn: $\frac{n}{64} + 64$ phép tính.
  $\Rightarrow$ Một truy vấn cần nhiều nhất $\frac{n}{64} + 128$ phép tính.

Vậy, ta cần nhiều nhất $\approx q(\frac{n}{64} + 128) \approx \frac{qn}{64}$ phép tính để giải bài toán. Với $q = n = 10^5$, $\frac{qn}{64} \approx 1.6 \times 10^8$, có khả năng chạy đủ nhanh trong 1 giây.
Độ phức tạp của thuật toán vẫn là $O(qn)$ nhưng với hằng số nhỏ.

Kĩ thuật 3: Chia căn

- Ta có thể biểu diễn b bóng đèn bằng một mảng `boolean` gồm b phần tử.

Kĩ thuật 3: Chia căn

- Ta có thể biểu diễn b bóng đèn bằng một mảng `boolean` gồm b phần tử. $\Rightarrow$ Ta cần $\frac{n}{b}$ mảng như vậy.

Kĩ thuật 3: Chia căn

- Ta có thể biểu diễn b bóng đèn bằng một mảng `boolean` gồm b phần tử. $\Rightarrow$ Ta cần $\frac{n}{b}$ mảng như vậy.
- Với mỗi một mảng, ta tạo thêm một biến cho mảng đó là số lượng bóng đèn bật trong mảng. Nếu số lượng bóng đèn là 0 thì ta coi toàn bộ bóng đèn biểu diễn bởi mảng đó tắt và số lượng bóng đèn là b thì ta coi toàn bộ bóng đèn biểu diễn bởi mảng đó bật.

Kĩ thuật 3: Chia căn

- Khi thực hiện một truy vấn bật / tắt, ta cần thay đổi giá trị của biến số lượng bóng đèn của nhiều nhất $\frac{n}{b}$ mảng, và thay đổi giá trị của nhiều nhất $2b$ biến `boolean` của hai mảng đầu và cuối.

Kĩ thuật 3: Chia căn

- Khi thực hiện một truy vấn bật / tắt, ta cần thay đổi giá trị của biến số lượng bóng đèn của nhiều nhất $\frac{n}{b}$ mảng, và thay đổi giá trị của nhiều nhất $2b$ biến `boolean` của hai mảng đầu và cuối. Tổng cộng ta cần nhiều nhất $\sim \frac{n}{b} + 2b$ phép tính.

Kĩ thuật 3: Chia căn

- Khi thực hiện một truy vấn bật / tắt, ta cần thay đổi giá trị của biến số lượng bóng đèn của nhiều nhất $\frac{n}{b}$ mảng, và thay đổi giá trị của nhiều nhất $2b$ biến `boolean` của hai mảng đầu và cuối. Tổng cộng ta cần nhiều nhất $\sim \frac{n}{b} + 2b$ phép tính.
- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{b}$ số nguyên để tìm mảng đầu tiên có bóng đèn bật. Sau khi tìm được mảng, ta duyệt tiếp b biến `boolean` để xác định bóng đèn cần tìm.

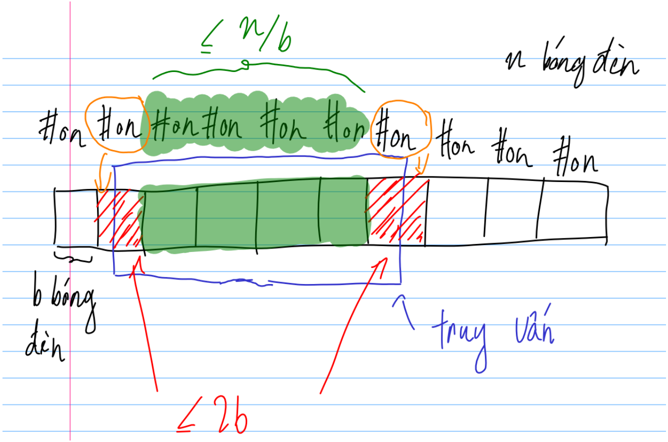
- Khi thực hiện một truy vấn bật / tắt, ta cần thay đổi giá trị của biến số lượng bóng đèn của nhiều nhất $\frac{n}{b}$ mảng, và thay đổi giá trị của nhiều nhất $2b$ biến `boolean` của hai mảng đầu và cuối. Tổng cộng ta cần nhiều nhất $\sim \frac{n}{b} + 2b$ phép tính.
- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{b}$ số nguyên để tìm mảng đầu tiên có bóng đèn bật. Sau khi tìm được mảng, ta duyệt tiếp b biến `boolean` để xác định bóng đèn cần tìm. Tổng cộng ta cần $\sim \frac{n}{b} + b$ phép tính.

