Thuật toán tìm kiếm

Nội dung

- Thuật toán và độ phức tạp
- Thuật toán tìm kiếm
- Bài tập thực hành

Thuật toán

📌 **Định nghĩa**

Thuật toán là **tập hợp hữu hạn các bước chỉ dẫn rõ ràng**, nhằm giải quyết một bài toán cụ thể.

📌 **Đặc điểm**

- **Tính hữu hạn**: phải kết thúc sau một số bước nhất định.
- **Tính xác định**: mỗi bước phải rõ ràng, không mơ hồ.
- **Tính hiệu quả**: các bước có thể thực hiện được trong thực tế.

📌 **Ví dụ**

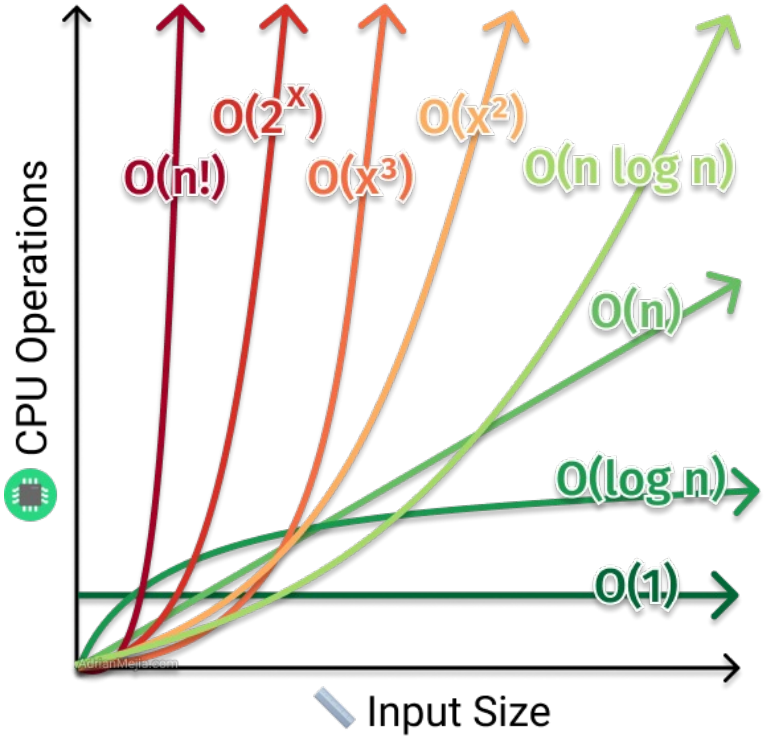
- Tìm số lớn nhất trong một dãy số.
- Thuật toán Euclid tìm **ước số chung lớn nhất**.
- Thuật toán sàng **Eratosthenes** tìm tất cả số nguyên tố từ 1 đến N

Độ phức tạp tính toán

- **Định nghĩa**: Đo lường thời gian hoặc bộ nhớ mà thuật toán cần khi kích thước đầu vào tăng.
- **Mục đích**: So sánh hiệu quả giữa các thuật toán.
- **Hai loại chính**:
  - **Time Complexity** (thời gian)
  - **Space Complexity** (bộ nhớ)

Big-O và một số độ phức tạp phổ biến

- Big-O: Tốc độ tăng trưởng khi n → ∞
- Một số dạng thường gặp:
  - O(1): Hằng số
  - O(log n): Logarit (Binary Search)
  - O(n): Tuyến tính (Linear Search)
  - O(n log n): Hiệu quả (Merge Sort)
  - O(n^2): Bình phương (Bubble Sort)
  - O(n^3): Lập phương (Nhân ma trận)
  - O(2^n): Bùng nổ (Dãy Fibonacci)
  - O(n!): Tồi tệ (Sinh hoán vị)



Ví dụ độ phức tạp

- Linear Search → O(n)
- Binary Search → O(log n)
- Bubble Sort → $O(n^2)$
- Merge Sort → O(n log n)
- Heap Sort → O(n log n)
- QuickSort → O(n log n)

💡 Bài học: chọn thuật toán đúng quan trọng hơn chọn máy tính mạnh

Cách tính độ phức tạp

- Bước 1: Đếm số phép toán chính (cộng, trừ, nhân, chia, gán, so sánh)
- Bước 2: Biểu diễn theo n (kích thước đầu vào)
- Bước 3: Bỏ hằng số và bậc thấp → lấy bậc cao nhất
  - Ví dụ: Vòng lặp lồng nhau → $O(n^2)$

Bài toán

Cho một danh sách gồm n phần tử - Không gian tìm kiếm

Tìm các phần tử thoả mãn điều kiện logic

Điều kiện logic có thể là:

