

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Bài 2: SEARCHING TECHNIQUES

Nội dung

2

1. Khái quát về tìm kiếm
2. Tìm tuyến tính (Linear Search)
3. Tìm nhị phân (Binary Search)

1. Khái quát về tìm kiếm

3

- Tìm kiếm là một yêu cầu rất thường xuyên trong đời sống hàng ngày cũng như trong tin học
- Tìm kiếm: Có trong hầu hết trong các hệ thống thông tin
- Ví dụ:
 - Tìm kiếm một sinh viên trong lớp
 - Tìm kiếm một tập tin, thư mục trong máy

1. Khái quát về tìm kiếm

4

- Ví dụ:
 - Cơ sở dữ liệu (Database): tìm 1 sinh viên, tìm 1 tài khoản ngân hàng, tài liệu, quyển sách,...
 - Internet: Google, Bing, Yahoo!, ...
- Để đơn giản ta xét bài toán tìm kiếm như sau:
 - Cho một dãy số gồm các phần tử a_1, a_2, \dots, a_n . Cho biết trong dãy này có phần tử nào có giá trị bằng X (cho trước) hay không?

1. Khái quát về tìm kiếm

5

- Xét hai cách tìm kiếm:
 - Tìm kiếm tuyến tính (**Linear Search**) hay còn gọi là tìm kiếm tuần tự (**Sequential Search**)
 - Tìm kiếm nhị phân (**Binary Search**)

Nội dung

6

1. Khái quát về tìm kiếm
2. Tìm tuyến tính (Linear Search)
3. Tìm nhị phân (Binary Search)

2. Tìm tuyến tính (Linear Search)

7

Ý tưởng:

- Bắt đầu từ phần tử đầu tiên của danh sách, so sánh lần lượt từng phần tử của danh sách với giá trị X cần tìm.
 - Nếu có phần tử bằng X, thuật toán dừng lại (tìm thấy)
 - Nếu đến cuối danh sách mà không có phần tử nào bằng X, thuật toán dừng lại (không tìm thấy)

2. Tìm tuyến tính (Linear Search)

8

Thuật toán:

B1: $i = 0$; // bắt đầu từ phần tử đầu tiên

B2: so sánh $A[i]$ với X , có 2 khả năng :

- ❑ $A[i] = X$: Tìm thấy. Dừng
- ❑ $A[i] \neq X$: Sang B3

B3: $i=i+1$ // Xét phần tử tiếp theo trong mảng

Nếu $i=n$: Hết mảng, không tìm thấy. Dừng

Ngược lại: lặp lại B2

2. Tìm tuyến tính (Linear Search)

10

5

Vị trí = 2

Khóa tìm

0	1	2	3	4	5	6	7
7	13	5	21	6	2	8	15



Tìm thành công

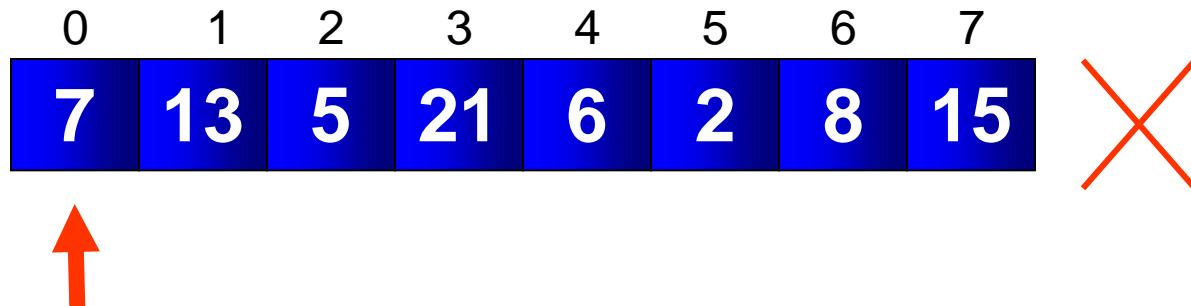
Số lần so sánh: 3

2. Tìm tuyến tính (Linear Search)

11

9

Khóa tìm



Không tìm thấy

Số lần so sánh: 8

2. Tìm tuyến tính (Linear Search)

12

```
int lsearch (int list[], int n, int key)
{
    int flag = -1; // giả sử lúc đầu chưa tìm thấy
    for(int i=0; i<n; i++)
        if (list[i] == key)
    {
        flag = i;    // tìm thấy
        break;
    }
    return flag;
}
```

