Chương 10

TÌM KIẾM

10.1 Bài toán tìm kiếm

Tìm kiếm (Searching) là một đòi hỏi rất thường xuyên trong đời sống hàng ngày cũng như trong xử lý tin học. Ngay trong chương trước ta cũng thấy xuất hiện các yêu cầu về tìm kiếm. Tuy nhiên, lúc đó vấn đề này đã được xét và giải quyết trong mối quan hệ mật thiết với phép xử lý chính, đó là phép sắp xếp. Còn bây giờ, trong chương này, bài toán tìm kiếm sẽ được đặt ra một cách độc lập và tổng quát, không liên quan đến mục đích xử lý cụ thể nào khác. Ta có thể phát biểu như sau:

"Cho một bảng gồm n bản ghi R_1, R_2, ..., R_n. Mỗi bản ghi R_i ($1 \le i \le n$) tương ứng với một khoá k_i. Hãy tìm bản ghi có giá trị khoá tương ứng bằng X cho trước".

X được gọi là *khoá tìm kiếm hay đối trị tìm kiếm* (argument).

Công việc tìm kiếm sẽ hoàn thành khi có một trong hai tình huống sau đây xảy ra:

1) Tìm được bản ghi có giá trị khoá tương ứng bằng X, lúc đó ta nói: phép *tìm kiếm được thoả* (successfull).

2) Không tìm thấy được bản ghi nào có giá trị khoá bằng X cả: Phép *tìm kiếm không thoả* (unsuccessfull). Sau một phép tìm kiếm không thoả có khi xuất hiện yêu cầu bổ sung thêm bản ghi mới có khoá bằng X vào bảng. Giải thuật thể hiện cả yêu cầu này được gọi là giải thuật "tìm kiếm có bổ sung".

Tương tự như sắp xếp, khoá của mỗi bản ghi chính là đặc điểm nhận biết của bản ghi đó trong tìm kiếm, ta sẽ coi nó như đại diện cho bản ghi ấy và trong các giải thuật, trong ví dụ ta cũng chỉ nói tới khoá. Để cho tiện, ta cũng coi các khoá $k_i (1 \le i \le n)$ là các số khác nhau. Ở đây ta cũng chỉ xét tới các phương pháp tìm kiếm cơ bản và phổ dụng, đối với dữ liệu ở bộ nhớ trong nghĩa là *tìm kiếm trong*, còn *tìm kiếm ngoài* sẽ được xét ở chương sau.

10.2 Tìm kiếm tuần tự (sequential searching)

10.2.1 Tìm kiếm tuần tự là kỹ thuật tìm kiếm rất đơn giản và cổ điển

Nội dung có thể tóm tắt như sau:

"Bắt đầu từ bản ghi thứ nhất, lần lượt so sánh khoá tìm kiếm với khoá tương ứng của các bản ghi trong bảng, cho tới khi tìm được bản ghi mong muốn hoặc đã hết bảng mà chưa thấy".

Sau đây là giải thuật:

Function: SEQUEN-SEARCH (k, n, X)

{Cho dãy khoá k gồm n phần tử. Thủ tục này sẽ được tìm kiếm trong dãy xem có khoá nào bằng X không. Nếu có nó sẽ đưa ra chỉ số của khoá ấy, nếu không nó sẽ đưa ra giá trị 0. Trong thủ tục này có sử dụng một khoá phụ k_{n+1} mà giá trị của nó chính là X}

1) {Khởi đầu}

i := 1; k[n + 1] := X;

2) {Tìm khoá trong dãy}

while k[i] ≠ X **do** i := i+1;

3) {Tìm thấy hay không?}

if i = n + 1 **then return** (o)

else return (i)

10.2.2. Ở đây, để đánh giá hiệu lực của phép tìm kiếm ta cũng dựa vào số lượng các phép so sánh. Ta thấy với giải thuật trên thuận lợi thì chỉ cần 1 phép so sánh: $C_{min} = 1$; còn xấu nhất thì $C_{max} = n + 1$. Nếu giả sử hiện tượng khoá tìm kiếm trùng với một khoá nào đó của bảng là đồng khả năng thì $C_{tb} = \frac{n+1}{2}$. Tóm lại cả trường hợp xấu nhất cũng như trung bình, cấp độ lớn của thời gian thực hiện giải thuật trên là O(n).

