

Chương 5. Các thuật toán sắp xếp

I. Khái niệm sắp xếp

1) Đặt vấn đề

- Giả sử cho dãy số sau: 3 2 4 1 6 3.

Yêu cầu của bài toán: Hãy tìm giá trị nhỏ nhất và lớn nhất của dãy số đã cho.

Thông thường ta sẽ duyệt từng phần tử một và so sánh chúng với nhau để tìm ra giá trị nhỏ nhất và lớn nhất của dãy số và như vậy ta sẽ mất rất nhiều thời gian. Nhưng nếu dãy số trên có thể như sau: 1 2 3 3 4 6 hoặc 6 4 3 3 2 1.

Ta thấy ngay rằng việc tìm giá trị nhỏ nhất hoặc lớn nhất của dãy số trên trở nên hết sức đơn giản, ta chỉ việc đưa ra phần tử đầu hoặc phần tử cuối tùy theo cách biến đổi dãy.

- Xét ví dụ khác: Quên lý danh sách các học sinh.

Cho danh sách các em học sinh có tên như sau:

Minh, Huệ, Bình, An, Tuấn.

Yêu cầu của bài toán: Hãy tìm tên của một bạn nào đó trong danh sách này?

Nếu phải tìm tên của một em trong danh sách với số lượng không lớn thì bài toán có thể giải quyết bằng nhiều cách. Nhưng với số lượng lớn các em thì việc tìm kiếm sẽ trở nên khó khăn hơn và việc ta đưa danh sách trên vào dùng như sau:

An, Bình, Huệ, Minh, Tuấn.

Đó có thể là một dãy số hay một danh sách như trên ta phải thay đổi vị trí của các phần tử trong dãy (hoặc danh sách) ban đầu thành một dãy (hoặc danh sách) mới. Vậy việc mà ta làm như thế đó gọi là sắp xếp.

2) Khái niệm

Sắp xếp là quá trình bố trí lại các phần tử của một danh sách để từng nào đó theo một thứ tự nhất định.

3) Mục đích

Sắp xếp nhằm giúp cho việc tìm kiếm dữ liệu trở nên dễ dàng và nhanh chóng hơn. Ngoài ra còn có các phương pháp khác nhau: sắp xếp các kết quả xếp lại in ra trên bảng biểu...

Nhìn chung, dữ liệu được sắp xếp có thể xuất hiện dưới nhiều dạng khác nhau, nhưng ở đây ta quy định tập dữ liệu sắp xếp là tập các bản ghi, mỗi bản ghi gồm một số trường, dữ liệu trường này với những thuộc tính khác nhau. Tuy nhiên không phải toàn bộ các trường dữ liệu của bản ghi đều được xem xét đến trong quá trình sắp xếp mà chỉ đưa vào một hoặc một vài trường nào đó, trường này với dữ liệu gì là khoá. Quá trình sắp xếp sẽ được tiến hành đưa vào giá trị của khoá.

Trong chương này, ta chỉ xét tới các phương pháp sắp xếp trong (internal sorting), nghĩa là các phương pháp tác động trên một tập các bản ghi lưu trữ dưới dạng thứ tự trong bộ nhớ trong mà ta gọi là bảng (table). Bên cạnh đó còn có các phương pháp sắp xếp ngoài (external sorting) tác động trên một tập các bản ghi lưu trữ ở bộ nhớ ngoài (file).

II. Ba phương pháp sắp xếp cơ bản

1. Sắp xếp kiểu lựa chọn (Selection Sort)

Một trong những phương pháp dễ dàng nhất để thực hiện sắp xếp một bảng là sắp xếp dựa trên phép lựa chọn

a) Nguyên tắc sắp xếp

Ở bước thứ i ($i=1, 2, \dots, n$), ta chọn trong dãy khoá $k_i, k_{i+1}, \dots, k_n$ khoá có giá trị nhỏ nhất và đổi chỗ nó với k_i .

Như vậy: sau j bước, j khoá nhỏ hơn đã nằm ở vị trí các vị trí thứ nhất, thứ hai, ..., thứ j theo đúng thứ tự sắp xếp.

Ví dụ

i	k _i	L- ít	1	2	3	4	...	9
1	42		11	11	11	11		11
2	23		23	23	23	23		23
3	74		74	74	36	36		36
4	11		42	42	42	42		42
5	65		65	65	65	65		58
6	58		58	58	58	58		65
7	94		94	94	94	94		74
8	36		36	36	74	74		87
9	99		99	99	99	99		94
10	87		87	87	87	87		99

b) Gi¶i thu¶t

D¶i ¶i đây là gi¶i thu¶t s¶p x¶p b¶ng ph¶¶ng pháp l¶a ch¶n d¶¶i d¶¶ng m¶t th¶ t¶c pascal:

Void SELECTION_SORT (k, n);

{ S¶p x¶p dãy khóa k có n ph¶¶n t¶ }

{

For (i=1; i<n-1; i++)

{

m:=i;

For (j=i+1; j<n; j++)

If (k[j] < k[m]) m:=j;

If (m <> i)

{

X:=k[i];

k[i] := k[m];

```

        k[m] := X;
    }
}

```

2. Sắp xếp kiểu thêm dần (Insertion Sort)

a) Nguyên tắc sắp xếp:

Nguyên tắc sắp xếp của phương pháp này dựa theo kinh nghiệm của nhúng ngỗng vào chiếc bài. Khi có $i-1$ lá bài đã được sắp xếp ở trên tay, nếu rút thêm lá bài tiếp theo vào thì sắp xếp thế nào? Câu trả lời là có thể so sánh lá bài mới lấy tới với lá bài thứ $i-1$, thứ $i-2$, ... để tìm ra vị trí thích hợp của lá bài mới và chèn vào vị trí đó.

Dựa trên nguyên tắc này, có thể triển khai một cách sắp xếp như sau:

- Đầu tiên, bảng được coi chứa một khóa k_1 đã được sắp xếp.
- Xét thêm k_2 , so sánh k_2 với k_1 để xác định vị trí chèn k_2 , ta được một bảng gồm 2 khóa đã được sắp xếp.
- Xét thêm k_3 , tiếp tục so sánh k_3 với k_2 và k_1 để xác định vị trí chèn k_3 , tiếp tục tiếp tục với các khóa $k_4, k_5, \dots$, và cuối cùng là k_n , ta được bảng khóa đã sắp xếp hoàn toàn

Ví dụ:

Lần	1	2	3	4	...	8	9	10
Khóa đưa vào	42	23	74	11		36	99	87
1	42	23↓	23	11↓		11	11	11
2	-	42	42	23↓		23	23	23
3	-	-	74	42		36	36	36
4	-	-	-	74		42	42	42

5	-	-	-	-		58	58	58
6	-	-	-	-		65	65	65
7	-	-	-	-		74	74	74
8	-	-	-	-		94	94	87
9	-	-	-	-		-	99	94
10	-	-	-	-		-	-	99

b) Giải thuật:

Đây là giải thuật sắp xếp bằng phương pháp thêm dần dãy i đang mở thuật toán pascal:

Void INSERTION_SORT (k, n);

{ Sắp xếp dãy khóa k có n phần tử }

{

For(i=1; i<n; i++)

{

X := k[i];

j := i-1;

While ((X < k[j]) &&(j > 0))

{

k[j+1] := k[j];

j := j-1;

}

k[j+1] := X;

}}

3. Sắp xếp kiếu nổi bọt (Bubble Sort)

Trong các phương pháp sắp xếp nêu trên, kỹ thuật đổi chỗ đầu đã được sử dụng nhưng chưa trở thành điểm nổi bật. Ở phương pháp này, việc đổi chỗ các cặp khóa kề cận khi chúng ngược thứ tự sẽ được thực hiện thông xuyên cho tới khi toàn bộ dãy khóa được sắp xếp.

a) Nguyên tắc sắp xếp:

Dãy khóa được duyệt từ dưới ngược lên, trong quá trình duyệt, nếu gặp hai khóa kề cận nhưng ngược thứ tự thì đổi chỗ chúng cho nhau.

Như vậy:

- Trong lượt duyệt thứ nhất, khóa nhỏ nhất sẽ chuyển dần lên đỉnh
- Trong lượt duyệt thứ hai, khóa nhỏ thứ hai sẽ chuyển dần lên vị trí thứ hai.

....

- Sau n-1 lượt duyệt, dãy khóa sẽ được sắp xếp hoàn toàn.

Nếu hình dung dãy khóa được thông đồng thì sau từng lượt sắp xếp, các giá trị khóa nhỏ sẽ nổi dần lên giống như các bọt nổi. Chính vì vậy, phương pháp này được gọi là sắp xếp kiếu nổi bọt (Bubble Sort).

Ví dụ:

i	k ₁	Lượt	1	2	3	4	5	...	9
1	42		11	11	11	11	11		11
2	23		42	23	23	23	23		23
3	74		23	42	36	36	36		36
4	11		74	36	42	42	42		42
5	65		36	74	58	58	58		58
6	58		65	58	74	65	65		65
7	94		58	65	65	74	74		74

8	36	94	87	87	87	87	87
9	99	87	94	94	94	94	94
10	87	99	99	99	99	99	99

b) Giíi thuít:

void BUBBLE_SORT (k, n);

{Sôp xôp dãy khóa k có n phần tử }

{

For (i =1; i<n; i++)

For(j =n-1; j>= i + 1; j--)

If (k [j] < k [j - 1])

{

X := k [j];

k [j] := k [j - 1];

k [j - 1] := X;

}

}

4. Đánh giá ba phương pháp sắp xếp đơn giản

Để i vớ i môt số phương pháp sắp xếp, khi xét tở i hiu lỏ c cỏ a nó, ngoài nhữ ng đánh giá vớ môt không gian nhữ cỏ n thiỏ t, ngỏ i ta thỏ ng lỏ u ý đở c biỏ t tở i chi phí vớ thỏ i gian. Mà thỏ i gian thì chỏ yỏ u phỏ thuỏ c vào viỏ c thỏ c hiỏ n các phép so sánh giá trỏ khóa và các phép chuyỏn chỏ bỏ ng ghi khi sắp xếp. Vì vớ y thông thỏ ng ngỏ i ta lỏ y số lỏ ng trung bình các phép so sánh và các phép chuyỏn chỏ làm đở i lỏ ng đở c trỏ ng cho chi phí vớ thỏ i gian thỏ c hiỏ n cỏ a tỏ ng phỏ ng pháp. Tuy nhiên ở đây ta chỏ xét tở i phép so sánh và coi nó nhỏ môt “phép toán tích cỏ c” cỏ a các giỏ i thuỏ t.

Đôi khi phép sắp xếp kiếu lựa chọn, ta thấy: i lần thứ i ($i = 1, 2, \dots, n-1$) để tìm khoá nhỏ nhất bao giờ cũng cần $C_i = (n-i)$ phép so sánh. Số lượng phép so sánh này không hề phụ thuộc gì vào tình trạng ban đầu của dãy khoá C . Từ đó suy ra:

$$C_{\min} = C_{\max} = C_{tb} = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2}$$

Còn với phép sắp xếp kiếu thêm dần (gọi tắt INSERT - SORT) thì có hai khác. Rõ ràng số lượng phép so sánh phụ thuộc vào dãy khoá ban đầu. Trường hợp thuận lợi nhất là khi với dãy khoá đã được sắp xếp rồi. Như vậy số lần lặp chỉ cần 1 phép so sánh. Do đó

$$C_{\min} = \sum_{i=2}^n 1 = n-1$$

Nhưng nếu dãy khoá ban đầu có thể ngược với thứ tự sắp xếp thì số lần lặp thứ i phải cần có: $C_i = (i-1)$ phép so sánh. Vì vậy:

$$C_{\max} = \sum_{i=2}^n (i-1) = \frac{n(n-1)}{2}$$

Nếu gọi số lần giá trị khoá đầu xuất hiện đang khả năng thì trung bình số lần lặp i có thể coi như $C_i = i/2$ phép so sánh. Suy ra:

$$C_{tb} = \sum_{i=2}^n \frac{i}{2} = \frac{n(n-1)}{2}$$

Nhìn vào các kết quả đánh giá ở trên ta thấy INSERT-SORT tỏ ra có “tốt hơn” so với hai phép sắp xếp kia. Tuy nhiên với n khá lớn, chi phí về thời gian thực hiện để đánh giá qua cặp dữ liệu, thì cả ba phép sắp xếp đều có cặp $O(n^2)$ và đây vẫn là mức chi phí cao so với một số phép sắp xếp mà ta sẽ xét thêm sau đây.

III. Sắp xếp bằng phép phân độ

1. Giới thiệu phép phân độ

2. Minh họa phép phân độ

3. Giới thiệu

Đây đây là giới thiệu sơ lược bằng phương pháp phân đoạn để viết bằng ngôn ngữ C, đây là một giới thiệu đơn giản.

```
#include "stdio.h"
```

```
typedef int mang[100];
```

```
void quick_sort(mang a,int l,int r)
```

```
{
```

```
    int tg,key,i,j;
```

```
    if (l>=r) return;
```

```
    key=a[(l+r)/2];
```

```
    i=l; j=r;
```

```
    do
```

```
    {
```

```
        while (a[i]<key) i++;
```

```
        while (a[j]>key) j--;
```

```
        if (i<=j)
```

```
        {
```

```
            tg=a[i];
```

```
            a[i]=a[j];
```

```
            a[j]=tg;
```

```
            i++;
```

```
            j--;
```

```
        }
```

```
    }
```

```
    while (i<=j);
```

```

        quick_sort(a,l,j);

        quick_sort(a,i,r);

    }

void main()

{

    int n,i,a[100];

    clrscr();

    printf("\n Nhap n=");scanf("%d",&n);

    for (i=0;i<n;i++)

    {

        printf("a[%d]= ",i);scanf("%d",&a[i]);

    }

    quick_sort(a,0,n-1);

    for (i=0;i<n;i++)

    {

        printf("%d ",a[i]);

    }

    getch();

}

```

4. Đánh giá

Trên cơ sở kết quả ta hãy để ý đến một vài chi tiết có ảnh hưởng tới hiệu quả của phương pháp, đồng thời cũng thấy hiện rõ được điểm của phương pháp này.

5. Vấn đề chọn “chốt”:

Theo giới thiệu chung thì chốt có thể chọn tuỳ ý trong dãy khoá cho, nhưng rõ ràng khi thấy hiện giới thuật ta phải định ra một cách chọn chốt

có thể. Nếu chúng ta chèn r vào đúng khoá nhỏ nhất (hoặc lớn nhất) của phân đồ nên cần xử lý thì sau mỗi lần ta chỉ tách ra được một phân đồ nên còn có kích thước nhỏ hơn trước là 1 (vì đã bớt đi một khoá chính là “chốt”) và chính phân đồ này sẽ được xử lý tiếp theo luôn. Như vậy ta đã quay trở lại phương pháp sắp xếp kiểu nổi bọt đơn giản. Việc chèn chốt nhỏ nhất này đã dẫn đến tình huống xấu nhất của phương pháp.

Nếu gọi “trung vị” (median) của một dãy khoá là khoá sẽ đứng ở giữa dãy đó sau khi dãy đã được sắp xếp, nghĩa là nó lớn hơn một nửa số khoá của dãy và nhỏ hơn số còn lại, thì thuật toán vẫn là chèn được đúng trung vị làm “chốt”. Lúc đó sau mỗi lần ta sẽ tách ra được hai phân đồ nên có độ dài gần nhau và phân đồ xử lý tiếp theo có kích thước chỉ bằng “một nửa” phân đồ đã chia nó.

Nhưng làm sao có thể chèn được đúng trung vị? Nếu gọi thuật: sẽ xuất hiện của các khoá trong dãy là đúng khi năng thì trung vị có thể là bất kỳ một khoá nào trong dãy. Trong giải thuật trên, ta chèn khoá đứng đầu làm chốt là dựa trên cơ sở này. Nhưng với cách chèn này, nếu dãy khoá có khuynh hướng đã theo thứ tự sắp xếp thì khả năng xấu nhất lại xuất hiện. Tuy nhiên nếu khuynh hướng này hay xuất hiện thì việc chèn khoá đang đứng ở giữa dãy lại gặp thuận lợi.

Đúng đắn với cách chèn như trên, đúng thì cũng đủ kết hợp với một đề nghị sau này của Hoare là: “Chèn trung vị của một dãy khoá nhỏ hơn, thu được dãy khoá cho, làm chốt”, R.C.Singleton đã đưa ra một cách chèn là:

Chèn K_q là “chốt” với K_q là trung vị của ba khoá K_1 , $K[(1+r)/2]$ và K_r trong đó l và r là chỉ số của khoá đầu và khoá cuối của dãy cho. Các kiểu chèn khoá chốt còn được nhiều tác giả khác nêu ra và cũng có nhiều kết quả đáng chú ý.

6. Vấn đề phân hoạch và cách sắp xếp khác

Khi kích thước của các phân đồ đã khá nhỏ, việc tiếp tục phân đồ nên theo QUICK-SORT thì có thể không có lợi. Lúc đó sẽ dùng một phương pháp

pháp sắp xếp đơn giản lại tốn kém. Vì vậy, sắp xếp NHANH thường không tốn hành trình đi mà dùng lại ở lúc cần thiết để giải quyết một phương pháp sắp xếp đơn giản, giao cho nó tiếp tục thực hiện sắp xếp với các phần đơn nhỏ hơn lại. Kunth (1974) có nêu: 9 có thể coi là kích thước giải quyết của phần đơn để sau đó QUICK-SORT giải quyết phương pháp sắp xếp đơn giản.

Ngoài ra còn để chọn phần đơn nào để xử lý tiếp theo cũng là một vấn đề cần được xem xét kỹ. Tuy nhiên ta sẽ không đi sâu vào đây.

Bây giờ ta xét sang việc đánh giá giải thuật QUICK-SORT. Do giải thuật là đệ quy nên ta sẽ áp dụng một cách tiếp cận khác một chút.

Giả sử $T(n)$ là thời gian thực hiện giải thuật để sắp xếp n khóa, $P(n)$ là thời gian để phân phần đơn một bảng n khóa thành hai bảng con. Ta có thể viết:

$$T(n) = P(n) + T(j-LB) + T(UB-j)$$

Chú ý rằng $P(n) = Cn$ với C là hằng số.

Trường hợp xấu nhất xảy ra khi bảng khóa n đã có thể sắp xếp: sau khi phân phần đơn một trong hai bảng con là rỗng ($j = LB$ hoặc $j = UB$)

Giả sử $j = LB$, ta có:

$$T_x(n) = P(n) + T_x(0) + T_x(n-1)$$

$$= Cn + T_x(n-1)$$

$$= Cn + C(n-1) + T_x(n-2)$$

...

...

...

$$= \sum_{k=1}^n C_k + T_x(0)$$

$$= C \cdot \frac{n(n+1)}{2} = O(n^2)$$

Trường hợp này QUICK-SORT không hơn gì các phương pháp đã nêu trước đây.

Trình độ phân hoạch tiếp tục xảy ra khi mảng luôn luôn được chia đôi, nghĩa là $j = \frac{LB+UB}{2}$. Lúc đó:

$$\begin{aligned}
 T_1(n) &= P(n)b + 2T_1(n/2) \\
 &= Cn + 2T_1(n/2) \\
 &= Cn + 2C(n/2) + 4T_1(n/4) = 2Cn + 2^2T_1(n/4) \\
 &= Cn + 2C(n/2) + 4C(n/4) + 8T_1(n/8) = 3Cn + 2^3T_1(n/8) \\
 &\dots \\
 &\dots \\
 &= (\log_2 n) Cn + 2^{\log_2 n} T_1(1) \\
 &= O(n \log_2 n)
 \end{aligned}$$

Vì xác định giá trị trung bình $T_{tb}(n)$ không còn đơn giản như hai trường hợp trên, nên ta sẽ không xét chi tiết. Kết quả mà ta cần ghi nhớ là: người ta đã chứng minh được:

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

Như vậy rõ ràng là khi n khá lớn thì QUICK-SORT đã tỏ ra có hiệu quả hơn hẳn ba phương pháp đã nêu.

IV. Heap Sort (Sắp xếp kiểu vun đống)

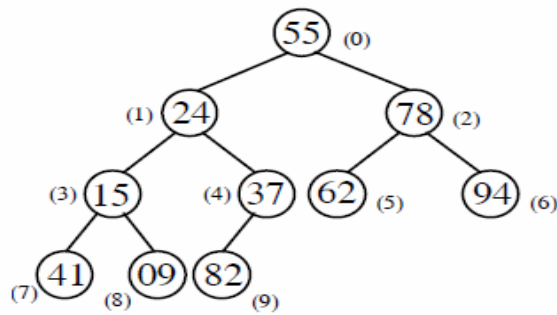
1. Định nghĩa Heap: Heap là cây nhị phân đầy đủ có cài đặt mảng mảng một chiều với các nút trên Heap có nội dung lớn hơn hay bằng nội dung của các nút con của nó.

Ví dụ: Ta có dãy số sau:

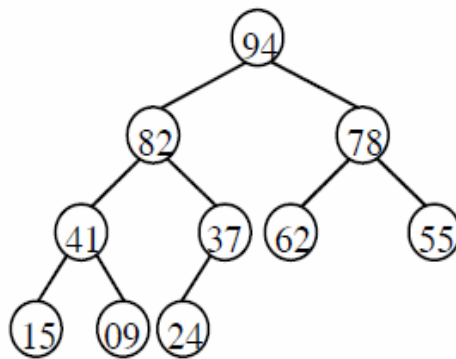
55 24 78 15 37 62 94 41 09 82

(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)

thì cây nhị phân ban đầu của dãy là :



và heap của dãy sau khi vun đống sẽ là :



2. Thuật toán Heap Sort:

a) Nội dung:

- Trước tiên ta cài đặt hàm Adjust (A, r,n) để điều chỉnh cây có gốc ở vị trí r với n nút thành đống với giả thiết hai cây con đã là đống.

Lưu ý: Do giả thiết là ta đã có hai cây con là đống nên ta phải vun đống từ dưới lên. Trong cây ch có các nút với vị trí $0, 1, \dots, n/2 - 1$ mới là các nút gốc có cây con. Do đó, ta chỉ cần vun đống các cây ở vị trí $0, 1, \dots, n/2 - 1$.

- Do sau khi vun đống thì nút ở vị trí 0 sẽ có nội dung lớn nhất nên ta sẽ hoán đổi nội dung của A_0 với A_{n-1} .

- Ta tiếp tục Adjust(0, n-1) để vun đống cây với gốc ở nút có vị trí 0 và sẽ nút là n-1 nút còn lại ($A_0, A_1, \dots, A_{n-2}$). Sau đó, sẽ hoán đổi nội dung của A_0 với A_{n-2} .

- Quá trình trên tiếp tục cho đến khi cây được vun đống chỉ còn 1 nút thì dừng lại. Cuối cùng, ta đã có dãy đã được sắp theo thứ tự tăng dần.

* Thuật toán Adjust (A,r,n)

- Ta so sánh nôi dung của 2 nút con tìm ra nôi dung của nút con lớn hơn
- So sánh tiếp nôi dung của nút con lớn hơn với nôi dung của nút gốc :
- + Nếu nôi dung của nút con đó mà nhỏ hơn nôi dung của nút gốc thì cây con đó là đúng.

168

- + Nếu nôi dung của nút con đó mà lớn hơn nôi dung của nút gốc thì dịch chuyển nôi dung của nút con lên vị trí của nút gốc r. Sau đó, điều chỉnh tiếp cây có gốc ở vị trí nút là nút con lớn hơn đó.

b) Cài đặt:

```
void Adjust(int A[], int r, int n)
{
    int j=2*r+1; // vị trí nút con bên trái
    int x=A[r];
    int cont=TRUE;
    while (j<=n-1 && cont)
    {
        if (j<n-1) // lúc này r mới có nút con bên phải. Nếu j=n-1 thì r không
        // có nút con bên phải thì không cần so sánh
        if (A[j] < A[j+1]) // tìm vị trí nút con lớn hơn
            j++;
        if (A[j] <=x)
            cont=FALSE;
        else
        { // di chuyển nút con j lên r
            A[r]= A[j];
            r=j ; // xem lại nút con có phải là đúng không
            j=2*r+1;
        }
    }
    A[r]=x;
}
```

```

void Heap_sort(int A[], int n)
{
    int i, temp;
    for (i=n/2-1; i>=0; i--) // Tao heap ban dau
        Adjust(A, i,n);
    for (i=n-2; i>=0; i--)
    {
        temp= A[0]; // Cho A[0] ve cuoi heap
        A[0] = A[i+1];
        A[i+1] = temp;
        Adjust(A,0,i+1); // Dieu chinh lai heap tai vi tri 0
        // Luc nay, 2 cay con o vi tri 1 va 2 da la heap
    }
}

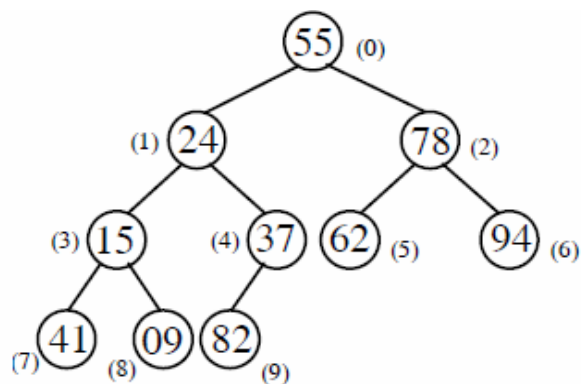
```

c) Ví dụ : Dùng thuật toán heap sort để sắp xếp dãy số sau theo chiều tăng dần:

55 24 78 15 37 62 94 41 09 82
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)

$n = 10$;

* Cây nhị phân ban đầu của dãy là :



Vun cây ban đầu thành đúng bằng gọi thuật Adjust

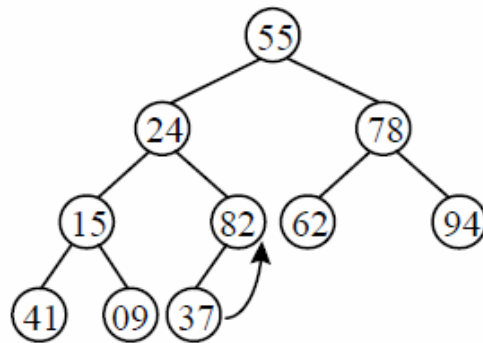
* Vun đúng : Vun từ dưới lên, trong cây chỉ có các nút 0, 1, 2, ..., $n/2-1=4$

mà là các nút gốc

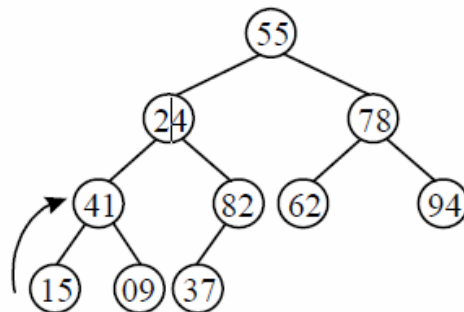
for (i=n/2-1; i>=0; i--) // Tao heap ban đầu

Adjust(A, i, n);

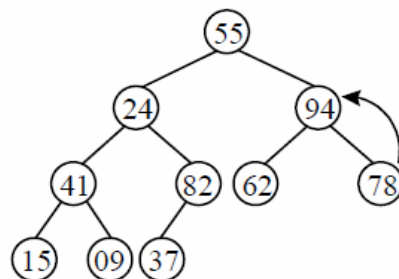
+ Vun cây có g c là $n/2 - 1 = 4$



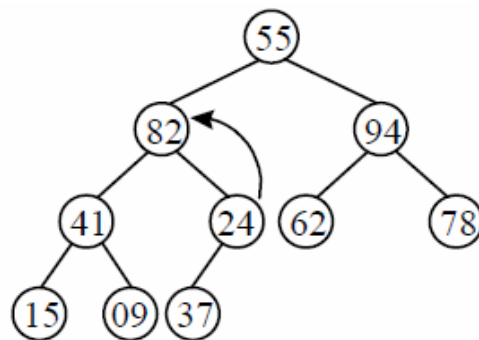
+ Vun cây có g c là $n/2 - 2 = 3$



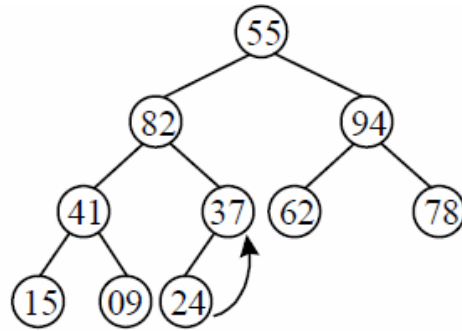
+ Vun cây có g c là $n/2 - 2 = 2$



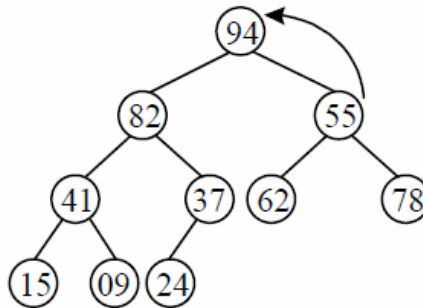
+ Vun cây có g c là $n/2 - 2 = 1$



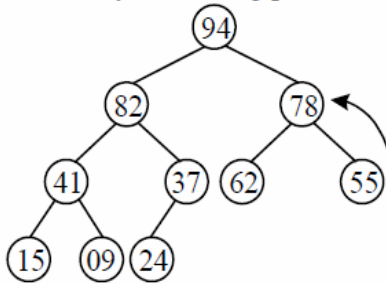
Do sau khi hoán đổi thì cây con không phải là đống nên ta Adjust lại:



+ Vun cây có gốc là $n/2 - 2 = 0$



Do sau khi hoán đổi thì cây con không phải là đống nên ta Adjust lại:



- Ta đổi chỗ $A[0]$ với $A[n-1]$ và sau đó Adjust lại cây ở vị trí 0 với số nút giảm đi 1, và quá trình trên sẽ dừng lại khi số nút bằng 0.

```
for (i=n-2; i>=0; i--)
```

```
{
```

```
temp= A[0]; // Cho A[0] về cuối heap
```

```
A[0] = A[i+1];
```

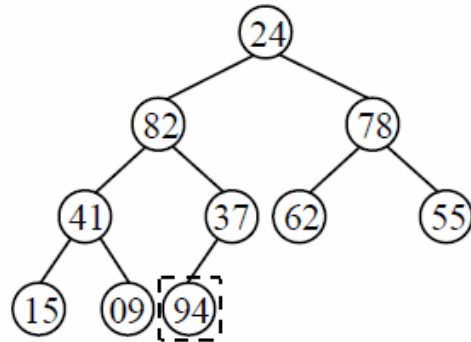
```
A[i+1] = temp;
```

```
Adjust(A,0,i+1); // Điều chỉnh lại heap tại vị trí 0
```

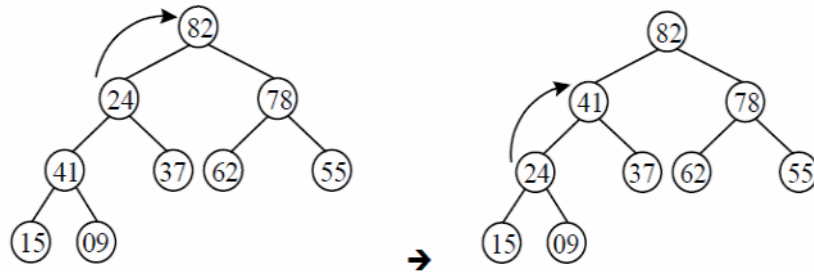
// Lúc này, 2 cây con ở vị trí 1 và 2 đã là heap

}

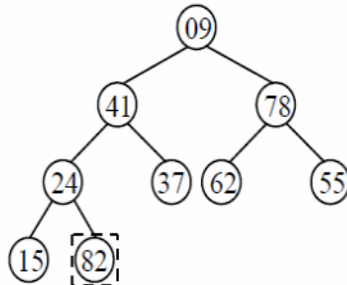
+ $i = n-2 = 8$: Đợi chờ $A[0]$ với $A[n-1]$ thì dãy $\{A[n-1]\}$ đã sắp xếp.



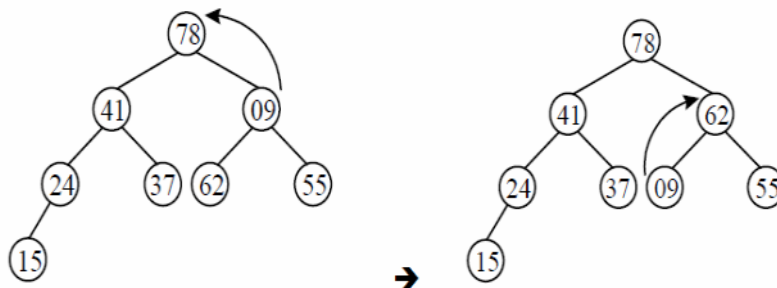
Sau đó, ta lấy vụn cây có gốc là 0 và số nút là $n-1 = 9$ thành đống.



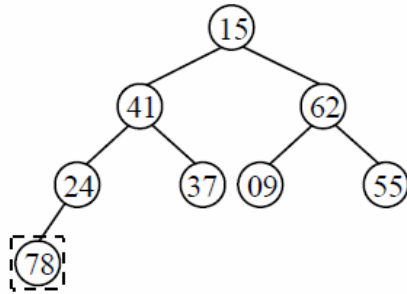
+ $i = 7$: Đợi chờ $A[0]$ với $A[n-2]$ thì dãy $\{A[n-2], A[n-1]\}$ đã sắp xếp.



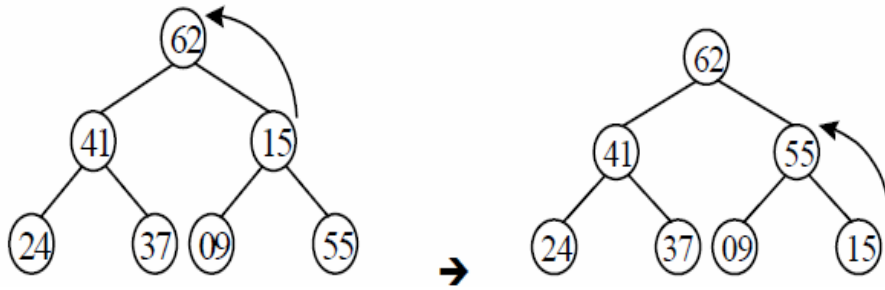
Sau đó, ta lấy vụn cây có gốc là 0 và số nút là $n-2=8$ thành đống.



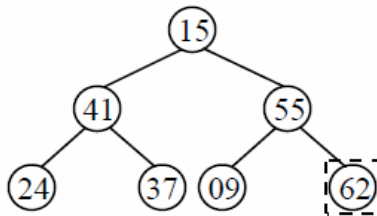
+ i = 6: Đòi hỏi chèn A[0] vào i A[n-3] thì dãy {A[n-3], A[n-2], A[n-1]} đã sắp xếp.



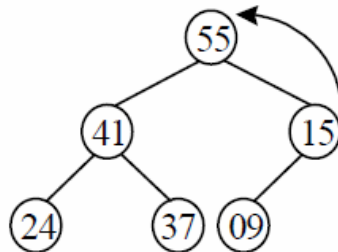
Sau đó, ta làm vụn cây có gốc là 0 và số nút là $n-3=7$ thành đúng.



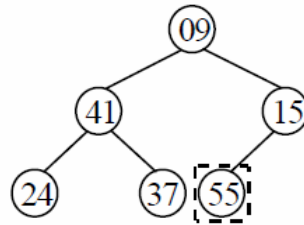
+ i = 5: Đòi hỏi chèn A[0] vào i A[n-4] thì dãy {A[n-4], A[n-3], A[n-2], A[n-1]} đã sắp xếp.



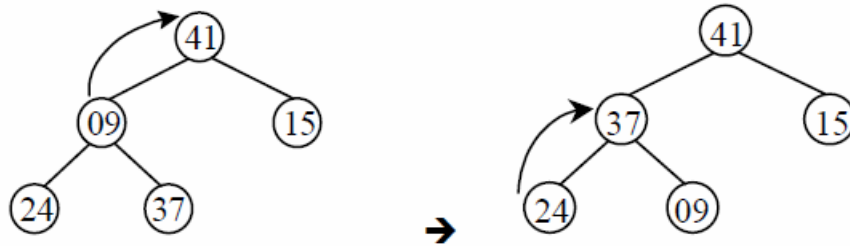
Sau đó, ta làm vụn cây có gốc là 0 và số nút là $n-4=6$ thành đúng.



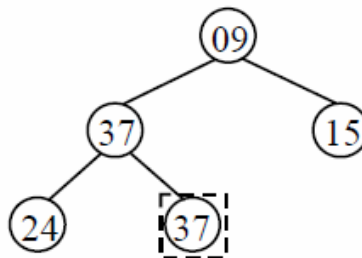
+ i = 4: Đòi hỏi chèn A[0] vào i A[n-5] thì dãy {A[n-5], A[n-4], A[n-3], A[n-2], A[n-1]} đã sắp xếp.



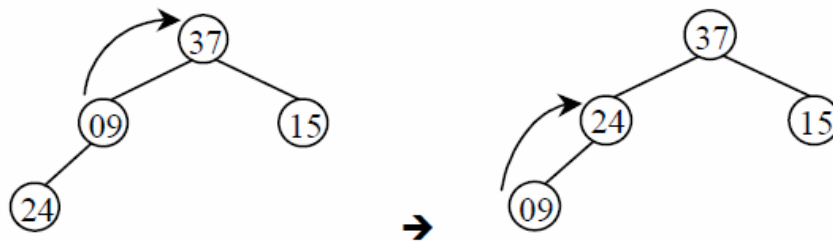
Sau đó, ta lồi vun cây có gốc là 0 và số nút là $n-5=5$ thành đồ ng.



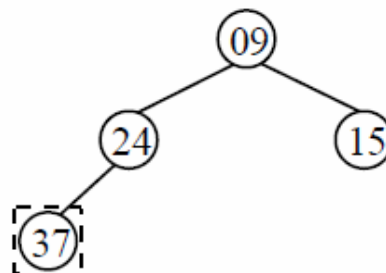
+ $i = 3$: Đ i ch $A[0]$ v i $A[n-6]$ thì dãy $\{A[n-6], A[n-5], A[n-4], A[n-3], A[n-2], A[n-1]\}$ đã s p x p.



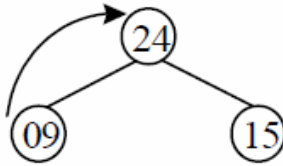
Sau đó, ta lồi vun cây có gốc là 0 và số nút là $n-6=4$ thành đồ ng.



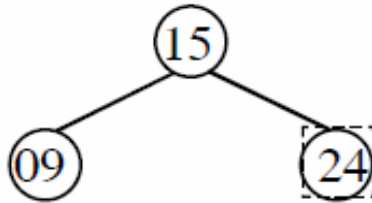
+ $i = 2$: Đ i ch $A[0]$ v i $A[n-7]$ thì dãy $\{A[n-7], A[n-6], A[n-5], A[n-4], A[n-3], A[n-2], A[n-1]\}$ đã s p x p.



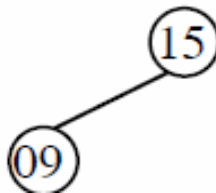
Sau đó, ta lùi vun cây có gốc là 0 và số nút là $n-7=3$ thành đúng.



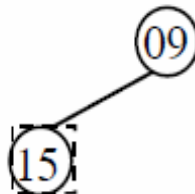
+ i = 1: Đợi ch $A[0]$ với $A[n-8]$ thì dãy $\{A[n-8], A[n-7], A[n-6], A[n-5], A[n-4], A[n-3], A[n-2], A[n-1]\}$ đã sắp xếp.



Sau đó, ta lùi vun cây có gốc là 0 và số nút là $n-7=3$ thành đúng.



+ i = 0: Đợi ch $A[0]$ với $A[n-9]$ thì dãy $\{A[n-9], A[n-8], A[n-7], A[n-6], A[n-5], A[n-4], A[n-3], A[n-2], A[n-1]\}$ đã sắp xếp.



Cuối cùng, dãy đã được sắp xếp theo thứ tự tăng dần.

d) Phân tích đánh giá

Đối với HEAP_SORT ta chú ý tới việc đánh giá trong trường hợp xấu nhất. Như ta đã biết: một cây nhị phân hoàn chỉnh có n nút thì chiều cao của cây đó là $\lceil \log_2(n+1) \rceil$. Khi đó cũng như khi vun lùi đúng trong giai đoạn sắp xếp, trường hợp xấu nhất thì số lượng phép so sánh cũng chính là chiều cao của cây. Do đó có thể suy ra, trong trường hợp xấu nhất, độ phức tạp thời gian thực hiện HEAP_SORT chính là $O(n \log_2 n)$. Việc đánh giá thời gian

thức hiện trung bình phức tạp hơn, ta không xét tới mà chỉ ghi nhận một kết quả đã được chứng minh là: cấp độ lớn của thời gian thức hiện trung bình giải thuật HEAP_SORT cũng là $O(n \log_2 n)$.

Có thể nhận xét thêm là: QUICK-SORT còn phải dùng thêm không gian nhớ cho stack, để lưu trữ thông tin về các phân đoạn sắp xếp tiếp theo (vì thức hiện đệ quy). Còn HEAP_SORT thì ngoài một nút nhớ phụ, để thức hiện đệ quy, nó không cần thêm gì nữa.

***Kết luận chung:**

Ta nhận thấy, với cùng một mục đích sắp xếp mà có rất nhiều phương pháp và kỹ thuật giải quyết khác nhau. Cấu trúc dữ liệu được lựa chọn để mô tả bài toán sắp xếp đã ảnh hưởng rất lớn tới việc lựa chọn giải thuật sắp xếp thích hợp. Các giải thuật sắp xếp được phân thành ba kỹ thuật cơ sở của sắp xếp (dựa vào phép so sánh các giá trị khóa). Tuy nhiên, cấp độ lớn của thời gian thức hiện chúng là (n^2) , do đó chúng nên sử dụng chúng khi cần thiết. Các giải thuật cải tiến như Quick_Sort, Heap_Sort đã đạt được thời gian thức hiện tốt hơn là $O(n \log_2 n)$, thời gian được sử dụng khi n lớn. Nếu dãy khóa cần sắp xếp vẫn có khuynh hướng hữu hạn đã được sắp xếp sẵn rồi thì Quick Sort lại không nên dùng. Ngược lại nếu ban đầu dãy có khuynh hướng ít nhiều có thể tăng dần về phía cuối thì Heap Sort lại tỏ ra thuận lợi. Việc không định một kỹ thuật sắp xếp nào nói trên là tốt hơn các phương pháp khác là không nên, mà việc chọn một phương pháp sắp xếp thích hợp thì tùy thuộc vào từng yêu cầu, từng điều kiện của bài toán.

Bài tập chương 5

1. Dãy cây nhị phân bất kỳ các đỉnh khi duyệt theo

a. Thứ tự trước: A D F G H K L P Q R W Z

Thứ tự giữa: G F H K D L A W R Q P Z

b. Theo thứ tự sau: F G H D A L P Q R Z W K

Thứ tự giữa: G F H K D L A W R Q P Z

2. Với mỗi biểu thức số học dưới đây, hãy vẽ cây nhị phân biểu diễn biểu thức ấy, rồi dùng các kỹ thuật để tìm biểu thức tiền tố và hậu tố tương ứng.

a. $a / (b - (c - (d - (e - f))))$

b. $((a * (b + c)) / (d - (e + f))) * (g / (h / (i * j)))$

3. Hãy trình bày các bước sau

a. Định nghĩa và đặc điểm của cây nhị phân tìm kiếm

b. Thao tác tìm kiếm nút trong cây

c. Hình thức của cây là gì?

4. Xét giải thuật tạo cây nhị phân tìm kiếm. Nếu thêm các khoá nhập vào là như sau:

8 3 5 2 20 11 30 9 18 4

thì hình ảnh cây được tạo như thế nào?

Sau đó, nếu huỷ bỏ nút các nút theo thứ tự như sau:

11 20 8

thì cây sẽ thay đổi như thế nào trong tình huống huỷ, vẽ cây minh hoạ qua tình huống.

5. Áp dụng thuật giải tạo cây AVL để dựng cây với thứ tự các khoá nhập vào như sau:

5 7 2 1 3 6 10

thì hình ảnh cây tạo được như thế nào? Giải thích rõ tình huống xảy ra khi thêm từng khoá vào cây và vẽ hình minh hoạ.

Sau đó, nếu huỷ bỏ nút các nút theo thứ tự như sau :

5 6 7

thì cây sẽ thay đổi như thế nào trong tình huống huỷ, vẽ sơ đồ và giải thích.

6. Vẽ các hàm (thuật toán) xác định các thông tin của cây nhị phân T

- Số nút lá

- Số nút có đúng một cây con
- Số nút có đúng hai cây con
- Số nút có khóa nhỏ hơn x (giả sử T là CNPTK)
- Số nút có khóa lớn hơn x (giả sử T là CNPTK)
- Chiều cao của cây
- In ra tất cả các nút thuộc thành phần của cây
- Kiểm tra xem T có phải là cây cân bằng hoàn toàn không

7. Xây dựng cấu trúc dữ liệu biểu diễn cây n- phân và tạo sinh cây nhị phân thông qua việc vẽ các khóa của cây n - phân.

Giả sử khóa được lưu trữ chiếm k byte, mỗi con trỏ chiếm 4 byte, vậy dùng cây nhị phân thay cho cây n- phân thì có lợi gì trong việc lưu trữ các khóa.

8. Viết hàm chuyển một cây n- phân thành cây nhị phân.

9. Hãy tìm một ví dụ về một cây AVL có chiều cao là 6 và khi huỷ một nút lá (chỉ có một) việc cân bằng lại lan truyền lên tận gốc của cây. Vẽ ra bảng biến đổi quá trình huỷ và cân bằng lại này.

10. Cài đặt chương trình mô phỏng thực quan các thao tác trên cây nhị phân tìm kiếm.

11. Cài đặt chương trình mô phỏng thực quan các thao tác trên cây AVL

Viết chương trình cho phép tạo, tra cứu và sửa chữa tệp dữ liệu Anh - Việt