2. Tìm tuyến tính (Linear Search)

13

```
int lsearch(int list[], int n, int key, int &ss)
{
    int find= -1;
    for(int i=0; i<n; i++) {
        ss = ss + 2;
        if (list[i] == key)
        {
            find = i;
            break;
        }
    }
    return find;
```

- Main
- Int ss=0;
- Vt = timTuyenTinh(a, n, x, ss);

2. Tìm tuyến tính (Linear Search)

15

Cài đặt thuật toán:

```
int LinearSearch (int A[], int n, int X)
{
    int i = 0;
    A[n]=x;
    while (A[i] != X)
        i++;
    if (i < n)
        return i; // trả về vị trí tìm thấy X
    return -1;
}
```

ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

16

- Phân tích, đánh giá thuật toán

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	n+1	Phần tử cuối cùng có giá trị x/không tìm thấy
Trung bình	(n+1)/2	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

- Vậy giải thuật tìm tuyến tính có độ phức tạp tính toán cấp n:
 $T(n) = O(n)$

❖ **Lưu ý:** Bài toán tìm max tìm min của một mảng, danh sách cũng áp dụng thuật toán tìm kiếm tuần tự:

- Số phần tử nhỏ ($<=10\ 000\ 000$): tốc độ chấp nhận được
- Kích thước lớn (hàng tỉ): thuật toán không hiệu quả

Ví dụ: *tìm tên của một người trên thế giới*

Cải tiến

18

Cải tiến thuật toán:

- Mỗi bước lặp với thuật toán trên cần thực hiện 2 phép so sánh \rightarrow ý tưởng giảm bớt phép so sánh bằng cách thêm vào mảng một phần tử cầm canh (sentinel/stand by) có giá trị bằng X để nhận diện ra sự hết mảng khi duyệt.

B1: $i = 1$

B2: $A[n] = X$

B3: Nếu $A[i] \neq X$

 Thì $i++$

 Ngược lại: Lặp lại B3

B4: Nếu $i < n$ Thì Tìm thấy phần tử có giá trị X ở vị trí i

B5: Ngược lại: Thì không tìm thấy phần tử có giá trị X

B6: Kết thúc

Tìm tuyến tính (Linear Search)

19

Cài đặt thuật toán cải tiến:

```
int LinearSearchCaitien (int A[], int n, int X)
{
    int i = 0;
    A[n] = X;
    while (A[i] != X)
        i++;
    if (i < n)
        return i;
    return -1;
}
```

Tìm tuyến tính (Linear Search)

20

Phân tích, đánh giá thuật toán cải tiến:

- Trường hợp tốt nhất (phần tử đầu tiên của mảng có giá trị = X)
 - ▣ Số phép gán $G_{\min} = 2$
 - ▣ Số phép so sánh $S_{\min} = 2$
- Trường hợp xấu nhất (không có phần tử nào của mảng có giá trị = X)
 - ▣ Số phép gán $G_{\max} = 2$
 - ▣ Số phép so sánh $S_{\max} = (N + 1) + 1 = N + 2$
- Trung bình
 - ▣ Số phép gán $G_{\text{avg}} = (2+2)/2=2$
 - ▣ Số phép so sánh $S_{\text{avg}} = ((N + 2)+2)/2=N/2+2$

Q & A

21



Nội dung

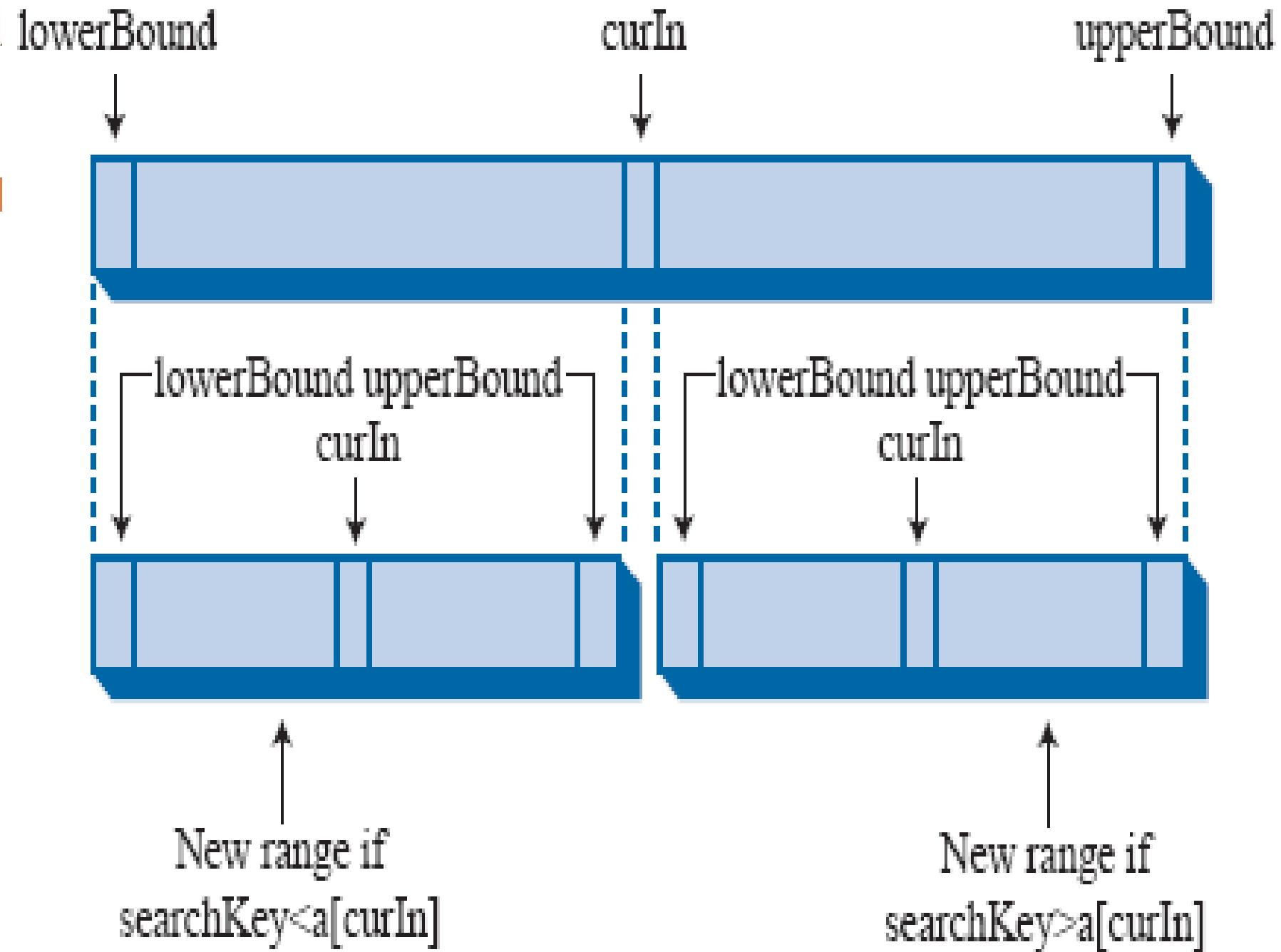
22

1. Khái quát về tìm kiếm
2. Tìm tuyến tính (Linear Search)
3. Tìm nhị phân (Binary Search)

3. Tìm nhị phân (Binary Search)

23

- Điều kiện:
 - Danh sách phải được sắp xếp trước
- Ý tưởng:
 - So sánh giá trị muốn tìm X với phần tử nằm ở vị trí giữa của danh sách:
 - Nếu bằng, tìm kiếm dừng lại (thành công)
 - Nếu X lớn hơn thì tiếp tục tìm kiếm ở phần danh sách bên phải phần tử giữa
 - Nếu X nhỏ hơn thì tiếp tục tìm kiếm ở phần danh sách bên trái phần tử giữa



