# Sắp Xếp và Tìm Kiếm

Tìm kiếm là quá trình xác định một đối tượng nào đó trong một tập các đối tượng. Kết quả trả về là đối tượng tìm được (nếu có) hoặc một chỉ số (nếu có)

xác định vị trí của đối tượng trong tập đó.

 Việc tìm kiếm dựa theo một trường nào đó của đối tượng

 VD: đối tượng sinh viên có các dữ liệu {MaSV, HoTen, DiaChi,…}. Khi đó có thể tìm kiếm trên danh sách sinh viên theo trường MaSV hoặc HoTen.

 Bài toán được mô tả như sau:

- Tập dữ liệu được lưu trữ là dãy a1 , a2 ,..,an . Giả sử chọn cấu trúc dữ liệu mảng để lưu trữ dãy số này trong bộ nhớ chính, có khai báo: int a[n];

- Giá trị cần tìm là x, có kiểu nguyên: int x;

Tìm Kiếm

Tìm Kiếm Nhị Phân

Tìm Kiếm Tuyến Tính

Tập dữ liệu đã được sắp xếp

Tập dữ liệu bất kỳ

**\* Tìm Kiếm Tuyến Tính**

 Ý tưởng chính: duyệt tuần tự từ phần tử đầu tiên, lần lượt so sánh khóa tìm kiếm với các phần tử trong danh sách. Cho đến khi gặp phần tử cần tìm hoặc đến khi duyệt hết danh sách.

 Các bước tiến hành:

- Bước 1: duyệt tuần tự từ phần tử đầu tiên

- Bước 2: so sánh các phần tử trong danh sách với khóa tìm kiếm có hai khả năng

 Nếu bằng nhau => tìm thấy => Dừng

 Nếu khác nhau chuyển sang bước 3

- Bước 3: xét phần tử kế tiếp trong mảng

Nếu hết mảng, không tìm thấy => Dừng

 Nếu chưa hết mảng quay lại bước 2

 Các bước tiến hành:

- Bước 1: i = 0;

- Bước 2: So sánh a[i] với x, có hai khả năng

 a[i] = x: Tìm thấy => Trả về i => Dừng

 a[i]  x: Sang bước 3

- Bước 3: i ++ // xét phần tử kế tiếp trong mảng

 Nếu i > N: Hết mảng, không tìm thấy => Trả về -1 => Dừng

 Nếu i <= N: Quay lại bước

 Ví dụ: Cho dãy số a, giá trị tìm x = 8

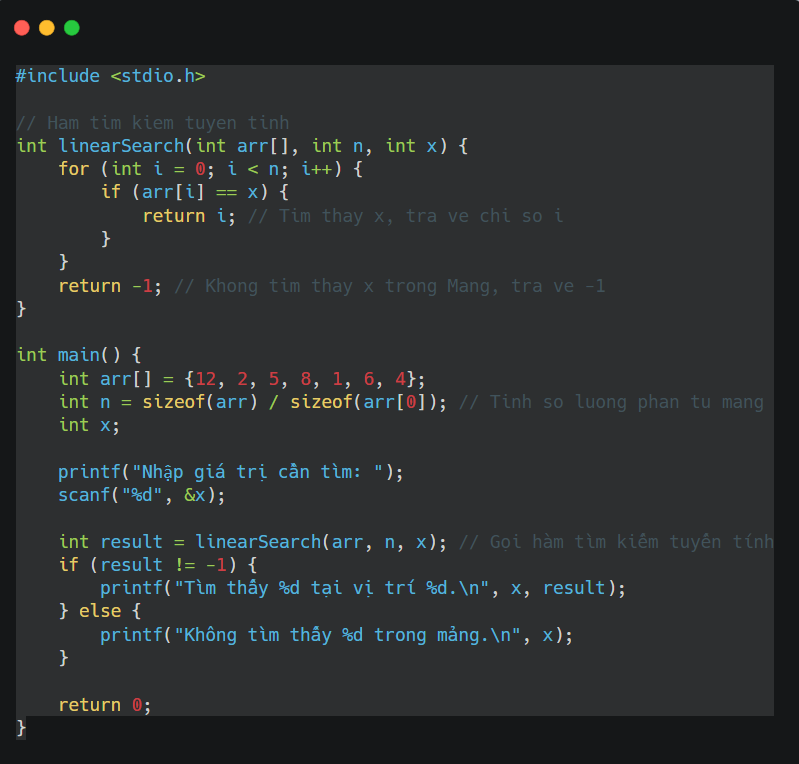
12 2 5 8 1 6 4

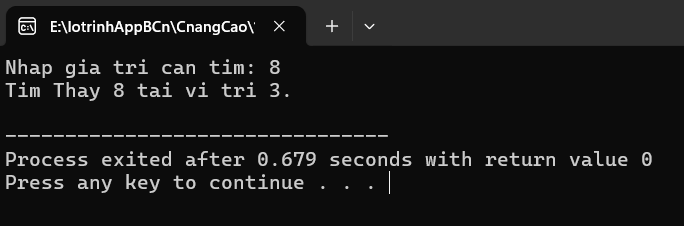
Minh Họa Tìm Kiếm Tuyến Tính

Tìm Thấy

X = 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 12 | 2 | 5 | 8 | 1 | 6 | 4 |





 Nhận xét

- Giải thuật tìm kiếm tuyến tính không phụ thuộc vào thứ tự của các phần tử trong mảng, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy bất kỳ.

**\* Tìm Kiếm Nhị Phân**

Phép tìm kiếm nhị phân được áp dụng trên dãy khóa đã có thứ tự:

**a[0]** <= **a[1]** <= … <= **a[n-1]**.

 Ý tưởng

- Giả sử ta cần tìm trong đoạn a[left,..,right] với khóa tìm kiếm là x, trước hết ta xét phần tử giữa là **a[mid]**, với mid=[left+right]/2.

 Nếu a[mid] = x thì việc tìm kiếm thành công.

 Nếu a[mid] < x thì có nghĩa là đoạn **a[left]** đến **a[mid]** chỉ chứa khóa < x, ta tiến hành tìm kiếm từ a[mid+1] đến a[right].

 Nếu a[mid] > x thì có nghĩa là đoạn **a[mid]** đến **a[right]** chỉ chứa khoá > x, ta tiến hành tìm kiếm từ a[left] đến a[mid-1].

- Quá trình tìm kiếm thất bại nếu left > right.

 Các bước tiến hành

- B1: left =0, right = n-1 // tìm kiếm trên tất cả phần tử

- B2: mid = (left + right)/2 // lấy mốc so sánh

So sánh a[mid] với x, có 3 khả năng

 a[mid] = x: **Tìm thấy** => **Trả về mid** => **Dừng**

 a[mid] > x: **right** = **mid -1**;

 a[mid] < x: **left** = **mid +1**;

- B3:

 Nếu left <= right // còn phần tử => tìm tiếp => Lặp B2

 Ngược lại: Trả về -1 => Dừng // đã xét hết các phần tử

Ví dụ: Cho dãy số gồm 8 phần tử bên dưới và x = 8:

1 2 4 5 6 8 12 15

**X = 8**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 5 | 6 | 8 | 12 | 15 |

Left = 0 mid = 3 right = 7

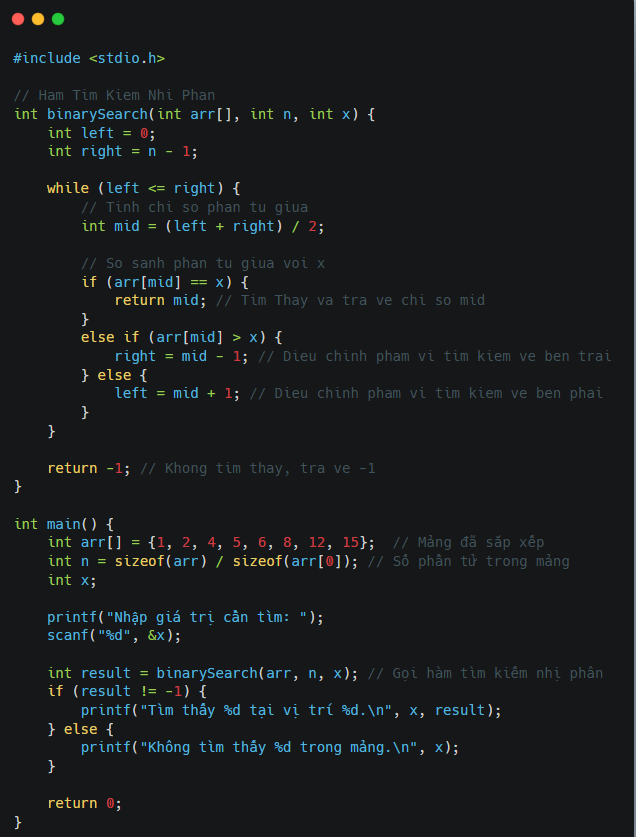
Đoạn Tìm Kiếm

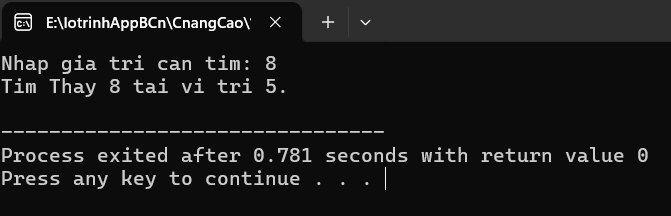
**X = 8**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 5 | 6 | 8 | 12 | 15 |

Left=4 Mid=5 Right=7

Đoạn Tìm Kiếm





 Nhận xét

- Giải thuật nhị phân dựa vào quan hệ giá trị của các phần tử trong mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được với dãy đã có thứ tự.

- Giải thuật nhị phân tìm kiếm nhanh hơn tìm kiếm tuyến tính.

- Tuy nhiên khi áp dụng giải thuật nhị phân thì cần phải quan tâm đến chi phí cho việc sắp xếp mảng. Vì khi mảng được sắp thứ tự rồi thì mới tìm kiếm nhị phân.

**\* Hàm qsort()**

Hàm qsort() nằm trong thư viện stdlib.h được cài đặt bằng thuật toán tối ưu và rất phổ biến là thuật toán sắp xếp nhanh - Quick sort.

Thông thường để thực hiện sắp mảng, dãy số thì bạn phải tự cài đặt các thuật toán này hoặc sử dụng hàm có sẵn.

Khuôn mẫu hàm qsort : qsort(array, elements, size, comparison\_function)

Trong đó :

* array : Mảng mà bạn muốn sắp xếp
* elements : Số lượng phần tử trong mảng mà bạn muốn sắp xếp
* size : Kích thước theo byte của 1 phần tử trong mảng
* comparison\_function : Hàm so sánh thể hiện logic mà bạn muốn sắp xếp

Hàm qsort() sẽ sắp xếp các phần tử trong mảng của bạn theo logic trong hàm so sánh, hay gọi là comparator hay comparison function.

Khuôn mẫu comparison function : int comparator(const void\* ptr1, const void\* ptr2);

Hàm này có kiểu trả về là số int và có 2 tham số là 2 con trỏ kiểu void. Khi sắp xếp bạn cần ép 2 con trỏ này sang con trỏ có kiểu dữ liệu phù hợp với mảng mà bạn muốn sắp xếp.

Hàm này có giá trị trả về dựa vào logic sau :

* < 0 : Nếu phần tử được trỏ bởi con trỏ ptr1 đứng trước phần tử được trỏ bởi con trỏ ptr2
* = 0 : Nếu phần tử được trỏ bởi con trỏ ptr1 bằng phần tử được trỏ bởi con trỏ ptr2
* > 0 : Nếu phần tử được trỏ bởi con trỏ ptr1 đứng sau phần tử được trỏ bởi con trỏ ptr2

Ví dụ 1 : Sắp xếp các phần tử trong mảng tăng, giảm dần

****

Mang sap tang dan : 1 2 3 4 5 6 7 8 9 10

Mang sap giam dan : 10 9 8 7 6 5 4 3 2 1

Giải thích :

Khi bạn sắp xếp tăng dần thì phần tử đứng trước phải nhỏ hơn phần tử đứng sau, khi đó \*x là giá trị trỏ bởi con trỏ x nếu nhỏ hơn \*y là giá trị trỏ bởi con trỏ y hàm so sánh sẽ trả về số âm, mình thường lấy là -1, các giá trị âm khác cũng được chấp nhận.

Ngược lại thì bạn trả về số dương, với việc sắp xếp tăng dần thì bạn chỉ cần return về \*x - \*y khi đó nếu \*x < \*y thì hàm sẽ trả về số âm để cho x đứng trước y, ngược lại nếu \*x > \*y thì hàm sẽ trả về số dương để x đứng sau y, khi \*x = \*y thì phần tử nào đứng trước cũng được vì nó bằng nhau.

 Khi sắp tăng dần mình thường return \*x - \*y còn khi sắp giảm dần thì return \*y - \*x