ĐỀ CƯƠNG BÀI GIẢNG

BÀI 3: CÁC PHƯƠNG PHÁP TÌM KIẾM VÀ SẮP XẾP

3.1. Các phương pháp tìm kiếm

3.1.1. Khái niệm tìm kiếm

Tìm kiếm là một đòi hỏi thường xuyên trong việc xử lý các bài toán tin học, nhất là đối với các bài toán quản lý. Tuy nhiên, ở đây ta chỉ xét các giải thuật tìm kiếm một cách tổng quát, không liên quan đến mục đích xử lý cụ thể nào.

Ta mô tả bài toán tìm kiếm như sau:

Cho một bảng gồm n bản ghi R_1, R_2, …, R_n. Mỗi bản ghi R_i có một khoá tìm kiếm là R_i.key ($1 \le i \le n$). Hãy tìm bản ghi có giá trị khoá tương ứng là KH cho trước.

KH được gọi là khoá tìm kiếm.

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 KH, lúc đó việc tìm kiếm là thành công.

2. Không tìm được bản ghi nào có khoá tương ứng bằng KH, lúc đó việc tìm kiếm là không thành công.

Cũng giống như việc sắp xếp, khoá của mỗi bản ghi chính là đặc điểm nhận biết của bản ghi đó trong tìm kiếm. Để đơn giản, trong các giải thuật ta coi nó là đại diện cho bản ghi đó và trong các ví dụ ta cũng chỉ nói tới khoá. Ta coi các khoá R_i.key ($1 \le i \le n$) là các số khác nhau. Trong chương này ta cũng chỉ xét các phương pháp tìm kiếm cơ bản, phổ dụng, đối với dữ liệu được lưu trữ ở bộ nhớ trong và được gọi là tìm kiếm trong. Có hai phương pháp tìm kiếm trong là tìm kiếm tuần tự và nhị phân tìm kiếm mà ta sẽ xét trong các phần sau đây.

3.1.2. Tìm kiếm tuần tự

3.1.2.1. Nguyên tắc tìm kiếm

Tìm kiếm tuần tự là phương pháp tìm kiếm rất đơn giản. Nguyên tắc tìm kiếm theo phương pháp này có thể tóm tắt như sau:

Bắt đầu từ bản ghi thứ nhất, lần lượt so sánh khoá tìm kiếm KH với các khoá tương ứng của các bản ghi trong bảng, cho đến khi tìm được một bản ghi mong muốn trong bảng (trường hợp tìm kiếm thành công, trả về vị trí của bản ghi tìm được) hoặc đã hết bảng mà không thấy (trường hợp tìm kiếm không thành công).

Ví dụ minh hoạ

Giả sử các khoá tương ứng của các bản ghi trong bảng có 6 bản ghi là dãy số:

38 12 -56 95 23 11 và khoá cần tìm là KH = 95, khi đó việc tìm kiếm được thực hiện như mô tả dưới đây.

Với i = 1 < n R_1.key = 38 ≠ KH, chuyển sang so sánh bản ghi tiếp theo.

Với i = 2 < n R_2.key = 12 ≠ KH, chuyển sang so sánh bản ghi tiếp theo.

Với i = 3 < n R_3.key = -56 ≠ KH, chuyển sang so sánh bản ghi tiếp theo.

Với i = 4 < n R_4.key = 95 = KH, kết thúc tìm kiếm và trả về vị trí tìm được là i = 4

Ví dụ trên đây minh hoạ cho một phép tìm kiếm thành công, bạn đọc tự lấy ví dụ minh hoạ cho phép tìm kiếm không thành công.

3.1.2.2. Giải thuật

Thủ tục dưới đây thể hiện giải thuật tìm kiếm tuần tự, tìm kiếm khoá KH trên một dãy khoá X, có n phần tử. Nếu có nó sẽ đưa ra chỉ số của khoá ấy, nếu không có nó đưa ra giá trị 0.

```
int Sequence_Search(X[], n, KH)
{
    //1. Khởi tạo
        i = 1;
    //2. Tìm khoá trong dãy
        while (X[i] != KH  && i <= n)  i = i + 1;
    //3. Kiểm tra và trả về kết quả tìm kiếm
        if (i <= n)  return (i);
        else return (0);
}
```

Để "tiết kiệm" thời gian ta có thể cải tiến giải thuật trên bằng cách thêm vào một khoá phụ X_{n+1}, có giá trị bằng KH như sau:

```
int Improvement_Sequence_Search(X[], n, KH)
{
    // 1. Khởi tạo
        i = 1; X[n+1] = KH;
    // 2. {tìm khoá trong dãy
        while (X[i] != KH)
            i = i + 1;
    // 3. Kiểm tra và trả về kết quả tìm kiếm
        if  (i == n + 1)  return (0);
        else return (i);
}
```

Với giải thuật cải tiến, ta giảm được một nửa số phép toán kiểm tra trong vòng lặp *while*, vì thế tiết kiệm được khá nhiều thời gian khi số lượng bản ghi trong dãy X thật sự lớn.

3.1.2.3. Đánh giá hiệu lực của giải thuật

Để đánh giá hiệu lực của phép tìm kiếm ta có thể dựa vào số lượng các phép so sánh. Ta thấy với giải thuật trên (cải tiến), trong trường hợp tốt nhất, chỉ cần 1 phép so sánh, $C_{min} = 1$, còn trong trường hợp xấu nhất số phép toán so sánh là $C_{max} = n + 1$. Nếu hiện tượng khoá tìm kiếm trùng với một khoá nào đó của bảng là đồng khả năng thì $C_{tb} = (n + 1) / 2$. Tóm lại, cả hai trường hợp xấu nhất cũng như trung bình, thời gian thực hiện tìm kiếm theo giải thuật trên là O(n).

Trong trường hợp dãy khoá đã được sắp xếp theo chiều tăng dần (hoặc giảm dần), thời gian trung bình thực hiện giải thuật sẽ nhỏ hơn. Tuy nhiên, trong trường hợp này ta có thể áp dụng phương pháp tìm kiếm khác, với thời gian nhanh hơn nhiều, đó là phương pháp nhị phân tìm kiếm.

3.1.3. Tìm kiếm nhị phân

3.1.3.1. Nguyên tắc

Nhị phân tìm kiếm là phương pháp tìm kiếm khá thông dụng. Trong phương pháp này ta áp dụng chiến lược "chia để trị" để giải quyết bài toán tìm kiếm. Việc này cũng giống như việc ta tìm một từ trong quyển từ điển. Ta có thể hiểu nôm na thế này: để tìm từ, ta mở từ điển vào trang "giữa", tìm từ trong trang này, nếu có, việc tìm kiếm là thành công, còn nếu không có ta tìm ở "nửa trước" hoặc nửa sau của từ điển. Việc tìm ở "nửa trước" hay "nửa sau" cũng được thực hiện giống như trên (nghĩa là ta cũng mở vào trang giữa ...). Việc tìm kiếm dừng lại khi tìm được từ trong một trang "giữa" nào đó (tìm kiếm thành công), hoặc khi từ điển chỉ còn một trang, mà không có từ cần tìm (tìm kiếm không thành công).

Như vậy, trong nhị phân tìm kiếm, giả sử với dãy khoá là X_l, X_{l+1}, …, X_r nó luôn chọn khoá ở giữa là X_j, với $j = [(l + r) / 2]$ để so sánh với khoá tìm kiếm KH. Tìm kiếm sẽ kết thúc nếu $KH = X_j$. Nếu $KH < X_j$ tìm kiếm được thực hiện tiếp với X_{l+1}, X_{l+2}, …, X_{j-1}, còn $KH > X_j$, tìm kiếm được thực hiện tiếp với X_{j+1}, X_{j+2}, …, X_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 là rỗng (không thấy).

<u>Ví dụ minh hoạ</u>

Giả sử các khoá tương ứng của các bản ghi trong bảng có 6 bản ghi là dãy số tăng dần:

-56 11 12 23 38 95 và khoá cần tìm là KH = 95, khi đó việc tìm kiếm được thực hiện như mô tả dưới đây.

X_1 X_2 X_3 X_4 X_5 X_6

[-56 11 **12** 23 38 95]

Bước 1: j = 3, khoá ở giữa là X_3 = 12, KH > X_3 Tìm kiếm tiếp tục với dãy khoá

[23 **38** 95]

Bước 2: j = 5, khoá ở giữa là X_5 = 38, KH > X_5 Tìm kiếm tiếp tục với dãy khoá

[95]

Bước 3: j = 6, khoá ở giữa là X_6 = 95, KH = X_5 Tìm kiếm thành công!