3. Tìm nhị phân (Binary Search)

25

Thuật toán:

B1: Left = 0, Right = n-1

B2: Mid = (Left + Right)/2 // lấy vị trí cận giữa

B3: So sánh X với A[Mid], có 3 khả năng xảy ra:

- A[Mid] = X // tìm thấy. Dừng thuật toán

- A[Mid] > X

- Right=Mid-1; // Tiếp tục tìm trong dãy A[0]... A[Mid-1]

- A[Mid] < X

- Left = Mid+1 // Tiếp tục tìm trong dãy A[Mid+1]... A[Right]

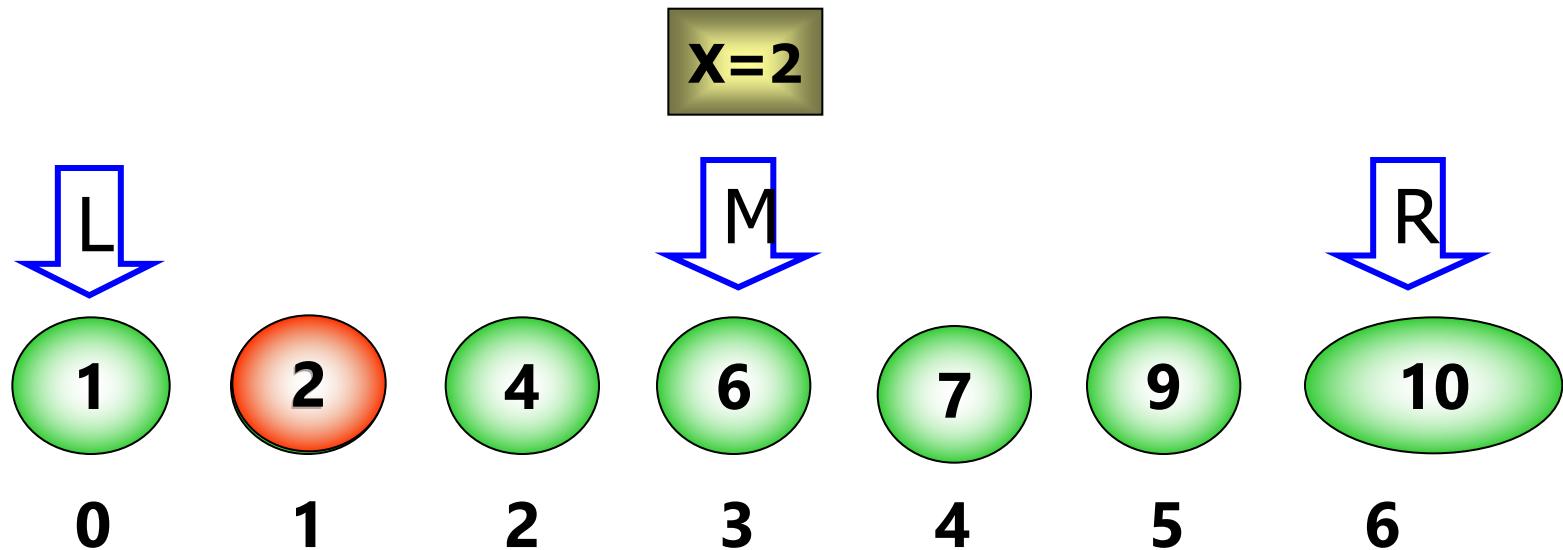
B4: Nếu (Left <= Right) // Còn phần tử chưa xét

