CHƯƠNG I: KỸ THUẬT PHÂN TÍCH GIẢI THUẬT

1. Mục tiêu
2. Kiến thức cơ bản cần có để học chương này
3. Tài liệu tham khảo có liên quan đến chương
4. Nội dung:
I.1 - Sự cần thiết phải phân tích giải thuật.
I.2 - Thời gian thực hiện của giải thuật.
I.3 - Tỷ suất tăng và độ phức tạp của giải thuật.
I.4 - Cách tính độ phức tạp.
I.5 - Phân tích các chương trình đệ quy.
5. Vấn đề nghiên cứu của trang kế tiếp

Trong chương này chúng ta sẽ nghiên cứu các vấn đề sau:

· Sự cần thiết phải phân tích các giải thuật.

· Thời gian thực hiện của chương trình.

· Tỷ suất tăng và độ phức tạp của giải thuật.

· Tính thời gian thực hiện của chương trình.

· Phân tích các chương trình đệ quy.

I.1- SỰ CẦN THIẾT PHẢI PHÂN TÍCH GIẢI THUẬT

Trong khi giải một bài toán chúng ta có thể có một số giải thuật khác nhau, vấn đề là cần phải đánh giá các giải thuật đó để lựa chọn một giải thuật tốt (nhất). Thông thường thì ta sẽ căn cứ vào các tiêu chuẩn sau:

1.- Giải thuật đúng đắn.

2.- Giải thuật đơn giản.

3.- Giải thuật thực hiện nhanh.

Với yêu cầu (1), để kiểm tra tính đúng đắn của giải thuật chúng ta có thể cài đặt giải thuật đó và cho thực hiện trên máy với một số bộ dữ liệu mẫu rồi lấy kết quả thu được so sánh với kết quả đã biết. Thực ra thì cách làm này không chắc chắn bởi vì có thể giải thuật đúng với tất cả các bộ dữ liệu chúng ta đã thử nhưng lại sai với một bộ dữ liệu nào đó. Vả lại cách làm này chỉ phát hiện ra giải thuật sai chứ chưa chứng minh được là nó đúng. Tính đúng đắn của giải thuật cần phải được chứng minh bằng toán học. Tất nhiên điều này không đơn giản và do vậy chúng ta sẽ không đề cập đến ở đây.

Khi chúng ta viết một chương trình để sử dụng một vài lần thì yêu cầu (2) là quan trọng nhất. Chúng ta cần một giải thuật dễ viết chương trình để nhanh chóng có được kết qủa , thời gian thực hiện chương trình không được đề cao vì dù sao thì chương trình đó cũng chỉ sử dụng một vài lần mà thôi.

Tuy nhiên khi một chương trình được sử dụng nhiều lần thì thì yêu cầu tiếit kiệm thời gian

thực hiện chương trình lại rất quan trọng đặc biệt đối với những chương trình mà khi thực hiện cần dữ liệu nhập lớn do đó yêu cầu (3) sẽ được xem xét một cách kỹ càng. Ta gọi nó là hiệu quả thời gian thực hiện của giải thuật.

I.2- THỜI GIAN THỰC HIỆN CỦA GIẢI THUẬT

I.2.1- Thời gian thực hiện chương trình.
I.2.2- Đơn vị đo thời gian thực hiện.
I.2.3- Thời gian thực hiện trong trường hợp xấu nhất.

Một phương pháp để xác định hiệu quả thời gian thực hiện của một giải thuật là lập trình nó và đo lường thời gian thực hiện của hoạt động trên một máy tính xác định đối với tập hợp được chọn lọc các dữ liệu vào.

Thời gian thực hiện không chỉ phụ thuộc vào giải thuật mà còn phụ thuộc váo tập các chỉ thị của máy tính, chất lượng của máy tính và kỹ xảo của người lập trình. Sự thi hành cũng có thể điều chỉnh để thực hiện tốt trên tập đặc biệt các dữ liệu vào được chọn. Để vượt qua các trở ngại này, các nhà khoa học máy tính đã chấp nhận tính phức tạp của thời gian được tiếp cận như một sự đo lường cơ bản sự thực thi của giải thuật. Thuật ngữ tính hiệu quả sẽ đề cập đến sự đo lường này và đặc biệt đối với sự phức tạp thời gian trong trường hợp xấu nhất.

I.2.1- Thời gian thực hiện chương trình.

Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu T(n) trong đó n là kích thước (độ lớn) của dữ liệu vào.

Ví dụ 1-1: Chương trình tính tổng của n số có thời gian thực hiện là T(n) = cn trong đó c là một hằng số.

Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0 \ \forall n \geq 0$.

I.2.2- Đơn vị đo thời gian thực hiện.

Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây... mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.

Ví dụ 1-2: Khi ta nói thời gian thực hiện của một chương trình là T(n) = cn thì có nghĩa là chương trình ấy cần cn chỉ thị thực thi.

I.2.3- Thời gian thực hiện trong trường hợp xấu nhất.

Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi ta cho vào dãy chưa có thứ tự, hoặc khi ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi ta cho vào một dãy đã có thứ tự giảm.

Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thuớc n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

I.3- TỶ SUẤT TĂNG VÀ ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

I.3.1- Tỷ suất tăng

Ta nói rằng hàm không âm T(n) có **tỷ suất tăng (growth rate)** f(n) nếu tồn tại các hằng số c và n_0 sao cho $T(n) \le f(n)$ với mọi $n \ge n_0$.

Ta có thể chứng minh được rằng "**Cho một hàm không âm T(n) bất kỳ, ta luôn tìm được tỷ suất tăng f(n) của nó**".

Ví dụ 1-3: Giả sử T(0) = 1, T(1) = 4 và tổng quát $T(n) = (n+1)^2$. Đặt $n_0 = 1$ và c = 4 thì với mọi $n \ge 1$ chúng ta dễ dàng chứng minh rằng $T(n) = (n+1)^2 \le 4n^2$ với mọi $n \ge 1$, tức là tỷ suất tăng của T(n) là n^2.

Ví dụ 1-4: Tỷ suất tăng của hàm $T(n) = 3n^3 + 2n^2$ là n^3. Thực vậy, cho $n_0 = 0$ và c = 5 ta dễ dàng chứng minh rằng với mọi $n \ge 0$ thì $3n^3 + 2n^2 \le 5n^3$

I.3.2- Khái niệm độ phức tạp của giải thuật

Giả sử ta có hai giải thuật P1 và P2 với thời gian thực hiện tương ứng là $T1(n) = 100n^2$ (với tỷ suất tăng là n^2) và $T2(n) = 5n^3$ (với tỷ suất tăng là n^3). Giải thuật nào sẽ thực hiện nhanh hơn? Câu trả lời phụ thuộc vào kích thước dữ liệu vào. Với n < 20 thì P2 sẽ nhanh hơn P1 (T2<T1), do hệ số của $5n^3$ nhỏ hơn hệ số của $100n^2$ (5<100). Nhưng khi n > 20 thì ngược lại do số mũ của $100n^2$ nhỏ hơn số mũ của $5n^3$ (2<3). Ở đây chúng ta chỉ nên quan tâm đến trường hợp n>20 vì khi n<20 thì thời gian thực hiện của cả P1 và P2 đều không lớn và sự khác biệt giữa T1 và T2 là không đáng kể..

Như vậy một cách hợp lý là ta xét tỷ suất tăng của hàm thời gian thực hiện chương trình thay vì xét chính bản thân thời gian thực hiện.

Cho một hàm T(n), T(n) gọi là **có độ phức tạp** f(n) nếu tồn tại các hằng c, N_0 sao cho $T(n) \le cf(n)$ với mọi $n \ge N_0$ (tức là T(n) có tỷ suất tăng là f(n)) và kí hiệu T(n) là O(f(n)) (đọc là "ô của f(n)")

Ví dụ 1-5: $T(n)= (n+1)^2$ có tỷ suất tăng là n^2 nên $T(n)= (n+1)^2$ là $O(n^2)$

Chú ý: O(c.f(n))=O(f(n)) với c là hằng số. Đặc biệt O(c)=O(1)

Nói cách khác độ phức tạp tính toán của giải thuật là một hàm chặn trên của hàm thời gian. Vì hằng nhân tử c trong hàm chặn trên không có ý nghĩa nên ta có thể bỏ qua vì vậy hàm thể hiện độ phức tạp có các dạng thường gặp sau: $\mathbf{log_2 n, n, n log_2 n, n^2, n^3, 2^2, n!, n^n}$. Ba hàm cuối cùng ta gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức. Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

Khi nói đến độ phức tạp của giải thuật là ta muốn nói đến hiệu quả của thời gian thực hiện của chương trình nên ta có thể xem việc xác định thời gian thực hiên của chương trình chính là xác định độ phức tạp của giải thuật.

I.4- CÁCH TÍNH ĐỘ PHỨC TẠP

I.4.3- Qui tắc tổng quát để phân tích một chương trình
I.4.4- Độ phức tạp của chương trình có gọi chương trình con không đệ qui

Cách tính độ phức tạp của một giải thuật bất kỳ là một vấn đề không đơn giản. Tuy nhiên ta có thể tuân theo một số nguyên tắc sau:

I.4.1- Qui tắc cộng:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n) thì thời gian thực hiện của đoạn hai chương trình đó **nối tiếp nhau** là T(n)=O(**max(f(n),g(n))**)

Ví dụ 1-6: Lệnh gán x:=15 tốn một hằng thời gian hay O(1)

Lệnh đọc dữ liệu READ(x) tốn một hằng thời gian hay O(1)

Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1))=O(1)

I.4.2- Qui tắc nhân:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n) thì thời gian thực hiện của đoạn hai đoạn chương trình đó **lồng nhau** là T(n) = O(**f(n).g(n)**)

I.4.3- Qui tắc tổng quát để phân tích một chương trình:

- Thời gian thực hiện của mỗi lệnh gán, READ, WRITE là O(1)

- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.

- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau THEN hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).

- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện than vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Ví dụ 1-7: Tính thời gian thực hiện của đoạn chương trình

```
procedure Bubble (var a: array[1..n] of integer);
var i,j,temp: integer;
begin
        {1} for i:=1 to n-1 do
        {2} for j:=n downto i+1 do
        {3} if a[j-1]>a[j] then begin
        { đổi chổ a[i], a[j] }
        {4} temp:=a[j-1];
        {5} a[j-1]:=a[j];
        {6} a[j]:=temp; end;
end;
```

Cả ba lệnh đổi chỗ {4} {5} {6} tốn O(1) thời gian, do đó lệnh {3} tốn O(1).

Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1)=O(n-i).

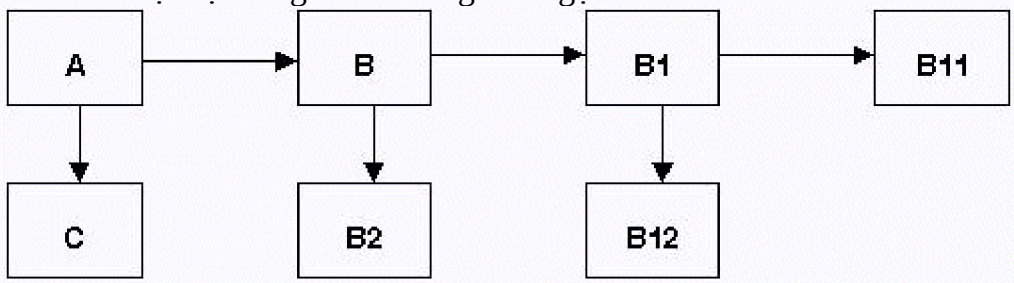
Vòng lặp {1} lặp (n-1) lần vậy độ phức tạp của giải thuật là:

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$$

I.4.4- Độ phức tạp của chương trình có gọi chương trình con không đệ qui

Nếu chúng ta có một chương trình với các chương trình con không đệ quy, để tính thời gian thực hiện của chương trình, trước hết chúng ta tính thời gian thực hiện của các chương trình con không gọi các chương trình con khác. Sau đó chúng ta tính thời gian thực hiện của các chương trình con chỉ gọi các chương trình con mà thời gian thực hiện của chúng đã được tính. Chúng ta tiếp tục quá trình đánh giá thời gian thực hiện của mỗi chương trình con sau khi thời gian thực hiện của tất cả các chương trình con mà nó gọi đã được đánh giá. Cuối cùng ta tính thời gian cho chương trình chính.

Giả sử ta cô một hệ thống các chương trình gọi theo sơ đồ sau:



Hình 1-1: Sơ đồ gọi thực hiện các chương trình con không đệ quy

Chương trình A gọi hai chương trình con là B và C, chương trình B gọi hai chương trình con là B1 và B2, chương trình B1 gọi hai chương trình con là B11 và B12.

Để tính thời gian thực hiện của A, ta tính theo các bước sau:

- Tính thời gian thực hiện của C, B2, B11 và B12.

- Tính thời gian thực hiện của B1.

- Tính thời gian thực hiện của B.

- Tính thời gian thực hiện của A.

Ví dụ 1-8: Ta có thể viết lại chương trình sắp xếp bubble như sau:

```
procedure Swap (var x, y: integer);
var temp: integer;
begin
        temp := x;
        x := y;
        y := temp;
end;
```

```
procedure Bubble (var a: array[1..n] of integer);
var i,j :integer;
begin
        {1} for i:=1 to n-1 do
        {2} for j:=n downto i+1 do
        {3} if a[j-1]>a[j] then Swap(a[j-1], a[j]);
end;
```

Trong cách viết trên, chương trình Bubble gọi chương trình con Swap, do đó để tính thời gian thực hiện của Bubble, trước hết ta cần tính thời gian thực hiện của Swap. Dễ thấy thời gian thực hiện của Swap là O(1) vì nó chỉ bao gồm 3 lệnh gán.

Trong Bubble, lệnh {3} gọi Swap nên chỉ tốn O(1), lệnh {2} thực hiện n-i lần, mỗi lần tốn O(1) nên tốn O(n-i). Lệnh {1} thực hiện n-1 lần nên

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$$

I.5- PHÂN TÍCH CÁC CHƯƠNG TRÌNH ĐỆ QUY

I.5.1- Thành lập phương trình đệ quy
I.5.2- Giải phương trình đệ quy

Với các chương trình có gọi các chương trình con đệ quy, ta không thể áp dụng cách tính như vừa trình bày trong mục I.4.4 bởi vì một chương trình đệ quy sẽ gọi chính bản thân nó.

Với các chương trình đệ quy, trước hết ta cần thành lập các phương trình đệ quy, sau đó giải phương trình đệ quy, nghiệm của phương trình đệ quy sẽ là thời gian thực hiện của chương trình đệ quy.

I.5.1- Thành lập phương trình đệ quy

Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) là thời gian thực hiện chương trình với kích thước dữ liệu nhập là n, T(k) thời gian thực hiện chương trình với kích thước dữ liệu nhập là k, với k < n. Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.

Ví dụ 1-9: Xét hàm tính giai thừa viết bằng giải thuật đệ quy như sau:

```
function Giai_thua(n:integer): integer;
begin
        if n=0 then Giai_thua :=1
        else Giai_thua := n* Giai_thua(n-1);
end;
```

Gọi T(n) là thời gian thực hiện việc tính n giai thừa, thì T(n-1) là thời gian thực hiện việc tính n-1 giai thừa. Trong trường hợp n=0 thì chương trình chỉ thực hiện một lệnh gán Giai_thua:=1, nên tốn O(1), do đó ta có T(0) = C1. Trong trường hợp n>0 chương trình phải gọi đệ quy Giai_thua(n-1), việc gọi đệ quy này tốn T(n-1), sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và gán cho Giai_thua. Thời gian để thực hiện phép nhân và phép gán là một hằng C2. Vậy ta có

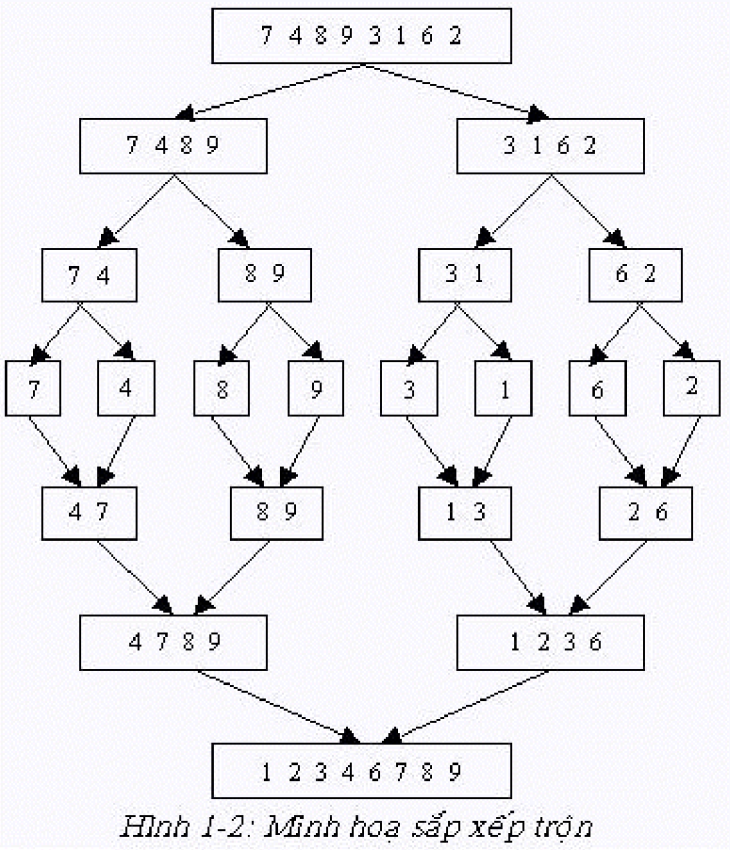
$$T(n) = \begin{cases} C1 & \text{nếu } n = 0 \\ T(n-1) + C2 & \text{nếu } n > 0 \end{cases}$$

Đây là phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy Giai_thua.

Ví dụ 1-10: Chúng ta xét thủ tục MergeSort một cách phác thảo như sau:

```
function MergeSort (L:List ; n:integer) : List;
var
        L1,L2 : List;
begin
        if n = 1 then return(L)
        else
        begin
        Chia L thành 2 nửa L1 và L2 , mỗi một nửa có độ dài n/2;
        return(Merge(MergeSort (L1 , n/2), MergeSort(L2, n/2)));
        end;
end;
```

Chẳng hạn để sắp xếp danh sách L gồm 8 phần tử 7, 4, 8, 9, 3, 1, 6, 2 ta có mô hình minh họa của MergeSort như sau:



Hình 1-2: Minh hoạ sắp xếp trộn

Hàm MergeSort nhận một danh sách có độ dài n và trả về một danh sách đã được sắp xếp. Thủ tục Merge nhận hai danh sách đã được sắp L1 và L2 mỗi danh sách có độ dài n/2, trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự. Giải thuật chi tiết của Merge ta sẽ bàn sau, chúng ta chỉ để ý rằng thời gian để Merge các danh sách có độ dài n/2 là O(n).

Gọi T(n) là thời gian thực hiện MergeSort một danh sách n phần tử thì T(n/2) là thời gian thực hiện MergeSort một danh sách n/2 phần tử , ta có thể viết phương trình đệ quy như sau:

$$T(n) = \begin{cases} c_1 & \text{nếu } n = 1 \\ 2T(n/2) + c_2 n & \text{nếu } n > 1 \end{cases}$$

Trong đó c1 là thời gian phải tốn khi L có độ dài 1. Trong trường hợp n > 1, thời gian của MergeSort được chia làm hai phần. Phần gọi đệ quy MergeSort một danh sách có độ dài n/2 là T(n/2) do đó ta có 2T(n/2). Phần thứ hai bao gồm phép thử n >1, chia danh sách L thành hai nửa bằng nhau và Merge. Ba thao tác này lấy thời gian không đổi đối với phép thử hoặc tỷ lệ với n đối với ngắt và Merge. Như vậy hằng c_2 được chọn và c_2n là thời gian tổng để làm các việc đó ngoại trừ gọi đệ quy.

I.5.2- Giải phương trình đệ quy

Có ba phương pháp giải phương trình đệ quy:

1.- Phương pháp truy hồi

2.- Phương pháp đoán nghiệm.

3.- Lời giải tổng quát của một lớp các phương trình đệ quy.

Phương pháp truy hồi

Dùng đệ quy để thay thế bất kỳ T(m) với m < n vào phía phải của phương trình cho đến khi tất cả T(m) với m > 1 được thay thế bởi biểu thức của các T(1). Vì T(1) luôn là hằng nên chúng ta có công thức của T(n) chứa các số hạng chỉ liên quan đến n và các hằng số.
Giải phương trình.

Ví dụ 1-10: Giải phương trình:

$$T(n) = \begin{cases} c_1 & \text{nếu } n = 1 \\ 2T(n/2) + c_2 n & \text{nếu } n > 1 \end{cases}$$

Ta có:

$$T(n) = 2T(\frac{n}{2}) + C_2 n$$

$$T(n) = 2[2T(\frac{n}{4}) + C_2 \frac{n}{2}] + C_2 n = 4T(\frac{n}{4}) + 2C_2 n$$

$$T(n) = 4[2T(\frac{n}{8}) + C_2 \frac{n}{4}] + 2C_2 n = 8T(\frac{n}{8}) + 3C_2 n$$

$$T(n) = 2^i T(\frac{n}{2^i}) + iC_2 n$$

Giả sử $n = 2^k$, quá trình suy rộng sẽ kết thúc khi i =k, khi đó ta có:

$$T(n) = 2^k T(1) + kC_2 n$$

Vì $2^k = n$ nên k = logn và với $T(1) = C_1$ nên ta có

$T(n) = C_1n + C_2nlogn$

Hay T(n) là O(nlogn).

Đoán nghiệm

Ta đoán một nghiệm f(n) và dùng chứng minh quy nạp để chứng tỏ rằng $T(n) \leq f(n)$ với mọi n.

Thông thường f(n) là một trong các hàm quen thuộc như **logn, n, nlogn, n^2, n^3, 2^n, n!, n^n**.

Đôi khi chúng ta chỉ đoán dạng của f(n) trong đó có một vài tham số chưa xác định (chẳng hạn $f(n) = an^2$ với a chưa xác định) và trong quá trình chứng minh quy nạp ta sẽ suy diễn ra giá trị thích hợp của các tham số.

Ví dụ 1-11: Giải phương trình đệ quy

$$T(n) = \begin{cases} c_1 & \text{nếu } n = 1 \\ 2T(n/2) + c_2 n & \text{nếu } n > 1 \end{cases}$$

Giả sử chúng ta đoán f(n) = anlogn. Với n = 1 ta thấy rằng cách đoán như vậy không được bởi vì anlog n có giá trị 0 không phụ thuộc vào giá trị của a. Vì thế ta thử tiếp theo f(n) = anlogn + b.

Với n = 1 ta có, $T(1) = C_1$ và f(1) = b, muốn $T(1) \leq f(1)$ thì $b \geq C_1$ (*)

Giả sử rằng $T(k) \leq aklogk + b$ với mọi k < n (I.2).Ta sẽ chứng minh $T(n) \leq anlogn + b$

Giả sử $n \geq 2$, từ (I.1) ta có $T(n) = 2T(n/2) + C_2n$

Áp dụng (I.2) với k = n/2 < n ta có:

$T(n) = 2T(n/2) + C_2n \leq 2[an/2log(n/2) + b] + C_2n$

$T(n) \leq anlogn - an + 2b + C_2n$

$T(n) \leq (anlogn + b) + [b + (C_2 - a)n]$. Nếu lấy $a \geq C_2 + b$ (**) ta được

$T(n) \leq (anlogn + b) + [b +(C_2 - C_2 - b)n]$

$T(n) \leq (anlogn + b) + (1-n) b$

$T(n) \leq an\ logn + b.$

Nếu ta lấy a và b sao cho cả (*) và (**) đều thoả mãn thì $T(n) \leq an\ logn + b$ với mọi n.

Ta giải hệ $\begin{cases} b \geq C_1 \\ a \geq C_2 + b \end{cases}$ Để đơn giản, ta giải hệ $\begin{cases} b = C_1 \\ a = C_2 + b \end{cases}$

Dễ dàng ta có $b = C_1$ và $a= C_1 +C_2$ ta được $T(n) \leq (C_1 + C_2)nlogn + C_1$ với mọi n.

Hay nói cách khác T(n) là O(nlogn).

Lời giải tổng quát cho một lớp các phương trình đệ quy

Để giải bài toán kích thước n, ta chia bài toán đã cho thành a bài toán con, mỗi bài tóan con có kích thước n/b. Giải các bài toán con này và tổng hợp kết quả lại để được kết quả của bài toán đã cho. Với các bài toán con chúng ta cũng làm như vậy. Kỹ thuật này sẽ dẫn chúng ta đến một chương trình đệ quy.

Giả thiết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian và thời gian để chia bài toán kích thước n thành các bài toán con kích thước n/b và tổng hợp kết quả từ các bài toán con để được lời giải của bài toán ban đầu là d(n). (Chẳng hạn đối với thí dụ MergeSort, chúng ta có a = b = 2, và $d(n) = C_2n/C_1$. Xem C_1 là một đơn vị).