Ví dụ minh hoạ tìm kiếm không thành công, bạn đọc xem như là một bài tập.

Sau đây là giải thuật nhị phân tìm kiếm.

3.1.3.2. Giải thuật.

Giải thuật được viết dưới dạng một thủ tục dạng đệ quy, tựa ngôn ngữ C. Nếu tìm thấy, giải thuật trả về vị trí của khoá được tìm thấy (giá trị j). Nếu không tìm thấy, giải thuật trả về giá trị 0.

```
int BINARY_SEARCH(X[], l, r, KH)
{
//1. Trường hợp dãy được xét rỗng, tìm kiếm không thành công
    if (l > r)
        return 0;
// 2. Tìm kiếm trong dãy được xét
    else {
        j = (l + r) % 2;
    // 2.1. Tìm kiếm thành công
        if (KH == X[j])      return j;
    //2.2. Tìm ở nửa dãy trái
        else
            if (KH < X[j])       return  BINARY_SEARCH(X,l,j-1,KH);
    //2.3. Tìm ở nửa dãy phải
            else    return BINARY_SEARCH(X,j+1,r,KH);
    }
}
```

Trong giải thuật trên *l*, r tương ứng là chỉ số của khoá đầu tiên và chỉ số của khoá cuối cùng của dãy đang xét.

Giải thuật nhị phân tìm kiếm cũng có thể được viết dưới dạng lặp (khử đệ quy), việc này bạn đọc có thể tự làm.

3.1.3.3. Đánh giá giải thuật

Trong giải thuật trên ta thấy số lượng phép so sánh phụ thuộc vào giá trị khoá KH. Trường hợp thuận lợi nhất đối với dãy khoá X_1, X_2, …, X_n, mà lời gọi sẽ là BINARY_SEARCH(X, 1, n, KH), là KH = X[(n + 1) / 2], nghĩa là chỉ cần một phép so sánh, lúc đó $T_1 = O(1)$. Trường hợp xấu nhất có phức tạp hơn. Ta gọi w(r – *l* + 1) là hàm biểu thị số lượng phép so sánh trong trường hợp xấu nhất ứng với một lời gọi BINARY_SEARCH(X, *l*, r, KH) và đặt n = r – *l* + 1 (ứng với dãy khoá mà *l* = 1, r = n) thì trong trường hợp xấu nhất ta sẽ phải gọi đệ quy, vậy ta có:

$$w(n) = 1 + w(n \text{ div } 2)$$

Với phương pháp truy hồi có thể viết

$$w(n) = 1 + 1 + w(n \text{ div } 2^2) \Rightarrow w(n) = 1 + 1 + 1 + w(n \text{ div } 2^3)$$

Như vậy w(n) có dạng $w(n) = k + w(n \text{ div } 2^k)$

Khi $(n \text{ div } 2^k) = 1$ ta có $w(n \text{ div } 2^k) = w(1)$ và khi đó tìm kiếm phải kết thúc. Song $(n \text{ div } 2^k) = 1$ 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 $k = \log_2 n$. Vì vậy, cuối cùng ta có $w(n) = \log_2 n + 1$ hay $T_X(n) = O(\log_2 n)$

Người ta cũng chứng minh được $T_{tb}(n) = O(\log_2 n)$

Rõ ràng so với tìm kiếm tuần tự (có thời gian trung bình là O(n)), chi phí nhị phân tìm kiếm ít hơn nhiều. Hiện tại không có phương pháp tìm kiếm nào, dựa trên việc so sánh giá trị khoá lại có thể đạt được kết quả tốt hơn.

Tuy nhiên, nên nhớ rằng nhị phân tìm kiếm chỉ thực hiện trên dãy khoá đã được sắp xếp, nên trong tìm kiếm cũng phải tính đến chi phí cho việc sắp xếp. Nếu dãy khoá luôn biến động, thì chi phí cho việc sắp xếp sẽ lớn, và đó là một trở ngại lớn với phương pháp tìm kiếm này.

Qua bài học này chúng tôi đã giới thiệu với bạn đọc một số phương pháp sắp xếp và tìm kiếm khá thông dụng, cùng với ưu và nhược điểm của mỗi phương pháp. Mong rằng bạn đọc sẽ tìm hiểu thật kỹ và thành công trong việc ứng dụng chúng trong các đề án tin học mà mình sẽ triển khai.