Lặp lại B2

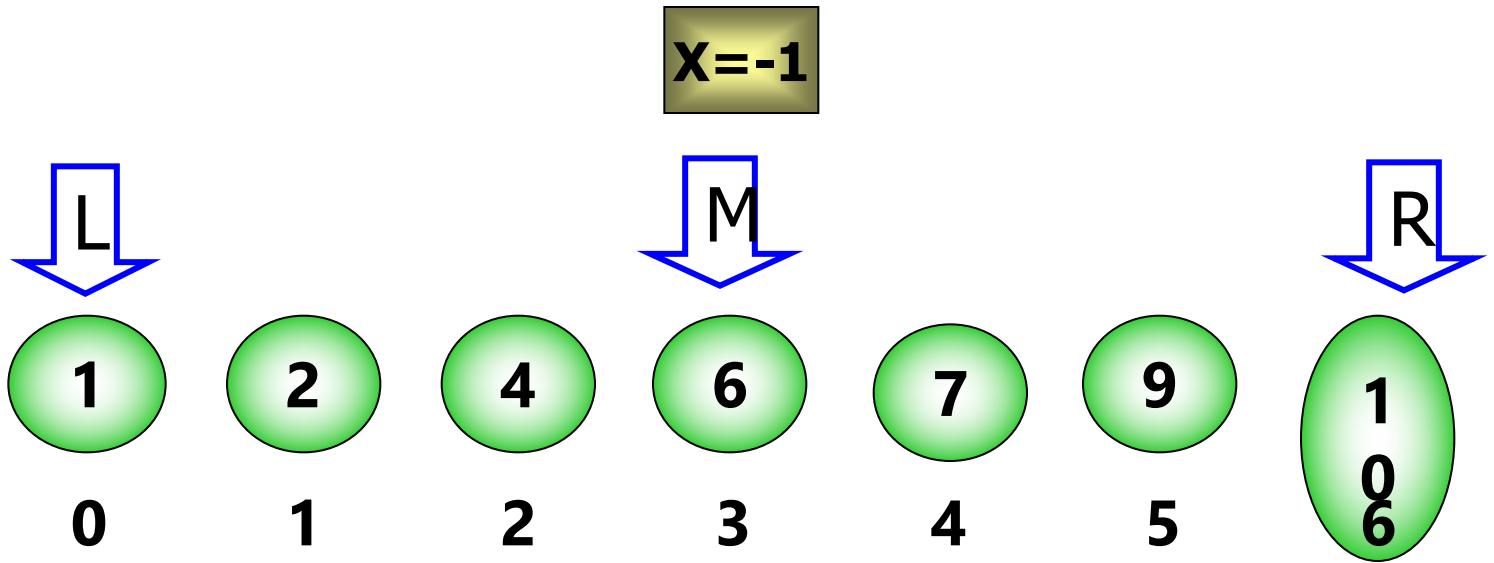
Ngược lại: Kết thúc

Minh Họa Thuật Toán Tìm Nhị Phân

Tìm thấy 2 tại vị trí 1



Minh Họa Thuật Toán Tìm Nhị Phân (tt)



$L=0$

$R=-1 \Rightarrow$ không tìm thấy $X=-1$

3. Tìm nhị phân (Binary Search)

28

Ví dụ:

0	1	2	3	4	5	6	7
1	2	4	5	6	8	12	15

X=8

Left=0, Right=7, Mid=3

X=8
↓

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

Left=4, Right=7, Mid=5

X=8
↓

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

Dừng

3. Tìm nhị phân (Binary Search)

29

Không đệ quy

```
int BSearch (int list[], int n, int key) {  
    int left=0, right=n-1, mid, flag = 0;  
    while (left <= right) {  
        mid = (left + right)/2;  
        if( list[mid] == key)  
            return mid;  
        else if (list[mid] < key)  
            left = mid +1;  
        else  
            right = mid -1;  
    }  
    return -1; //không tìm thâ  
}
```

