Chương 7

TÌM KIẾM

1. Nhu cầu tìm kiếm và sắp xếp dữ liệu trong một hệ thống thông tin

Trong hầu hết các hệ lưu trữ, quản lí dữ liệu, thao tác tìm kiếm thường hay được thực hiện để khai thác thông tin:

Ví dụ: tra cứu từ điển, tìm sách trong thư viện,...

Do các hệ thống thông tin thường phải lưu trữ một khối lượng dữ liệu đáng kể, nên việc xây dựng các giải thuật cho phép tìm kiếm nhanh sẽ có ý nghĩa rất lớn. Nếu dữ liệu trong hệ thống đã được tổ chức theo một trật tự nào đó, thì việc tìm kiếm sẽ tiến hành nhanh chóng và hiệu quả hơn.

Ví dụ: Các từ trong từ điển được sắp xếp theo từng vần, trong mỗi vần lại được sắp xếp theo trình tự alphabet; sách trong thư viện được xếp theo chủ đề,...

Vì thế, khi xây dựng một hệ quản lí thông tin trên máy tính, bên cạnh các thuật toán tìm kiếm, các thuật toán sắp xếp dữ liệu cũng là một trong những chủ đề được quan tâm hàng đầu.

Hiện nay đã có nhiều giải thuật tìm kiếm và sắp xếp xây dựng, mức độ hiệu quả của từng giải thuật còn phụ thuộc vào tính chất của cấu trúc dữ liệu cụ thể mà nó tác động đến. Dữ liệu được lưu trữ chủ yếu trong bộ nhớ chính và trên bộ nhớ phụ, do đặc điểm khác nhau của thiết bị lưu trữ, các thuật toán tìm kiếm và sắp xếp được xây dựng cho các cấu trúc lưu trữ trên bộ nhớ chính hoặc phụ cũng có những đặc thù khác nhau. Chương 7 và 8 sẽ trình bày các thuật toán sắp xếp và tìm kiếm dữ liệu được lưu trữ trên bộ nhớ chính - gọi là các giải thuật *tìm kiếm và sắp xếp nội*.

2. Các giải thuật tìm kiếm nội

Có 2 giải thuật thường được áp dụng để tìm kiếm dữ liệu là *tìm tuyến tính* và *tìm nhị phân*. Để đơn giản trong việc trình bày giải thuật, bài toán được đặc tả như sau:

Tập dữ liệu được lưu trữ là dãy số $a_1, a_2,..., a_N$.

Giả sử chọn cấu trúc dữ liệu mảng để lưu trữ dãy số này trong bộ nhớ chính và có khai báo:

```
int a[N];
```

Lưu ý: Các bản cài đặt trong giáo trình sử dụng ngôn ngữ C, do đó chỉ số của mảng mặc định bắt đầu từ 0, nên các giá trị của các chỉ số có chênh lệch so với thuật toán, nhưng ý nghĩa không đổi.

Khoá cần tìm là x, được khai báo như sau:

```
int x;
```

2.1. Tìm kiếm tuyến tính

Giải thuật:

Tìm tuyến tính là một kĩ thuật tìm kiếm rất đơn giản và cổ điển. Thuật toán tiến hành so sánh x lần lượt với phần tử thứ nhất, thứ hai,... của mảng a cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy x. Các bước tiến hành như sau:

Bước 1:

i = 1; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2:

So sánh a[i] với x, có 2 khả năng:

a[i] = x: Tìm thấy. Dừng

a[i]!= x: Sang Bước 3.

Bước 3:

i = i+1; // xét tiếp phần tử kế trong mảng

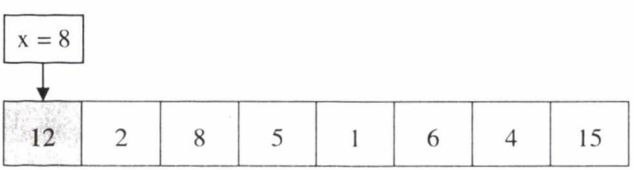
Nếu i >N: Hết mảng, không tìm thấy. Dừng

Ngược lại: Lặp lại Bước 2.

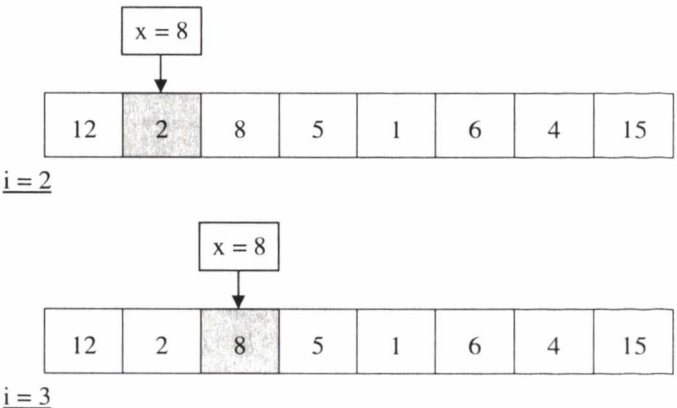
Ví dụ: Cho dãy số a

12 2 8 5 1 6 4 15

Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau:



i = 1



Dừng.

Cài đặt:

Từ mô tả trên đây của thuật toán tìm tuyến tính, có thể cài đặt hàm LinearSearch để xác định vị trí của phần tử có khoá x trong mảng a:

```
int LinearSearch(int a[], int N, int x)
{   int i=0;
 while ((i<N) && (a[i]!=x)) i++;
 if(i==N)   return -1;  // tìm hết mảng nhưng không có x
     else   return i;   // a[i] là phần tử có khoá x
}
```

Trong cài đặt trên đây, nhận thấy mỗi lần lặp của vòng lặp while phải tiến hành kiểm tra 2 điều kiện (i<N) - điều kiện biên của mảng - và (a[i]!=x) - điều kiện kiểm tra chính. Nhưng thật sự chỉ cần kiểm tra điều kiện chính (a[i]!=x), để cải tiến cài đặt, có thể dùng phương pháp *"lính canh"* - đặt thêm một phần tử có giá trị x vào cuối mảng, như vậy bảo đảm luôn tìm thấy x trong mảng, sau đó dựa vào vị trí tìm thấy để kết luận. Cài đặt cải tiến sau đây của hàm LinearSearch giúp giảm bớt một phép so sánh trong vòng lặp:

```
int LinearSearch(int a[],int N,int x)
{   int i=0;          // mảng gồm N phần tử từ a[0]..a[N-1]
 a[N] = x;  // thêm phần tử thứ N+1
 while (a[i]!=x) i++;
```

```
    if (i==N)
        return -1;  // tìm hết mảng nhưng không có x
    else
        return i;   // tìm thấy x tại vị trí i
}
```

Đánh giá giải thuật:

Có thể ước lượng độ phức tạp của giải thuật tìm kiếm qua số lượng các phép so sánh được tiến hành để tìm ra x. Trường hợp giải thuật tìm tuyến tính, có:

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
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)

Nhận xét:

- Giải thuật tìm tuyến tính không phụ thuộc vào thứ tự của các phần tử mảng, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy số bất kì.
- Một thuật toán có thể được cài đặt theo nhiều cách khác nhau, kĩ thuật cài đặt ảnh hưởng đến tốc độ thực hiện của thuật toán.

2.2. Tìm kiếm nhị phân

Giải thuật:

Đối với những dãy số đã có thứ tự (giả sử thứ tự tăng), các phần tử trong dãy có quan hệ $a_{i-1} < a_i < a_{i+1}$, từ đó kết luận được nếu $x > a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_{i+1},a_N]$ của dãy, ngược lại nếu $x < a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_1,a_{i-1}]$ của dãy. Giải thuật tìm nhị phân áp dụng nhận xét trên đây để tìm cách giới hạn phạm vi tìm kiếm sau mỗi lần so sánh x với một phần tử trong dãy. Ý tưởng của giải thuật là tại mỗi bước tiến hành so sánh x với phần tử nằm ở vị trí giữa của dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này để quyết định giới hạn dãy tìm kiếm ở bước kế tiếp là nửa trên hay nửa dưới của dãy tìm kiếm hiện hành. Giả sử dãy tìm kiếm hiện hành bao gồm các phần tử $a_{left},..., a_{right}$, các bước tiến hành như sau:

Bước 1: left = 1; right = N; // tìm kiếm trên tất cả các phần tử

Bước 2:

mid = (left+right)/2; // lấy mốc so sánh

So sánh a[mid] với x, có 3 khả năng:

a[mid] = x: Tìm thấy. Dừng

a[mid] > x: //tìm tiếp x trong dãy con $a_{left}, \ldots, a_{mid-1}$:

right =midle - 1;

a[mid] < x: //tìm tiếp x trong dãy con $a_{mid+1}, \ldots, a_{right}$:

left = mid+ 1;

Bước 3:

Nếu left < right //còn phần tử chưa xét, tìm tiếp.