- Có giá trị bằng X cho trước
- Có giá trị lớn hơn X cho trước

Tìm kiếm tuần tự

19

19 19 19 19 19

Giá trị	34	28	27	7	19	24	29	16
Vị trí	0	1	2	3	4	5	6	7

Tìm kiếm tuần tự

là một phương pháp tìm kiếm bằng cách duyệt lần lượt từng phần tử của danh sách cho đến lúc tìm thấy giá trị mong muốn hay đã duyệt qua toàn bộ danh sách.

```
int search(int arr[], int n, int x) {
    for (int i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

O(n)

Bài toán

Một cuốn sách được đánh số trang từ 1 đến 500

Hãy tìm trang số 315

Tìm kiếm nhị phân

Tư duy “Chia để trị” (Devide and Conquer)

34

7	10	15	19	23	28	32	34	42	49	52
						32	34	42	49	52
						32	34			
							34			

Tìm kiếm nhị phân

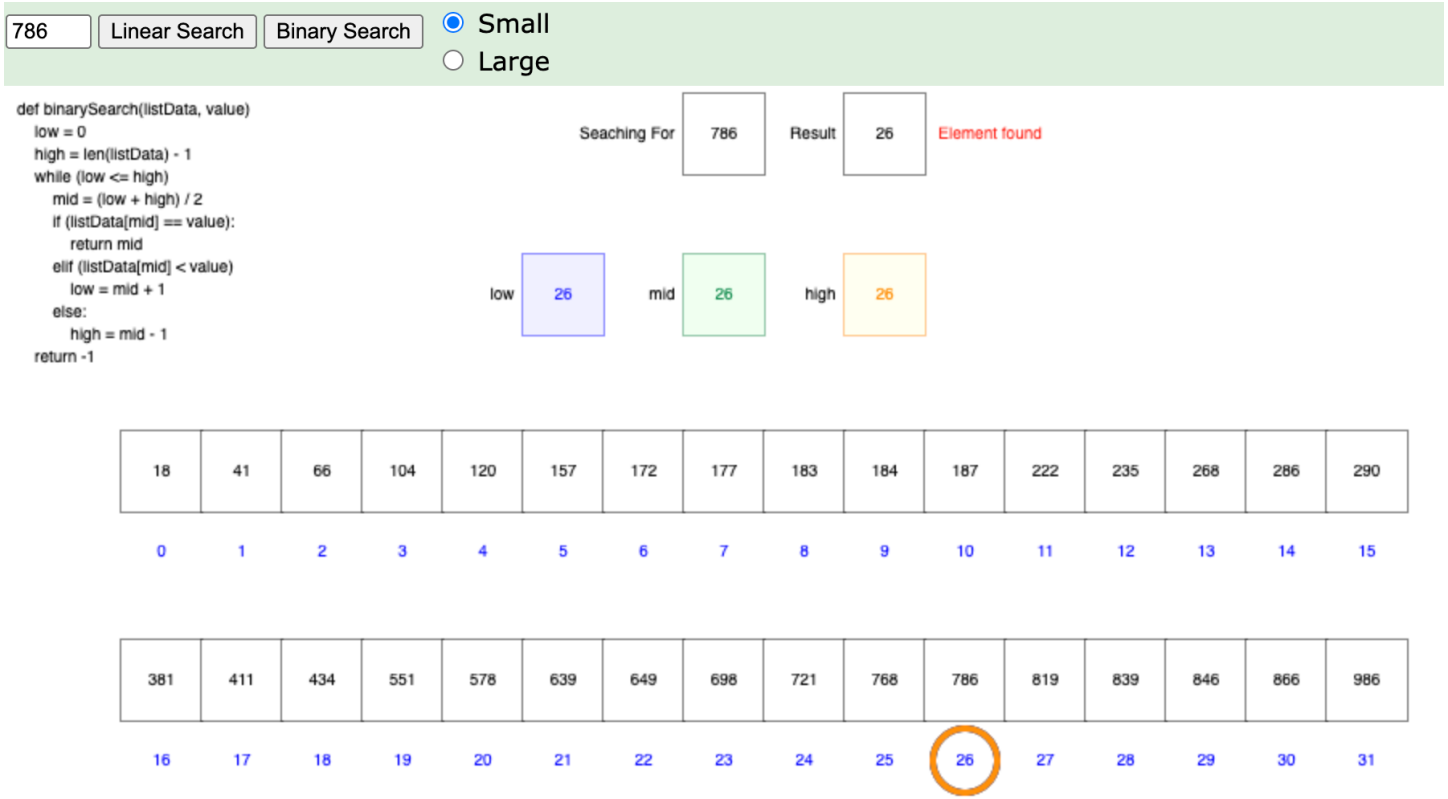
```
BinarySearch(Arr, l, r, x)
        if (l <= r)
                mid = (l + r) / 2
                if (Arr[mid] == x)
                        return mid
                else
                        if (Arr[mid] > x)
                                BinarySearch(Arr, l, mid - 1, x)
                        else
                                BinarySearch(Arr, mid + 1, r, x)
        else
                return -1
```

Tìm kiếm nhị phân

```
BinarySearch(Arr, l, r, x)
        while (l <= r)
                mid = (l + r) / 2
                if (Arr[mid] == x)
                        return mid
                else
                        if (Arr[mid] > x)
                                r = mid - 1
                        else
                                l = mid + 1
        return -1
```

Tìm kiếm nhị phân

Trực quan hóa: https://www.cs.usfca.edu/~galles/visualization/Search.html



Tính độ phức tạp của thuật toán tìm kiếm nhị phân