3. Tìm nhị phân (Binary Search)

30

Đệ quy

```
int BSearch_Recursion (int list[], int key, int left, int right)
{
    if (left <= right)
    {
        int mid = (left + right)/2;
        if (key == list[mid])
            return mid;      // trả về vị trí tìm thấy key
        else if (key < list[mid])
            return BSearch_Recursion (list, key, left, mid-1);
        else return BSearch_Recursion (list, key, mid+1, right);
    }
    return -1; // không tìm thấy
}
```

3. Tìm nhị phân (Binary Search)

31

Phân tích, đánh giá thuật toán không đệ quy:

- Trường hợp tốt nhất (phần tử đầu tiên của mảng có giá trị = X)
 - Số phép gán $G_{\min} = 3$
 - Số phép so sánh $S_{\min} = 2$
- Trường hợp xấu nhất (không có phần tử nào của mảng có giá trị = X)
 - Số phép gán $G_{\max} = 2\log_2 N + 4$
 - Số phép so sánh $S_{\max} = 3\log_2 N + 1$
- Trung bình
 - Số phép gán $G_{\text{avg}} = \log_2 N + 3.5$
 - Số phép so sánh $S_{\text{avg}} = \frac{1}{2}(3\log_2 N + 3)$

3. Tìm nhị phân (Binary Search)

32

Phân tích, đánh giá thuật toán đệ quy:

- Trường hợp tốt nhất (phần tử đầu tiên của mảng có giá trị = X)
 - Số phép gán $G_{\min} = 1$
 - Số phép so sánh $S_{\min} = 2$
- Trường hợp xấu nhất (không có phần tử nào của mảng có giá trị = X)
 - Số phép gán $G_{\max} = \log_2 N + 1$
 - Số phép so sánh $S_{\max} = 3\log_2 N + 1$
- Trung bình
 - Số phép gán $G_{\text{avg}} = 1/2\log_2 N + 1$
 - Số phép so sánh $S_{\text{avg}} = 1/2(3\log_2 N + 3)$

3. Tìm nhị phân (Binary Search)

33

- Phân tích, đánh giá thuật toán:

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 (n/2)$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

- Vậy giải thuật tìm nhị phân có độ phức tạp tính toán cấp n: $T(n) = O(\log_2 n)$

Nhận xét

34

- Tìm Tuyến Tính không phụ thuộc vào thứ tự của các phần tử, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy bất kỳ
- Tìm Nhị Phân dựa vào quan hệ giá trị của các phần tử mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được cho những dãy đã có thứ tự
- Giải thuật Tìm Nhị Phân tiết kiệm thời gian hơn rất nhiều so với giải thuật Tìm Tuyến Tính do:

$$T_{\text{nhi phân}}(n) = O(\log_2 n) < T_{\text{tuyến tính}}(n) = O(n)$$

COMPLEXITY OF ALGORITHMS

• Linear Search	n	$\log_2 n$
– $O(n)$.	10	3
	100	6
• Binary Search	1,000	9
– $O(\log_2 N)$	10,000	13
	100,000	16

Nhận xét

36

- Tuy nhiên khi muốn áp dụng giải thuật tìm Nhị Phân cần phải xét đến thời gian sắp xếp dãy số để thỏa điều kiện dãy số có thứ tự
- Thời gian này không nhỏ, và khi dãy số biến động cần phải tiến hành sắp xếp lại
- Tất cả các nhu cầu đó tạo ra khuyết điểm chính cho giải thuật tìm Nhị Phân
- Ta cần cân nhắc nhu cầu thực tế để chọn một trong hai giải thuật tìm kiếm trên sao cho có lợi nhất

Q & A

37

