Chương 7: Giải thuật tìm kiếm

1. Bài toán tìm kiếm

* Bài toán tìm kiếm:

  Cho dãy khóa k là các số nguyên có n phần tử. Tìm khóa có giá trị bằng x cho trước.

* Gọi x là khoá tìm kiếm hay giá trị tìm kiếm. Công việc tìm kiếm sẽ hoàn thành khi có một trong 2 tình huống sau xảy ra:

1

1- Tìm được khóa có giá trị bằng x.

2- Không tìm được khóa nào có giá trị bằng x.

Sau phép tìm kiếm không thấy, nếu có yêu cầu bổ sung khoá x vào dãy khóa thì giải thuật này gọi là "Tìm kiếm có bổ sung".

2

2. Tìm kiếm tuần tự (Sequential searching)

2.1. Phương pháp

Đây là giải thuật đơn giản, cổ điển.

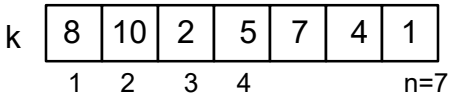
Ý tưởng giải thuật: Bắt đầu từ khóa thứ nhất, lần lượt so sánh khoá tìm kiếm x với các khóa trong dãy khóa cho đến khi tìm thấy hoặc đã hết dãy khóa mà chưa thấy.

* Giải thuật:

Cho dãy khoá k có n phần tử lưu trữ trên mảng một chiều k có n ô nhớ với chỉ số từ 1 đến n. Tìm khoá có giá trị bằng x, nếu tìm thấy thì trả về thứ tự của khoá, nếu không tìm thấy thì trả về 0.

3

Minh họa ý tưởng



x=5

4

```
Function  sequenceSearch(k,n,x)
    1. { Khởi tạo }
    i:=1; k[n+1]:=x;
    2. {Tìm kiếm trong dãy}
    While  k[i] <> x  Do  i:=i+1;
    3. { Trả về kết quả tìm kiếm }
    If  i=n+1  then  Return(0)  Esle  Return(i);
Return
```

5

2.2. Đánh giá

Trong giải thuật này phép toán tích cực là phép so sánh k[i] <> x .

- Trường hợp thuận lợi ta tìm thấy ngay ở phần tử đầu nên $C_{min} = 1$
- Trường hợp xấu nhất ta phải so sánh n+1 lần nên $C_{max} = n+1$
- Giả sử hiện tượng khoá tìm kiếm x trùng với một khoá nào đó của bảng là đồng khả năng thì $C_{tb} = (n+1)/2$

Do đó $T_{tb} = O(n)$.

- Nếu trường hợp hiện tượng khoá tìm kiếm x trùng với một khoá nào đó của bảng là không đồng khả năng, mà xác suất để xuất hiện $k_i = x$ là p_i, $p_i \neq 1/n$. Khi đó $C_{tb}=1*p_1 + 2p_2 + \ldots + n*p_n$

với $\sum_{i=1}^{n} p_i = 1$

Từ công thức trên ta nhận thấy với $p_1>p_2 > \ldots >p_n$ thì thời gian trung bình sẽ nhỏ nhất. Như vậy ta phải sắp xếp trước khi tìm kiếm.

6

3. Tìm kiếm nhị phân trên mảng

3.1 Phương pháp

- **Phương pháp tìm kiếm thực hiện trên dãy khóa đã sắp xếp tăng dần:**

  - Điều kiện: dãy khóa trên mảng sắp xếp tăng dần.

  - Tương tự như tra tìm từ trong từ điển hoặc danh bạ điện thoại. Chỉ khác là trong tra cứu ta chọn từ ngẫu nhiên, còn trong tìm kiếm nhị phân luôn chọn khoá "ở giữa" dãy khoá.

  - Giả sử có dãy khoá $k_L, \ldots, k_R$ đã được sắp xếp tăng dần, có khoá ở giữa là k_m với

  m=(L+R) div 2

7

+ Tìm kiếm sẽ kết thúc nếu: $x=k_m$

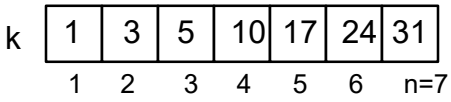
+ Nếu $x<k_m$ tìm kiếm sẽ được thực hiện tiếp với $k_L, \ldots, k_{m-1}$ với cách tương tự.

+ Nếu $x>k_m$ tìm kiếm sẽ được thực hiện tiếp với $k_{m+1}, \ldots, k_R$ với cách tương tự.

Qúa trình tìm kiếm kết thúc khi tìm thấy một khoá mong muốn hoặc dãy khoá rỗng (không tìm thấy).

8

Minh họa ý tưởng



x=3
m = (1+7) div 2 = 4
k[m] = 10

9

*** Giải thuật:**

Cho dãy k gồm n khoá, sắp xếp theo thứ tự tăng dần. Tìm khoá có giá trị =x.

