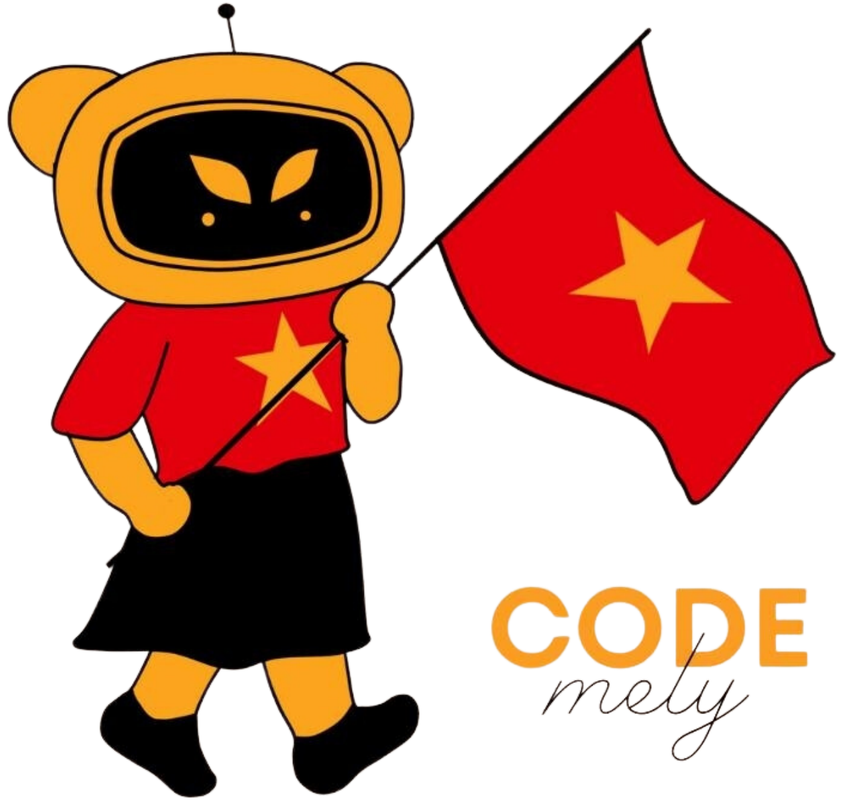
CODEMELY

THUẬT TOÁN TÌM KIẾM

Nguyễn Minh Quang



THUẬT TOÁN TÌM KIẾM LÀ GÌ ?

Thuật toán tìm kiếm được thiết kế để kiểm tra 1 phần tử hoặc truy xuất một phần tử từ bất kỳ cấu trúc dữ liệu nào mà nó được lưu trữ.

LEARNING TARGETS

01 Tìm kiếm tuần tự (Sequential Search)

02 Tìm kiếm khoảng (Interval Search)

03 Thực hành

01 TÌM KIẾM TUẦN TỰ SEQUENTIAL SEARCH

Ví dụ: Tìm kiếm tuần tự (Linear Search)

Tìm kiếm tuần tự (Linear Search) được xác định là một thuật toán tìm kiếm tuần tự **bắt đầu từ một đầu** và **duyệt qua từng phần tử của danh sách cho đến khi tìm thấy phần tử mong muốn**, **nếu không** thì tìm kiếm tiếp tục cho **đến cuối tập dữ liệu.**

01 LINEAR SEARCH HOẠT ĐỘNG NHƯ THẾ NÀO ?

Trong thuật toán Tìm kiếm tuần tự:

- Mỗi phần tử được xem xét như là một phần tử có thể khớp với khóa (key) và kiểm tra điều này.
- Nếu bất kỳ phần tử nào được tìm thấy bằng giá trị bằng với khóa, thì tìm kiếm thành công và trả về chỉ số của phần tử đó.
- Nếu không có phần tử nào được tìm thấy có giá trị bằng với khóa, thì tìm kiếm trả về kết quả "Không tìm thấy khớp".

01 LINEAR SEARCH HOẠT ĐỘNG NHƯ THẾ NÀO ?

Ví dụ: Xem xét mảng arr[] = {10, 50, 30, 70, 80, 20, 90, 40} và khóa (key) = 30.