Nhưng nếu xác suất để xuất hiện $k_i = X$ mà $p_i \neq \frac{1}{n}$ thì sao? Lúc đó ta sẽ có $C_{tb} = 1 * p_1 + 2 * p_2 + ...+ n * p_n$

với: $\sum_{i=1}^{n} p_i = 1$

Rõ ràng là nếu $p_1 \geq p_2 \geq ... \geq p_n$ thì thời gian trung bình sẽ nhỏ hơn. Nhưng muốn như vậy thì phải sắp xếp trước!

10.3 Tìm kiếm nhị phân (Binary searching)

10.3.1. Tìm kiếm nhị phân là một phương pháp tìm kiếm khá thông dụng. Nó tương tự như cách thức ta đã làm khi tra tìm số điện thoại của một cơ quan, trong bảng danh mục điện thoại hay khi tìm một từ trong từ điển. Chỉ có một điều hơi khác là trong các công việc trên để so sánh với khoá tìm kiếm ta chọn hú hoạ một phần tử, còn với phép tìm kiếm nhị phân thì nó luôn luôn chọn khoá "ở giữa" dãy khoá đang xét để thực hiện so sánh với khoá tìm kiếm. Giả sử dãy khoá đang xét là k_l, ..., k_r thì khoá ở giữa dãy sẽ là k_i với $i = \left\lfloor \frac{l+r}{2} \right\rfloor$. Tìm kiếm sẽ kết thúc nếu $X = k_i$. Nếu $X < k_i$ tìm kiếm sẽ được thực hiện tiếp với k_l,..., k_{i-1}; còn nếu $X > k_i$ tìm kiếm lại được làm với k_{i+1},..., k_r. Với dãy khoá sau, một kỹ thuật tương tự lại được sử dụng. Quá trình tìm kiếm được tiếp tục khi tìm thấy khoá mong muốn hoặc dãy khoá xét đó trở nên rỗng (không thấy).

Giải thuật sau đây thể hiện phép tìm kiếm này.

Function BINARY_SEARCH (k, n, X)

{Cho dãy k gồm n khoá đã được sắp xếp theo thứ tự tăng dần. Giải thuật này tìm xem trong dãy có khoá nào bằng giá trị X hay không. Ở đây dùng các biến *l*, r, m để ghi nhận chỉ số của phần tử đầu, phần tử cuối và phần tử giữa của dãy khoá k. Nếu phép tìm kiếm được thoả, giá trị cho ra là chỉ số của khoá đã tìm thấy, nếu không thì cho ra giá trị 0}.

1. {Khởi đầu}
   l := 1;
   r := n;
2. {Tìm}
   while *l* ≤ r **do begin**
3. {Tính chỉ số giữa}
   m := ⌊(*l*+r)/2⌋;
4. {So sánh}
   if X < k[m] **then** r := m-1
   else
   if X > k[m] **then** *l* := m+1
   else return (m)
   end
5. {Tìm kiếm không thoả mãn}
   return(0)

*Ví dụ: Với dãy khoá

11 23 36 42 58 65 74 87 94 99

a) Nếu X = 23: phép tìm kiếm được thoả mãn và các bước sẽ như sau:

[11 23 36 42 **58** 65 74 87 94 99]

[11 **23** 36 42]

b) Nếu X = 71: phép tìm kiếm không thoả

[11 23 36 42 **58** 65 74 87 94 99]

[65 74 **87** 94 99]

[**65** 74]

[**74**]

(Ở đấy dấu [ứng với *l*, dấu] ứng với r, dấu – ứng với m)

Giải thuật tìm kiếm nhị phân có thể viết dưới dạng đệ quy như sau:

Function RBINARY_SEARCH (*l*, r, k, X)