Kĩ thuật 3: Chia căn

- Khi thực hiện một truy vấn bật / tắt, ta cần thay đổi giá trị của biến số lượng bóng đèn của nhiều nhất $\frac{n}{b}$ mảng, và thay đổi giá trị của nhiều nhất $2b$ biến `boolean` của hai mảng đầu và cuối. Tổng cộng ta cần nhiều nhất $\sim \frac{n}{b} + 2b$ phép tính.
- Để tìm bóng đèn bật có chỉ số thấp nhất, ta duyệt mảng $\frac{n}{b}$ số nguyên để tìm mảng đầu tiên có bóng đèn bật. Sau khi tìm được mảng, ta duyệt tiếp b biến `boolean` để xác định bóng đèn cần tìm. Tổng cộng ta cần $\sim \frac{n}{b} + b$ phép tính.

Vậy ta cần nhiều nhất $q(\frac{n}{b} + 2b)$ phép tính.

Kĩ thuật 3: Chia căn



Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$
$$\Rightarrow q(\frac{n}{b} + 2b) \geq 2\sqrt{2}q\sqrt{n}$$

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$
$$\Rightarrow q(\frac{n}{b} + 2b) \geq 2\sqrt{2}q\sqrt{n}$$

Đẳng thức xảy ra khi $\frac{n}{b} = 2b \Rightarrow 2b^2 = n \Rightarrow b = \sqrt{\frac{n}{2}}$

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$
$$\Rightarrow q(\frac{n}{b} + 2b) \geq 2\sqrt{2}q\sqrt{n}$$

Đẳng thức xảy ra khi $\frac{n}{b} = 2b \Rightarrow 2b^2 = n \Rightarrow b = \sqrt{\frac{n}{2}}$
(Khi $n = 10^5$, ta chọn $b = 224$)

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$
$$\Rightarrow q(\frac{n}{b} + 2b) \geq 2\sqrt{2}q\sqrt{n}$$

Đẳng thức xảy ra khi $\frac{n}{b} = 2b \Rightarrow 2b^2 = n \Rightarrow b = \sqrt{\frac{n}{2}}$
(Khi $n = 10^5$, ta chọn $b = 224$)
Khi ta chọn $b = \sqrt{\frac{n}{2}}$ thì ta sẽ cần $2\sqrt{2}q\sqrt{n} \in O(q\sqrt{n})$ phép tính.

Chọn b thế nào?

Do $\frac{n}{b} > 0$ và $2b > 0$, theo bất đẳng thức AM - GM, ta có

$$\frac{n}{b} + 2b \geq 2\sqrt{\frac{n}{b} \cdot 2b} = 2\sqrt{2}\sqrt{n}$$
$$\Rightarrow q(\frac{n}{b} + 2b) \geq 2\sqrt{2}q\sqrt{n}$$

Đẳng thức xảy ra khi $\frac{n}{b} = 2b \Rightarrow 2b^2 = n \Rightarrow b = \sqrt{\frac{n}{2}}$
(Khi $n = 10^5$, ta chọn $b = 224$)
Khi ta chọn $b = \sqrt{\frac{n}{2}}$ thì ta sẽ cần $2\sqrt{2}q\sqrt{n} \in O(q\sqrt{n})$ phép tính.
Như vậy là ta đã giảm được độ phức tạp từ $O(qn)$ xuống $O(q\sqrt{n})$

Một số lưu ý

- Sau khi chọn được b cho giá trị n lớn nhất, ta có thể đặt b làm hằng số và dùng luôn giá trị b này cho các giá trị n nhỏ hơn.

Một số lưu ý

- Sau khi chọn được b cho giá trị n lớn nhất, ta có thể đặt b làm hằng số và dùng luôn giá trị b này cho các giá trị n nhỏ hơn.
- Không phải lúc nào chọn $b \in O(\sqrt{n})$ cũng đúng. Ví dụ: Machine Learning - Codeforces Round #466.

Một số lưu ý

- Sau khi chọn được b cho giá trị n lớn nhất, ta có thể đặt b làm hằng số và dùng luôn giá trị b này cho các giá trị n nhỏ hơn.
- Không phải lúc nào chọn $b \in O(\sqrt{n})$ cũng đúng. Ví dụ: Machine Learning - Codeforces Round #466.
- Có nhiều cách chia căn khác nhau
  $\Rightarrow$ Vào `https://oj.vnoi.info/tags/`, sau đó nhấn Chia căn (Sqrt Decomposition) để tìm bài luyện tập.

Kĩ thuật 4: Hai con trỏ

Bài toán

Cho một mảng a gồm n số nguyên dương và một số nguyên dương S. Tìm một dãy con liên tiếp của mảng a sao cho tổng của dãy con bằng S

Kĩ thuật 4: Hai con trỏ

Bài toán

Cho một mảng a gồm n số nguyên dương và một số nguyên dương S. Tìm một dãy con liên tiếp của mảng a sao cho tổng của dãy con bằng S

- Thuật $O(n^3)$?