- **Bước 1:** Bắt đầu từ phần tử đầu tiên (chỉ số 0) và so sánh khóa (key) với mỗi phần tử (arr[i]).
  - So sánh khóa với phần tử đầu tiên arr[0]. Vì không bằng nhau, trình lặp di chuyển đến phần tử tiếp theo như một phần tử có thể khớp.
  - So sánh khóa với phần tử tiếp theo arr[1]. Vì không bằng nhau, trình lặp di chuyển đến phần tử tiếp theo như một phần tử có thể khớp.
- **Bước 2:** Bây giờ, khi so sánh arr[2] với khóa (key), giá trị khớp. Vì vậy, thuật toán Tìm kiếm tuần tự sẽ trả về một thông báo thành công và trả về chỉ số của phần tử khi tìm thấy khóa (ở đây là 2).

01 PHÂN TÍCH ĐỘ PHỨC TẠP CỦA TÌM KIẾM TUẦN TỰ:

Phân tích Độ phức tạp của Tìm kiếm tuần tự:

Độ phức tạp thời gian:

- Trường hợp tốt nhất: O(1).
- Trường hợp xấu nhất: O(N), trong đó N là kích thước của danh sách.
- Trường hợp trung bình: O(N)

Không gian phụ: O(1) vì ngoài biến để lặp qua danh sách, không có biến nào được sử dụng.

01 ƯU VÀ NHƯỢC ĐIỂM

Ưu điểm:

- Tìm kiếm tuần tự có thể được sử dụng mà không cần xem xét danh sách có được sắp xếp hay không. Nó có thể được sử dụng trên mảng của bất kỳ loại dữ liệu nào.
- Không yêu cầu bộ nhớ bổ sung.
- Nó là một thuật toán phù hợp cho các tập dữ liệu nhỏ.

Nhược điểm:

- Tìm kiếm tuần tự có độ phức tạp thời gian là O(N), điều này làm cho nó chậm đối với các tập dữ liệu lớn.
- Không phù hợp cho các mảng lớn.

01 KỸ THUẬT LÍNH CANH

Bài toán: Tìm kiếm vị trí phần tử x trên mảng một chiều có n phần tử.

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

```
i = 0;
while (i < n && arr[i] != x)
    i++;
if (i < n) return i;
else return -1;
```

Đặt vấn đề: Tốn bao nhiêu phép so sánh trong trường hợp xấu nhất ?

01 KỸ THUẬT LÍNH CANH

Ta có: Kỹ thuật lính canh

```
arr[n] = target;
int i = 0;
while (arr[i] != target)
    i++;
if (i < n) return i;
else return -1;
```

Đặt vấn đề: có tối ưu hơn 2 cái trước đó không nhỉ ?

02

TÌM KIẾM KHOẢNG
INTERVAL SEARCH

Tìm kiếm khoảng: Những thuật toán này được thiết kế đặc biệt để tìm kiếm trong cấu trúc dữ liệu đã được sắp xếp. Loại tìm kiếm này hiệu quả hơn nhiều so với Tìm kiếm tuần tự vì chúng nhắm đến trung tâm của cấu trúc tìm kiếm liên tục và chia không gian tìm kiếm làm hai nửa. Ví dụ: Tìm kiếm nhị phân (Binary Search).

02 BINARY SEARCH LÀ GÌ ?

Tìm kiếm nhị phân được xác định là một thuật toán tìm kiếm được sử dụng trong một mảng **đã được sắp xếp** bằng cách liên tục **chia đôi khoảng tìm kiếm**.

02 BINARY SEARCH HOẠT ĐỘNG NHƯ THẾ NÀO ?

1. Chia không gian tìm kiếm thành hai phần bằng cách tìm chỉ số giữa "mid".
2. Tìm chỉ số giữa "mid" trong thuật toán Tìm kiếm nhị phân.
3. So sánh phần tử ở giữa của không gian tìm kiếm với khóa cần tìm.
4. Nếu khóa được tìm thấy ở phần tử giữa, quá trình được kết thúc.
5. Nếu khóa không được tìm thấy ở phần tử giữa, chọn nửa nào sẽ được sử dụng làm không gian tìm kiếm tiếp theo.
6. Nếu khóa nhỏ hơn phần tử giữa, sử dụng phần bên trái cho tìm kiếm tiếp theo.
7. Nếu khóa lớn hơn phần tử giữa, sử dụng phần bên phải cho tìm kiếm tiếp theo.
8. Quá trình này tiếp tục cho đến khi khóa được tìm thấy hoặc không gian tìm kiếm tổng cộng được dùng hết.