{*l*, r là chỉ số dưới và chỉ số trên của dãy k, m vẫn là chỉ số giữa. Ở đây dùng thêm biến nguyên LOC để đưa ra chỉ số ứng với khoá cần tìm, nếu tìm kiếm không thoả LOC có giá trị 0}

```
1- if l> r then LOC := 0
          else m := ⌊(l+ r)/2⌋;
   if x < k[m]
          then LOC := RBINARY_SEARCH (l, m-1, k, X)
          else if X> k[m]
                 then
                  LOC := RBINARY_SEARCH (m+1,r,k, X)
                 else LOC := m;
2- return (LOC)
```

10.3.2 Phân tích đánh giá

Ta thấy số lượng phép toán so sánh phụ thuộc vào X. Với giải thuật đệ quy nêu trên, trường hợp thuận lợi nhất đối với dãy khoá $k_1,..., k_n$ (mà lời gọi sẽ là: POS := RBINARY-SEARCH (1, n, k, X)) là X = k[⌊(n+1)/2⌋] nghĩa là chỉ cần một phép so sánh, lúc đó $T_t(n) = O(1)$. Trường hợp xấu nhất xét có hơi phức tạp hơn. Giả sử ta gọi w(r-*l*+1) là hàm biểu thị số lượng phép so sánh trong trường hợp xấu nhất ứng với một phép gọi RBINARY_SEARCH (*l*, r, k, X) và đặt n = r - *l* + 1 (ứng với dãy khoá mà

$l = 1$, r = n) thì trong trường hợp xấu nhất ta phải gọi đệ quy cho tới khi dãy khoá xét chỉ còn là 1 phần tử và vì vậy ta có:

$$w(n) = 1 + w(\lfloor n / 2 \rfloor)$$

Với phương pháp truy hồi, ta có thể viết

$$w(n) = 1 + 1 + w(\lfloor n/2^2 \rfloor)$$
$$= 1 + 1 + 1 + w(\lfloor n/2^3 \rfloor)$$

Như vậy w(n) có thể viết dưới dạng $w(n) = k + w(\lfloor n/2^k \rfloor)$

Khi $\lfloor n/2^k \rfloor = 1$ ta có $w(\lfloor n/2^k \rfloor) = w(1)$ mà $w(1) = 1$ và khi đó tìm kiếm phải kết thúc. Song $\lfloor n/2^k \rfloor = 1$ thì suy ra $2^k \le n \le 2^{k+1}$, do đó $k \le \log_2 n < k+1$, nghĩa là có thể viết $k = \lfloor \log_2 n \rfloor$. Vì vậy cuối cùng ta có:

$$w(n) = \lfloor \log_2 n \rfloor + 1$$

hay

$$T_x(n) = O(\log_2 n)$$

Người ta cũng chứng minh được

$$T_{tb}(n) = O(\log_2 n)$$

Rõ ràng là so với tìm kiếm tuần tự, chi phí tìm kiếm nhị phân ít hơn khá nhiều. Sau này ta sẽ thấy rằng không có một phương pháp tìm kiếm nào dựa trên so sánh giá trị khoá lại có thể đạt được kết quả tốt hơn.

Tuy nhiên ta cũng không nên quên rằng trước khi sử dụng tìm kiếm nhị phân dãy khoá đã phải được sắp xếp rồi, nghĩa là thời gian chi phí cho sắp xếp cũng phải kể đến. Nếu dãy khoá luôn biến động (được bổ sung thêm hoặc loại bớt đi) thì lúc đó chi phí cho sắp xếp lại nổi lên rất rõ và chính điều ấy đã bộc lộ nhược điểm của phương pháp tìm kiếm này.

10.4 Cây nhị phân tìm kiếm (binary search tree)

Để khắc phục nhược điểm vừa nêu trên đối với tìm kiếm nhị phân và đáp ứng yêu cầu tìm kiếm đối với bảng biến động, một phương pháp mới đã được hình thành dựa trên cơ sở bảng được tổ chức dưới dạng cây nhị phân (mỗi bản ghi ứng với một nút trên cây đó) mà ta gọi *cây nhị phân tìm kiếm*.

10.4.1 Định nghĩa cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm ứng với n khoá $k_1, ..., k_n$ là một cây nhị phân mà mỗi nút của nó đều được gán một giá trị khoá nào đó trong các giá trị khoá đã cho và đối với mọi nút trên cây tính chất sau đây luôn được thoả mãn: