CHƯƠNG 3
SẮP XẾP VÀ TÌM KIẾM NÂNG CAO

GV. Ngô Công Thắng
Bộ môn Công nghệ phần mềm
Khoa Công nghệ thông tin
Website: dse.vnua.edu.vn/ncthang
Email: ncthang@vnua.edu.vn

Nội dung Chương 3

1. Sắp xếp nhanh (Quick Sort)
2. Sắp xếp vun đống (Heap Sort)
3. Sắp xếp hòa nhập (Merge Sort)
4. Tìm kiếm nhị phân
5. Cây nhị phân tìm kiếm

1. Sắp xếp nhanh (Quick Sort)

1.1. Phương pháp

- Sắp xếp nhanh (quick sort) còn được sắp xếp phân đoạn (partition sort).
- Ý tưởng thuật toán:
  - Chọn ngẫu nhiên một phần tử x.
  - Duyệt từ bên trái mảng cho tới khi có một phần tử a_i>=x
  - Sau đó duyệt từ bên phải mảng cho tới khi có một phần tử aj=<x
  - Đổi chỗ a_i và a_j
  - Tiếp tục duyệt và đổi chỗ cho tới khi 2 phía gặp nhau.

1.1. Phương pháp (*tiếp*)

- Kết quả mảng được chia thành 2 phần: bên trái là các phần tử < x, bên phải là các phần tử > x.

Thủ tục sắp xếp nhanh

```
Procedure  Q_sort(L,R);
    1) If  L>=R then return;
    2) i:=L; j:=R ; k:=(L+R) div 2;
    3) x:=a[k];
    4) Repeat
            While a[i] <x  Do  i:=i+1;
            While a[j] >x  Do  j:=j-1;
            If  i<j  then  a[i] ↔ a[j]
       Until  i=j
    5) Call  Q_sort(L,j-1);  { Thực hiện trên nửa <x }
    6) Call  Q_sort(j+1,R);  { Thực hiện trên nửa >x }
Return
```


1.2. Đánh giá

- Người ta đã chứng minh được thời gian trung bình thực hiện giải thuật là:

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

- Như vậy, với n khá lớn Quick sort có hiệu lực hơn 3 thuật giải trên.

2. Sắp xếp vun đống (Heap Sort)

2.1. Phương pháp

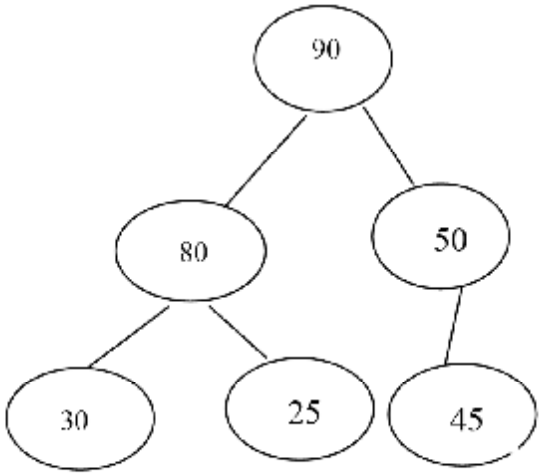
- Một cây nhị phân có chiều cao h được gọi là đống khi:
  - Là cây nhị phân hoàn chỉnh mà các nút lá ở mức h-1 phải nằm phía bên trái.
  - Khoá ở nút cha bao giờ cũng lớn hơn khoá ở nút con.

2. Sắp xếp vun đống (Heap Sort)

2.1. Phương pháp

- Thuật toán sắp xếp vun đống chia làm 2 giai đoạn.
- Giai đoạn 1: Tạo đống.

Ví dụ 1: Cây sau đây là một đống.



Khoá ở nút gốc của đống chính là khoá lớn nhất (khoá trội) so với các khoá trên cây.