Lặp lại Bước 2.

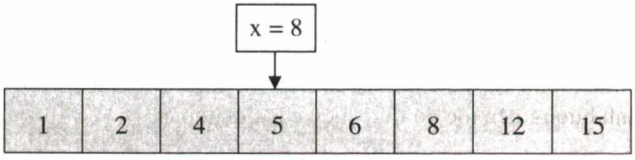
Ngược lại: Dừng; //đã xét hết tất cả các phần tử.

Ví dụ: Cho dãy số a gồm 8 phần tử:

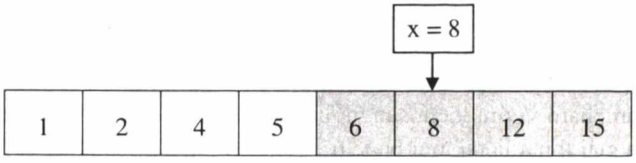
1 2 4 5 6 8 12 15

Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau:

left = 1, right = 8, midle = 4



left = 5, right = 8, midle = 6



Dừng.

Cài đặt:

Thuật toán tìm nhị phân có thể được cài đặt thành hàm BinarySearch:

```
int BinarySearch(int a[],int N,int x)
{   int left =0; right = N-1;
int  midle;
 do {
     mid = (left + right)/2;
     if (x = a[midle])   return midle;//Thấy x tại mid
     else
         if (x < a[midle])   right = midle -1;
         else left = midle +1;
 }while (left <= right);
 return -1; // Tìm hết dãy mà không có x
}
```

Đánh giá giải thuật:

Trường hợp giải thuật tìm nhị phân, có bảng phân tích sau:

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:

- Giải thuật 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 $T_{\text{nhị phân}}(n) = O(\log_2 n) < T_{\text{tuyến tính}}(n) = O(n)$. 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.

BÀI TẬP

1. Cho dãy khóa

10, 20, 30, 40, 50, 60, 70, 80, 90

Hãy minh họa các bước tìm kiếm vị trí của khóa có giá trị 90 bằng

a) Thuật toán tìm kiếm tuyến tính.

b) Thuật toán tìm kiếm nhị phân.

2. Hãy dựng cây nhị phân tìm kiếm ứng với dãy khóa cho như sau: 13, 35, 67,12, 69, 80, 47, 50.

a) Đánh dấu đường đi trên cây này khi thực hiện tìm kiếm khóa 47.

b) Minh họa các bước tìm kiếm khóa 45 và bổ sung vào cây nhị phân tìm kiếm.

3. Hãy dựng cây nhị phân tìm kiếm ứng với dãy khóa cho như sau: 54, 16, 58, 90, 100, 30, 58, 10, 50, 90, 98.

Chương 8

SẮP XẾP

1. Định nghĩa bài toán sắp xếp

Sắp xếp là quá trình xử lí một danh sách các phần tử (hoặc các mẫu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu giữ tại mỗi phần tử.

Tại sao cần phải sắp xếp các phần tử thay vì để nó ở dạng tự nhiên (chưa có thứ tự) vốn có? Ví dụ của bài toán tìm kiếm với phương pháp tìm kiếm nhị phân và tuần tự đủ để trả lời câu hỏi này.

Khi khảo sát bài toán sắp xếp, ta sẽ phải làm việc nhiều với một khái niệm gọi là *nghịch thế*.

Định nghĩa: Xét một mảng các số $a_0, a_1,\ldots,a_n$. Nếu có $i < j$ và $a_i > a_j$, thì ta gọi đó là một nghịch thế.

Mảng chưa sắp xếp sẽ có nghịch thế.