Gọi T(n) là thời gian để giải bài toán kích thước n thì ta có phương trình đệ quy:

$$\begin{cases} T(1) = 1 \\ T(n) = aT(\frac{n}{b}) + d(n) \end{cases} \qquad (I.1)$$

Ta sử dụng phương pháp truy hồi để giải phương trình này

$$T(n) = aT(n/b) + d(n)$$

$$T(n) = a[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)] + d(n) = a^2T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = a^2[aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right)] + ad\left(\frac{n}{b}\right) + d(n) = a^3T\left(\frac{n}{b^3}\right) + a^2d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$= \cdots\cdots$$

$$= a^iT\left(\frac{n}{b^i}\right) + \sum_{i=0}^{i-1} a^id\left(\frac{n}{b^i}\right)$$

Giả sử $n = b^k$ ta được: $T(n/b^k) = T(1) = 1$. Thay vào trên với i = k ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^jd\left(b^{k-j}\right) \qquad (I.2)$$

Hàm tiến triển, nghiệm thuần nhất và nghiệm riêng

Trong phương trình đệ quy (I.1) hàm thời gian d(n) được gọi là **hàm tiến triển** (driving function)

Trong công thức (I.2), $a^k = n^{\log b^a}$ được gọi là **nghiệm thuần nhất** (homogeneous solutions).

Nghiệm thuần nhất là nghiệm chính xác khi d(n) = 0 với mọi n. Nói một cách khác, nghiệm thuần nhất biểu diễn thời gian để giải tất cả các bài toán con.

Trong công thức (I.2), $a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$ được gọi là **nghiệm riêng** (particular solutions).

Nghiệm riêng biểu diễn thời gian phải trả để tạo ra các bài toán con và tổng hợp các kết quả của chúng. Nhìn vào công thức ta thấy nghiệm riêng phụ thuộc vào hàm tiến triển, số lượng và kích thước các bài toán con.

Khi tìm nghiệm của phương trình (I,1), chúng ta phải tìm nghiệm riêng và so sánh với nghiệm thuần nhất. Nếu nghiệm nào lớn hơn, ta lấy nghiệm đó làm nghiệm của phương trình (I,1).

Việc xác định nghiệm riêng không đơn giản chút nào, tuy vậy, chúng ta cũng tìm được một lớp các hàm tiến triển có thể dễ dàng xác định nghiệm riêng.

Hàm nhân

Một hàm f(n) được gọi là **hàm nhân** (multiplicative function) nếu f(m.n) = f(m).f(n) với mọi số nguyên dương m và n.

Ví dụ 1-12: Hàm $f(n) = n^k$ là một hàm nhân, vì $f(m.n) = (m.n)^k = m^k.n^k = f(m)\ f(n)$

Nếu d(n) trong (I.1) là một hàm nhân thì theo tính chất của hàm nhân ta có

$d(b^{k-j}) = (d(b))^{k-j}$ và nghiệm riêng của (I.2) là:

$$\sum_{j=0}^{k-1} a^j d(b^{k-j}) = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j = d(b)^k \frac{\left[\frac{a}{d(b)}\right]^k - 1}{\frac{a}{d(b)} - 1} = \frac{a^k - d(b)^k}{\frac{a}{d(b)} - 1} \qquad (I.3)$$

Xét ba trường hợp sau:

1.- Nếu a > d(b) thì nghiệm riêng là $O(a^k) = O(n^{\log_b a})$. Như vậy nghiệm riêng và nghiệm thuần nhất bằng nhau do đó T(n) là $O(n^{\log_b a})$.

Ta cũng thấy thời gian thực hiện chỉ phụ thuộc vào a, b mà không phụ thuộc vào hàm tiến triển d(n). Vì vậy để cải tiến giải thuật ta cần giảm a hoặc tăng b.

2.- Nếu a < d(b) thì nghiệm riêng là $O(d(b)^k) = O(n^{\log_b d(b)})$. Trong trường hợp này nghiệm riêng lớn hơn nghiệm thuần nhất nên $T(n) = O(n^{\log_b d(b)})$.

Để cải tiến giải thuật chúng ta cần để ý đến cả d(n), a và b cùng mức độ như nhau.

Trường hợp đặc biệt quan trọng khi $d(n) = n^\alpha$. Khi đó $d(b) = b^\alpha$ và $\log_b(b^\alpha) = \alpha$. Vì thế nghiệm riêng là $O(n^\alpha)$ và do vậy T(n) là $O(n^\alpha)$.

3.- Nếu a = d(b) thì công thức (I.5) không xác đinh nên ta tính trực tiếp nghiệm riêng:

$$\text{Nghiệm riêng} = \sum_{j=0}^{k-1} a^j (d(b))^{k-j} = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right] = d(b)^k \sum_{j=0}^{k-1} 1 = d(b)^k k = n^{\log_b d(b)} \log_b n$$

Vì a= d(b) nên nghiệm riêng là $n^{\log_b a}\log_b n$ và nghiệm này lớn gấp $\log_b n$ lần nghiệm thuần nhất. Do đó $T(n) = O(n^{\log_b a}\log_b n)$.

Trong trường hợp đặc biệt $d(n) = n^\alpha$ ta được $T(n) = O(n^\alpha \log n)$.

Chú ý khi giải một phương trình đệ quy cụ thể, ta phải xem phương trình đó có thuộc dạng phương trình tổng quát hay không. Nếu có thì phải xét xem hàm tiến triển có phải là hàm nhân không. Nếu có thì ta xác định a, d(b) và dựa vào sự so sánh giữa a và d(b) mà vận dụng một trong ba trường hợp nói trên.

Ví dụ 1-13: Giải các phương trình đệ quy sau với T(1) = 1 và

1/- $T(n) = 4T(n/2) + n$

2/- $T(n) = 4T(n/2) + n^2$

3/- $T(n) = 4T(n/2) + n^3$

Trong mỗi trường hợp, a=4, b=2 và nghiệm thuần nhất là n^2. Với d(n) = n ta có d(b) = 2 vì a = 4 > d(b) nên nghiệm riêng cũng là n^2 và $T(n) = O(n^2)$ trong phương trình (1).

Trong phương trình (3), $d(n) = n^3$, d(b) = 8 và a < d(b). Vì vậy nghiệm riêng là $O(n^{\log_b d(b)}) = O(n^3)$ và T(n) của (3) là $O(n^3)$.

Trong phương trình (2) chúng ta có d(b) = 4 = a nên $T(n) = O(n^2 \log n)$.

Các hàm tiến triển khác

Ta xét hai trường hợp dưới dạng hai ví dụ, trường hợp 1 là tổng quát hóa của hàm bất kỳ là tích của một hàm nhân với một hằng lớn hơn hoặc bằng 1. Trường hợp thứ hai là hàm tiến triển không phải là một hàm nhân.

Ví dụ 1-14: Giải pgương trình đệ quy sau :

T(1) = 1

$T(n) = 3T(n/2) + 2n^{1.5}$

Ở đây, $2n^{1.5}$ không phải là hàm nhân nhưng $n^{1.5}$ là hàm nhân. Đặt U(n) = T(n)/2 với mọi n thì

U(1) = 1/2

$U(n) = 3U(n/2) + n^{1.5}$

Nghiệm thuần nhất khi U(1) = 1 là $n^{\log 3} = n^{1.59}$; vì U(1) = 1/2 nên nghiệm thuần nhất là $n^{1.59/2}$ là $O(n^{1.59})$. Vì a = 3 và b = 2 và $b^{1.5} = 2.82 < a$, nghiệm riêng cũng là $O(n^{1.59})$ và do đó $U(n) = O(n^{1.59})$. Vì

T(n) = 2U(n) nên T(n) = O($n^{1.59}$) hay T(n) = O($n^{\log 3}$).

Ví dụ 1-15: Giải phương trình đệ quy sau :

T(1) = 1

T(n) = 2T(n/2) + nlogn

Vì a = b = 2 nên nghiệm thuần nhất là n. Tuy nhiên, d(n) = nlogn không phải là hàm nhân ta phải tính nghiệm riêng bằng cách xét trực tiếp:

$$\sum_{j=0}^{k-1} 2^j 2^{k-j} \log(2^{k-j}) = 2^k \sum_{j=0}^{k-1} (k-j) = 2^{k-1} k(k+1)$$

Vì k = logn chúng ta có nghiệm riêng là O(nlog^{2}n), nghiệm này lớn hơn nghiệm thuần nhất và T(n) = O(nlog^{2}n).

BÀI TẬP Chương I: KỸ THUẬT PHÂN TÍCH GIẢI THUẬT

Bài 1: Tính thời gian thực hiện của các đoạn chương trình sau:

a) Tính tổng của các số

```
Sum := 0;
for i:=1 to n do begin
        readln(x);
        Sum := Sum + x;
end;
```

b) Tính tích hai ma trận vuông cấp n C = A*B:

```
for i := 1 to n do
        for j := 1 to n do begin
                c[i,j] := 0;
                for k := 1 to n do c[i,j] := c[i,j] + a[i,k] * b[k,j];
end;
```

Bài 2: Giải các phương trình đệ quy sau với T(1) = 1 và

a) T(n) = 3T(n/2) + n

b) T(n) = 3T(n/2) + n^2

c) T(n) = 8T(n/2) + n^3

Bài 3: Giải các phương trình đệ quy sau với T(1) = 1 và

a) $T(n) = 4T(n/3) + n$

b) $T(n) = 4T(n/3) + n^2$

c) $T(n) = 9T(n/3) + n^2$

Bài 4: Giải các phương trình đệ quy sau với T(1) = 1 và

a) $T(n) = T(n/2) + 1$

b) $T(n) = 2T(n/2) + \log n$

c) $T(n) = 2T(n/2) + n$

d) $T(n) = 2T(n/2) + n^2$

Bài 5: Giải các phương trình đệ quy sau bằng phương pháp đoán nghiệm:

a) T(1) = 2 và $T(n) = 2T(n-1) + 1$ với $\forall\ n \geq 2$

b) T(1) = 1 và $T(n) = 2T(n-1) + n$ với $\forall\ n \geq 2$

Bài 6: Cho một mảng n số nguyên được sắp thứ tự tăng. Viết hàm tìm một số nguyên trong mảng đó, nếu tìm thấy thì trả về TRUE, ngược lại trả về FALSE.

Sử dụng hai phương pháp tìm kiếm tuần tự và tìm kiếm nhị phân. Với mỗi phương pháp hãy viết một hàm tìm và tính thời gian thực hiện của hàm đó.

Bài 7: Tính thời gian thực hiện của giải thuật đệ quy giải bài toán Tháp Hà nội với n tầng?

Bài 8: Xét định nghĩa số tổ hợp chập k của n như sau:

$$C_n^k = \begin{cases} 1 \text{ nếu } k = 0 \text{ hoặc } k = n \\ C_n^{k-1} + C_{n-1}^{k-1} \text{ nếu } 0 < k < n \end{cases}$$

a) Viết một hàm đệ quy để tính số tổ hợp chập k của n.

Tính thời gian thực hiện của giải thuật nói trên.

CHƯƠNG II: SẮP XẾP

Trong chương này chúng ta sẽ nghiên cứu các vấn đề sau:

· Bài toán sắp xếp.

· Một số giải thuật sắp xếp đơn giản.

· QuickSort

· HeapSort

· BinSort

II.1- BÀI TOÁN SẮP XẾP

II.1.1 Tầm quan trọng của bài toán sắp xếp

Sắp xếp một danh sách các đối tượng theo một thứ tự nào là một bài toán thường được vận dụng trong các ứng dụng tin học. Ví dụ ta cần sắp xếp danh sách thí sinh theo tên với thứ tự Alphabet, hoặc sắp xếp danh sách sinh viên theo điểm trung bình với thứ tự từ cao đến thấp. Một ví dụ khác là khi cần tìm kiếm một đối tượng trong một danh sách các đối tượng bằng giải thuật tìm kiếm nhị phân thì danh sách các đối tượng này phải được sắp xếp trước đó.

Tóm lại sắp xếp là một yêu cầu không thể thiếu trong khi thiết kế các phần mềm.

II.1.2 Sắp xếp trong và sắp xếp ngoài

Sắp xếp trong là sự sắp xếp dữ liệu được tổ chức trong bộ nhớ trong cuả máy tính, ở đó ta có thể sử dụng khả năng truy nhập ngẫu nhiên của bộ nhớ và do vậy sự thực hiện rất nhanh.

Sắp xếp ngoài là sự sắp xếp được sử dụng khi số lượng đối tượng được sắp xếp lớn không thể lưu trữ trong bộ nhớ trong mà phải lưu trữî trên bộ nhớ ngoài. Cụ thể là ta sẽ sắp xếp dữ liệu được lưu trữ trong các tập tin.

Chương này tập trung giải quyết vấn đề sắp xếp trong còn sắp xếp ngoài sẽ được nghiên cứu trong chương IV.

II.1.3 Tổ chức dữ liệu và ngôn ngữ cài đặt

Các đối tượng cần được sắp xếp là các mẩu tin gồm một hoặc nhiều trường. Một trong các trường được gọi là khóa (key), kiểu của nó là một kiểu có quan hệ thứ tự (như các kiểu số nguyên, số thực, chuỗi ký tự...).

Danh sách các đối tượng cần sắp xếp sẽ là một mảng của các mẩu tin vừa nói ở trên. Mục đích của việc sắp xếp là tổ chức lại các mẩu tin sao cho các khóa của chúng được sắp thứ tự tương ứng với quy luật sắp xếp.

Để trình bày các ví dụ minh họa chúng ta sẽ dùng PASCAL làm ngôn ngữ thể hiện và sử dụng khai báo sau:

```
const N = 100;
type
        KeyType = integer;
        OtherType = real;

        RecordType = Record
                Key : KeyType;
                OtherFields : OtherType;
                end;
var
        a : array[1..N] of RecordType;

procedure Swap(var x,y:RecordType);
        var
                temp : RecordType;
begin
        temp := x;
        x := y;
        y := temp;
end;
```

Cần thấy rằng thủ tục Swap lấy O(1) thời gian vì chỉ thực hiện 3 lệnh gán nối tiếp nhau.

II.2- CÁC PHƯƠNG PHÁP SẮP XẾP ĐƠN GIẢN

II.2.1- Sắp xếp chọn
II.2.2- Sắp xếp xen
II.2.3- Sắp xếp nổi bọt

Các giải thuật đơn giản thường lấy $O(n^2)$ thời gian để sắp xếp n đối tượng và các giải thuật này thường chỉ dùng để sắp các danh sách có ít đối tượng.

Với mỗi giải thuật chúng ta sẽ nghiên cứu các phần: giải thuật, ví dụ, chương trình và phân tích đánh giá.

II.2.1- Sắp xếp chọn (Selection Sort)

Giải thuật

Đây là phương pháp sắp xếp đơn giản nhất được tiến hành như sau:

· Đầu tiên chọn phần tử có khóa nhỏ nhất trong n phần tử từ a[1] đến a[n] và hoán vị nó với phần tử a[1].

· Chọn phần tử có khóa nhỏ nhất trong n-1phần tử từ a[2] đến a[n] và hoán vị nó với a[2].

· Tổng quát ở bước thứ i, chọn phần tử có khoá nhỏ nhất trong n-i+1 phần tử từ a[i] đến a[n] và hoán vị nó với a[i].

· Sau n-1 bước này thì mảng đã được sắp xếp.

Phương pháp này được gọi là phương pháp chọn bởi vì nó lặp lại quá trình chọn phần tử nhỏ nhất trong số các phần tử chưa được sắp.

Ví dụ 2-1: Sắp xếp mảng gồm 10 mẩu tin có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Khoá Bước	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	6	5	2	10	12	9	10	9	3
Bước 2		**2**	5	6	10	12	9	10	9	3
Bước 3			**3**	6	10	12	9	10	9	5
Bước 4				**5**	10	12	9	10	9	6
Bước 5					**6**	12	9	10	9	10
Bước 6						**9**	12	10	9	10
Bước 7							**9**	10	12	10
Bước 8								**10**	12	10
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-1: Sắp xếp chọn

Chương trình:

```
procedure SelectionSort;
var
        i,j,LowIndex: integer;
        LowKey: KeyType;
begin
(1)     for i := 1 to n-1 do begin
(2)         LowIndex := i;
(3)         LowKey := a[i].key;
(4)         for j := i+1 to n do
(5)             if a[j].key < LowKey then
```

```
                begin
(6)                 LowKey := a[j].key;
(7)                 LowIndex := j;
                end;
(8)         Swap(a[i] , a[LowIndex]);
        end;
end;
```

Đánh giá: *Phương pháp sắp xếp chọn lấy* ***O(n^2)*** *để sắp xếp n phần tử.*

Trước hết ta có thủ tục Swap lấy một hằng thời gian như đã nói ở mục II.1.3.

Các lệnh (2), (3) đều lấy O(1) thời gian. Vòng lặp for (4) - (7) thực hiện n-i lần, vì j chạy từ i+1 đến n, mỗi lần lấy O(1), nên lấy O(n-i) thời gian. Do đó thời gian tổng cộng là:

$$T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2} \text{ tức là } O(n^2).$$

II.2.2- Sắp xếp xen (Insertion Sort)

Giải thuật

Trước hết ta xem phần tử a[1] là một dãy đã có thứ tự.

· Bước 1, xen phần tử a[2] vào danh sách đã có thứ tự a[1] sao cho a[1], a[2] là một danh sách có thứ tự.

· Bước 2, xen phần tử a[3] vào danh sách đã có thứ tự a[1], a[2] sao cho a[1], a[2], a[3] là một danh sách có thứ tự.

· Tổng quát, bước i, xen phần tử a[i+1] vào danh sách đã có thứ tự a[1],a[2],..a[i] sao cho a[1], a[2],.. a[i+1] là một danh sách có thứ tự.

· Phần tử đang xét a[j] sẽ được xen vào vị trí thích hợp trong danh sách các phần tử đã được sắp trước đó a[1],a[2],..a[j-1] bằng cách so sánh khoá của a[j] với khoá của a[j-1] đứng ngay trước nó. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j-1] và a[j] cho nhau và tiếp tục so sánh khoá của a[j-1] (lúc này a[j-1] chứa nội dung của a[j]) với khoá của a[j-2] đứng ngay trước nó...

Ví dụ 2-2: Sắp xếp mảng gồm 10 mẩu tin đã cho trong ví dụ 2-1.

Khoá Bước	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**5**	**6**								
Bước 2	**2**	**5**	**6**							
Bước 3	**2**	**2**	**5**	**6**						
Bước 4	**2**	**2**	**5**	**6**	**10**					
Bước 5	**2**	**2**	**5**	**6**	**10**	**12**				

Bước 6	**2**	**2**	**5**	**6**	**9**	**10**	**12**			
Bước 7	**2**	**2**	**5**	**6**	**9**	**10**	**10**	**12**		
Bước 8	**2**	**2**	**5**	**6**	**9**	**9**	**10**	**10**	**12**	
Bước 9	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-2: Sắp xếp xen

Chương trình

```
procedure InsertionSort;
var
        i,j: integer;
begin
{1}     for i := 2 to n do begin
{2}         J := i;
{3}         while (j>1) and (a[j].key < a[j-1].key) do begin
{4}             swap(a[j], a[j-1]);
{5}             j := j-1;
             end;
        end;
end;
```

Đánh giá: *Phương pháp sắp xếp chọn lấy* ***O(n^2)*** *để sắp xếp n phần tử.*

Ta thấy các lệnh (4) và (5) đều lấy O(1). Vòng lặp (3) chạy nhiều nhất i-1 lần, mỗi lần tốn O(1) nên (3) lấy i-1 thời gian. Lệnh (2) và (3) là hai lệnh nối tiếp nhau, lệnh (2) lấy O(1) nên cả hai lệnh này lấy i-1.

Vòng lặp (1) có i chạy từ 2 đến n nên nếu gọi T(n) là thời gian để sắp n phần tử thì ta có

$$T(n)=\sum_{i=2}^{n-1}(n-i)=\frac{n(n+1)}{2} - 1 \text{ tức là } O(n^2).$$

II.2.3- Sắp xếp nổi bọt (Bubble Sort)

Giải thuật

Chúng ta tưởng tượng rằng các mẩu tin được lưu trong một mảng dọc, qua quá trình sắp, mẩu tin nào có khóa "nhẹ" sẽ được nổi lên trên. Chúng ta duyệt tòan mảng, từ dưới lên trên. Nếu hai phần tử ở cạnh nhau mà không đúng thứ tự tức là nếu phần tử "nhẹ hơn" lại nằm dưới thì phải cho nó "nổi lên" bằng cách đổi chỗ hai phần tử này cho nhau. Cụ thể là:

· Bước 1: Xét các phần tử từ a[n] đến a[2], với mỗi phần tử a[j], so sánh khoá của nó với khoá của phần tử a[j-1] đứng ngay trước nó. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j] và a[j-1] cho nhau.

· Bước 2: Xét các phần tử từ a[n] đến a[3], và làm tương tự như trên.

· Tổng quát ở bước thứ i, ta sẽ xét các phần tử từ a[n] đến a[i+1].

Sau n bước ta thu được mảng có thứ tự

Ví dụ 2-3: Sắp xếp mảng gồm 10 mẩu tin đã cho trong ví dụ 2-1.

Khoá Bước	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	5	6	2	3	10	12	9	10	9
Bước 2		**2**	5	6	3	9	10	12	9	10
Bước 3			**3**	5	6	9	9	10	12	10
Bước 4				**5**	6	9	9	10	10	12
Bước 5					**6**	9	9	10	10	12
Bước 6						**9**	9	10	10	12
Bước 7							**9**	10	10	12
Bước 8								**10**	10	12
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-3: Sắp xếp nổi bọt

Chương trình

```
procedure BubbleSort;
var
        i,j: integer;
begin
(1)     for i := 1 to n-1 do
(2)         for j := n downto i+1 do
(3)             if a[j].key < a[j-1].key then
(4)                 Swap(a[j],a[j-1]);
end;
```

Đánh giá: *Phương pháp sắp xếp nổi bọt lấy* $O(n^2)$ *để sắp n phần tử.*

Dòng lệnh (3) lấy một hằng thời gian. Vòng lặp (2) thực hiện (n-i) bước, mỗi bước lấy O(1) nên lấy O(n-i) thời gian. Như vậy đối với toàn bộ chương trình ta có:

$$T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}=O(n^2).$$

II.3- QUICKSORT

II.3.1- Ý tưởng
II.3.2- Thiết kế giải thuật
II.3.3- Cài đặt giải thuật
II.3.4- Thời gian thực hiện của QuickSort

Trong phần này chúng ta sẽ nghiên cứu một giải thuật sắp xếp được dùng một cách phổ biến là Quick Sort do A.R. Hoare phát minh vào năm 1960. Quick Sort được đánh giá tốt nhờ vào sự phân tích toán học và các khẳng định về khả năng của nó. Quick Sort đã được cải tiến để trở thành phương pháp được chọn trong các ứng dụng sắp xếp thực tế khác nhau.

II.3.1- Ý tưởng

Chúng ta vẫn xét mảng a các mẩu tin a[1]..a[n]. Giả sử v là 1 giá trị khóa mà ta gọi là chốt (pivot). Ta phân hoạch dãy a[1]..a[n] thành hai mảng con "bên trái" và "bên phải". Mảng con "bên trái" bao gồm các phần tử có khóa nhỏ hơn chốt, mảng con "bên phải" bao gồm các phần tử có khóa lớn hơn hoặc bằng chốt.

Sắp xếp mảng con “bên trái” và mảng con “bên phải” thì mảng đã cho sẽ được sắp bởi vì tất cả các khóa trong mảng con “bên trái “ đều nhỏ hơn các khóa trong mảng con “bên phải”.

Việc sắp xếp các mảng con “bên trái” và “bên phải” cũng được tiến hành bằng phương pháp nói trên.

Một mảng chỉ gồm một phần tử hoặc gồm nhiều phần tử có khóa bằng nhau thì xem như đã có thứ tự.

II.3.2- Thiết kế giải thuật

Vấn đề chọn chốt

Chọn khóa lớn nhất trong hai phần tử có khóa khác nhau đầu tiên kể từ trái qua. Nếu mảng chỉ gồm một phần tử hay gồm nhiều phần tử có khóa bằng nhau thì không có chốt.

Vấn đề phân hoạch

Để phân hoạch mảng ta dùng 2 "con nháy" L và R trong đó L từ bên trái và R từ bên phải, ta cho L chạy sang phải cho tới khi gặp phần tử có khóa ≥ chốt và cho R chạy sang trái cho tới khi gặp phần tử có khóa < chốt. Tại chỗ dừng của L và R nếu L<R thì hoán vị a[L],a[R]. Lặp lại quá trình dịch sang phải, sang trái của 2 "con nháy" L và R cho đến khi L>R. Khi đó L sẽ là điểm phân hoạch, cụ thể là a[L] là phần tử đầu tiên của mảng con “bên phải”.