- Giai đoạn 2:

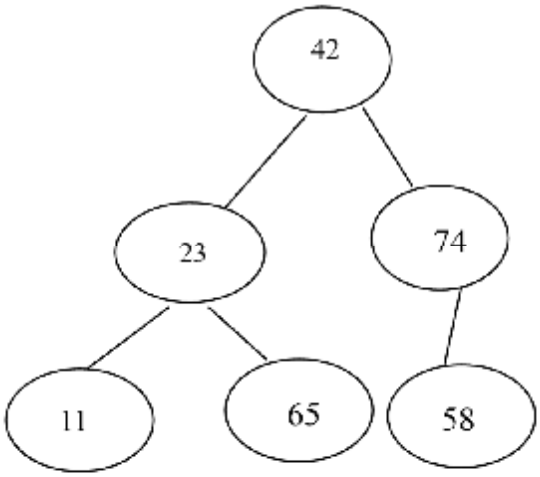
+ Đưa khoá trội về vị trí của nó bằng cách đổi chỗ cho khoá ở vị trí đó.

+ Vun lại đống với cây gồm các khoá còn lại (sau khi đã loại khoá trội)

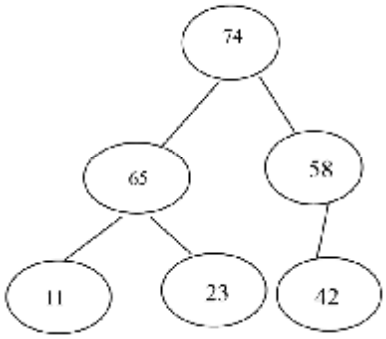
Quá trình trên lại được lặp lại.

Ví dụ: Có dãy khoá 42, 23, 74, 11, 65, 58

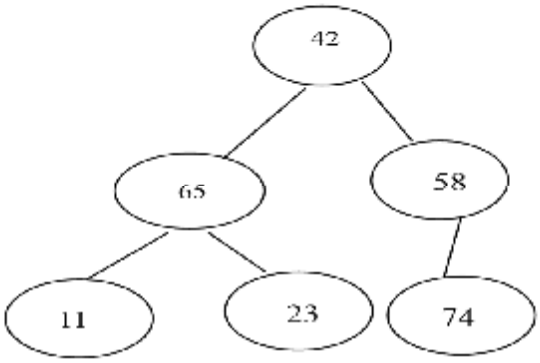
- Ta có cây hoàn chỉnh biểu diễn diễn dãy khoá:



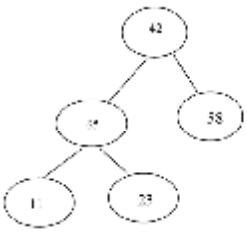
- Giai đoạn đầu: Vun đống ban đầu

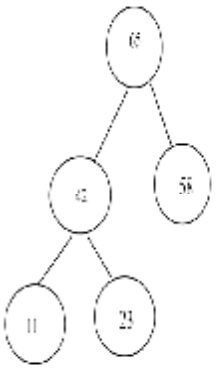


- Giai đoạn 2: Đổi chỗ 74 cho 42

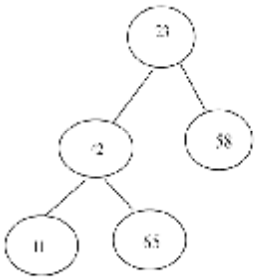


- Loại khoá trội 74 . Dãy đã sắp s= (74)

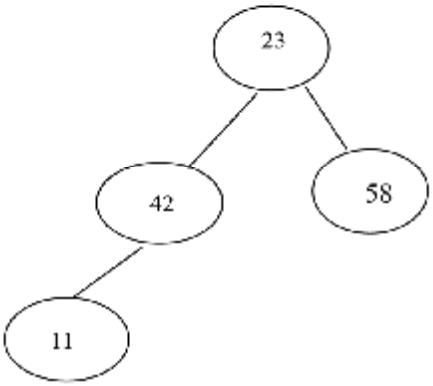




- Đổi chỗ 65 cho 23:



- Loại khoá trội 65 . Dãy đã sắp s= (65, 74)



- Lặp lại các bước tương tự cho các cây còn lại.

Cuối cùng ta thu được dãy đã sắp là s=(11, 23, 42, 58, 65, 74)

* Giải thuật vun đống:

- Một lá coi như cây con là một đống.

- Thuật toán tiến hành từ đáy lên: Chuyển đổi thành đống cho một cây con mà cây con trái và cây con phải của gốc đã là một đống.

Cây nhị phân hoàn chỉnh có n nút thì với chỉ số [n/2] trở lên có thể là nút cha: [n/2], [n/2]-1, . . . , 1.

a) Thủ tục vun đống:

Chỉnh lý cây nhị phân con hoàn chỉnh gốc i trên cây nhị phân có n nút để trở thành "đống" với điều kiện cây con trái và cây con phải có gốc là 2i và 2i+1 đã là đống.

```
Procedure  ADJUST(i,n)
1. { Khởi đầu }
Key:=K[i]; j:=2*i;
2. { Chọn con ứng với khoá lớn nhất trong 2 con của i }
While  j<=n  Do
  Begin
     If  (j<n) and (K[j]<K[j+1])  then  j:=j+1;
```

a) Thủ tục vun đống:

```
3. { So sánh khoá cha với khoá lớn nhất }
    If  Key > K[j]  then  Begin
            K[j/2]:=Key;
            Return;
        End;
    K[j/2]:=K[j]; j:=2*j;
 End; {Kết thúc while}
4. { Đưa Key vào vị trí của nó }
K[j/2]:=Key;
5. Return;
```


b) Thủ tục sắp xếp vun đống:

```
Procedure  Heap_Sort(K,n)
1. { Tạo đống ban đầu }
For  i:=[n/2]  Downto  1  Do
      Call  ADJUST(i,n)
2. { Sắp xếp }
For  i:= n-1  Downto  1 Do
 Begin
     tg:=K[1]; K[1]:=K[i+1];
     K[i+1]:=tg;
     Call  ADJUST(1,i);
 End;
3.  Return
```

5.2. Đánh giá

Thời gian thực hiện trung bình của giải thuật này là $O(n\log_2^n)$.

3. Sắp xếp kiểu hoà nhập (MERGE SORT)

3.1. Phép hoà nhập 2 đường

Thực hiện hợp nhất các bản ghi của 2 bảng đã được sắp xếp thành 1 bảng được sắp.

a) Phương pháp:

So sánh 2 khoá nhỏ nhất (hoặc lớn nhất của 2 bảng) để đưa vào miền sắp xếp.

Quá trình cứ tiếp tục cho tới khi 2 bảng đã cạn.

b) Giải thuật:

Bảng 1: $(x_b, ..., x_m)$

Bảng 2: $(x_{m+1}, ..., x_n)$

Bảng sắp: $(z_b, ..., z_n)$

Ví dụ: Bảng 1: (3, 5, 10, 16)

Bảng 2: (1, 4, 15)

Bảng sắp: (1, 3, 4, 5, 10, 15, 16)

* Thủ tục như sau:

```
Procedure  MERGE(X,b,m,n,Z);
1. i:=k:=b; j:=m+1;
2. While  (i<=m) and (j<=n)  Do
      Begin  If  x[i]<=x[j]  then
            Begin  Z[k]:=x[i];
                  i:=i+1;
            End
            Else  Begin  z[k]:=x[j];
                        j:=j+1;
                  End;
            k:=k+1;
      End;
```

* Thủ tục như sau:

3. { Một trong 2 bảng con đã cạn }

If i>m then (zk, ..., zn):= (xj, ..., xn)

Else (zk, ..., zn):= (xi, ..., xm)

4. Return

3.2. Sắp xếp kiểu hòa nhập trực tiếp (Straight two way merge)

* Bảng con đã được sắp gọi là một mạch (run).