Mảng đã có thứ tự sẽ không chứa nghịch thế. Khi đó a_0 sẽ là phần tử nhỏ nhất rồi đến a_1, a_2,...

Như vậy, để sắp xếp một mảng, ta có thể tìm cách giảm số các nghịch thế trong mảng này bằng cách hoán vị các cặp phần tử a_i, a_j nếu có $i < j$ và $a_i > a_j$ theo một quy luật nào đó.

Cho trước một dãy số $a_0, a_1, a_2,..., a_N$ được lưu trữ trong cấu trúc dữ liệu mảng

Int a[N];

Sắp xếp dãy số $a_0, a_1, a_2,...,a_N$ là thực hiện việc bố trí lại các phần tử sao cho hình thành được dãy mới $a_{k1}, a_{k2},...,a_{kN}$ có thứ tự (giả sử xét thứ tự tăng) nghĩa là $a_{ki} > a_{ki-1}$. Để quyết định được những tình huống cần thay đổi vị trí các phần tử trong dãy, cần dựa vào kết quả của một loạt phép so sánh. Chính vì vậy, hai thao tác so sánh và gán là các thao tác cơ bản của hầu hết các thuật toán sắp xếp.

Khi xây dựng một thuật toán sắp xếp cần chú ý tìm cách giảm thiểu những phép so sánh và đổi chỗ không cần thiết để tăng hiệu quả của thuật toán. Đối với các dãy số được lưu trữ trong bộ nhớ chính, nhu cầu tiết kiệm bộ nhớ được đặt nặng, do vậy những thuật toán sắp xếp đòi hỏi cấp phát thêm vùng nhớ để lưu trữ dãy kết quả ngoài vùng nhớ lưu trữ dãy số ban đầu thường ít được quan tâm. Thay vào đó, các thuật toán sắp xếp trực tiếp trên dãy số ban đầu – gọi là các thuật toán sắp xếp tại chỗ (sắp xếp nội) - lại được đầu tư phát triển. Phần này giới thiệu một

số giải thuật sắp xếp từ đơn giản đến phức tạp có thể áp dụng thích hợp cho việc sắp xếp nội.

Sau đây là một số phương pháp sắp xếp thông dụng sẽ được đề cập đến trong giáo trình này:

- Chon trực tiếp - Selection Sort
- Chèn trực tiếp - Insertion Sort
- Nổi bọt - Bubble Sort
- Heapsort
- Quicksort
- Merge Sort

Trong đó, chúng ta sẽ lần lượt khảo sát các thuật toán trên. Các thuật toán như Insertion sort, Selection sort, Bubble sort là những thuật toán đơn giản dễ cài đặt nhưng chi phí cao. Các thuật toán Heapsort, Quicksort, Merge sort phức tạp hơn nhưng hiệu suất cao hơn nhóm các thuật toán đầu. Cả hai nhóm thuật toán trên có một điểm chung là đều được xây dựng dựa trên cơ sở việc so sánh giá trị của các phần tử trong mảng (hay so sánh các khóa tìm kiếm).

2. Các phương pháp sắp xếp $O(n^2)$

2.1. Phương pháp chọn trực tiếp (Selection Sort)

Giải thuật:

Ta thấy rằng, nếu mảng có thứ tự tăng dần, phần tử a_i luôn là $\min(a_i, a_{i+1},.., a_N)$. Ý tưởng của thuật toán chọn trực tiếp mô phỏng một trong những cách sắp xếp tự nhiên nhất trong thực tế: Chọn phần tử nhỏ nhất trong N phần tử ban đầu, đưa phần tử này về vị trí là đầu dãy hiện hành; sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn N-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2; lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử. Dãy ban đầu có N phần tử, vậy tóm tắt ý tưởng thuật toán là thực hiện N-1 lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí ở đầu dãy. Các bước tiến hành như sau:

- Bước 1: i = 1;
- Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]
- Bước 3: Hoán vị a[min] và a[i]
- Bước 4: Nếu $i \leq N - 1$ thì i = i + 1; Lặp lại Bước 2

Ngược lại: Dừng. //N-1 phần tử đã nằm đúng vị trí.

Ví dụ:

Cho dãy số a: 12 2 8 5 1 6 4 15