02

VÍ DỤ

Xem xét một mảng arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}, và mục tiêu (target) là 23.

1. **Bước đầu tiên:**
   - Tính toán chỉ số giữa (mid) và so sánh phần tử giữa với khóa cần tìm (key). Nếu khóa nhỏ hơn phần tử giữa (mid), di chuyển sang phía trái, và nếu khóa lớn hơn phần tử giữa, di chuyển không gian tìm kiếm sang phía phải.
   - Khóa (tức là 23) lớn hơn phần tử giữa hiện tại (tức là 16). Không gian tìm kiếm di chuyển sang bên phải.
   - Khóa nhỏ hơn phần tử giữa hiện tại là 56. Không gian tìm kiếm di chuyển sang bên trái.
2. **Bước thứ hai:** Nếu khóa khớp với giá trị của phần tử giữa, phần tử đó được tìm thấy và tìm kiếm dừng lại.

LÀM THẾ NÀO ĐỂ CÀI ĐẶT TÌM KIẾM NHỊ PHÂN?

01 Tìm kiếm nhị phân lặp lại
(Iterative Binary Search Algorithm)

02 Tìm kiếm nhị phân đệ quy
(Recursive Binary Search Algorithm)

02

TÌM KIẾM NHỊ PHÂN LẶP (ITERATIVE BINARY SEARCH)

```
int search(int arr[], int n, int x)
{
    int l = 0, r = n - 1;
    while (l <= r)
    {
        int mid = (l+r)/2;
        if (arr[mid] == x) return mid; // Nếu phần tử thứ arr thứ mid sẽ trả về giá trị mid
        if (arr[mid] < x) l = mid + 1; // nếu arr[mid] nhỏ hơn thì dịch chuyển qua phần bên phải để tìm kiếm
        else r = mid - 1; // ngược lại thì dịch chuyển qua phần bên trái để tìm kiếm
    }
    return -1;
}
```

02

TÌM KIẾM NHỊ PHÂN BẰNG ĐỆ QUY (RECURSIVE BINARY SEARCH)

```
int search_ver2(int arr[], int l, int r, int x)
{
    if (l <= r)
    {
        int mid = (l+r)/2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) return search_ver2(arr,mid+1,r,x);
        else return search_ver2(arr, l, mid - 1, x);
    }
    else return -1;
}
```

02 ƯU ĐIỂM

1. Tìm kiếm nhị phân nhanh hơn tìm kiếm tuyến tính, **đặc biệt là đối với mảng có kích thước lớn.**
2. Hiệu quả hơn so với các thuật toán tìm kiếm khác có cùng độ phức tạp thời gian, như tìm kiếm nội suy hoặc tìm kiếm mũ.
3. Tìm kiếm nhị phân thích hợp cho việc tìm kiếm trong các tập dữ liệu lớn được lưu trữ trong bộ nhớ bên ngoài, chẳng hạn như trên ổ cứng hoặc trong đám mây.

02 NHƯỢC ĐIỂM

1. Mảng phải được sắp xếp.
2. Tìm kiếm nhị phân yêu cầu cấu trúc dữ liệu đang được tìm kiếm được lưu trữ trong các vị trí bộ nhớ liền kề.
3. Tìm kiếm nhị phân yêu cầu các phần tử trong mảng có thể so sánh được, nghĩa là chúng phải có thể được sắp xếp.

BINARY SEARCH FUNCTIONS IN C++ STL

01

binary_search:
binary_search(start_ptr, end_ptr, num):

02

Lower_bound:
lower_bound(start_ptr, end_ptr, num):

03

Upper_bound:
upper_bound(start_ptr, end_ptr, num)

PRACTICE WITH CODEMELY

Nguyễn Minh Quang

THANK YOU
HẸN GẶP LẠI NHÉ