Giải thuật QuickSort

Để sắp xếp mảng a[i]..a[j] ta tiến hành các bước sau:

· Xác định chốt,

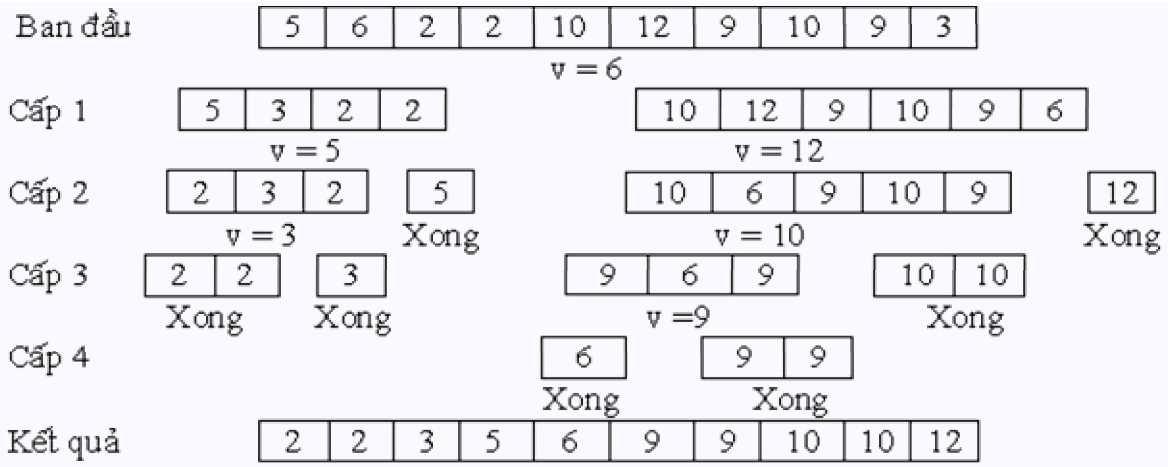
· Phân hoạch mảng đã cho thành hai mảng con a[i]..a[k-1] và a[k]..a[j].

· Sắp xếp mảng a[i]..a[k-1] (Đệ quy).

· Sắp xếp mảng a[k]..a[j] (Đệ quy).

Quá trình đệ quy sẽ dừng khi không còn tìm thấy chốt.

Ví dụ 2-4: Ta cần sắp một mảng mà khóa là các số nguyên đã được trình bày trong ví dụ 2-1. Hai phần tử đầu tiên có khóa khác nhau là 5 và 6, ta chọn 6 làm chốt và tiến hành phân hoạch mảng ban đầu làm hai mảng con và đệ quy cho hai mảng con này.



Hình 2-14: Sắp xếp nhanh.

II.3.3- Cài đặt giải thuật

Hàm FindPivot

Ta thiết kế hàm FindPivot để xác định trong dãy a[i]..a[j] xem có hay không hai phần tử có khóa khác nhau. Nếu không tìm thấy hai phần tử có khóa khác nhau thì trả về giá trị 0 (không tìm thấy chốt), ngược lại hàm trả về giá trị là **chỉ số** của phần tử có khóa lớn hơn trong hai phần tử có khóa khác nhau đầu tiên. Khóa lớn hơn này sẽ trở thành phần tử chốt mà ta sẽ xác định trong thủ tục QuickSort.

Để tiện so sánh ta sử dụng biến FirstKey để lưu giữ khóa của phần tử đầu tiên trong mảng a[i]..a[j] (FirstKey chính là a[i].key).

Ta sẽ dùng một chỉ số k để dò tìm trong mảng a[i]..a[j], kể từ vị trí i+1 đến hết mảng, một phần tử a[k] mà a[k].key <> FirstKey. Nếu không tìm thấy một a[k] như thế thì hoặc là mảng chỉ gồm một phần tử hoặc gồm nhiều phần tử có khóa bằng nhau. Trong trường hợp đó thì không tìm thấy chốt và hàm FindPivot sẽ trả về 0. Ngược lại ta sẽ phải xét xem a[k].key có lớn hơn FirstKey hay không, nếu đúng như thế thì chốt sẽ là khóa của a[k] và hàm FindPivot sẽ trả về k, nếu không thì hàm FindPivot sẽ trả về i.

```
Function FindPivot(i,j:integer): integer;
var
        FirstKey : KeyType;
        k : integer;
begin
        k := i+1;
        FirstKey := a[i].key;
        while (k<=j) and (a[k].key = FirstKey) do k:= k+1;
        if k > j then FindPivot := 0
        else
            if a[k].key > FirstKey then FindPivot := k
            else FindPivot := i;
end;
```

Hàm Partition

Hàm Partition nhận vào ba tham số i, j và Pivot để thực hiện việc phân hoạch theo mảng a[i]..a[j] theo chốt Pivot và trả về giá trị l là chỉ số đầu tiên của mảng "bên phải".

Hai con nháy L, R sẽ được sử dụng để thực hiện việc phân hoạch như đã trình bày trong phần II.3.2.

```
Function Partition(i,j:integer; pivot :KeyType):integer ;
var l,r : integer;
begin
        l := i; {Đặt con nháy L ởi bên trái}
        r := j; {Đặt con nháy R ởi bên phải}
        while l <= r do begin
            {L tiến sang phải}
            while a[l].key < pivot do l := l+1;
            {R tiến sang trái}
            while a[r].key >= pivot do r := r-1;
            if l <r then Swap(a[l],a[r]);
        end;
        Partition :=l;
end;
```

QuickSort

Bây giờ chúng ta trình bày thủ tục cuối cùng có tên là **QuickSort** và chú ý rằng để sắp xếp mảng A các record gồm n phần tử của kiểu Recordtype ta chỉ cần gọi **QuickSort(1,n)**.

Ta sẽ sử dụng biến PivotIndex để lưu giữ kết quả trả về của hàm FindPivot, nếu biến PivotIndex nhận được một giá trị khác 0 thì mới tiến hành phân hoạch mảng. Biến Pivot sẽ được sử dụng để lưu giữ giá trị chốt và biến k để lưu giữ giá trị của điểm phân hoạch do hàm Partition trả về. Sau khia đã phân hoạch xong ta sẽ gọi đệ quy QuickSort cho mảng con "bên trái" a[i]..a[k-1] và mảng con "bên phải" a[k]..a[j].

```
procedure Quicksort(i,j:integer);
var
        Pivot : KeyType;
        PivotIndex, k : integer;
begin
(1)     PivotIndex := FindPivot(i,j);
(2)     if PivotIndex <> 0 then
        begin
(3)         Pivot := a[PivotIndex].key;
(4)         k := Partition(i,j,Pivot);
(5)         QuickSort(i,k-1);
(6)         QuickSort(k,j);
        end;
end;
```

II.3.4- Thời gian thực hiện của QuickSort

*QuickSort lấy **O(nlogn)** thời gian để sắp xếp n phần tử trong trường hợp tốt nhất và **O(n^2)**. trong trường hợp xấu nhất.*

Hàm **Partition** lấy thời gian tỷ lệ với số phần tử của mảng. Như vậy nếu mảng có n phần tử thì Partition lấy P(n)= n đơn vị thời gian.

Gọi T(n) là thời gian thực hiện của QuickSort thì T(n) phải là tổng của P(n) và thời gian QuickSort đệ quy cho hai mảng con.

Giả sử các giá trị khóa của mảng khác nhau. Trong trường hợp xấu nhất là ta luôn chọn phải phần tử có khóa lớn nhất làm chốt, lúc bấy giờ việc phân hoạch bị lệch tức là mảng bên phải chỉ gồm một phần tử chốt, còn mảng bên trái gồm n-1 phần tử còn lại. Khi đó ta có thể thành lập phương trình đệ quy như sau:

$$T(n) = \begin{cases} 1 & \text{nếu } n = 1 \\ T(n-1) + T(1) + n & \text{nếu } n > 1 \end{cases}$$

Giải phương trình này bằng phương pháp truy hồi

Ta có T(n) = T(n-1) + T(1) +n = T(n-1) + (n+1)

= [T(n-2) + T(1) +(n-1)] + (n+1) = T(n-2) + n + (n+1)

= [T(n-3) + T(1) +(n-2)] + n + (n+1) = T(n-3) +(n-1) + n + (n+1)

. . . . . .

= T(n-i) + (n-i+2) + (n-i+3) + ... + n + (n+1) = T(n-i) + $\sum_{j=n-1+2}^{n+1} j$

Quá trình trên kết thúc khi i=n-1, khi đó ta có

$$T(n) = T(1) + \sum_{j=3}^{n+1} j = 1 + \frac{(n-1)(n+4)}{2} = \frac{n^2 + 3n - 2}{2} = O(n^2)$$

Trong trường hợp tốt nhất khi ta chọn được chốt sao cho hai mảng con có kích thước bằng nhau và bằng n/2. Lúc đó ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} 1 & \text{nếu } n = 1 \\ 2T(n/2) + n & \text{nếu } n > 1 \end{cases}$$

Giải phương trình đệ quy này (xem I.4.2) ta được T(n) = O(nlogn).

Người ta cũng chứng minh được rằng trong trường hợp trung bình QuickSort lấy T(n) = O(nlogn).

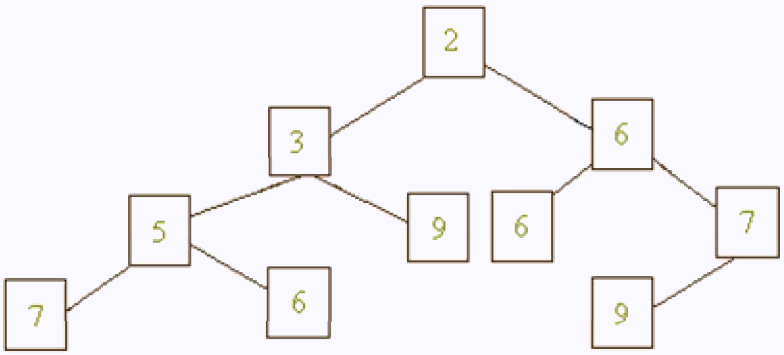
II.4- HEAPSORT

II.4.1- Heap

Cây sắp thứ tự bộ phận hay còn gọi là heap là cây nhị phân mà giá trị tại mỗi nút (khác nút lá) đều không lớn hơn giá trị của các con của nó.

Ta có nhận xét rằng nút gốc a[1] của cây sắp thứ tự bộ phận có giá trị nhỏ nhất.

Ví dụ 2-5: Cây sau là một heap.



Hình 2-5: Một heap

II.4.2- Ý tưởng

(1) Xem mảng ban đầu là một cây nhị phân. Mỗi nút trên cây lưu trữ một phần tử mảng, trong đó a[1] là nút gốc và mỗi nút không là nút lá a[i] có con trái là a[2i] và con phải là a[2i+1]. Với cách tổ chức này thì cây nhị phân thu được sẽ có các nút trong là các nút a[1],..,a[n DIV 2]. Tất cả các nút trong đều có 2 con, ngoại trừ nút a[n DIV 2] có thể chỉ có một con trái (trong trường hợp n là một số chẵn).

(2) Sắp xếp cây ban đầu thành một heap căn cứ vào giá trị khoá của các nút.

(3) Hoán đổi a[1] cho cho phần tử cuối cùng.

(4) Sắp lại cây sau khi đã bỏ đi phần tử cuối cùng để nó trở thành một heap mới.

Lặp lại quá trình (3) và (4) cho tới khi cây rỗng ta sẽ được mảng sắp theo thứ tự giảm.

II.4.3- Thiết kế và cài đặt giải thuật

Thủ tục PushDown

Giả sử a[first],..,a[last] đã đúng vị trí (giá trị khoá tại mỗi nút nhỏ hơn hoặc bằng giá trị khoá tại các nút con của nó) ngoại trừ a[first]. PushDown dùng để đẩy phần tử a[first] xuống đúng vị trí của nó trong cây (và có thể gây ra việc đẩy xuống các phần tử khác).

Xét a[first], có hai khả năng có thể xảy ra:

· Nếu a[first] có khoá lớn hơn con trái của nó (a[first].key > a[2*first].key) và khoá của con trái không lớn hơn khoá của con phải (a[2*first].key ≤ a[2*first+1].key) thì hoán đổi a[first] cho con trái a[2*first] của nó, việc này có thể gây ra tình trạng con trái sẽ không đúng vị trí nên phải xem xét lại con trái.

· Ngược lại, nếu a[first] có khoá lớn hơn khoá của con phải của nó (a[first].key > a[2*first+1].key) và khoá của con phải nhỏ hơn khoá của con trái (a[2*first+1].key < a[2*first].key) thì hoán đổi a[first] cho con phải a[2*first+1] của nó, việc này có thể gây ra tình trạng con phải sẽ không đúng vị trí nên phải tiếp tục xem xét con phải.

· Nếu cả hai trường trên đều không xẩy ra thì a[first] đã đúng vị trí.

· Như trên ta thấy việc đẩy a[first] xuống có thể gây ra việc đẩy xuống một số phần tử khác, nên tổng quát là ta sẽ xét việc đẩy xuống của một phần tử a[r] bất kỳ, bắt đầu từ a[first].

```
procedure PushDown(first,last:integer);
var r:integer;
begin
        r:= first; {Xét nút a[first] trước hết}
            while r <= last div 2 do
            {Trong khi a[r] còn là nút trong}
            if last = 2*r then begin {nút r chỉ có con trái }
                if a[r].key > a[last].key then swap(a[r],a[last]);
                r:=last;
        end
        else
            if (a[r].key>a[2*r].key)and(a[2*r].key  ≤  a[2*r+1].key)then begin
                swap(a[r],a[2*r]);
                r := 2*r ; {Xét tiếp nút con trái }
            end
            else
                if (a[r].key > a[2*r+1].key) and (a[2*r+1].key < a[2*r].key) then begin
                swap(a[r],a[2*r+1]);
                r := 2*r+1 ; {Xét tiếp nút con phải }
            end
            else
                r := last; {Nút r đã đúng vị trí }
end;
```

Thủ tục HeapSort

· Việc sắp xếp cây ban đầu thành một heap được tiến hành bằng cách sử dụng thủ tục PushDown để đẩy tất cả các nút trong chưa đúng vị trí xuống đúng vị trí của nó, khởi đầu từ nút a[n DIV 2], lần ngược tới gốc.

· Lặp lại việc hoán đổi a[1] cho a[i], sắp xếp cây a[1]..a[i-1] thành heap, i chạy từ n đến 2.

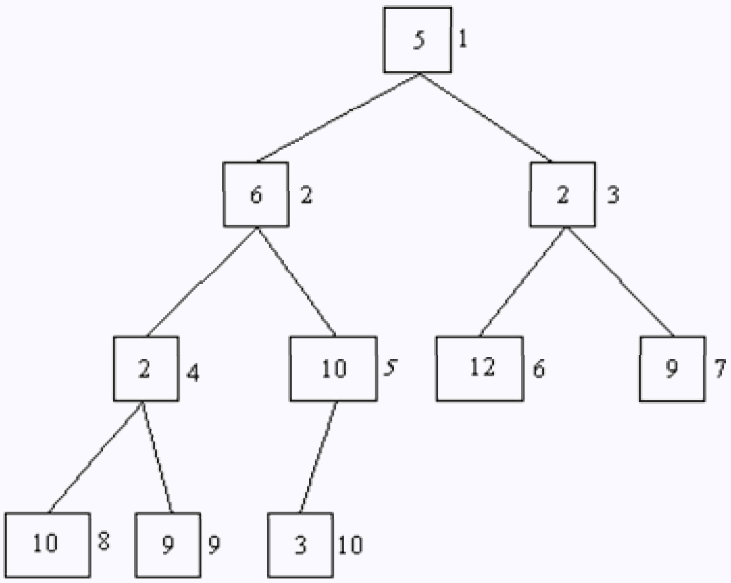
```
procedure HeapSort;
var
        i:integer;
begin
(1)     for i := (n div 2) downto 1 do
(2)         PushDown(i,n);
(3)     for i := n downto 2 do begin
(4)         swap(a[1],a[i]);
```

```
(5)         pushdown(1,i-1);
        end;
end;
```

Ví dụ 2-6: Sắp xếp mảng bao gồm 10 phần tử có khoá là các số nguyên như trong các ví dụ trước:

Khoá	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3

Mảng này được xem như là một cây nhị phân ban đầu như sau:

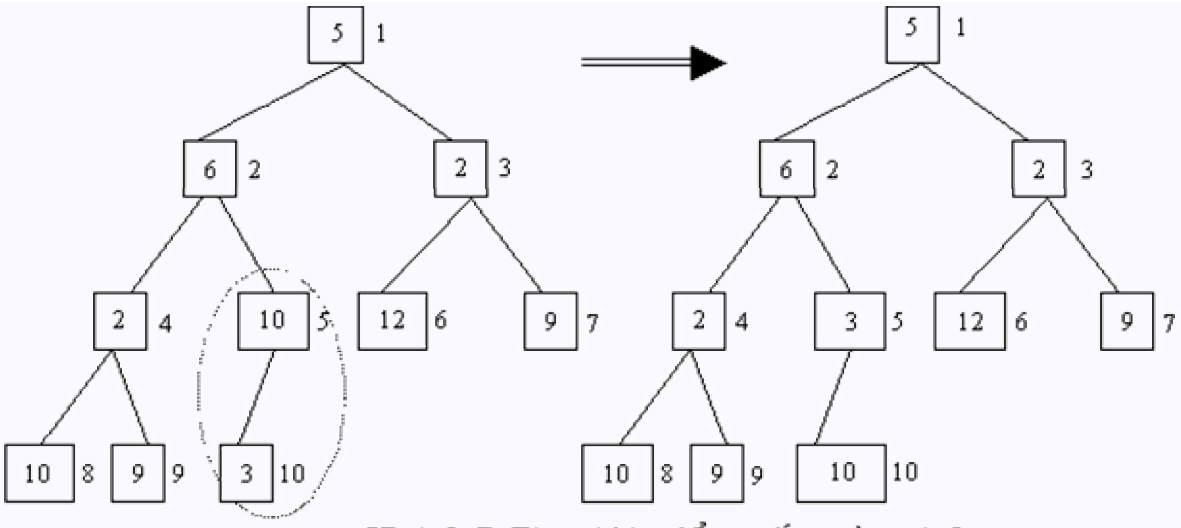


Hình 2-6: Cây ban đầu

Trong cây trên, giá trị ghi trong các nút là khoá của các phân tử mảng, giá trị ghi bên ngoài các nút là chỉ số của các phần tử mảng.

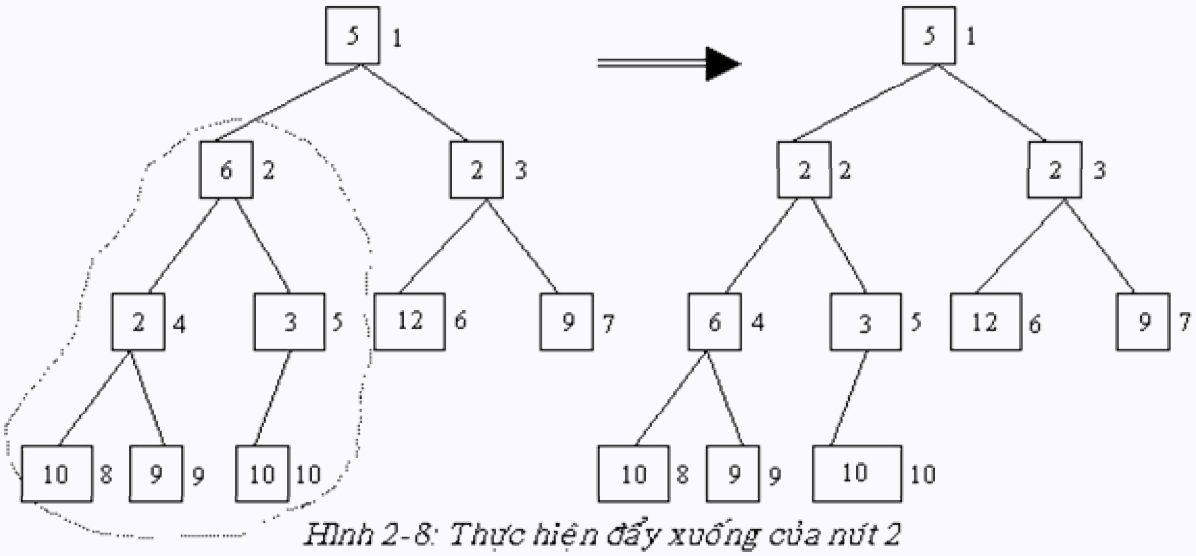
Việc sắp xếp cây này thành một heap sẽ bắt đầu từ việc đẩy xuống nút a[5] (vì 5 = 10 DIV 2)

Xét nút 5 ta thấy a[5] chỉ có một con trái và giá trị khóa tương ứng của nó lớn hơn con trái của nó nên ta đổi hai nút này cho nhau và do con trái của a[5] là a[10] là một nút lá nên việc đẩy xuống của a[5] kết thúc.



Hình 2-7: Thực hiện đẩy xuống của nút 5

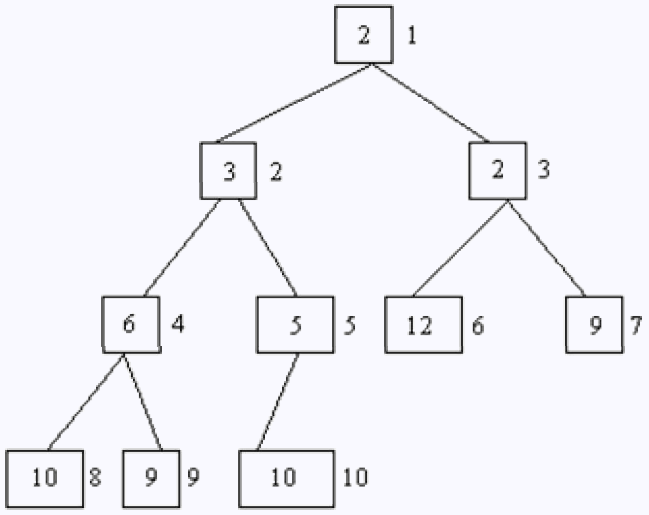
Nút 4 và nút 3 đã đúng vị trí nên không phải thực hiện sự hoán đổi. Tại nút 2, giá trị khóa của nó lớn hơn khoá con trái và khoá của con trái nhỏ hơn khoá của con phải nên ta hóan đổi nút 2 cho con trái của nó (nút 4), sau khi hoán đổi, ta xét lại nút 4, thấy nó vẫn đúng vị trí nên kết thúc việc đẩy xuống của nút 2.



Hình 2-8: Thực hiện đẩy xuống của nút 2

Cuối cùng ta xét nút 1, ta thấy giá trị khoá của nút 1 lớn hơn khoá của con trái và con trái có khoá bằng khoá của con phải nên hóan đổi nút 1 cho con trái của nó (nút 2).

Sau khi thực hiện phép hóan đổi nút 1 cho nút 2, ta thấy nút 2 có giá trị khoá lớn hơn khoá của con phải của nó (nút 5) và con phải có khoá nhỏ hơn khoá của con trái nên phải thực hiện phép hoán đổi nút 2 cho nút 5. Xét lại nút 5 thì nó vẫn đúng vị trí nên ta được cây mới trong hình 2-9.



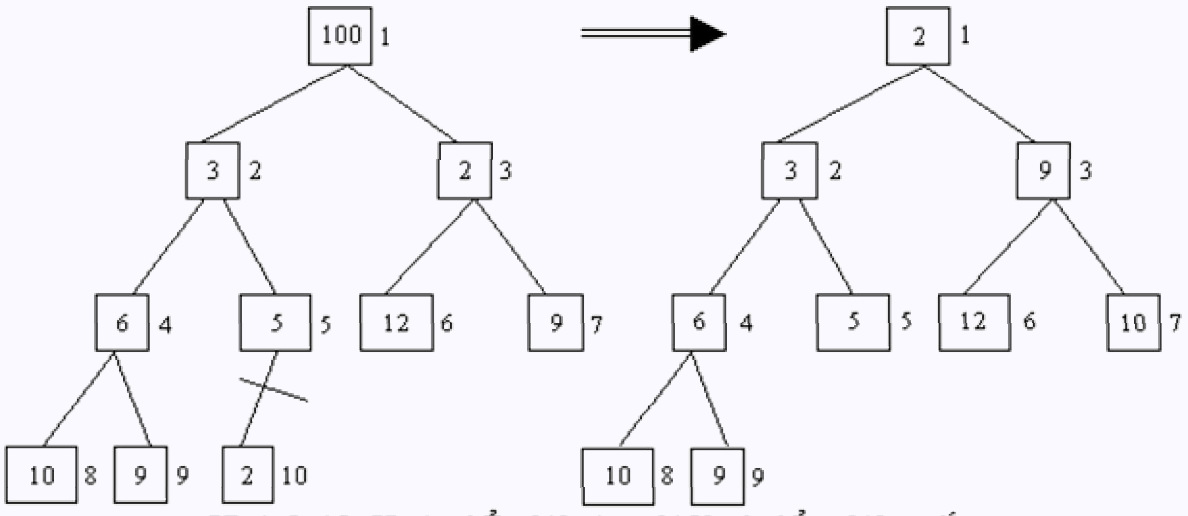
Hình 2-9: Cây ban đầu đã được tạo thành heap

Cây này là một heap tương ứng với mảng

Khoá	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Heap	2	3	2	6	5	12	9	10	9	10

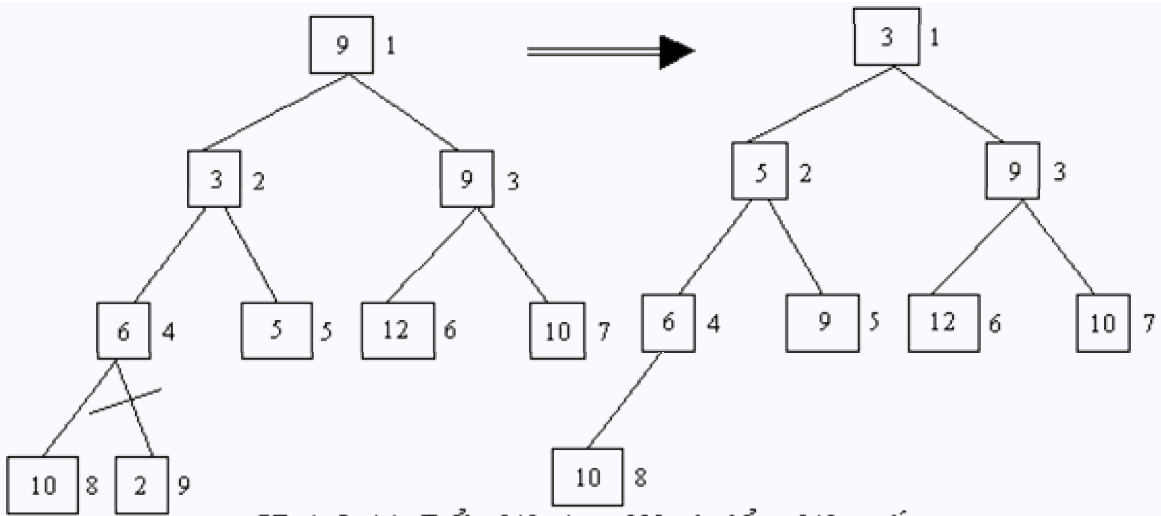
Từ heap đã có ở trên, hoán đổi a[1] cho a[10] ta có a[10] là nút có khóa nhỏ nhất, cắt bỏ nút a[10] ra khỏi cây. Như vậy phần cuối mảng chỉ gồm một phần tử a[10] đã được sắp.

Thực hiện việc đẩy a[1] xuống đúng vị trí của nó trong cây a[1]..a[9] ta được cây:



Hình 2-10: Hoán đổi a[1] cho a[10] và đẩy a[1] xuống

Hoán đổi a[1] cho a[9] và cắt a[9] ra khỏi cây. Ta được phần cuối mảng bao gồm hai phần tử a[9]..a[10] đã được sắp. Thực hiện việc đẩy a[1] xuống đúng vị trí của nó trong cây a[1]..a[8] ta được cây



Hình 2-11: Đổi a[1] cho a[9] và đẩy a[1] xuống

Tiếp tục quá trình trên ta sẽ được một mảng có thứ tự giảm.

II.4.4- Phân tích HeapSort

Thời gian thực hiện của **HeapSort** là **O(n logn)**

Trong thủ tục **PushDown** mỗi lần lặp, r có ít nhất 2 giá trị, vì r bắt đầu bằng first nên sau i lần lặp, chúng ta có r ≥ first * 2^i. Thủ tục dừng khi r > last/2 do vậy khi thủ tục dừng thì ta có first*2^i > last/2 hay 2^{i+1} > last/first. Suy ra i > log(last/first) - 1 tức là số lần lặp nhiều nhất của thủ tục PushDown là log(last/first).

Vì first ≥1 và last ≤ n trong mỗi lần gọi PushDown của thủ tục HeapSort tại dòng (2) hoặc (5) nên số lần lặp ≤ logn, mỗi bước lặp lấy một thời gian O(1) nên thời gian thực hiện của PushDown là O(logn).

Trong thủ tục **HeapSort** dòng lệnh (1)-(2) lặp n/2 lần mà mỗi lần lấy O(logn) nên thời gian thực hiện (1)-(2) là O(n logn). Vòng lặp (3)-(4)-(5) lặp n-1 lần, mỗi lần lấy O(logn) nên thời gian thực hiện (3)-(4)-(5) là O(n logn). Tóm lại thời gian thực hiện HeapSort là O(n logn)

II.5- BINSORT

II.5.1- Giải thuật

Nói chung các giải thuật đã trình bày ở trên đều có độ phức tạp là O(n2) hoặc O(nlogn). Tuy nhiên khi kiểu dữ liệu của trường khoá là một kiểu đặc biệt , việc sắp xếp có thể chỉ chiếm O(n) thời gian. Sau đây ta sẽ xét một số trường hợp.

Trường hợp đơn giản: Giả sử ta phải sắp xếp một mảng A gồm n phần tử có khoá là các số nguyên có giá trị khác nhau và là các giá trị từ 1 đến n. Ta sử dụng B là một mảng cùng kiểu với A và phân phối vào phần tử b[j] một phần tử a[i] mà a[i].key = j. Khi đó mảng B lưu trữ kết quả đã được sắp xếp của mảng A.

Ví dụ 2-7: Sắp xếp mảng A gồm 10 phần tử có khoá là các số nguyên có giá trị là các số 4, 5,

2, 1, 7, 8, 10, 9, 6 và 3

Ta sử dụng mảng B có cùng kiểu với A và thực hiện việc phân phối a[1] vào b[4] vì a[1].key = 4, a[2] vào b[5] vì a[2].key = 5, a[3] vào b[2] vì a[3].key = 2,...

A	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Khoá của A	4	5	2	1	7	8	10	9	6	3
B	b[4]	b[5]	b[2]	b[1]	b[7]	b[8]	b[10]	b[9]	b[6]	b[3]

Hình 2-12: Binsort trong trường hợp đơn giản

Để thực hiện việc phân phối này ta chỉ cần một lệnh lặp:

```
for i:=1 to n do B[A[i].key] := A[i]
```

Đây cũng là lệnh chính trong chương trình sắp xếp. Lệnh này lấy O(n) thời gian.

Các phần tử b[j] được gọi là các bin và phương pháp sắp xếp này được gọi là bin sort.

Trường hợp tổng quát: Là trường hợp có thể có nhiều phần tử có chung một giá trị khóa, chẳng hạn để sắp một mảng A có n phần tử mà các giá trị khóa của chúng là các số nguyên lấy giá trị trong khoảng 1..m với $m \leq n$. Trong trường hợp này ta không thể sử dụng các phần tử của mảng B làm bin được vì nếu có hai phần tử của mảng A có cùng một khoá thì không thể lưu trữ trong cùng một bin.

Để giải quyết sự đụng độ này ta chuẩn bị một cấu trúc có m bin, mỗi bin có thể lưu trữ nhiều hơn một phần tử. Cụ thể là bin thứ j sẽ lưu các phần tử có khóa là j ($1 \leq j \leq m$) sau đó ta sẽ nối các bin lại với nhau để được một dãy các phần tử được sắp.

Cách tốt nhất là ta thiết kế mỗi bin là một danh sách liên kết của các phần tử mà mỗi phần tử có kiểu RecordType. Ta sẽ gọi kiểu của danh sách này là ListType.

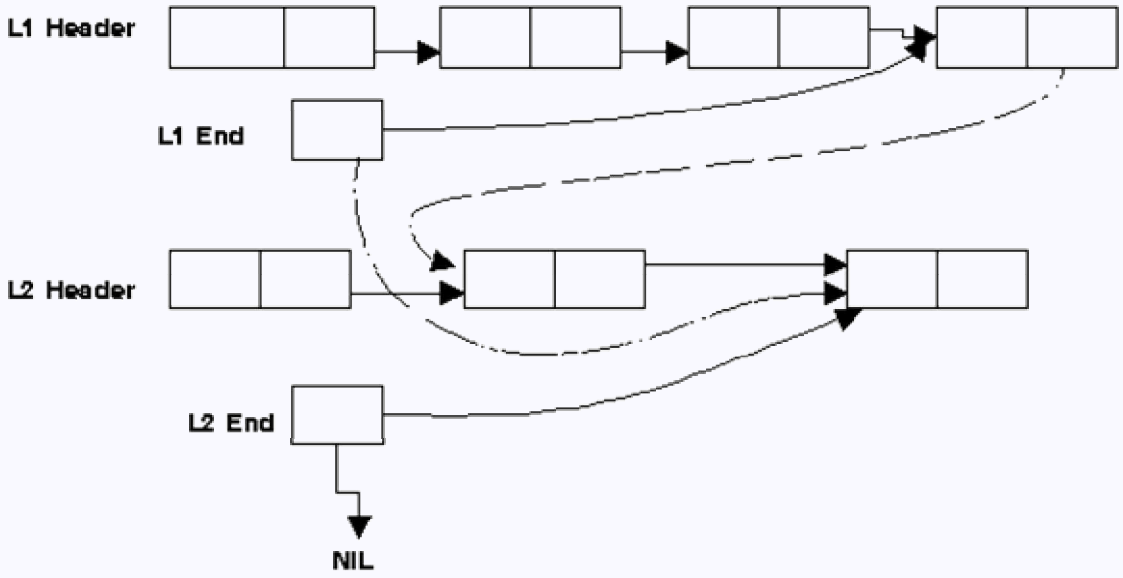
Ta có thể tạo kiểu ListType bằng cách ghép RecordType với một con trỏ để trỏ tới phần tử kế tiếp.

Lấy B là một mảng kiểu Array[KeyType] of ListType. Như vậy B là mảng các bin, mỗi bin là một danh sách. B được đánh chỉ số bởi KeyType, như thế có ít nhất một bin cho mỗi giá trị khoá.

Ta vẫn sẽ phân phối phần tử a[i] vào bin b[j] nếu j = a[i]key. Dĩ nhiên mỗi bin b[j] có thể chứa nhiều phần tử của mảng A. Các phần tử mới sẽ được đưa vào cuối danh sách b[j].

Sau khi tất cả các phần tử của mảng A đã được phân phối vào trong các bin, công việc cuối cùng là ta phải nối các bin lại với nhau, ta sẽ được một danh sách có thứ tự. Ta sẽ dùng thủ tục concatenate(L_1,L_2) để nối hai danh sách L_1, L_2. Nó thay thế danh sách L_1 bởi danh sách nối L_1L_2. Việc nối sẽ được thực hiện bằng cách gắn con trỏ của phần tử cuối cùng của L_1 vào đầu của L_2. Ta biết rằng để đến được phần tử cuối cùng của danh sách liên kết L1 ta phải duyệt qua tất cả các phần tử của nó. Để cho có hiệu quả, ta thêm một con trỏ nữa, trỏ đến phần tử cuối cùng của mỗi danh sách, điều này giúp ta đi thẳng tới phần tử cuối cùng mà không phải duyệt qua toàn bộ danh sách.

Hình 2-13 minh họa việc nối hai danh sách.



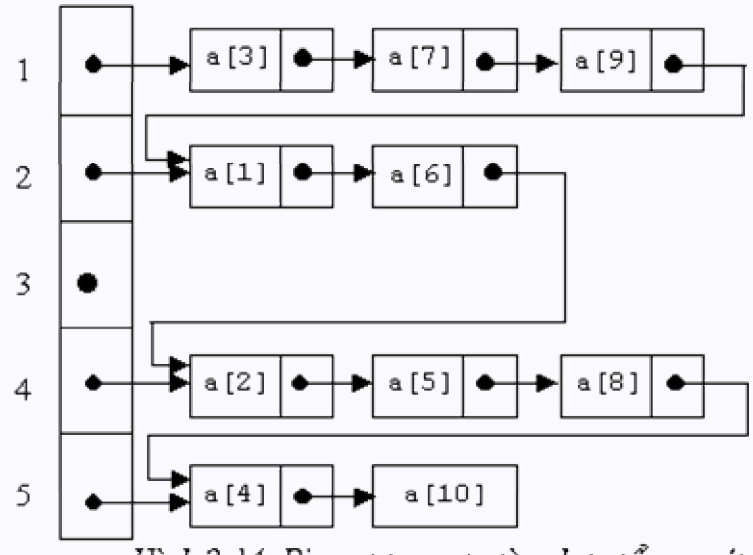
Hình 2-13: Nối các bin

Sau khi nối thì header và end của danh sách L2 không còn tác dụng nữa.

Ví dụ 2-8: Sắp xếp mảng A gồm 10 phần tử có khoá là các số nguyên có giá trị là các số 2, 4, 1, 5, 4, 2, 1, 4, 1, 5.

A	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Khoá của A	2	4	1	5	4	2	1	4	1	5

Ta thấy các giá trị khoá nằm trong khoảng 1..5. Ta tổ chức một mảng B gồm 5 phần tử, mỗi phần tử là một con trỏ, trỏ đến một danh sách liên kết .



Hình 2-14: Binsort trong trường hợp tổng quát

Chương trình sử dụng cấu trúc danh sách liên kết làm các bin

```
VAR
A: array[1..n] of RecordType;
```

```
B: array[keytype] of ListType;
{Ta giả thiết keytype là kiểu miền con 1..m }
procedure BinSort;
var
        i:integer;
        j: KeyType;
begin
{1}for i:=1 to n do
        INSERT(A[i], END(B[A[i].key]), B[A[i}.key]);
{2}for j:= 2 to m do
        CONCATENATE(B[1], B[j]);
end;
```

II.5.2- Phân tích Bin Sort

*Bin sort lấy **O(n)** thời gian để sắp xếp mảng gồm n phần tử.*

Trước hết thủ tục INSERT cần một thời gian O(1) để xen một phần tử vào trong danh sách. Do cách tổ chức danh sách có giữ con trỏ đến phần tử cuối cùng nên việc nối hai danh sách bằng thủ tục CONCATENATE cũng chỉ mất O(1) thời gian. Ta thấy vòng lặp {1} thực hiện n lần, mỗi lần tốn O(1) = 1 nên lấy O(n) đơn vị thời gian. Vòng lặp {2} thực hiện m-1 lần, mỗi lần O(1) nên tốn O(m) đơn vị thời gian. Hai lệnh {1} và {2} nối tiếp nhau nên thời gian thực hiện của BinSort là T(n) = O(max(n,m)) = O(n) vì $m \leq n$.

II.5.3- Sắp xếp tập giá trị có khoá lớn

Nếu m số các khoá không lớn hơn n số các phần tử cần sắp xếp, khi đó O(max(n,m)) thực sự là O(n). Nếu giả sử $m = n^2$ khi đó O(max(n, n^2)) là O(n^2) như vậy Bin sort lấy O(n^2) thời gian.

Tuy nhiên ta vẫn có thể tổng quát hoá kỹ thuật bin sort để nó vẫn lấy O(n) thời gian.

Giả sử ta cần sắp xếp n phần tử có các giá trị khoá thuộc 0..n^2-1.

Ta sẽ sử dụng n bin B[0], B[1],...B[n-1] và tiến hành việc sắp xếp trong hai kỳ.

Kỳ 1: Phân phối phần tử a[i] vào bin B[j] mà j = a[i].key MOD n.

Kỳ 2: Phân phối các phân tử trong danh sách kết quả của kỳ 1 vào các bin. Phần tử a[i] sẽ được phân phối vào bin B[j] mà j = a[i].key DIV n.

Chú ý rằng trong cả hai kỳ, ta xen các phần tử mới được phân phối vào cuối danh sách.

Ví dụ 2-9: Cần sắp xếp mảng gồm 10 phần tử có khoá là các số nguyên: 36, 9, 0, 25, 1, 49, 64, 16, 81 và 4.

Ta sử dụng 10 bin được đánh số từ 0 đến 9. Kỳ một ta phân phối phần tử a[i] vào bin có chỉ số a[i].key MOD 10. Nối các bin lại với nhau ta được danh sách có khóa là: 0 1 81 64 4 25 36 16 9 49. Kỳ hai sử dụng kết quả của kỳ 1 để sắp tiếp. Phân phối phần tử a[i] vào bin có chỉ số a[i].key DIV 10. Nối các bin lại với nhau ta được danh sách có thứ tự.

Kỳ 1 Kỳ 2

Bin		
0	0	
1	1	81
2		
3		
4	64	4
5	25	
6	36	16
7		
8		
9	9	49

Bin				
0	0	1	4	9
1	16			
2	25			
3	36			
4	49			
5				
6	64			
7				
8	81			
9				

Hình 2-13: Sắp xếp theo hai kỳ

Theo sự phân tích giải thuật Bin Sort thì mỗi kỳ lấy O(n) thời gian, hai kỳ này nối tiếp nhau nên thời gian tổng cộng là O(n).

Chứng minh giải thuật đúng

Để thấy tính đúng đắn của giải thuật ta xem các các giá trị khóa nguyên từ 0 đến n^2-1 như **các số có hai chữ số trong hệ đếm cơ số n**. Xét hai số k = sn + t và l = un + v trong đó s, t, u, v là các số 0..n-1. Giả sử k < l, ta cần chứng minh rằng sau 2 kỳ sắp thì k phải đứng trước l.

Vì k < l nên s ≤ u. Nếu s < u thì k đứng trước l trong danh sách kết qủa vì k được sắp vào bin b[s] và l được sắp vào bin b[u] mà b[s] đứng trước b[u]. Nếu s=u thì t<v. Sau kỳ 1 thì k đứng trước l, vì k được sắp vào trong bin b[t] và l trong bin b[v]. Đến kỳ 2 mặc dù cả k và l đều được sắp vào trong bin b[s], nhưng k được xen vào trước l nên kết quả là k đứng trước l.

Chú ý Từ chứng minh trên ta thấy để sắp các phần tử có khóa là các số nguyên (hệ đếm cơ số 10) từ 0 đến 99 ta dùng 10 bin có chỉ số từ 0 đến 9. Để sắp các phần tử có khóa là các số nguyên từ 0 đến 999 ta dùng 100 bin có chỉ số từ 0 đến 99...

Bài 1: Sắp xếp mảng gồm 12 phần tử có khóa là các số nguyên : 5, 15, 12, 2, 10, 12, 9, 1, 9, 3, 2, 3 bằng cách sử dụng:

a) Sắp xếp chọn.

b) Sắp xếp xen.

c) Sắp xếp nổi bọt.

d) QuickSort.

e) HeapSort.

Bài 2: Có một biến thể của sắp xếp chọn như sau: Trong bước thứ i ta chọn một phần tử có

khóa nhỏ nhất và một phần tử có khóa lớn nhất trong a[i]...a[n-i+1] và hóan đổi phần tử nhỏ nhất cho a[i] và phần tử lớn nhất cho a[n-i+1]. Hãy viết thủ tục sắp xếp chọn theo giải thuật này và tính thời gian thực hiện.

Bài 3: Viết thủ tục sắp xếp trộn (xem giải thuật thô trong chương 1).

Bài 4: Có một biến thể của QuickSort như sau: Chọn chốt là khóa của phần tử nhỏ nhất trong hai phần tử có khóa khác nhau đầu tiên. Mảng con bên trái gồm các phần tử có khóa nhỏ hơn hoặc bằng chốt, mảng con bên phải gồm các phần tử có khóa lớn hơn chốt. Hãy viết lại các thủ tục cần thiết cho biến thể này.

Bài 5: Một biến thể khác của QuickSort là chọn khóa của phần tử đầu tiên làm chốt. Hãy viết lại các thủ tục cần thiết cho biến thể này.

Bài 6: Hãy viết lại thủ tục PushDown trong HeapSort để có thể sắp xếp theo thứ tự tăng.

CHƯƠNG III: KỸ THUẬT THIẾT KẾ GIẢI THUẬT

Nói chung khi thiết kế một giải thuật chúng ta thường dựa vào một số kỹ thuật nào đó. Chương này sẽ trình bày một số kỹ thuật quan trọng để thiết kế giải thuật như: Chia để trị (Divide-and-Conquer), quy hoạch động (dynamic programming), kỹ thuật tham ăn (greedy techniques), quay lui (backtracking) và tìm kiếm địa phương (local search). Các kỹ thuật này được áp dụng vào một lớp rộng các bài toán, trong đó có những bài toán nổi tiếng như bài toán tìm đường đi ngắn nhất của người giao hàng, bài toán cây phủ tối tiểu...

III.1- GIẢI THUẬT CHIA ĐỂ TRỊ

III.1.1- Nội dung kỹ thuật

Có thể nói rằng kỹ thuật quan trọng nhất, được áp dụng rộng rãi nhất để thiết kế các giải thuật có hiệu quả là kỹ thuật "chia để trị" (divide and conquer). Nội dung của nó là: Để giải một bài toán kích thước n, ta chia bài toán đã cho thành một số bài toán con có kích thuóc nhỏ hơn. Giải các bài toán con này rồi tổng hợp kết quả lại để được lời giải của bài toán ban đầu. Đối với các bài toán con, chúng ta lại sử dụng kỹ thuật chia để trị để có được các bài toán kích thước nhỏ hơn nữa. Quá trình trên sẽ dẫn đến những bài toán mà lời giải **chúng là hiển nhiên** hoặc **dễ dàng thực hiện**, ta gọi các bài toán này là **bài toán cơ sở**.

Tóm lại kỹ thuật chia để trị bao gồm hai quá trình: **Phân tích** bài toán đã cho thành các bài toán cơ sở và **tổng hợp kết quả** từ bài toán cơ sở để có lời giải của bài toán ban đầu. Tuy nhiên đối với một số bài toán, thì quá trình phân tích đã chứa đựng việc tổng hợp kết quả do đó nếu chúng ta đã giải xong các bài toán cơ sở thì bài toán ban đầu cũng đã được giải quyết. Ngược lại có những bài toán mà quá trình phân tích thì đơn giản nhưng việc tổng hợp kết quả lại rất khó khăn. Trong các phần tiếp sau ta sẽ trình bày một số ví dụ để thấy rõ hơn điều này.

Kỹ thuật này sẽ cho chúng ta một giải thuật đệ quy mà việc xác định độ phức tạp của nó sẽ phải giải một phương trình đệ quy như trong chương I đã trình bày.

III.1.2 Nhìn nhận lại giải thuật MergeSort và QuickSort

Hai giải thuật sắp xếp đã được trình bày trong các chương trước (MergeSort trong chương I và QuickSort trong chương II) thực chất là đã sử dụng kỹ thuật chia để trị.

Với MergeSort, để sắp một danh sách L gồm n phần tử, chúng ta chia L thành hai danh sách con L1 và L2 mỗi danh sách có n/2 phần tử. Sắp xếp L1, L2 và trộn hai danh sách đã được sắp này để được một danh sách có thứ tự. Quá trình phân tích ở đây là quá trình chia đôi một danh sách, quá trình này sẽ dẫn đến bài toán sắp xếp một danh sách có độ daì bằng 1, đây chính là bài toán cơ sở vì việc sắp xếp danh sách này là "không làm gì cả". Việc tổng hợp các kết quả ở đây là "trộün 2 danh sách đã được sắp để được một danh sách có thứ tự".

Với QuickSort, để sắp xếp một danh sách gồm n phần tử, ta tìm một giá trị chốt và phân hoạch danh sách đã cho thành hai danh sách con "bên trái" và "bên phải ". Sắp xếp "bên trái" và "bên phải" thì ta được danh sách có thứ tự. Quá trình phân chia sẽ dẫn đến các bài toán sắp xếp một danh sách chỉ gồm một phần tử hoặc gồm nhiều phần tử có khoá bằng nhau, đó chính là các bài toán cơ sở, vì bản thân chúng đã có thứ tự rồi. Ở đây chúng ta cũng không có việc tổng hợp kết quả một cách tường minh, vì việc đó đã được thực hiện trong quá trình phân hoạch.

III.1.3- Bài toán nhân các số nguyên lớn

Xét bài toán nhân hai số nguyên n chữ số X và Y. Theo giải thuật nhân hai số thông thường thì cần n^2 phép nhân và n phép cộng nên tốn $O(n^2)$ thời gian. Áp dụng kỹ thuật "chia để trị" vào phép nhân các số nguyên, ta chia mỗi số nguyên X và Y thành các số nguyên có n/2 chữ số. Để đơn giản ta giả sử n là luỹ thừa của 2

$X = A10^{n/2} + B$

$Y = C10^{n/2} + D$

Trong đó A, B, C, D là các số nguyên có n/2 chữ số.

Chẳng hạn với X = 1234 thì A = 12 và B = 34 bởi vì $X = 12 * 10^2 + 34$.

Tích của X và Y có thể được viết thành: $XY = AC10^n + (AD + BC)10^{n/2} + BD$ (III.1)

Với mỗi số có n/2 chữ số, chúng ta lại tiếp tục phân tích theo cách trên, quá trình phân tích sẽ dẫn đến bài toán cơ sở là nhân các số nguyên chỉ gồm một chữ số mà ta dễ dàng thực hiện. Việc tổng hợp kết quả chính là thực hiện các phép toán theo công thức (III.1).

Theo (III.1) thì chúng ta phải thực hiện 4 phép nhân các số nguyên n/2 chữ số (AC, AD, BC, BD), sau đó tổng hợp kết quả bằng 3 phép cộng các số nguyên n chữ số và 2 phép nhân với 10^n và $10^{n/2}$.

Các phép cộng các số nguyên n chữ số dĩ nhiên chỉ cần O(n). Phép nhân với 10^n có thể thực hiện một cách đơn giản bằng cách thêm vào n chữ số 0 và do đó cũng chỉ lấy O(n). Gọi T(n) là thời gian để nhân hai số nguyên, mỗi số có n chữ số thì từ (III.1) ta có:

$T(1) = 1$

$T(n) = 4T(n/2) + cn$ (III.2)

Giải (III.1) ta được $T(n) = O(n^2)$. Như vậy thì chẳng cải tiến được chút nào so với giải thuật nhân hai số bình thường. Để cải thiện tình hình, chúng ta có thể viết lại (III.1) thành dạng:

$$XY = AC10^n + [(A-B)(D-C) + AC + BD]\ 10^{n/2} + BD \qquad \text{(III.3)}$$

Công thức (III.3) chỉ đòi hỏi 3 phép nhân của các số nguyên n/2 chữ số là: AC, BD và (A-B)(D-C), 6 phép cộng trừ và 2 phép nhân với 10^n. Các phép toán này đều lấy O(n) thời gian. Từ (III.3) ta có phương trình đệ quy:

$T(1) = 1$

$T(n) = 3T(n/2) + cn$

Giải phương trình đệ quy này ta được nghiệm $T(n) = O(n^{\log 3}) = O(n^{1.59})$. Giải thuật này rõ ràng đã được cải thiện rất nhiều.

Giải thuật thô để nhân hai số nguyên (dương hoặc âm) n chữ số là:

```
function Mult(x,y:integer; n:integer) : integer;
var
        m1,m2,m3,A,B,C,D: integer;
        s:integer;{Lưu trữ dấu của tích xy}
begin
        s := sign(x)*sign(y);
        {Hàm sign có giá trị 1 nếu x dương và –1 nếu x âm}
        x := ABS(x);{Lấy trị tuyệt đối của x}
        y := ABS(y);
        if n = 1 then mult:=x*y*s
        else begin
                A := left( x, n DIV 2);
                B := right(x, n DIV 2);
                C := left(y, n DIV 2);
                D := right(y, n DIV 2);
                m1 := mult(A,C, n DIV 2);
                m2 := mult(A-B,D-C, n DIV 2);
                m3 := mult(B,D, n DIV 2);
                mult := (s * (m1 * 10n + (m1+m2+m3)* 10 n DIV 2 + m3));
        end
end;
```

III.1.4- Xếp lịch thi đấu thể thao

Kỹ thuật chia để trị không những chỉ có ứng dụng trong thiết kế giải thuật mà còn trong nhiều lĩnh vực khác của cuộc sống. Chẳng hạn xét việc xếp lịch thi đấu thể thao theo thể thức đấu vòng tròn 1 lượt cho n cầu thủ. Mỗi cầu thủ phải đấu với các cầu thủ khác, và mỗi cầu thủ chỉ đấu nhiều nhất một trận mỗi ngày. Yêu cầu là xếp một lịch thi đấu sao cho số ngày thi đấu là ít nhất. Ta dễ dàng thấy rằng tổng số trận đấu của toàn giải là n(n-1)/2. Như vậy nếu n là một số chẵn thì ta có thể sắp n/2 cặp thi đấu trong một ngày và do đó cần ít nhất n-1 ngày. Ngược lại nếu n là một số lẻ thì n-1 là một số chẵn nên ta có thể sắp (n-1)/2 cặp thi đấu trong một ngày và do đó ta cần n ngày. Giả sử $n = 2^k$ thì n là một số chẵn

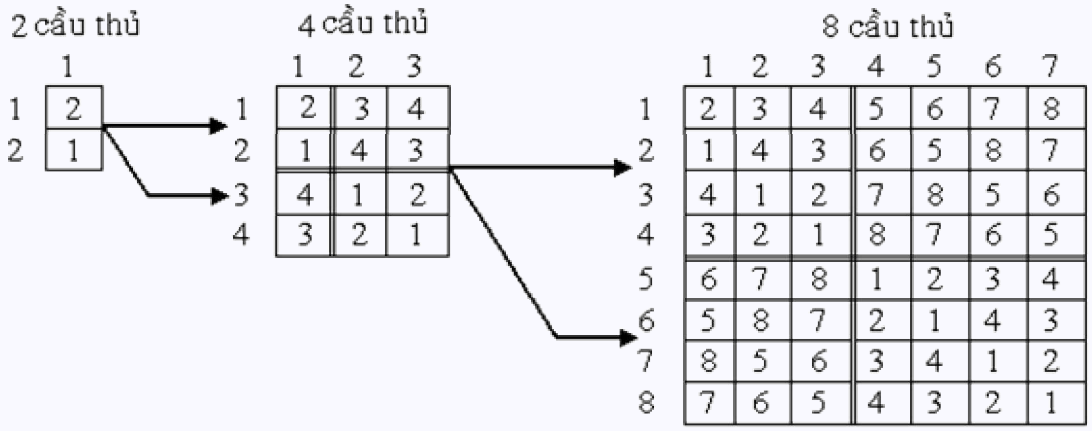
và do đó cần tối thiểu n-1 ngày.

Lịch thi đấu là một bảng n dòng và n-1 cột. Các dòng được đánh số từ 1 đến n và các cột được đánh số từ 1 đến n-1, trong đó dòng i biểu diễn cho cầu thủ i, cột j biểu diễn cho ngày thi đấu j và ô(i,j) ghi cầu thủ phải thi đấu với cầu thủ i trong ngày j.

Chiến lược chia để trị xây dựng lịch thi đấu như sau: Để sắp lịch cho n cầu thủ, ta sẽ sắp lịch cho n/2 cầu thủ, để sắp lịch cho n/2 cầu thủ, ta sẽ sắp lịch cho n/4 cầu thủ... Quá trình này sẽ dẫn đến bài toán cơ sở là sắp lịch thi đấu cho 2 cầu thủ. Hai cầu thủ này sẽ thi đấu một trận trong một ngày, lịch thi đấu cho họ thật dễ sắp. Khó khăn chính là ở chỗ từ các lịch thi đấu cho hai cầu thủ, ta tổng hợp lại để được lịch thi đấu của 4 cầu thủ, 8 cấu thủ, ...

Xuất phát từ lịch thi đấu cho hai cầu thủ ta có thể xây dựng lịch thi đấu cho 4 cầu thủ như sau: Lịch thi đấu cho 4 cầu thủ sẽ là một bảng 4 dòng, 3 cột. Lịch thi đấu cho 2 cầu thủ 1 và 2 trong ngày thứ 1 chính là lịch thi đấu của hai cầu thủ (bài toán cơ sở). Như vậy ta có Ô(1,1) = “2” và Ô(2,1) = “1”. Tương tự ta có lịch thi đấu cho 2 cầu thủ 3 và 4 trong ngày thứ 1. Nghĩa là Ô(3,1) =”4” và Ô(4,1) = “3”. (Ta cố thể thấy rằng Ô(3,1) = Ô(1,1) + 2 và Ô(4,1) = Ô(2,1) + 2). Bây giờ để hoàn thành lịch thi đấu cho 4 cầu thủ, ta lấy góc trên bên trái của bảng lắp vào cho góc dưới bên phải và lấy góc dưới bên trái lắp cho góc trên bên phải.

Lịch thi đấu cho 8 cầu thủ là một bảng gồm 8 dòng, 7 cột. Góc trên bên trái chính là lịch thi đấu trong 3 ngày đầu của 4 cầu thủ từ 1 đến 4. Các ô của góc dưới bên trái sẽ bằng các ô tương ứng của góc trên bên trái cộng với 4. Đây chính là lịch thi đấu cho 4 cầu thủ 5, 6, 7 và 8 trong 3 ngày đầu. Bây giờ chúng ta hoàn thành việc sắp lịch bằng cách lấp đầy góc dưới bên phải bởi góc trên bên trái và góc trên bên phải bởi góc dưới bên trái.



Hình 3-1: Lịch thi đấu của 2, 4 và 8 cầu thủ

III.1.5- Bài toán con cân bằng (Balancing Subproblems)

Đối với kỹ thuật chia để trị, nói chung sẽ tốt hơn nếu ta chia bài toán cần giải thành các bài toán con có kích thước gần bằng nhau. Ví dụ, sắp xếp trộn (MergeSort) phân chia bài toán thành hai bài toán con có cùng kích thước n/2 và do đó thời gian của nó chỉ là O(nlogn). Ngược lại trong trường hợp xấu nhất của QuickSort, khi mảng bị phân hoạch lệch thì thời gian thực hiện là $O(n^2)$.

Nguyên tắc chung là chúng ta tìm cách chia bài toán thành các bài toán con có kích thước xấp xỉ bằng nhau thì hiệu suất sẽ cao hơn.

III.2- QUY HOẠCH ĐỘNG

III.2.1- Nội dung kỹ thuật

Như trong III.1 chúng ta đã nói, kỹ thuật chia để trị thường dẫn chúng ta tới một giải thuật đệ quy. Trong các giải thuật đó, có thể có một số giải thuật thời gian mũ. Tuy nhiên, thường chỉ có một số đa thức các bài toán con, điều đó có nghĩa là chúïng ta đã phải giải một số bài toán con nào đó trong nhiều lần. Để tránh việc giải dư thừa một số bài toán con, chúng ta tạo ra một bảng lưu tất cả kết quả của các bài toán con và khi cần chúng ta chỉ cần tham khảo tới kết quả đã được lưu trong bảng mà **không cần phải giải lại bài toán** đó. Lấp đầy bảng kết quả các bài toán con theo một quy luật nào đó để nhận được kết quả của bài toán ban đầu (cũng đã được lưu trong một ô nào đó của bảng) được gọi là quy hoạch động.

III.2.2- Bài toán tính số tổ hợp

Một bài toán khá quen thuộc với chúng ta là tính số tổ hợp chập k của n theo công thức:

$C^k_n = 1$ nếu k=0 hoặc k = n

$C^k_n = C^{k-1}_{n-1} + C^k_{n-1}$ nếu 0 < k < n

Công thức trên đã gợi ý cho chúng ta một giải thuật đệ quy như sau:

```
function Comb(n,k : integer) : Integer;
begin
        if (k=0) or (k=n) then Comb := 1
        else Comb := Comb(n-1, k-1) + Comb(n-1,k);
end;
```

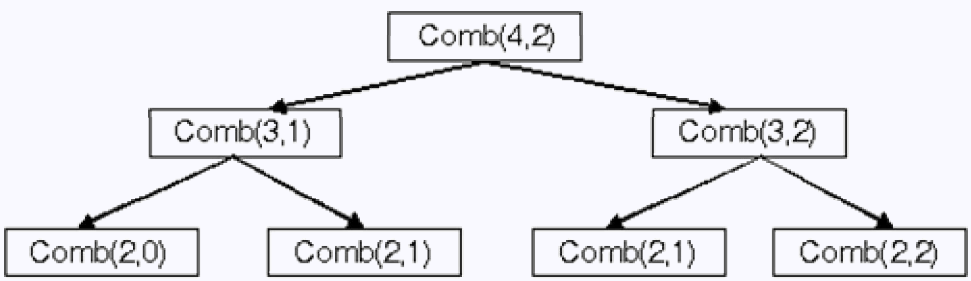
Gọi T(n) là thời gian để tính số tổ hợp chập k của n, thì ta có phương trình đệ quy:

T(1) = C1

T(n) = 2T(n-1) + C2

Giải phương trình này ta được $T(n) = O(2^n)$, như vậy là một giải thuật thời gian mũ, trong khi chỉ có một đa thức các bài toán con. Điều đó chứng tỏ rằng có những bài toán con được giải nhiều lần.

Chẳng hạn để tính Comb(4,2) ta phải tính Comb(3,1) và Comb(3,2). Để tính Comb(3,1) ta phải tính Comb(2,0) và Comb(2,1). Để tính Comb(3,2) ta phải tính Comb(2,1) và Comb(2,2). Như vậy để tính Comb(4,2) ta phải tính Comb(2,1) hai lần. Hình 3-2 sau minh hoạ rõ điều đó.



Hình 3-2: Sơ đồ gọi thực hiện Com (4,2)

Áp dụng kỹ thuật quy hoạch động để khắc phục tình trạng trên, ta xây dựng một bảng gồm n+1 dòng (từ 0 đến n) và n+1 cột (từ 0 đến n) và điền giá trị cho O(i,j) theo quy tắc sau: (Quy tắc tam giác Pascal):

O(0,0) = 1;

i \ j	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

Hình 3-3: Tam giác Pascal

O(i,0) =1;

O(i,i) = 1 với 0 < i (n;

O(i,j) = O(i-1,j-1) + O(i-1,j) với 0 < j < i £ n.

Chẳng hạn với n = 4 ta có bảng bên.

O(n,k) chính là Comb(n,k) và ta có giải thuật như sau:

```
function Comb(n, k : Integer) : Integer
var C: array[0..n, 0..n] of integer;
        i,j : integer;
begin
{1}     C[0,0] := 1;
{2}     for i := 1 to n do begin
{3}         C[i,0] := 1;
{4}         C[i,i] := 1;
{5}         for j := 1 to i-1 do C[i,j] := C[i-1,j-1] + C[i-1,j];
        end;
{6}     Comb := C[n,k];
end;
```

Vòng lặp {5} thực hiện i-1 lần, mỗi lần O(1). Vòng lặp {2} có i chạy từ 1 đến n, nên nếu gọi T(n) là thời gian thực hiện giải thuật thì ta có:

$$T(n) = \sum_{i=1}^{n} (i-1) = \frac{n(n-1)}{2} = O(n^2)$$

III.3- KỸ THUẬT "THAM ĂN"

III.3.1- Bài toán tối ưu tổ hợp

Là một dạng của bài toán tối ưu, nó có dạng tổng quát như sau:

· Cho phiếm hàm $f(X) = \sum_{i=1}^{n} c_i x_i$ xác định trên một tập hữu hạn các phần tử D. Hàm f(X) được gọi là **hàm mục tiêu**.

· Mỗi phần tử X $\in$D có dạng X= (x_1, x_2, .. x_n) được gọi là một phương án.

· Cần tìm một phương án X $\in$D sao cho hàm f(X) đạt min (max). Phương án X như thế được gọi là **phương án tối ưu**.

Ta có thể tìm thấy phương án tối ưu bằng phương pháp "vét cạn" nghĩa là xét tất cả các phương án trong tập D (hữu hạn) để xác đinh phương án tốt nhất. Mặc dù tập hợp D là hữu hạn nhưng để tìm phương án tối ưu cho một bài toán kích thước n bằng phương pháp "vét cạn" ta có thể cần một thời gian mũ.

Các phần tiếp theo của chương này sẽ trình bày một số kỹ thuật giải bài toán tối ưu tổ hợp mà thời gian có thể chấp nhận được.

III.3.2- Nội dung kỹ thuật tham ăn

Kỹ thuật tham ăn thường được vận dụng để giải bài toán tối ưu tổ hợp bằng cách xây dựng một phương án X. Phương án X được xây dựng bằng cách lựa chọn từng thành phần xi của X cho đến khi hoàn chỉnh (đủ n thành phần). Với mỗi xi , ta sẽ chọn xi tối ưu. Với cách này thì có thể ở bước cuối cùng ta không còn gì để chọn mà phải chấp nhận một giá trị cuối cùng còn lại.

Áp dụng kỹ thuật tham ăn sẽ cho một giải thuật thời gian đa thức, tuy nhiên **nói chung chúng ta chỉ đạt được một phương án tốt chứ không phải là tối ưu**.

Có rất nhiều bài toán mà ta có thể giải bằng kỹ thuật này, sau đây là một số ví dụ.

III.3.3- Bài toán đường đi của người giao hàng

Chúng ta sẽ xét một bài toán rất nổi tiếng có tên là bài toán tìm đường đi của người giao hàng (TSP - Traveling Salesman Problem): Có một người giao hàng cần đi giao hàng tại n thành phố. Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, khoảng cách từ một thành phố đến các thành phố khác là xác định được. Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, có thể là cước phí di chuyển hoặc thời gian di chuyển. Ta gọi chung là độ dài. Hãy tìm một chu trình (một đường đi khép kín thỏa mãn điều kiện trên) sao cho tổng độ dài các cạnh là nhỏ nhất. Hay còn nói là tìm một phương án có giá nhỏ nhất. Bài toán này cũng được gọi là bài toán người du lịch.

Với phương pháp vét cạn ta xét tất cả các chu trình, mỗi chu trình tính tổng độ dài các cạnh của nó rồi chọn một chu trình có tổng độ dài nhỏ nhất. Tuy nhiên chúng ta cần xét tất cả là $\frac{(n-1)!}{2}$ chu trình. Thực vậy, do mỗi chu trình đều đi qua tất cả các đỉnh (thành phố) nên ta có thể cố định một đỉnh. Từ đỉnh này ta có n-1 cạnh tới n-1 đỉnh khác, nên ta có n-1 cách chọn cạnh đầu tiên của chu trình. Sau khi đã chọn được cạnh đầu tiên, chúng ta còn n-2 cách chọn cạnh thứ hai, do đó ta có (n-1)(n-2) cách

chọn hai cạnh. Cứ lý luận như vậy ta sẽ thấy có (n-1)! cách chọn một chu trình. Tuy nhiên với mỗi chu trình ta chỉ quan tâm đến tổng độ dài các cạnh chứ không quan tâm đến hướïng đi theo chiều dương hay âm vì vậy có tất cả $\frac{(n-1)!}{2}$ phương án. Đó là một giải thuật thời gian mũ !!!.

Kỹ thuật tham ăn áp dụng vào đây là:

1. Tính độ dài của tất cả các cạnh (có tất cả $\frac{n(n-1)}{2}$ cạnh).

2. Xét các cạnh có độ dài từ nhỏ đến lớn để đưa vào chu trình.

3. Một cạnh sẽ được đưa vào chu trình nếu cạnh đó thỏa mãn hai điều kiện sau:

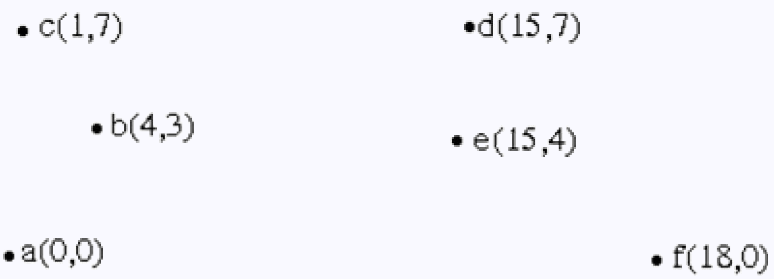
· Không tạo thành một chu trình thiếu (không đi qua đủ n đỉnh)

· Không tạo thành một đỉnh có cấp ≥ 3 (tức là không được có nhiều hơn hai cạnh xuất phát từ một đỉnh, do yêu cầu của bài toán là mỗi thành phố chỉ được đến một lần: một lần đến và một lần đi)

4. Lặp lại bước 3 cho đến khi xây dựng được một chu trình.

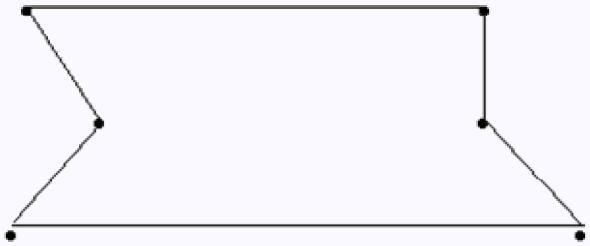
Với kỹ thuật này ta chỉ cần n(n-1)/2 phép chọn nên ta có một giải thuật cần $O(n^2)$ thời gian.

Ví dụ 3-1: Cho bài toán TSP với 6 đỉnh được cho bởi các tọa độ như sau:



Hình 3-4: Sáu thành phố được cho bởi toạ độ

Do có 6 đỉnh nên có tất cả 15 cạnh. Đó là các cạnh: ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df và ef. Độ dài các cạnh ở đây là khoảng cách Euclide. Trong 15 cạnh này thì de = 3 là nhỏ nhất, nên de được chọn vào chu trình. Kế đến là 3 cạnh ab, bc và ef đều có độ dài là 5. Cả 3 cạnh đều thỏa mãn hai điều kiện nói trên, nên đều được chọn vào chu trình. Cạnh có độ dài nhỏ kế tiếp là ac = 7.08, nhưng không thể đưa cạnh này vào chu trình vì nó sẽ tạo ra chu trình thiếu (a-b-c-a). Cạnh df cũng bị loại vì lý do tương tự. Cạûnh be được xem xét nhưng rồi cũng bị loại do tạo ra đỉnh b và đỉnh e có cấp 3. Tương tự chúng ta cũng loại bd. cd là cạnh tiếp theo được xét và được chọn. Cuối cùng ta có chu trình a-b-c-d-e-f-a với tổng độ dài là 50.



Hình 3-5:

Đây chỉ là một phương án tốt.

Phương án tối ưu là chu trình a-c-d-e-f-b-a với tổng độ dài là 48.39.

Giải thuật sơ bộ như sau:

```
procedure TSP;
begin
        {E là tập các cạnh }
        {Sắp xếp các cạnh trong E theo thứ tự tăng của độ dài }
        Chu_Trinh := F;
        Gia := 0.0;
        while E <> F do begin
            if cạnh e có thể chọn then begin
                Chu_Trinh := Chu_Trinh + e ;
                Gia := Gia + độ dài của e;
            end;
            E := E-e;
        end ;
end;
```

Một cách tiếp cận khác của kỹ thuật tham ăn vào bài toán này là:

1. Xuất phát từ một đỉnh bất kỳ, chọn một cạnh có độ dài nhỏ nhất trong tất cả các cạnh đi ra từ đỉnh đó để đến đỉnh kế tiếp.

2. Từ đỉnh kế tiếp ta lại chọn một cạnh có độ dài nhỏ nhất đi ra từ đỉnh này thoả mãn hai điều kiện nói trên để đi đến dỉnh kế tiếp.

3. Lặp lại bước 2 cho đến khi đi tới đỉnh n thì quay trở về đỉnh xuất phát.

III.3.4- Bài toán cái ba lô

Cho một cái ba lô có thể đựng một trọng lượng W và n loại đồ vật, mỗi đồ vật i có một trọng lượng gi và một giá trị vi. Tất cả các loại đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá W và tổng giá trị là lớn nhất.

Theo yêu cầu của bài toán thì ta cần những đồ vật có giá trị cao mà trọng lượng lại nhỏ để sao cho có thể mang được nhiều "đồ quý", sẽ là hợp lý khi ta quan tâm đến yếu tố "đơn giá" của từng loại đồ vật tức là tỷ lệ giá trị/trọng lượng. Đơn giá càng cao thì đồ càng quý. Từ đó ta có kỹ thuật greedy áp

dụng cho bài toán này là:

1. Tính đơn giá cho các loại đồ vật.

2. Xét các loại đồ vật theo thứ tự đơn giá từ lớn đến nhỏ.

3. Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà trọng lượng còn lại của ba lô cho phép.

4. Xác định trọng luợng còn lại của ba lô và quay lại bước 3 cho đến khi không còn có thể chọn được đồ vật nào nữa.

Ví dụ 3-2: Ta có một ba lô có trọng lượng lài 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng (hình 3-6)

Loại đồ vật	Trọng lượng	Giá trị
A	15	30
B	10	25
C	2	2
D	4	6

Hình 3-6

Từ bảng đã cho ta tính đơn giá cho các loại đồ vật và sắp xếp các loại đồ vật này theo thứ tự đơn giá giảm dần ta có bảng (hình 3-7).

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

Hình 3-7

Theo đó thì thứ tự ưu tiên để chọn đồ vật là là B, A, D và cuối cùng là C.

Vật B được xét đầu tiên và ta chọn tối đa 3 cái vì mỗi cái vì trọng lượng mỗi cái là 10 và ba lô có trọng lượng 37. Sau khi đã chọn 3 vât loại B, trọng lượng còn lại trong ba lô là 37 - 3*10 = 7. Ta xét đến vật A, vì A có trọng lượng 15 mà trọng lượng còn lại của balô chỉ còn 7 nên không thể chọn vật A. Xét vật D và ta thấy có thể chọn 1 vật D, khi đó trọng lượng còn lại của ba lô là 7-4 = 3. Cuối cùng ta chọn được một vật C.

Như vậy chúng ta đã chọn 3 cái loại B, một cái loại D và 1 cái loại C. Tổng trọng lương là 3*10 + 1*4 + 1*2 = 36 và tổng giá trị là 3*25+1*6+1*2 = 83.

Chú ý có một số biến thể của bài toán cái ba lô như sau:

1. Mỗi đồ vật i chỉ có một số lượng si. Với bài toán này khi lựa chọn vật i ta không được lấy một số lượng vượt quá si.

2. Mỗi đồ vật chỉ có một cái. Với bài toán này thì với mỗi đồ vật ta chỉ có thể chọn hoặc không chọn.

III.4- KỸ THUẬT QUAY LUI

III.4.1- Định trị cây biểu thức số học
III.4.2- Kỹ thuật cắt tỉa Alpha-Beta
III.4.3- Kỹ thuật nhánh cận

Kỹ thuật quay lui (backtracking) như tên gọi của nó, là một quá trình phân tích đi xuống. Tại mỗi bước phân tích chúng ta chưa giải quyết được vấn đề do còn thiếu cứ liệu nên cứ phải phân tích cho tới các điểm dừng, nơi chúng ta xác định được lời giải của chúng hoặc là xác định được là không thể (hoặc không nên) tiếp tục theo hướng này. Từ các điểm dừng này chúng ta quay ngược trở lại theo con đường mà chúng ta đã đi qua để giải quyết các vấn đề còn tồn đọng và cuối cùng ta sẽ giải quyết được vấn đề ban đầu.