* Mỗi bản ghi coi như 1 mạch có độ dài (kích thước) là 1. Nếu hoà nhập 2 bảng như vậy ta được 1 mạch mới có độ dài =2. Hoà nhập 2 mạch có độ dài là 2 ta được một mạch có độ dài là 4, ...

* Thủ tục MPASS thực hiện một bước của sắp xếp hoà nhập. Nó hòa nhập từng cặp mạch kề cận nhau, có độ dài L, từ bảng X sang bảng Y, n là số lượng khoá (bản ghi) trong X.

```
Procedure MPASS(X,Y,n,L)
    1. i:=1;
    2. Hoà nhập cặp mạch có độ dài L }
     While  i<= n-(2L-1)  Do
          Begin  Call  MERGE(X,i,i+L-1,i+2L-1, Y)
               i:=i+2L;
          End;
    3. { Hoà nhập cặp mạch còn lại cuối cùng
         với tổng độ dài <2L}
    If  i+L-1 <n  then  Call  MERGE(X,i,i+L-1,n,Y)
    Else  (y_i, ..., y_n):= (x_i, ..., x_n);
Return
```


Thủ tục MPASS trên sẽ được gọi tới trong thủ tục sắp xếp kiểu hoà nhập trình bày ở sau đây.

* Thủ tục sắp xếp kiểu hoà nhập trực tiếp:

Thủ tục này lưu trữ các mạch mới khi hoà nhập dùng biến phụ là Y(n). trong đó X và Y được dùng luân phiên. L là kích thước của mạch, sau mỗi lượt L được tăng gấp đôi.

```
Procedure  STRAIGHT_MSORT(X,n)
1. L:=1;
2. While  L<n  Do
     Begin
          Call  MPASS(X,Y,n,L);  L:=L+L;
          Call  MPASS(Y,X,n,L);  L:=L+L;
     End;
3. Return
```

Ví dụ: X= 9, 1, 7, 6, 4, 3, 8, 7

Bước 1:	1, 9, 6, 7, 3, 4, 7, 8	đưa vào Y	L=1
Bước 2:	1, 6, 7, 9, 3, 4, 7, 8	đưa vào X	L=2
Bước 3:	1, 3, 4, 6, 7, 7, 8, 9	đưa vào Y	L=4

3.3. Đánh giá

Thời gian thực hiện trung bình của giải thuật là:

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

* Nhận xét chung:

- Với n nhỏ có thể dùng các phương pháp: chọn trực tiếp, chèn trực tiếp, đổi chỗ trực tiếp.
- Với n lớn: Nếu dãy khoá không sắp dùng Quick sort, nếu dãy khoá có sắp dùng Heap sort.

4. Tìm kiếm nhị phân

Bài toán tìm kiếm

* Bài toán tìm kiếm được phát biểu như sau:

Cho một bảng gồm n bản ghi $r_1, r_2, \ldots, r_n$; r_i ($1 <= i <= n$) tương ứng với một khoá k_i . Hãy tìm bản ghi có giá trị khoá tương ứng 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- Tìm được bản ghi có giá trị khoá tương ứng bằng x. Lúc đó ta nói phép tìm kiếm được thoả.

2- Không tìm được bản ghi nào có giá trị khoá bằng x . Khi đó ta nói phép tìm kiếm không thoả.

Sau phép tìm kiếm không thoả nếu có yêu cần bổ sung bản ghi mới có khoá x vào bảng. Giải thuật này gọi là " Tìm kiếm có bổ sung".

Khoá của mỗi bản ghi chính là đặc điểm nhận biết của bản ghi đó trong tìm kiếm, ta coi nó là đại diện của bản ghi trong giải thuật.

4.1. Ý tưởng giải thuật

* Phương pháp tìm kiếm thực hiện trên dãy khóa đã sắp xếp, có nội dung như sau:

- 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" dẫy khoá.