Kĩ thuật 4: Hai con trỏ

Bài toán

Cho một mảng a gồm n số nguyên dương và một số nguyên dương S. Tìm một dãy con liên tiếp của mảng a sao cho tổng của dãy con bằng S

- Thuật $O(n^3)$?
- Thuật $O(n^2)$?

Kĩ thuật 4: Hai con trỏ

Bài toán

Cho một mảng a gồm n số nguyên dương và một số nguyên dương S. Tìm một dãy con liên tiếp của mảng a sao cho tổng của dãy con bằng S

- Thuật $O(n^3)$?
- Thuật $O(n^2)$?
- Thuật $O(n)$?

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \le j \le r : \sum_{i=l+1}^{j} a_i \ne S$

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$
Chọn j bất kì sao cho $l+1 \leq j \leq r$

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$
Chọn j bất kì sao cho $l+1 \leq j \leq r$
Ta có $\sum_{i=l+1}^{j} a_i < a_l + \sum_{i=l+1}^{r} a_i$ (do $a_l > 0$)

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$
Chọn j bất kì sao cho $l+1 \leq j \leq r$
Ta có $\sum_{i=l+1}^{j} a_i < a_l + \sum_{i=l+1}^{r} a_i$ (do $a_l > 0$)
$\Rightarrow \sum_{i=l+1}^{j} a_i < \sum_{i=l}^{r} a_i < S$

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$
Chọn j bất kì sao cho $l+1 \leq j \leq r$
Ta có $\sum_{i=l+1}^{j} a_i < a_l + \sum_{i=l+1}^{r} a_i$ (do $a_l > 0$)
$\Rightarrow \sum_{i=l+1}^{j} a_i < \sum_{i=l}^{r} a_i < S$
$\Rightarrow \sum_{i=l+1}^{j} \neq S$

Kĩ thuật 4: Hai con trỏ

Nhận xét

Nếu $\sum_{i=l}^{r} a_i < S$ thì $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i \neq S$

Giả sử $\sum_{i=l}^{r-1} a_i < S$.
Ta cần chứng minh $\forall l+1 \leq j \leq r : \sum_{i=l+1}^{j} a_i < S$
Chọn j bất kì sao cho $l+1 \leq j \leq r$
Ta có $\sum_{i=l+1}^{j} a_i < a_l + \sum_{i=l+1}^{r} a_i$ (do $a_l > 0$)
$\Rightarrow \sum_{i=l+1}^{j} a_i < \sum_{i=l}^{r} a_i < S$
$\Rightarrow \sum_{i=l+1}^{j} \neq S$
và ta có điều phải chứng minh.

Kĩ thuật 4: Hai con trỏ

Kĩ thuật 4: Hai con trỏ

Ta thấy j sẽ dừng ở các điểm $j_1, j_2, ..., j_n$ sao cho $1 \leq j_1 \leq j_2 ... \leq j_n = n$

Ta thấy j sẽ dừng ở các điểm $j_1, j_2, ..., j_n$ sao cho
$1 \leq j_1 \leq j_2 ... \leq j_n = n$
Đặt $j_0 = 1$.

Kĩ thuật 4: Hai con trỏ

Ta thấy j sẽ dừng ở các điểm $j_1, j_2, ..., j_n$ sao cho
$1 \le j_1 \le j_2 ... \le j_n = n$
Đặt $j_0 = 1$.
Số phép tính cần thực hiện là:

Ta thấy j sẽ dừng ở các điểm $j_1, j_2, ..., j_n$ sao cho
$1 \leq j_1 \leq j_2 ... \leq j_n = n$
Đặt $j_0 = 1$.
Số phép tính cần thực hiện là:
$\sum_{i=1}^{n} \sum_{j=j_{i-1}}^{j_i} 1 = \sum_{i=1}^{n} (j_i - j_{i-1} + 1) =$
$\sum_{i=1}^{n} (j_i - j_{i-1}) + \sum_{i=1}^{n} 1 = n + n = 2n \in O(n)$

Kĩ thuật 4: Hai con trỏ

Ta thấy j sẽ dừng ở các điểm $j_1, j_2, ..., j_n$ sao cho
$1 \leq j_1 \leq j_2 ... \leq j_n = n$
Đặt $j_0 = 1$.
Số phép tính cần thực hiện là:
$\sum_{i=1}^{n} \sum_{j=j_{i-1}}^{j_i} 1 = \sum_{i=1}^{n} (j_i - j_{i-1} + 1) =$
$\sum_{i=1}^{n} (j_i - j_{i-1}) + \sum_{i=1}^{n} 1 = n + n = 2n \in O(n)$
Vậy thuật toán sử dụng kĩ thuật hai con trỏ có độ phức tạp $O(n)$