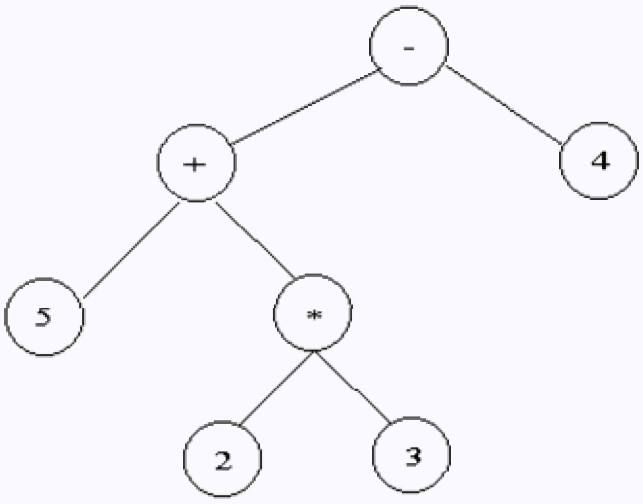
Ở đây chúng ta sẽ xét 3 kỹ thuật quay lui: “vét cạn” là kỹ thuật phải đi tới tất cả các điểm dừng rồi mới quay lui. “Cắt tỉa Alpha-Beta” và “Nhánh-Cận” là hai kỹ thuật cho phép chúng ta không cần thiết phải đi tới tất cả các điểm dừng, mà chỉ cần đi đến một số điểm nào đó và dựa vào một số suy luận để có thể quay lui sớm. Các kỹ thuật này sẽ được trình bày thông qua một số bài toán cụ thể sau.

III.4.1- Định trị cây biểu thức số học

Trong các ngôn ngữ lập trình đều có các biểu thức số học, việc dịch các biểu thức này đòi hỏi phải đánh giá (định trị) chúng. Để làm được điều đó cần phải có một biểu diễn trung gian cho biểu thức. Một trong các biểu diễn trung gian cho biểu thức là cây biểu thức.

Cây biểu thức số học là một cây nhị phân, trong đó các nút lá biểu diễn cho các toán hạng, các nút trong biểu diễn cho các toán tử.

Ví dụ 3-3: Biểu thức 5 + 2 * 3 - 4 sẽ được biểu diễn bởi cây trong hình 3-8:



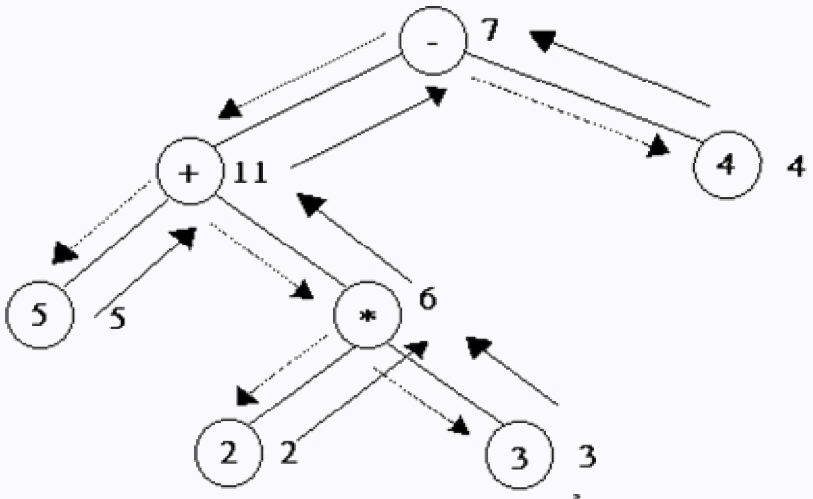
Hình 3-8: Một cây biểu thức số học

Trị của một nút lá chính là trị của toán hạng mà nút đó biểu diễn. Trị của một nút trong có được bằng cách lấy toán tử mà nút đó biểu diễn áp dụng vào các con của nó.

Trị của nút gốc chính là trị của biểu thức.

Như vậy để định trị cho nút gốc, chúng ta phải định trị cho hai con của nó, đối với mỗi con ta xem nó có phải là nút lá hay không, nếu không phải ta lại phải xét hai con của nút đó. Quá trình cứ tiếp tục như vậy cho tới khi gặp các nút lá mà giá trị của chúng đã được biết, quay lui để định trị cho các nút cha của các nút lá và cứ như thế mà định trị cho tổ tiên của chúng. Đó chính là kỹ thuật quay lui vét cạn, vì chúng ta phải lần đến tất cả các nút lá mới định trị được cho các nút trong và do thế mới định trị được cho nút gốc.

Ví dụ 3-4: Với cây biểu thức trong ví dụ 3-3. Để định trị cho nút - chúng ta phải định trị cho nút + và nút 4. Nút 4 là nút lá nên giá trị của nó là 4. Để định trị cho nút + ta phải định trị cho nút 5 và nút *. Nút 5 là nút lá nên giá trị của nó là 5. Để định trị cho nút *, ta phải định trị cho nút 2 và nút 3. Cả hai nút này đều là lá nên giá trị của chúng tương ứng là 2 và 3. Quay lui lại nút *, lấy toán tử * áp dụng cho hai con của nó là 2 và 3 ta được trị của nút * là 6. Quay lui về nút +, lại áp dụng toán tử + vào hai con của nó là 5 và 6 được trị của nút + là 11. Cuối cùng quay về nút -, áp dụng toán tử - vào hai con của nó là 11 và 4 ta được trị của nút - (nút gốc) là 7. Đó chính là trị của biểu thức. Trong hình 3-9, mũi tên nét đứt minh họa quá trình đi tìm nút lá và mũi tên nét liền minh họa quá trình quay lui để định trị cho các nút, các số bên phải mỗi nút là trị của nút đó.



Hình 3-9: Quy trình định trị cây biểu thức SH

Giải thuật sơ bộ để định trị một nút bất kỳ như sau:

```
function Eval(n : node): real;
begin
        if n là lá then return (trị của toán hạng trong n)
        else return (Toán tử trong n (Eval (Con trái của n), Eval (Con phải của n)) );
end;
```

Muốn định trị cho cây biểu thức T, ta gọi Eval(ROOT(T)).

III.4.2- Kỹ thuật cắt tỉa Alpha-Beta

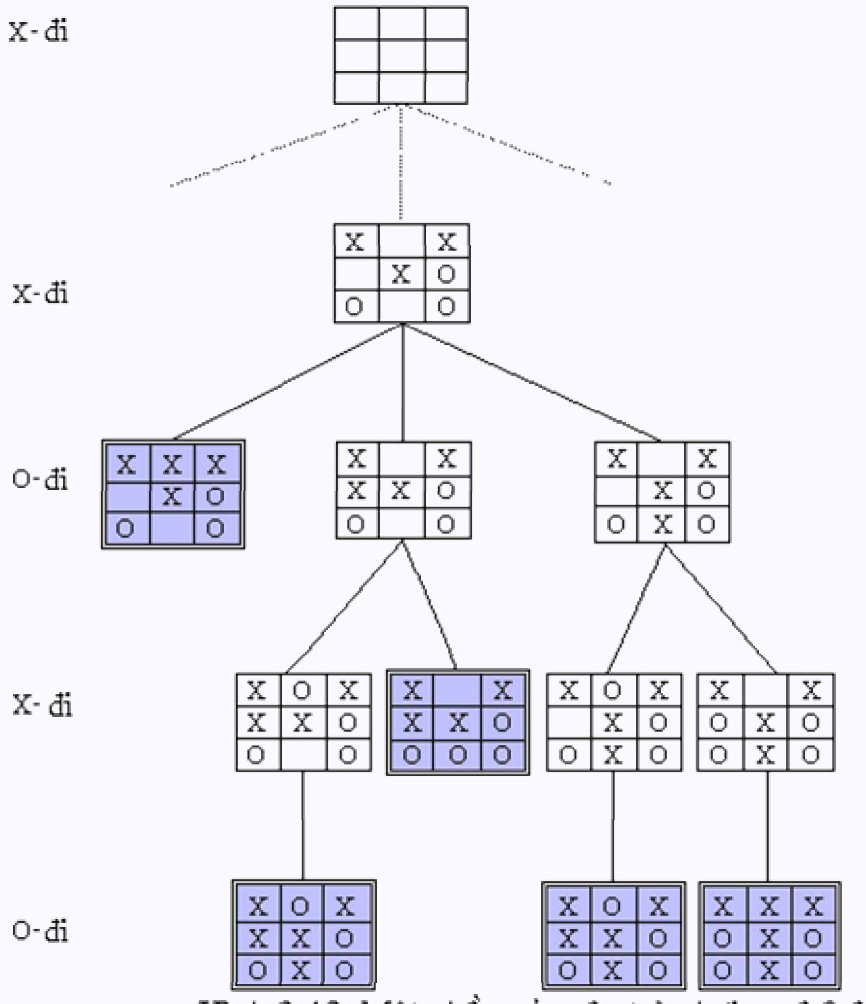
Cây trò chơi

Xét một trò chơi trong đó hai người thay phiên nhau đi nước của mình như cờ vua, cờ tướng, carô,... Trò chơi có một trạng thái bắt đầu và mỗi nước đi sẽ biến đổi trạng thái hiện hành thành một trạng thái mới. Trò chơi sẽ kết thúc theo một quy định nào đó, theo đó thì cuộc chơi sẽ

dẫn đến một trạng thái phản ánh có một người thắng cuộc hoặc một trạng thái mà cả hai đấu thủ không thể phát triển được nước đi của mình, ta gọi nó là trạng thái hòa cờ. Ta tìm cách phân tích xem từ một trạng thái nào đó sẽ dẫn đến đấu thủ nào sẽ thắng với điều kiện cả hai đấu thủ đều có trình độ như nhau.

Một trò chơi như vậy có thể được biểu diễn bởi một cây, gọi là cây trò chơi. Mỗi một nút của cây biểu diễn cho một trạng thái. Nút gốc biểu diễn cho trạng thái bắt đầu của cuộc chơi. Mỗi nút lá biểu diễn cho một trạng thái kết thúc của trò chơi (trạng thái thắng thua hoặc hòa). Nếu trạng thái x được biểu diễn bởi nút n thì các con của n biểu diễn cho tất cả các trạng thái kết quả của các nước đi có thể xuất phát từ trạng thái x.

Ví dụ 3-5: Xét trò chơi carô có 9 ô. Hai người thay phiên nhau đi X hoặc O. Người nào đi được 3 ô thẳng hàng (ngang, dọc, xiên) thì thắng cuộc. Nếu đã hết ô đi mà chưa phân thắng bại thì hai đấu thủ hòa nhau. Một phần của trò chơi này được biểu diễn bởi cây sau:



Hình 3-10: Một phần của cây trò chơi carô 9 ô

Trong cây trò chơi trên, các nút lá được tô nền và viền khung đôi để dễ phân biệt với các nút khác. Ta có thể gán cho mỗi nút lá một giá trị để phản ánh trạng thái thắng thua hay hòa của các đấu thủ. Chẳng hạn ta gán cho nút lá các giá trị như sau:

· 1 nếu tại đó người đi X đã thắng,

· -1 nếu tại đó người đi X đã thua và

· 0 nếu hai đấu thủ đã hòa nhau.

Như vậy từ một trạng thái bất kỳ, đến lượt mình, người đi X sẽ chọn cho mình một nước đi sao cho dẫn đến trạng thái có giá trị lớn nhất (trong trường hợp này là 1). Ta nói X chọn nước đi MAX, nút mà từ đó X chọn nước đi của mình được gọi là nút MAX. Người đi O đến lượt mình sẽ chọn một nước đi sao cho dẫn đến trạng thái có giá trị nhỏ nhất (trong trường hợp này là -1, khi đó X sẽ thua và do đó O sẽ thắng). Ta nói O chọn nước đi MIN, nút mà từ đó O chọn nước đi của mình được gọi là nút MIN. Do hai đấu thủ luân phiên nhau đi nước của mình nên các mức trên cây trò chơi cũng luân phiên nhau là MAX và MIN. Cây trò chơi vì thế còn có tên là cây MIN-MAX. Ta có thể đưa ra một quy tắc định trị cho các nút trên cây để phản ánh tình trạng thắng thua hay hòa và khả năng thắng cuộc của hai đấu thủ.

Nếu một nút là nút lá thì trị của nó là giá trị đã được gán cho nút đó. Ngược lại, nếu nút là nút MAX thì trị của nó bằng giá trị lớn nhất của tất cả các trị của các con của nó. Nếu nút là nút MIN thì trị của nó là giá trị nhỏ nhất của tất cả các trị của các con của nó.

Quy tắc định trị này cũng gần giống với quy tắc định trị cho cây biểu thức số học, điểm khác biệt ở đây là các toán tử là các hàm lấy max hoặc min và mỗi nút có thể có nhiều con. Do vậy ta có thể dùng kỹ thuật quay lui để định trị cho các nút của cây trò chơi.

Để cài đặt ta có một số giả thiết sau:

· Ta có một hàm Payoff nhận vào một nút lá và cho ta giá trị của nút lá đó.

· Các hằng ∞ và -∞ tương ứng là các trị Payoff lớn nhất và nhỏ nhất.

· Khai báo kiểu ModeType = (MIN, MAX) để xác định định trị cho nút là MIN hay MAX.

· Một kiểu NodeType được khai báo một cách thích hợp để biểu diễn cho một nút trên cây phản ánh một trạng thái của cuộc chơi.

· Ta có một hàm is_leaf để xác định xem một nút có phải là nút lá hay không?

· Hàm max và min tương ứng lấy giá trị lớn nhất và giá trị nhỏ nhất của hai giá trị.

Giải thuật vét cạn định trị cây trò chơi

Hàm Search nhận vào một nút n và kiểu mode của nút đó (MIN hay MAX) trả về giá trị của nút.

Nếu nút n là nút lá thì trả về giá trị đã được gán cho nút lá. Ngược lại ta cho n một giá trị tạm value là -∞ hoặc ∞ tùy thuộc n là nút MAX hay MIN và xét con của n. Sau khi một con của n có giá trị V thì đặt lại value = max(value,V) nếu n là nút MAX và value = min(value,V) nếu n là nút MIN. Khi tất cả các con của n đã được xét thì giá trị tạm value của n trở thành giá trị của nó.

```
function Search(n : NodeType; mode: ModeType): real;
var C : NodeType ; { C là một nút con của nút n}
    Value : real;
{Lúc đầu ta cho value một giá trị tạm, sau khi đã xét hết tất cả các con của nút n thì value là giá
```

```
trị của nút n }
begin
        if is_leaf(n) then return ( Payoff(n) )
        else begin
                {Khởi tạo giá trị tạm cho n }
                if mode = MAX then value := -∞ else value := ∞;

{Xét tất cả các con của n, mỗi lần xác định được giá trị của một nút con, ta phải đặt lại giá trị
tạm value. Khi đã xét hết tất cả các con thì value là giá trị của n}

        for với mỗi con C của n do
                    if mode = MAX then
                    Value := max(Value, Search(C, MIN) )
                    else Value := min(Value, Search(C, MAX) );
                return (value);
        end;
end;
```

Kỹ thuật cắt tỉa Alpha-Beta (Alpha-Beta Pruning)

Trong giải thuật vét cạn ở trên, ta thấy để định trị cho một nút nào đó, ta phải định trị cho tất cả các nút con cháu của nó, và muốn định trị cho nút gốc ta phải định trị cho tất cả các nút trên cây. Số lượng các nút trên cây trò chơi tuy hữu hạn nhưng không phải là ít. Chẳng hạn trong cây trò chơi ca rô nói trên, nếu ta có bàn cờ bao gồm n ô thì có thể có tới n! nút trên cây (trong trường hợp trên là 9!). Đối với các loại cờ khác như cờ vua chẳng hạn, thì số lượng các nút còn lớn hơn nhiều. Ta gọi là một sự bùng nổ tổ hợp các nút.

Chúng ta cố gắng tìm một cách sao cho khi định trị một nút thì không nhất thiết phải định trị cho tất cả các nút con cháu của nó. Trước hết ta có nhận xét như sau:

Nếu P là một nút MAX và ta đang xét một nút con Q của nó (dĩ nhiên Q là nút MIN). Giả sử Vp là một giá trị tạm của P, Vq là một giá trị tạm của Q và nếu ta có $Vp \geq Vq$ thì ta không cần xét các con chưa xét của Q nữa. Vì nếu có xét thì giá trị của Q cũng sẽ nhỏ hơn hoặc bằng Vq và do đó không ảnh hưởng gì đến Vp. Tương tự nếu P là nút MIN (tất nhiên Q là nút MAX) và $Vp \leq Vq$ thì ta cũng không cần xét đến các con chưa xét của Q nữa. Việc không xét tiếp các con chưa được xét của nút Q gọi là việc cắt tỉa Alpha-Beta các con của nút Q.

Trên cơ sở nhận xét đó, ta nêu ra quy tắc định trị cho một nút không phải là nút lá trên cây như sau:

1. Khởi đầu nút MAX có giá trị tạm là -∞ và nút MIN có giá trị tạm là ∞.

2. Nếu tất cả các nút con của một nút đã được xét hoặc bị cắt tỉa thì giá trị tạm của nút đó trở thành giá trị của nó.

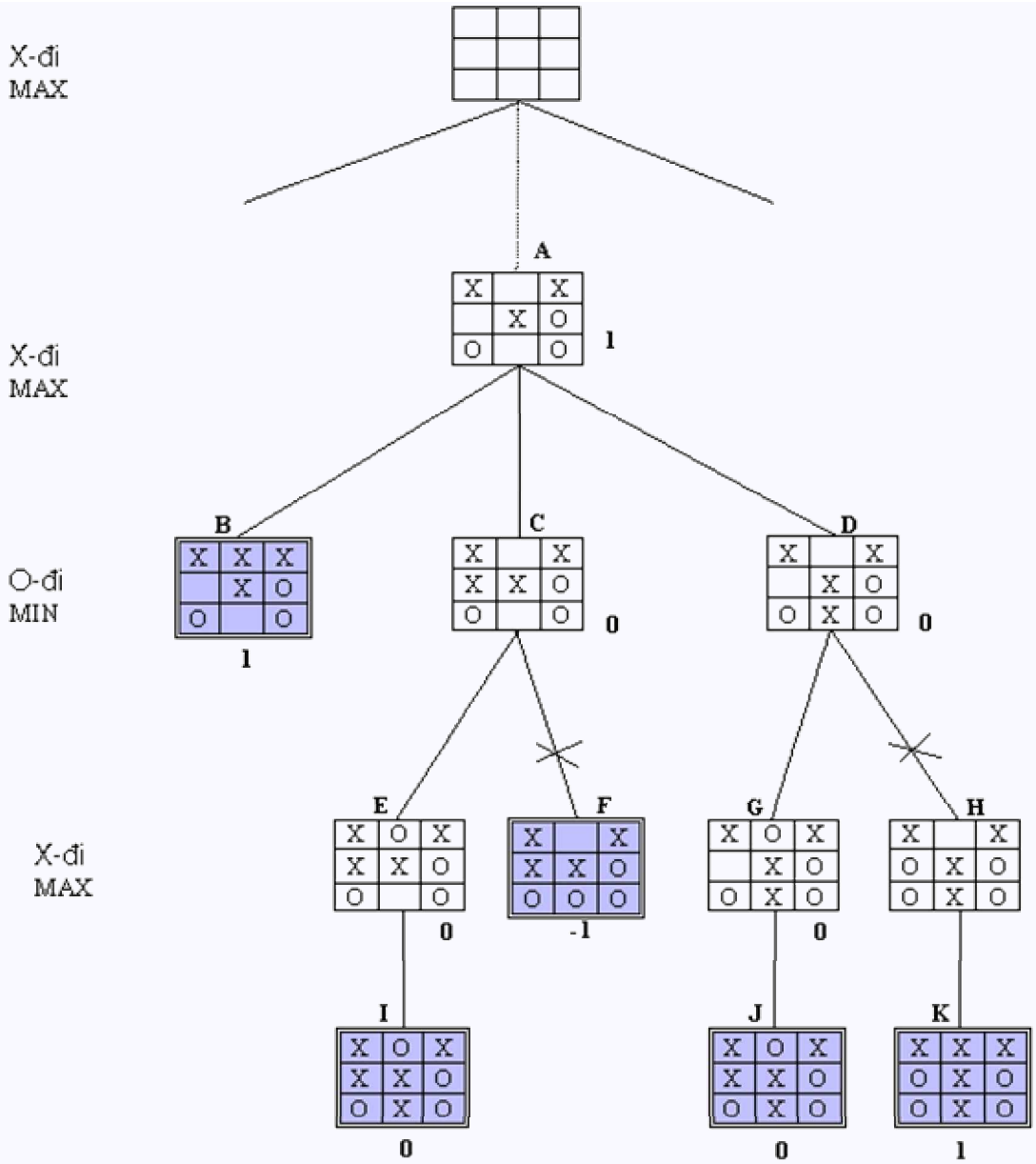
3. Nếu một nút MAX n có giá trị tạm là V1 và một nút con của nó có giá trị là V2 thì đặt giá trị tạm mới của n là max (V1,V2). Nếu n là nút MIN thì đặt giá trị tạm mới của n là min (V1,V2).

4. Vận dụng quy tắc cắt tỉa Alpha-Beta nói trên để hạn chế số lượng nút phải xét.

```
function cat_tia(Q: NodeType; mode: ModeType; Vp: real): real;
var C : NodeType ; { C là một nút con của nút Q}
        Vq : real;
{Vq là giá trị tạm của Q, sau khi tất cả các con của nút Q đã xét hoặc bị cắt tỉa thì Vq là giá trị của nút Q}
begin
        if is_leaf(Q) then return ( Payoff(Q) )
        else begin
                { Khởi tạo giá trị tạm cho Q }
                if mode = MAX then Vq := -∞ else Vq := ∞;
{Xét các con của Q, mỗi lần xác định được giá trị của một nút con của Q, ta phải đặt lại giá trị tạm Vq và so sánh với Vp để có thể cắt tỉa hay không}
                Xét C là con trái nhất của Q;
        while C là con của Q DO
                        if mode = MAX then begin
                        Vq:= max(Vq, Cat_tia(C, MIN, Vq));
                        if Vp ≤ Vq then return(Vq);
                end
                        else begin Vq := min(Vq, Cat_tia(C, MAX, Vq));
                                if Vp ≥ Vq then return(Vq);
                            end;
                return (Vq);
        end;
end;
```

Ví dụ 3-6: Vận dụng quy tắc trên để định trị cho nút A trong cây trò chơi trong ví dụ 3-5.

A là nút MAX, lúc đầu nó có giá trị tạm là -∞, xét B là con của A, B là nút lá nên giá trị của nó là giá trị đã được gán 1, giá trị tạm của A bây giờ là max (-∞,1) = 1. Xét con C của A, C là nút MIN, giá trị tạm lúc đầu của C là ∞. Xét con E của C, E là nút MAX, giá trị tạm của E là -∞. Xét con I của E, I là nút lá nên giá trị của nó là 0. Quay lui lại E, giá trị tạm của E bây giờ là max (-∞,0) = 0. Vì E chỉ có một con là I đã xét nên giá trị tạm 0 trở thành giá trị của E. Quay lui lại C, giá trị tạm mới của C là min (∞,0) = 0. A là nút MAX có giá trị tạm là 1, C là con của A, có giá trị tạm là 0, 1>0 nên ta không cần xét con F của C nữa. Nút C có hai con là E và F, trong đó E đã được xét, F đã bị cắt, vậy giá trị tạm 0 của C trở thành giá trị của nó. Sau khi có giá trị của C, ta phải đặt lại giá trị tạm của A, nhưng giá trị tạm này không thay đổi vì max (1,0) = 1. Tiếp tục xét nút D, D là nút MIN nên giá trị tạm là ∞, xét nút con G của D, G là nút MAX nên giá trị tạm của nó là -∞, xét nút con J của G. Vì J là nút lá nên có giá trị 0. Quay lui lại G, giá trị tạm của G bây giờ là max (-∞,0) = 0 và giá trị tạm này trở thành giá trị của G vì G chỉ có một con J đã xét. Quay lui về D, giá trị tạm của D bây giờ là min (∞,0) = 0. Giá trị tạm này của D nhỏ hơn giá trị tạm của nút A MAX là cha của nó nên ta cắt tỉa con H chưa được xét của D và lúc này D có giá trị là 0. Quay lui về A, giá trị tạm của nó vẫn không thay đổi, nhưng lúc này cả 3 con của A đều đã được xét nên giá trị tạm 1 trở thành giá trị của A. Kết quả được minh họa trong hình sau:



Hình 3-11: Định trị cây trò chơi bằng kỹ thuật cắt tỉa alpha-beta

III.4.3-Kỹ thuật nhánh cận

Với các bài toán tìm phương án tối ưu, nếu chúng ta xét hết tất cả các phương án thì mất rất nhiều thời gian, nhưng nếu sử dụng phương pháp tham ăn thì phương án tìm được chưa hẳn đã là phương án tối ưu. Nhánh cận là kỹ thuật xây dựng cây tìm kiếm phương án tối ưu, nhưng không xây dựng toàn bộ cây mà sử dụng giá trị cận để hạn chế bớt các nhánh.