Nếu tìm thấy thì trả về vị trí của khóa, nếu không tìm thấy thì trả về 0.

Dùng biến L, R, m: chỉ số đầu, chỉ số cuối, chỉ số giữa của dãy khoá k.

10

```
Function  binarySearch(k,n,x)
        1. {Khởi tạo}
        L := 1;  R := n;
        2. {Tìm kiếm}
        While  L <= R  Do
        Begin
                {Tính chỉ số giữa}
                m := (L+R) div 2;
                {So sánh}
                If  x < k[m]  then  R := m-1
                Else  IF  x > k[m]  then  L := m+1
                        Else  Return (m);
        End;
        3. {Không tìm thấy}
Return (0)
```

11

*** Giải thuật viết dạng đệ quy như sau:**

L, r là chỉ số đầu, chỉ số cuối của dãy K, biến nguyên Loc để đưa ra chỉ số ứng với khoá cần tìm, nếu không tìm thấy thì Loc =0.

```
Function  binarySearch(L,R,x)
    If  L>R then  Loc:=0
    Else
            begin m:=(L+R) div 2;
               If  x<k[m]   then
                    Loc:=binarySearch(L,m-1,x)
               Else  If  x>k[m]  then
                    Loc:=binarySearch(m+1,R,x)
                    Else  Loc:=m;
            end;
  Return(Loc)
```

12

3.2. Đánh giá

Phép tính tích cực là phép so sánh L<= r

C_{min}=1

Người ta đã tính được

C_{max}=[log_2n]

T_{tb}=O(log_2n)

Tìm kiếm nhị phân tốt hơn tìm kiếm tuần tự nhưng dãy k phải được sắp.

13

4. Tìm kiếm nhị phân trên cây

4.1. Định nghĩa cây nhị phân tìm kiếm

* Cây nhị phân tìm kiếm ứng với n khoá k_1, k_2, ..., k_n là một cây nhị phân mà mỗi nút của nó đều được định danh bởi một khoá nào đó trong các khoá đã cho. Đối với mọi nút trên cây tính chất sau đây luôn được thoả mãn:

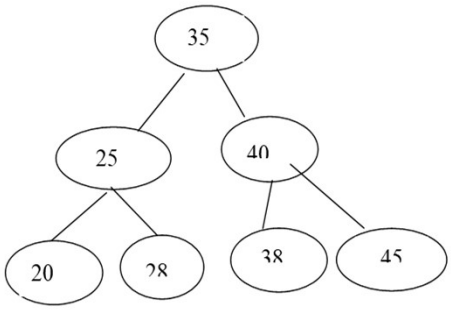
  - Mọi khoá thuộc cây con trái của một nút đều nhỏ hơn khoá ứng với nút đó.

  - Mọi khoá thuộc cây con phải của một nút đều lớn hơn khoá ứng với nút đó.

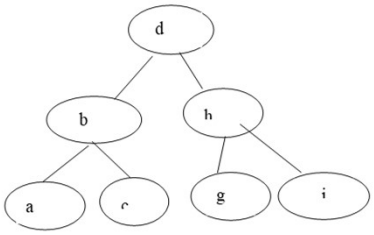
Chú ý : Khoá là số thì so sánh số bình thường, Khoá là chữ thì ta so sánh xâu kí tự.

14

Ví dụ có các cây nhị phân tìm kiếm sau:



a



b

15

4.2. Giải thuật tìm kiếm nhị phân trên cây

* Dãy khóa lưu trữ trên cây nhị phân.
* Điều kiện: Cây phải là cây nhị phần tìm kiếm
* Đối với một cây nhị phân để tìm kiếm xem một khoá x nào đó có trên cây đó không? Ta có thể thực hiện như sau:

  So sánh x với khoá ở gốc và một trong 4 tình huống sau đây sẽ xuất hiện:

  1- Không có gốc cây (cây rỗng): X không có trên cây, phép tìm kiếm không thoả mãn.

16

2- X trùng với khoá ở gốc: Phép tìm kiếm được thoả mãn.

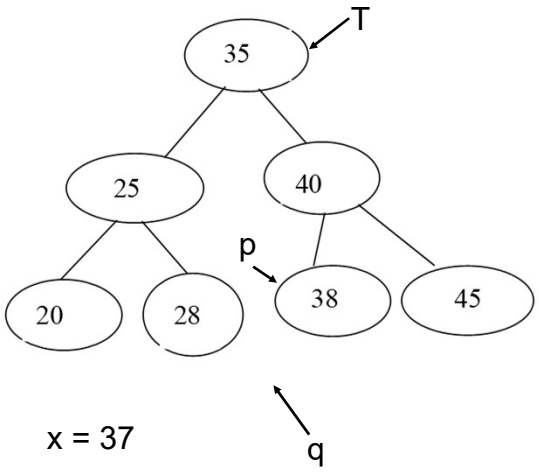
3- X nhỏ hơn khoá ở gốc: Tìm kiếm được thực hiện tiếp tục bằng cách xét cây con trái của gốc với cách làm tương tự.