- Giả sử có dãy khoá $k_L, \ldots, k_R$ thì khoá ở giữa là k_i với

i=(L+R) div 2

+ Tìm kiếm sẽ kết thúc nếu: $x=k_i$

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

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

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

* Giải thuật:

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

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

Nếu tìm thấy cho ra chỉ số của khoá đó, nếu không tìm thấy cho ra 0.

```
Function  Binary_search(K,n,x)
    1. { Khởi tạo }
    L:=1;  R:=n;
    2. { Tìm kiếm }
    While  L<= R  Do
    Begin
    3. { Tính chỉ số giữa }
            m:=( L+R) div 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; {End of While}
5. { Không tìm thấy }
Return (0)
```


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

4.2. 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.

5. Cây nhị phân tìm kiếm

5.1. Định nghĩa cây nhị phân tìm kiếm

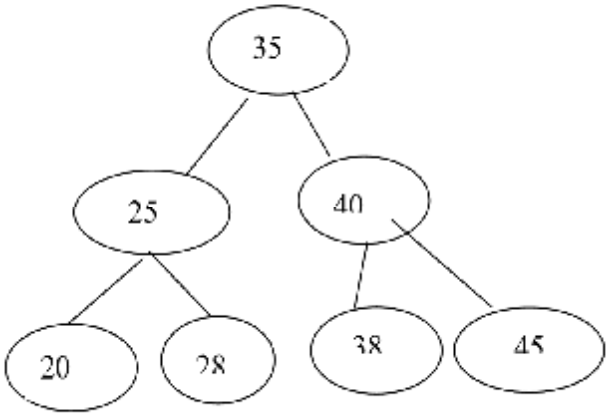
* Cây nhị phân tìm kiếm ứng với n khoá k_1, k_2, ..., k_n là một cây nhị phân mà mỗi nút của nó đều được định danh bởi một khoá nào đó trong các khoá đã cho. Đối với mọi nút trên cây tính chất sau đây luôn được thoả mãn:

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

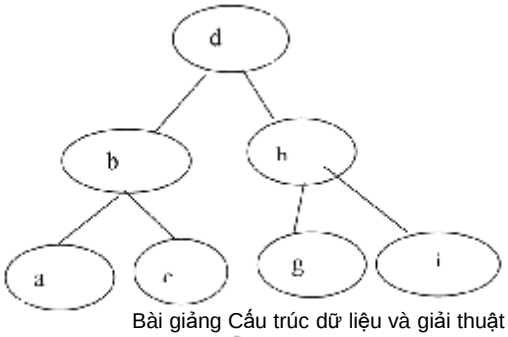
  - Mọi khoá thuộc cây con phải của một nút đều lớn hơn khoá ứng với nút đó.

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

Ví dụ có các cây nhị phân tìm kiếm sau:



a



b

5.2. Giải thuật 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.

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.

Ví dụ Tìm x=28 trên cây a: So x và 35, x<35 nên ta tìm trên cây con trái của 35

X>25 nên lại tìm trong cây con phải. So sánh ta có x=cây con phải cũng là 28 nên phép tìm kiếm được thoả mãn.

Nếu tìm x=m trên cây b thì phép tìm kiếm không thoả mãn.

* 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:

LPTR	KEY	INFOR	RPTR

LPTR: con trỏ trỏ tới gốc cây con trái.
RPTR: con trỏ trỏ tới gốc cây con phải.
KEY: khoá của các nút.
INFOR: thông tin.

* 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 kiếm được thoả thì con trỏ trỏ tới nút đó, giá trị trả về là q, đó là địa chỉ của nút ta tìm được (con trỏ q trỏ tới nút đó)

Nếu không tìm dược ta bổ sung x vào cây T, con trỏ q trỏ vào nút mới đó.

```
Function BST(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:=LPTR(q)
        X=key(q): Return(q)
        X>key(q): p:=q; q:=RPTR(q)
     End Case
```



```
3. {Bo sung}
    q ⇐ AVAIL
    Key(q):=X
    LPTR(q):=RPTR(q):= Null
    Case
        T=Null: T:=q
        X<key(p): LPTR(p):=q
        Else: RPTR(p):=q
    End case
Return(q)
```