```
def binary_search(arr, x):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

- **Mỗi vòng lặp**:
  - Tính mid
  - So sánh arr[mid] với x
  - Giảm nửa khoảng tìm kiếm
  - 👉 Số phép toán trong 1 vòng lặp là **O(1)**.
- **Số vòng lặp tối đa**:
  - Ban đầu có n phần tử
  - Sau mỗi bước, chỉ còn n/2, rồi n/4, n/8, …
  - Sau k bước: còn n / 2^k
  - Khi n / 2^k = 1 ⇒ 2^k = n ⇒ k = $\log_2 n$
- **Tổng số phép toán**:
  - ≈ $\log_2 n$ vòng lặp × O(1) phép toán/vòng

=> Độ phức tạp = O(log n)

So sánh độ phức tạp

	1 lần tìm kiếm	m lần tìm kiếm
Tìm kiếm tuần tự	O(n)	O(m*n)
Tìm kiếm nhị phân	O(n * log(n))	O((n + m) * log(n))

Bài tập thực hành

Bài 1: Thuật toán Sàng Eratosthenes tìm số nguyên tố
Viết chương trình sử dụng **thuật toán Sàng Eratosthenes** để liệt kê tất cả các số nguyên tố nhỏ hơn hoặc bằng một số nguyên dương n cho trước.
Phương pháp

- Khởi tạo một mảng đánh dấu từ 2 đến n (ban đầu tất cả coi là số nguyên tố).
- Bắt đầu từ số p = 2, gạch bỏ tất cả bội số của p lớn hơn p.
- Tìm số tiếp theo chưa bị gạch, đặt làm p mới, tiếp tục gạch bội số của nó.
- Dừng lại khi $p^2 > n$.
- Các số chưa bị gạch bỏ chính là số nguyên tố.

📩 **Output (Minh họa)**

Ví dụ với **n = 30**:
Các số nguyên tố: 2 3 5 7 11 13 17 19 23 29

Bài tập thực hành

Bài 2: Tìm vị trí (Cơ bản)

Cho một mảng số nguyên `nums` đã được **sắp xếp theo thứ tự tăng dần** và một số nguyên `target`. Bạn hãy viết một hàm để tìm `target` trong `nums`.

- Nếu `target` tồn tại, trả về chỉ số (index) của nó.
- Nếu `target` không tồn tại, trả về `-1`.

Yêu cầu

Thuật toán của bạn phải có độ phức tạp thời gian là O(logn).

Ví dụ

- **Input:** `nums = [-1, 0, 3, 5, 9, 12], target = 9`
- **Output:** `4`
- **Giải thích:** Số 9 tồn tại trong mảng `nums` tại vị trí 4.
- **Input:** `nums = [-1, 0, 3, 5, 9, 12], target = 2`
- **Output:** `-1`
- **Giải thích:** Số 2 không tồn tại trong mảng `nums`.

Bài tập thực hành

Bài 3: Đoán ngày sinh

Bài toán này yêu cầu bạn áp dụng tư duy "chia để trị" của tìm kiếm nhị phân vào một không gian lời giải.

Đề bài

A và B chơi trò đoán ngày sinh (ngày, tháng). Luật chơi như sau: A hỏi B một số câu hỏi, B trả lời bằng cách nói “Đúng” hoặc “Sai”, căn cứ vào đó A điều chỉnh câu hỏi của mình…

- Hãy trình bày và lập trình thực hiện thuật toán giúp A tìm ra ngày sinh của B sau ít bước nhất có thể.
- Số câu hỏi tối đa mà A cần hỏi để biết chính xác ngày sinh của B là bao nhiêu?

Bài tập thực hành

Bài 4: Tìm vị trí đầu tiên và cuối cùng (Nâng cao) 🌟

Cho một mảng số nguyên nums đã được **sắp xếp tăng dần**, *có thể chứa các giá trị trùng lặp*. Hãy tìm dãy con trong *nums* có giá trị target cho trước.

Nếu target được tìm thấy, trả về một mảng [first_position, last_position].

Nếu target không được tìm thấy, trả về [-1, -1].

Yêu cầu: Thuật toán phải có độ phức tạp thời gian là O(logn).

Ví dụ

- **Input:** nums = [5, 7, 7, 8, 8, 10], target = 8
- **Output:** [3, 4]
- **Input:** nums = [5, 7, 7, 8, 8, 10], target = 6
- **Output:** [-1, -1]
- **Input:** nums = [], target = 0
- **Output:** [-1, -1]

Gợi ý: Thực hiện tìm kiếm nhị phân hai lần, một lần để tìm vị trí **đầu tiên** (cận trái) của target, một lần nữa để tìm vị trí **cuối cùng** (cận phải) của target.

Contact

TS. Nguyễn Thanh Bình
binh.nguyenthanh@phenikaa-uni.edu.vn