Cây tìm kiếm phương án có nút gốc biểu diễn cho tập tất cả các phương án có thể có, mỗi nút lá biểu diễn cho một phương án nào đó. Nút n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n. Kỹ thuật này gọi là phân nhánh.

Với mỗi nút trên cây ta sẽ xác định một giá trị cận. Giá trị cận là một giá trị gần với giá của các phương án. Với bài toán tìm **min** ta sẽ xác định **cận dưới** còn với bài toán tìm **max** ta sẽ xác định **cận trên**. Cận dưới là giá trị nhỏ hơn hoặc bằng giá của phương án, ngược lại cận trên là giá trị lớn hơn hoặc bằng giá của phương án.

Để dễ hình dung ta sẽ xét hai bài toán quen thuộc là bài toán TSP và bài toán cái ba lô.

Bài toán đường đi của người giao hàng
Phân nhánh

Cây tìm kiếm phương án là cây nhị phân trong đó:

· Nút gốc là nút biểu diễn cho cấu hình bao gồm tất cả các phương án.

· Mỗi nút sẽ có hai con, con trái biểu diễn cho cấu hình bao gồm tất cả các phương án chứa một cạnh nào đó, con phải biểu diễn cho cấu hình bao gồm tất cả các phương án không chứa cạnh đó (các cạnh để xét phân nhánh được thành lập tuân theo một thứ tự nào đó, chẳng hạn thứ tự từ điển).

· Mỗi nút sẽ kế thừa các thuộc tính của tổ tiên của nó và có thêm một thuộc tính mới (chứa hay không chứa một cạnh nào đó).

· Nút lá biểu diễn cho một cấu hình chỉ bao gồm một phương án.

· Để quá trình phân nhánh mau chóng tới nút lá, tại mỗi nút ta cần có một quyết định bổ sung dựa trên nguyên tắc là mọi đỉnh trong chu trình đều có cấp 2 và không tạo ra một chu trình thiếu.

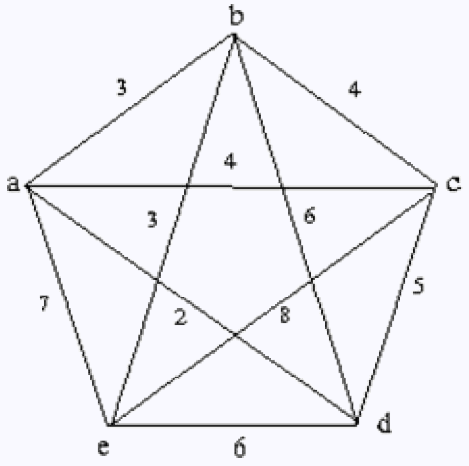
Ví dụ 3-7: Xét bài toán TSP có 5 đỉnh với độ dài các cạnh được cho trong hình 3-12.

Các cạnh theo thứ tự từ điển để xét là:

ab, ac, ad, ae, bc, bd, be, cd, ce và de.

Nút gốc A của cây bao gồm tất cả các phương án.

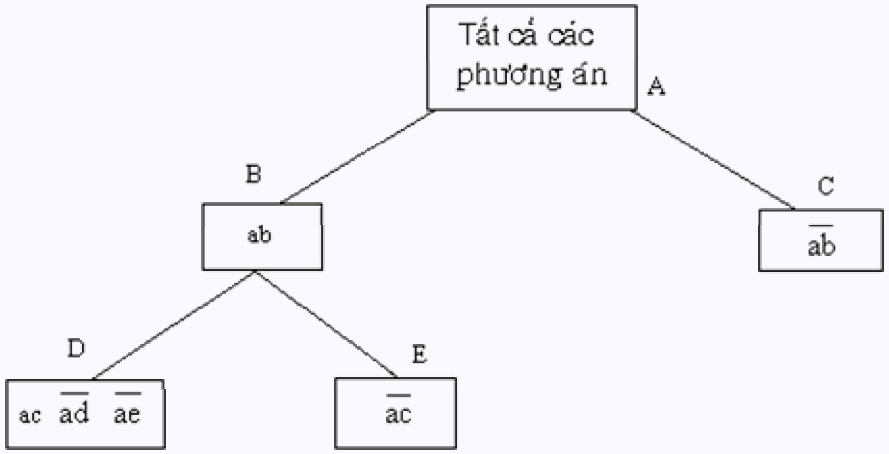
Hai con của A là B và C, trong đó B bao gồm tấtcả các phương án chứa cạnh ab, C bao gồm tất cả các phương án không chứa ab.



Hình 3-12

Hai con của B là D và E. Nút D bao gồm tất cả các phương án chứa ac. Vì các phương án này vừa chứa ab (kế thừa của B) vừa chứa ac nên đỉnh a đã đủ cấp hai nên D không thể chứa ad và ae. Nút E bao gồm tất cả các phương án không chứa ac..

Ta được cây (chưa đầy đủ) trong hình 3-13



Hình 3-13

Trong đó ký hiệu $\overline{ab}$có nghĩa là không chứa cạnh ab.

Tính cận dưới

Đây là bài toán tìm min nên ta sử dụng **cận dưới**. Cận dưới tại mỗi nút là một số nhỏ hơn hoặc bằng giá của tất cả các phương án được biểu diễn bởi nút đó. Giá của một phương án ở đây là tổng độ dài của một chu trình.

Để tính cận dưới của nút gốc, mỗi đỉnh ta chọn hai cạnh có độ dài nhỏ nhất. Cận dưới của nút gốc bằng tổng độ dài tất cả các cạnh được chọn chia cho 2.

Ví dụ 3-8: Với số liệu cho trong ví dụ 3-7 nói trên, ta tính cận dưới của nút gốc A (hình 3-13) như sau:

· Đỉnh a chọn ad = 2, ab = 3

· Đỉnh b chọn ba = 3, be = 3

· Đỉnh c chọn ca = 4, cb = 4

· Đỉnh d chọn da = 2, dc = 5

· Đỉnh e chọn eb = 3, ed = 6

Tổng độ dài các cạnh được chọn là= 35, cận dưới của nút gốc A là 35/2 = 17.5

Đối với các nút khác, chúng ta phải lựa chọn hai cạnh có độ dài nhỏ nhất thỏa điều kiện ràng buộc (phải chứa cạnh này, không chứa cạnh kia).

Ví dụ 3-9: Tính cận dưới cho nút D trong hình 3-13. Điều kiện ràng buộc là phải chứa ab, ac và không chứa ad,ae.

· Đỉnh a chọn ab = 3, ac = 4, do hai cạnh này buộc phải chọn.

· Đỉnh b chọn ba = 3, be = 3

· Đỉnh c chọn ca = 4, cb = 4

· Đỉnh d chọn de = 6, dc = 5, do không được chọn da nên ta phải chọn de.

· Đỉnh e chọn eb = 3, ed = 6

Tổng độ dài các cạnh được chọn là 41, cận dưới của nút D là 41/2 = 20.5

Kỹ thuật nhánh cận

Bây giờ ta sẽ kết hợp hai kỹ thuật trên để xây dựng cây tìm kiếm phương án. Quy tắc như sau:

· Xây dựng nút gốc, bao gồm tất cả các phương án, tính cận dưới cho nút gốc.

· Sau khi phân nhánh cho mỗi nút, ta tính cận dưới cho cả hai con.

· Nếu cận dưới của một nút con lớn hơn hoặc bằng **giá nhỏ nhất tạm thời** của một phương án đã được tìm thấy thì ta không cần xây dựng các cây con cho nút này nữa (Ta gọi là cắt tỉa các cây con của nút đó).

· Nếu cả hai con đều có cận dưới nhỏ hơn giá nhỏ nhất tạm thời của một phương án đã được tìm thấy thì nút con nào có cận dưới nhỏ hơn sẽ được ưu tiên phân nhánh trước.

· Mỗi lần quay lui để xét nút con chưa được xét của một nút ta phải xem xét lại nút con đó để có thể cắt tỉa các cây của nó hay không vì có thể một phương án có giá nhỏ nhất tạm thời vừa được tìm thấy.

· Sau khi tất cả các con đã được phân nhánh hoặc bị cắt tỉa thì phương án có giá nhỏ nhất trong các phương án tìm được là phương án cần tìm.

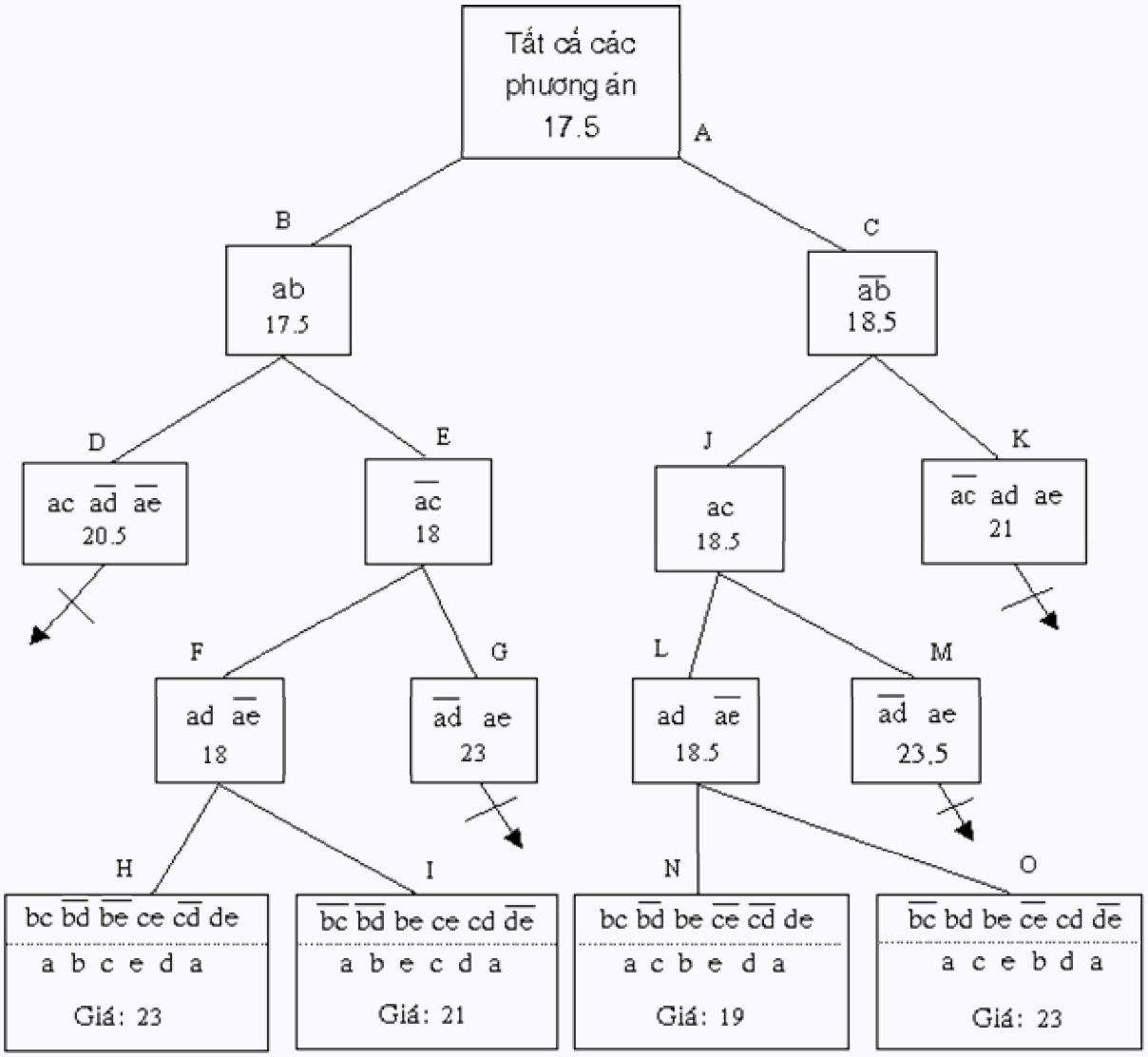
Trong quá trình xây dựng cây có thể ta đã xây dựng được một số nút lá, như ta biết mỗi nút lá biểu diễn cho một phương án. Giá nhỏ nhất trong số các giá của các phương án này được gọi là giá nhỏ nhất tạm thời.

Ví dụ 3-10: Xét bài toán TSP trong ví dụ 3-7 nói trên.

Tập hợp các cạnh để xét phân nhánh là ab, ac, ad, ae, bc, bd, be, cd, ce và de. Điều kiện bổ sung ở đây là mỗi đỉnh phải được chọn hai cạnh, bị loại hai cạnh và không được tạo ra chu trình thiếu.

Nút gốc A bao gồm tất cả các phương án, có cận dưới là 17.5. Phân nhánh cho A, xây dựng hai con là B và C. Tính cận dưới cho hai nút này được cận dưới của B là 17.5 và C là 18.5. Nút B có cận dưới nhỏ hơn nên được phân nhánh trước. Hai con của B là D và E. Các ràng buộc của D và E giống như ta đã nói trong ví dụ của phần phân nhánh. Tính cận cho D và E, được cận dưới của D là 20.5 và của E là 18. Nút E được xét trước. Phân nhánh cho nút E theo cạnh ad, hai con của E là F và G. F chứa ad và G không chứa ad. Do F kế thừa các thuộc tính của E và B, nên F là tập hợp các phương án chứa ab, ad và không chứa ac, đỉnh a đã đủ cấp 2 vậy F không chứa ae. Tương tự G chứa ab, không chứa ac, không chứa ad nên phải chứa ae. Tính cận dưới cho F và G được cận dưới của F là 18 và của G là 23. Tiếp tục xây dựng hai con cho F theo cạnh bc là H và I. H chứa bc và I không chứa bc. Do H kế thừa các thuộc tính của B, E và F nên H là các phương án chứa ab, ad, không chứa ac và chứa bc. Như vậy

đỉnh a đã thỏa điều kiện là được chọn hai cạnh (ab và ad) và bị loại hai cạnh (ac và ae), Đỉnh b đã được chọn 2 cạnh (ba và bc) nên bd và be bị loại. Đỉnh c đã được chọn cb, bị loại ca, ta có thể chọn cd hoặc ce. Nếu chọn cd thì sẽ có một chu trinh thiếu a b c d a, như vậy cd bị loại nên phải chọn ce. Đỉnh d có db và dc đã bị loại, da đã được chọn nên phải chọn thêm de. Lúc đó đỉnh e cũng đã có hai cạnh được chọn là ec và ed, hai cạnh bị loại là eb và ea. Tóm lại H là tập chỉ bao gồm một phương án a b c e d a có giá là 23. Đối với I ta đã có I chứa ab, không chứa ac, chứa ad, không chứa ae và không chứa bc. Bằng lý luận tương tự ta có I không chứa bd, chứa be, cd, ce và không chứa de. Một phương án mới là a b e c d a với giá 21. Đây là giá nhỏ nhất tạm thời mới được tìm thấy.



Hình 3-14: Tìm phương án tối ưu bằng kỹ thuật nhánh cận cho bài toán TSP

Bây giờ ta quay lui về E và xét nút con của nó là G. Vì G có cận dưới là 23 lớn hơn giá thấp nhất tạm thời 21 nên cắt tỉa các con của G.

Quay lui về B và xét nút con D của nó. Cận dưới của D là 20.5 không lớn hơn 21. Nhưng vì độ dài các cạnh trong bài toán đã cho là số nguyên nên nếu ta triển khai các con của D tới nút lá gồm một phương án. Giá của phương án này phải là một số nguyên lớn hơn 20.5 hay lớn hơn hoặc bằng 21. Vậy ta cũng không cần xây dựng các con của D nữa.

Tiếp tục quay lui đến A và xét con C của nó. Phân nhánh C theo cạnh ac thành hai con J và K.

J chứa ac có cận dưới là 18.5. K không chứa ac nên phải chứa ad và ae, cận dưới của K là 21 bằng giá nhỏ nhất tạm thời nên cắt tỉa các con của K.

Hai con của J là L và M. M không chứa ad, ab, chứa ac và ae có cận dưới 23.5 nên bị cắt tỉa các con.Hai con của L là N và O, N chứa bc và O không chứa bc.

Xét nút N ta có: Đỉnh a được chọn hai cạnh ac và ad, bị loại hai cạnh ab và ae. Đỉnh b đã được chọn bc, bị loại ba, ta có thể chọn bd hoặc be. Nếu chọn bd thì sẽ có một chu trình thiếu là a c b d a, vậy phải loại bd và chọn be. Đỉnh c đã được chọn ca, cb nên phải loại cd và ce. Đỉnh d đã được chọn da, bị loại db và dc nên phải chọn de. Khi đó đỉnh e có đủ hai cạnh được chọn là eb, ed và hai cạnh bị loại là ea và ec. Vậy N bao gồm chỉ một phương án là a c b e d a với giá 19.

Tương tự nút O bao gồm chỉ một phương án a c e b d a có giá là 23.

Tất cả các nút con của cây đã được xét hoặc bị cắt tỉa nên phương án cần tìm là a c b e d a với giá 19.

Hình 3-14 minh họa cho những điều ta vừa nói.

Bài toán cái ba lô

Ta thấy đây là một bài toán tìm max. Danh sách các đồ vật được sắp xếp **theo thứ tự giảm** của đơn giá để xét phân nhánh.

1. Nút gốc biểu diễn cho trạng thái ban đầu của ba lô, ở đó ta chưa chọn một vật nào. Tổng giá trị được chọn TGT=0. Cận trên của nút gốc CT = W * Đơn giá lớn nhất.

2. Nút gốc sẽ có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số:

· TGT = TGT (cũ) + số đồ vật được chọn * giá trị mỗi vật.

· W = W (cũ) - số đồ vật được chọn * trọng lượng mỗi vật.

· CT = TGT + W (mới) * Đơn giá của vật sẽ xét kế tiếp.

3. Trong các nút con, ta sẽ **ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn** trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút ta lại phải xác định lại các thông số TGT, W, CT theo công thức đã nói trong bước 2.

4. Lặp lại bước 3 với chú ý: đối với những nút có **cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời** của một phương án đã được tìm thấy thì ta không cần phân nhánh cho nút đó nữa.

5. Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá lớn nhất là phương án cần tìm.

Ví dụ 3-11: Với bài toán cái ba lô đã cho trong ví dụ 3-2 , sau khi tính đơn giá cho các đồ vật và sắp xếp các đồ vật theo thứ tự giảm dần của đơn giá ta được bảng sau.

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
b	10	25	2.5
a	15	30	2.0
d	4	6	1.5
c	2	2	1

Hình 3-15: Các đồ vật được sắp theo thứ tự giảm dần của đơn giá

Gọi x1, x2, x3, x4 là số lượng cần chọn tương ứng của các đồ vật b, a, d, c.

Nút gốc A biểu diễn cho trạng thái ta chưa chọn bất cứ một đồ vật nào. Khi đó tổng giá trị TGT =0, trọng lượng của ba lô W=37 (theo đề ra) và cận trên CT = 37*2.5 = 92.5, trong đó 37 là W, 2.5 là đơn giá của đồ vật b.

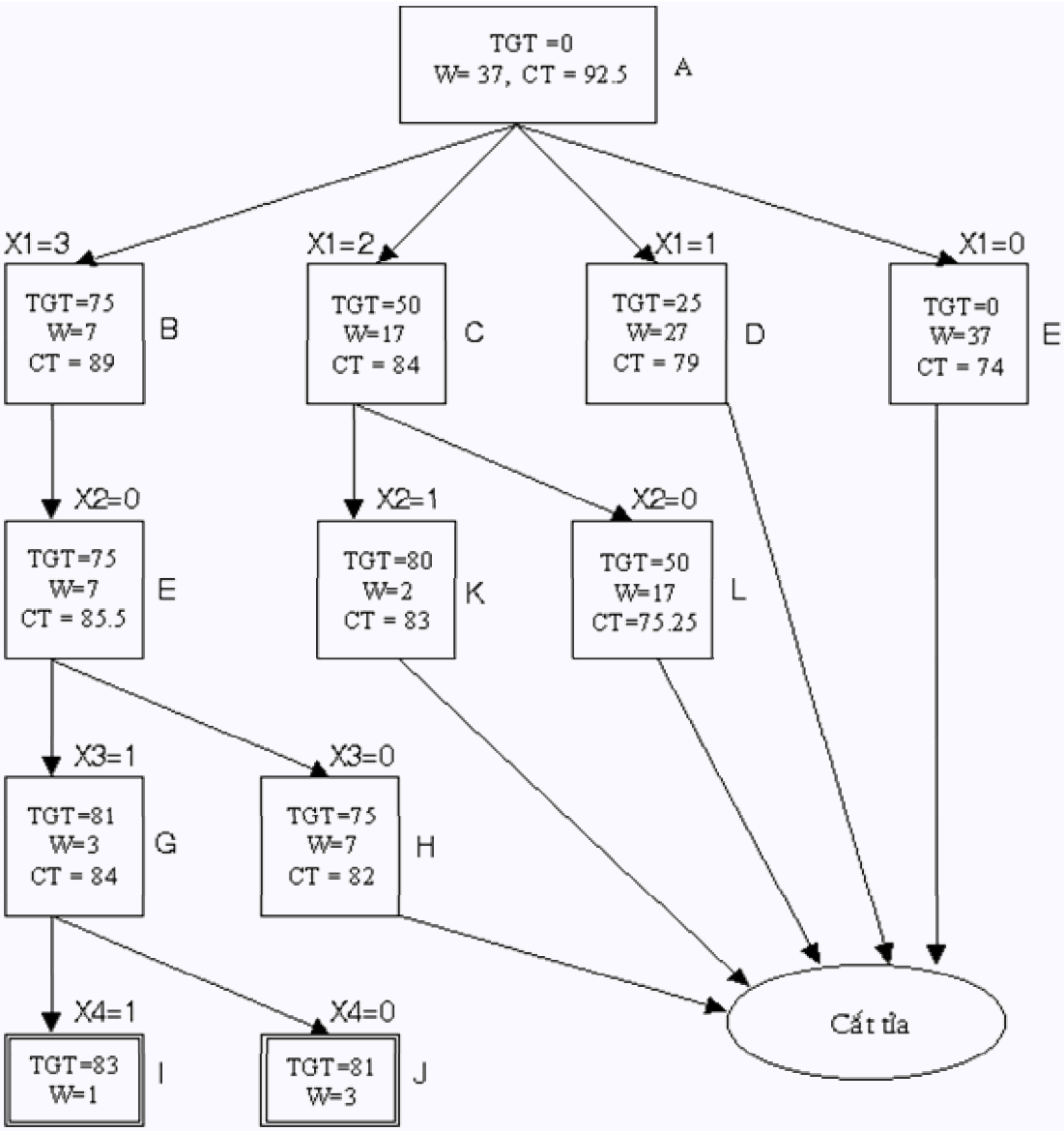
Với đồ vật b, ta có 4 khả năng: chọn 3 đồ vật b (X1=3), chọn 2 đồ vật b (X1=2), chọn 1 đồ vật b (X1=1) và không chọn đồ vật b (X1=0). Ứng với 4 khả năng này, ta phân nhánh cho nút gốc A thành 4 con B, C, D và E.

Với nút con B, ta có TGT = 0+ 3*25 = 75, trong đó 3 là số vật b được chọn, 25 là giá trị của mỗi đồ vật b. W = 37- 3*10 = 7, trong đó 37 là trọnh lượng ban đầu của ba lô, 3 là số vật b được, 10 là trọng lượng mõi đồ vật b. CT = 75 + 7*2 = 89, trong đó 75 là TGT, 7 là trọng lượng còn lại của ba lô và 2 là đơn giá của đồ vật a. Tương tự ta tính được các thông số cho các nút C, D và E, trong đó cận trên tương ứng là 84, 79 và 74.

Trong các nút B, C, D và E thì nút B có cận trên lớn nhất nên ta sẽ phân nhánh cho nút B trước với hy vọng sẽ có được phương án tốt từ hướng này. Từ nút B ta chỉ có một nút con F duy nhất ứng với X2=0 (do trọng lượng còn lại của ba lô là 7, trong khi trọng lượng của mỗi đồ vật a là 15). Sau khi xác định các thông số cho nút F ta có cận trên của F là 85.5. Ta tiếp tục phân nhánh cho nút F. Nút F có 2 con G và H tương ứng với X3=1 và X3=0. Sau khi xác định các thông số cho hai nút này ta thấy cận trên của G là 84 và của H là 82 nên ta tiếp tục phân nhánh cho nút G. Nút G có hai con là I và J tương ứng với X4=1 và X4=0. Đây là hai nút lá (biểu diễn cho phương án) vì với mỗi nút thì số các đồ vật đã được chọn xong. Trong đó nút I biểu diễn cho phương án chọn X1=3, X2=0, X3=1 và X4=1 với giá 83, trong khi nút J biểu diễn cho phương án chọn X1=3, X2=0, X3=1 và X4=01 với giá 81. Như vậy giá lớn nhất tạm thời ở đây là 83.

Quay lui lên nút H, ta thấy cận trên của H là 82<83 nên cắt tỉa nút H.