4- X lớn hơn khoá ở gốc: Tìm kiếm được thực hiện tiếp tục bằng cách xét cây con phải của gốc với cách làm tương tự.

Cứ tiếp tục như vậy cho tới khi tìm thấy hoặc cây rỗng.

17

Minh họa ý tưởng



18

* Nếu phép tìm kiếm không thoả mãn ta bổ sung luôn x vào cây nhị phân tìm kiếm. Phép bổ sung không ảnh hưởng đến cây, đến vị trí các khoá trên cây.
* Mỗi nút của cây nhị phân tìm kiếm có dạng sau:

left	key	right

left: con trỏ trỏ tới gốc cây con trái.
right: con trỏ trỏ tới gốc cây con phải.
key: khoá của các nút.

* Thuật giải tìm kiếm trên cây nhị phân tìm kiếm như sau:

Cho cây nhị phân tìm kiếm có gốc được trỏ bởi T. Tìm nút trên cây có khoá bằng x.

Nếu tìm thấy thì trả về con trỏ trỏ tới nút đó, nếu không thấy thì bổ sung x x vào cây và trả về con trỏ tới nút mới.

19

```
Function BST(Var T; x)
1. {Khởi tạo con trỏ}
    p := Null ;
    q := T;
2. { Tìm kiếm }
    While q# Null Do
     Case
        x < key(q): p:=q;q:=left(q)
        x = key(q): Return(q)
        x > key(q): p:=q; q:=right(q)
     End Case
```

20

```
3. {Bo sung}
    q ⇐ AVAIL
    key(q):=x
    left(q):=right(q):= Null
    Case
        T=Null: T:=q
        x<key(p): left(p):=q
        Else: right(p):=q
    End case
Return(q)
```

$O(\log_2(n))$

21

Một số bài tập suy luận

1) Viết giả mã tạo cây nhị phân tìm kiếm cho dãy khóa k có n phần tử

2) Sửa lại giả mã BST thành BST2 cho trường hợp không bổ sung x vào cây, cho biết tìm thấy hay không tìm thấy x.

3) Sửa lại giả mã BST có bổ sung nhưng cho bên ngoài biết là có tìm thấy hay không tìm thấy x trên cây nhị phân tìm kiếm.

4) Cho cây nhị phân tìm kiếm T, viết giả mã đưa ra các khóa trên cây theo thứ tự tăng dần.

22

Câu 1

- Viết giả mã BST
- Viết giả mã tạo cây nhị phân tìm kiếm

Procedure createBST(Var T; k, n)

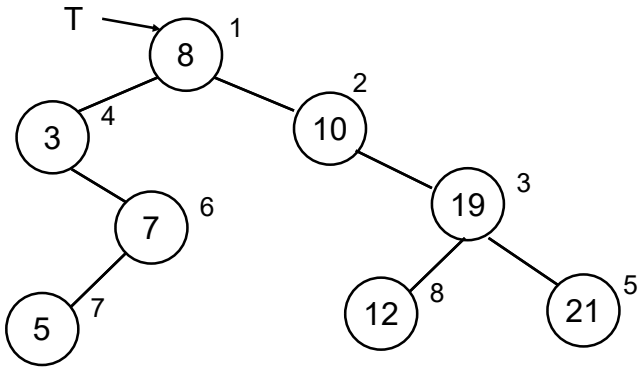
    For i:=1 to n do BST(T, k[i]);

Return

Ví dụ: Áp dụng vẽ cây nhị phân cho dãy khóa sau: 8 10 19 3 21 7 5 12

23

Dãy khóa k: 8 10 19 3 21 7 5 12



inOrder: 3 5 7 8 10 12 19 21

24

Bài tập

- Viết lệnh C++ cài đặt cấu trúc lưu trữ của cây nhị phân tìm kiếm có khóa là số nguyên.

25

Bài tập lập trình

- (ctdlgt-caynhiphantimkiem.cpp) Cài đặt cấu trúc dữ liệu cây nhị phân có phần tử là số nguyên. Ứng dụng vào tạo cây nhị phân tìm kiếm cho dãy khóa k có n phần tử là các số nguyên đọc vào từ tệp văn bản “daykhoa.txt”. Tìm khóa trên có giá trị bằng x, nếu không tìm thấy thì bổ sung x trở thành khóa trên cây, cho biết x có trên cây hay không. Đưa các khóa trên cây ra màn hình theo thứ tự tang dần.

26