Quay lui lên nút C, ta thấy cận trên của C là 84>83 nên tiếp tục phân nhánh cho nút C. Nút C có hai con là K và L ứng với X2=1 và X2=0. Sau khi tính các thông số cho K và L ta thấy ận trên của K là 83 và của L là 75.25. Cả hai giá trị này đều không lớn hơn 83 nên cả hai nút này đều bị cắt tỉa. Cuối cùng các nút D và E cũng bị cắt tỉa. Như vậy tất cả các nút trên cây đều đã được phân nhánh hoặc bị cắt tỉa nên phương án tốt nhất tạm thời là phương án cần tìm. Theo đó ta cần chọn 3 đồ vật loại b, 1 đồ vật loại d và một đồ vật loại c với tổng giá trị là 83, tổng trọng lượng là 36. Xem hình 3-16.



Hình 3-16: Kỹ thuật nhánh cận áp dụng cho bài toán cái ba lô

III.5- KỸ THUẬT TÌM KIẾM ĐỊA PHƯƠNG

III.5.1- Nội dung kỹ thuật

Kỹ thuật tìm kiếm địa phương (local search) thường được áp dụng để giải các bài toán tìm lời giải tối ưu. Phương pháp như sau:

· Xuất phát từ một phương án nào đó.

· Áp dụng một phép biến đổi lên phương án hiện hành để được một phương án mới tốt hơn phương án đã có.

· Lặp lại việc áp dụng phép biến đổi lên phương án hiện hành cho đến khi không còn có thể cải thiện được phương án nữa.

Thông thường một phép biến đổi chỉ thay đổi một bộ phận nào đó của phương án hiện hành để

được một phương án mới nên phép biến đổi được gọi là phép biến đổi địa phương và do đó ta có tên kỹ thuật tìm kiếm địa phương. Sau đây ta sẽ trình bày một số ví dụ áp dụng kỹ thuật tìm kiếm địa phương.

III.5.2- Bài toán cây phủ tối thiểu

Cho G = (V,E) là một đồ thị vô hướng liên thông, trong đó V là tập các đỉnh và E là tập các cạnh. Các cạnh của đồ thị G đều có trọng số. Cây T có tập hợp các nút là V được gọi là cây phủ (spaning tree) của đồ thị G.

Cây phủ tối thiểu là một cây phủ của G mà tổng độ dài (trọng số) các cạnh nhỏ nhất.

Bài toán cây phủ tối thiểu thường được áp dụng trong việc thiết kế một mạng lưới giao thông giữa các thành phố hay thiết kế một mạng máy tính.

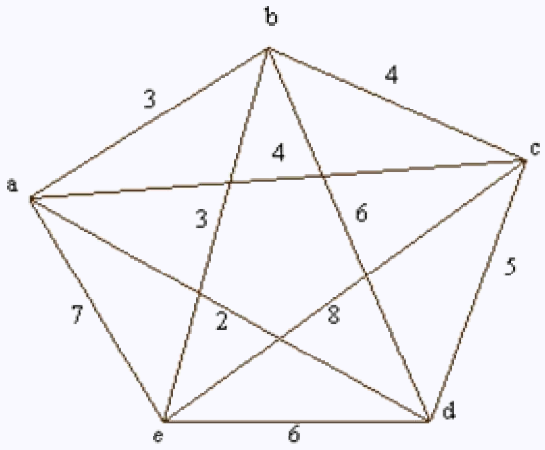
Kỹ thuật tìm kiếm địa phương áp dụng vào bài toán này như sau:

· Phương án ban đầu là một cây phủ nào đó.

· Thành lập tập tất cả các cạnh theo thứ tăng dần của độ dài (có $\frac{n(n-1)}{2}$ cạnh đối với đồ thị có n đỉnh).

· Phép biến đổi địa phương ở đây là: Chọn một cạnh có độ dài nhỏ nhất trong tập các cạnh chưa sử dụng để thêm vào cây. Trong cây sẽ có một chu trình, loại khỏi chu trình cạnh có độ dài lớn nhất trong chu trình đó. Ta được một cây phủ mới. Lặp lại bước này cho đến khi không còn cải thiện được phương án nữa.

Ví dụ 3-12: Cho đồ thị G bao gồm 5 đỉnh a, b, c, d,e và độ dài các cạnh được cho trong hình 3-17.



Hình 3-17

Tập hợp các cạnh để xét được thành lập theo thứ tự từ nhỏ đến lớn là ad, ab, be, bc, ac, cd, bd, de, ae và ce.

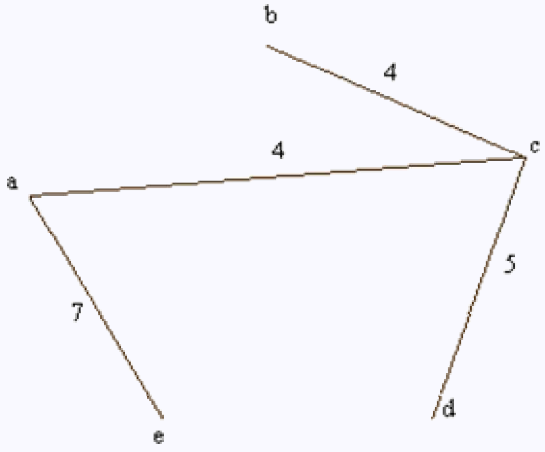
Cây xuất phát với giá là 20 (Hình 3-18a). Thêm cạnh ad = 2, bỏ cạnh cd = 5 ta được cây mới có giá là 17 (Hình 3-18b)

Lại thêm cạnh ab = 3, bỏ cạnh bc = 4 ta được cây có giá là16 (Hình 3-18c).

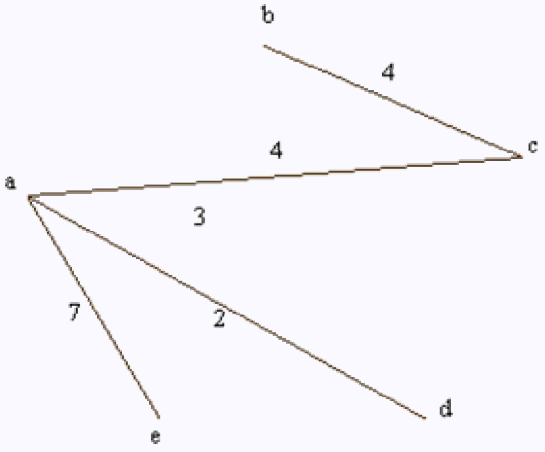
Thêm cạnh be = 3, bỏ cạnh ae = 7 ta được cây có giá là 12. (Hình 3-18d).

Việc áp dụng các phép biến đổi đến đây dừng lại vì nêuú tiếp tục nũa thì cũng không cải thiện được phương án.

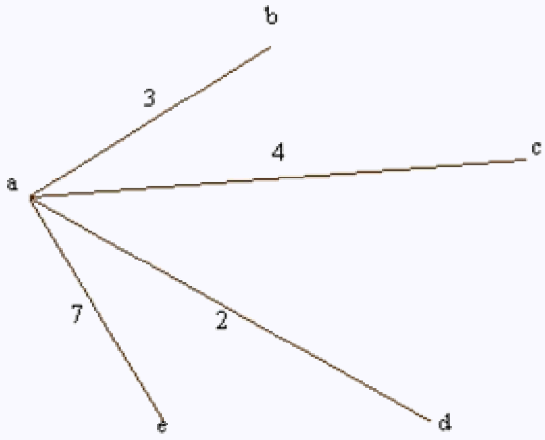
Vậy cây phủ tối thiểu cần tìm là cây trong hình 3-18d



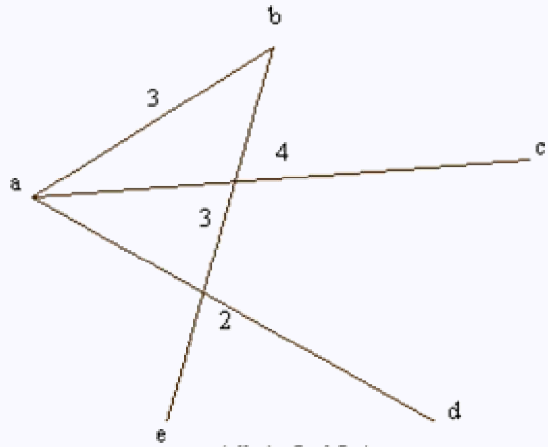
Hình 3-18a



Hình 3-18b



Hình 3-18c



Hình 3-18d

III.5.3- Bài toán đường đi của người bán hàng.

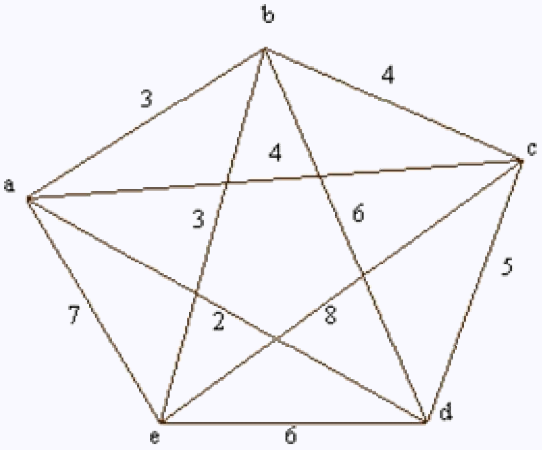
Ta có thể vận dụng kỹ thuật tìm kiếm địa phương để giải bài toán tìm đường đi ngắn nhất của người bán hàng (TSP).

· Xuất phát từ một chu trình nào đó.

· Bỏ đi hai cạnh có độ dài lớn nhất không kề nhau, nối các đỉnh lại với nhau sao cho vẫn tạo ra một chu trình đủ.

· Tiếp tục quá trình biến đổi trên cho đến khi nào không còn cải thiện được phương án nũa.

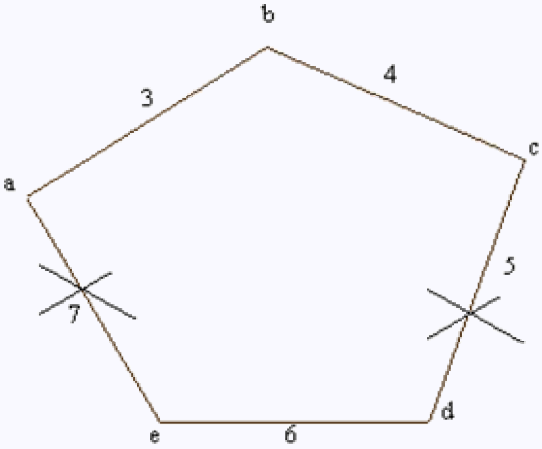
Ví dụ 3-13: Bài toán TSP có 5 đỉnh và các cạnh có độ dài được cho trong hình 3-19



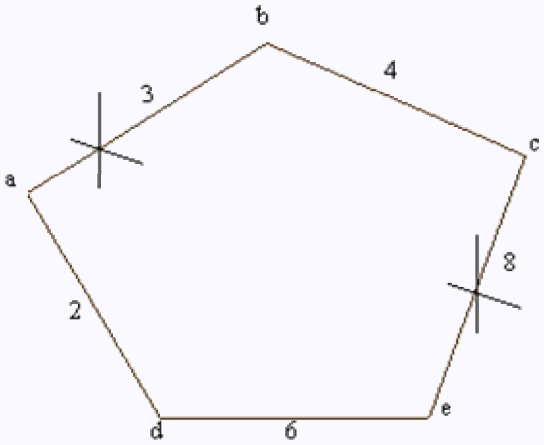
Hình 3-19

Phương án ban đầu là chu trình (a b c d e a) có giá (tổng độ dài) là 25. (Hình 3-20a).

Bỏ hai cạnh có độ dài lớn nhất không kề nhau là ae và cd, nối a với d và e với c. ta được chu trình mới (a b c e d a) với giá = 23 (Hình 3-20b).

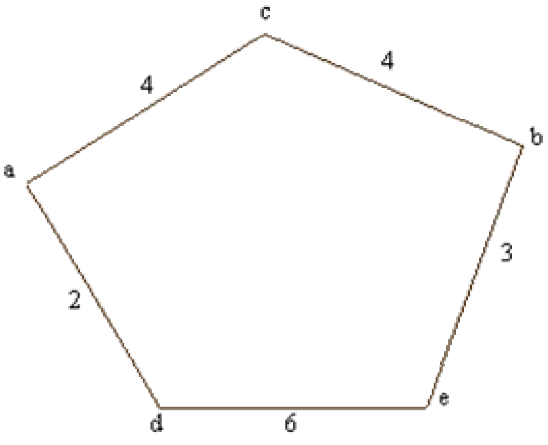


Hình 3-20a



Hình 3-20b

Bỏ hai cạnh có độ dài lớn nhất, không kề nhau là ce và ab, ta được chu trình mới (a c b e d a) có giá = 19. (Hình 3-20c). Quá trình kết thúc vì nếu tiếp tục thì giá sẽ tăng lên.



Hình 3-20c

BÀI TẬP Chương III: KỸ THUẬT THIẾT KẾ GIẢI THUẬT

Bài 1: Giả sử có hai đội A và B tham gia một trận thi đấu thể thao, đội nào thắng trước n hiệp thì sẽ thắng cuộc. Chẳng hạn một trận thi đấu bóng chuyền 5 hiệp, đội nào thắng trước 3 hiệp thì sẽ tháng cuộc. Giả sử hai đội ngang tài ngang sức. Gọi P(i,j) là xác suất để đội A cần i hiệp nữa để chiến thắng, B cần j hiệp.

Để tính P(i,j) chúng ta thấy như sau rằng nếu i= 0 và j>0 tức là đội A đã thắng nên P(0,j) = 1. Tương tự P(0,j) = 0 với i>0. Nếu i và j đều lớn hơn không thì ít nhất còn một hiệp nữa phải đấu và hai đội có khả năng 5 ăn, 5 thua trong hiệp này. Như vậy P(i,j) là trung bình cộng của P(i-1,j) và P(i,j-1). Trong đó P(i-1,j) là xác suất để đội A thắng cuộc nếu nó thắng hiệp đó và P(i,j-1) là xác suất để A thắng cuộc nếu nó thua hiệp đó. Tóm lại ta có công thức tính P(i,j) như sau:

P(i,j) = 1 Nếu i = 0 và j > 0

P(i,j) = 0 Nếu i > 0 và j = 0

P(i,j) = (P(i-1,j) + P(i,j-1))/2 Nếu i > 0 và j > 0

1. Viết một hàm đệ quy để tính P(i,j). Tính độ phức tạp của hàm đó.

2. Dùng kỹ thuật quy hoạch động để viết hàm tính P(i,j). Tính độ phức tạp của hàm đó.

Bài 2: Bài toán phân công lao động: Có n công nhân có thể làm n công việc. Công nhân i làm công việc j trong một khoảng thời gian tij. Phải tìm một phương án phân công như thế nào để các công việc đều được hoàn thành, các công nhân đều có việc làm, mỗi công nhân chỉ làm một công việc và mỗi công việc chỉ do một công nhân thực hiện đồng thời tổng thời gian là nhỏ nhất.

1. Mô tả kỹ thuật "háu ăn" (greedy) cho bài toán phân công lao động.

2. Tìm phương án theo giải thuật "háu ăn" cho bài toán phân công lao động được cho trong bảng sau. Trong đó mỗi dòng là một công nhân, mỗi cột là một công việc, ô (i,j) ghi thời gian tij mà công nhân i cần để hoàn thành công việc j. (Cần chỉ rõ công nhân nào làm công việc gì và tổng thời gian là bao nhiêu).

Công việc / Công nhân	1	2	3	4	5
1	5	6	4	7	2
2	5	2	4	5	1
3	4	5	4	6	3
4	5	5	3	4	2
5	3	3	5	2	5

Bài 3: Cho bài toán đường đi của người bán hàng (TSP) với 7 thành phố a, b, c, d, e, f, g. Ma trận khoảng cách giữa các thành phố được cho trong bảng sau:

	a	b	c	d	e	f	g

a							
b	15						
c	12	6					
d	6	23	16				
e	4	5	13	26			
f	8	4	10	14	24		
g	11	1	2	9	5	7	

Hãy sử dụng kỹ thuật "tham ăn" (greedy) để tìm một chu trình có tổng độ dài các cạnh ngắn nhất.

Bài 4: Bài toán đóng gói: Giả sử ta có các hộp có cùng kích thước T_0 và n đối tượng O_1, O_2, ...O_n. Mỗi đối tượng O_i có kích thước t_i . Dĩ nhiên $t_i <= T_0$ (i=1..n) và các t_i này có thể bằng nhau.

Tìm phương án xếp tất cả n đối tượng này vào trong các hộp sao cho tốn ít số hộp nhất.

1. Hãy nêu kỹ thuật "tham ăn" (greedy) để giải bài toán này.

2. Giải bài toán cụ thể với ï các hộp kích thước 13, và 9 đối tượng cói kích thước được cho trong bảng sau:

Đối tượng	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9
Kích thước	7	8	4	9	7	4	6	2	3

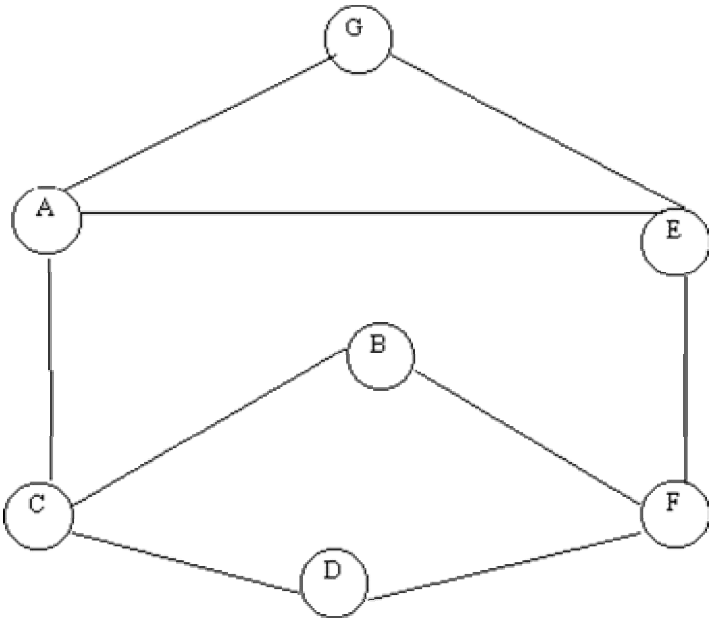
Bài 5: Bài toán tô màu bản đồ thế giới

Người ta muốn tô màu bản đồ các nước trên thế giới, mỗi nước đều được tô màu và hai nước có biên giới chung nhau thì không được có màu giống nhau (các nước không chung biên giới có thể được tô màu giông nhau). Tìm một phương án tô màu sao cho số loại màu phải dùng ít nhất.

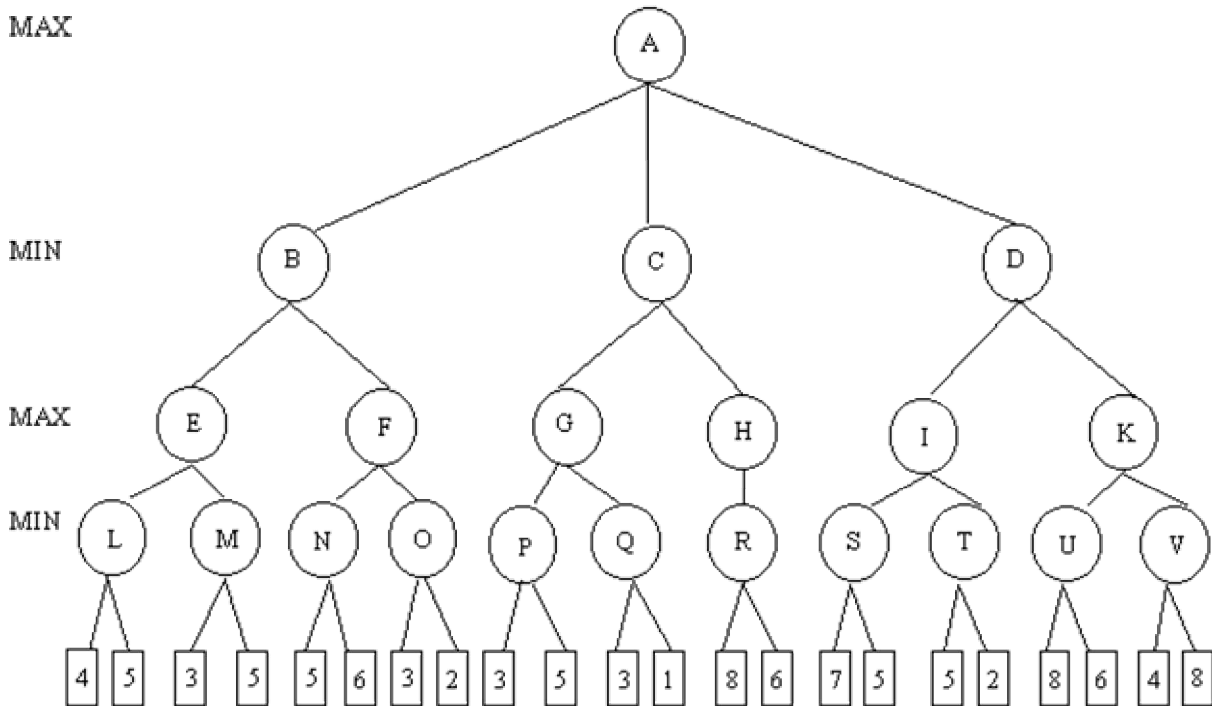
Người ta có thể mô hình hóa bản đồ thế giới bằng một đồ thị không có hướng, trong đó mỗi đỉnh biểu diễn cho một nước, biên giới của hai nước được biểu diễn bằng cạnh nối hai đỉnh. Bài toán tô màu bản đồ thế giới trở thành bài toán tô màu các đỉnh của đồ thi: Mỗi đỉnh của đồ thị phải được tô màu và hai đỉnh có chung một cạnh thì không được tô cùng một màu (cá đỉnh không chung cạnh có thể được tô cùng một màu). Tìm một phương án tô màu sao cho số loại màu phải dùng là ít nhất.

a) Hãy mô tả kỹ thuật "háu ăn" (Greedy) để giải bài toán tô màu cho đồ thị.

b) Áp dụng kỹ thuật háu ăn để tô màu cho các đỉnh của đồ thị sau (các màu có thể sử dung để tô là: ĐỎ, CAM, VÀNG, XANH, ĐEN, NÂU, TÍM)



Bài 6: Dùng kỹ thuật cắt tỉa alpha-beta để định trị cho nút gốc của cây trò chơi sau (các nút lá đã được gán trị:



Bài 7: Xét một trò chơi có 6 viên bi, hai người thay phiên nhau nhặt từ 1 đến 3 viên. Người phải nhặt viên bi cuối cùng thì bị thua.

1. Vẽ toán bộ cây trò chơi

2. Sử dụng kỹ thuật cắt tỉa alpha-beta định trị cho nút gốc

3. Ai sẽ thắng trong trò chơi này nếu hai người đều đi những nước tốt nhất. Hãy cho một nhận xét về trường hợp tổng quát khi ban đầu có n viên bi và mỗi lần có thể nhặt từ 1 đến m viên.

Bài 8: Xét một trò chơi có 7 cái đĩa. Người chơi 1 chia thành 2 chồng có số đĩa không bằng

nhau. Người chơi 2 chọn một chồng trong số các chồng cos thể chia và tiếp tục chia thành hai chồng không bằng nhau. Hai người luân phiên nhau chia đĩa như vậy cho đến khi không thể chia được nữa thì thua.

1. Vẽ toàn bộ cây trò chơi.

2. Sử dụng kỹ thuật cắt tỉa alpha-beta định trị cho nút gốc

3. Ai sẽ thắng trong trò chơi này nếu hai người đều đi những nước tốt nhất.

Bài 9: Cho một đồ thị vô hướng, có trọng số với 7 đỉnh a, b, c, d, e, f, g. Ma trận trọng số được cho trong bảng sau:

	a	b	c	d	e	f	g
a							
b	15						
c	12	6					
d	6	23	16				
e	4	5	13	26			
f	8	4	10	14	24		
g	11	1	2	9	5	7	

Hãy sử dụng kỹ thuật tìm kiếm địa phương để xác định cây phủ tối thiểu trên đồ thị đó.

CHƯƠNG IV: GTDL & GIẢI THUẬT LƯU TRỮ NGOÀI

1. Mục tiêu
2. Kiến thức cơ bản cần có để học chương này
3. Tài liệu tham khảo có liên quan đến chương
4. Nội dung:

Trong chương này chúng ta sẽ nghiên cứu hai vấn đề chính là sắp xếp dữ liệu được lưu trong bộ nhớ ngoài và kỹ thuật lưu trữ tập tin. Trong kỹ thuật lưu trữ tập tin chúng ta sẽ sử dụng các cấu trúc dữ liệu tuần tự, bảng băm, tập tin chỉ mục và cấu trúc B-cây.

IV.1- MÔ HÌNH XỬ LÝ NGOÀI

Trong các giải thuật mà chúng ta đã đề cập từ trước tới nay, chúng ta đã giả sử rằng số lượng các dữ liệu vào là khá nhỏ để có thể chứa hết ở bộ nhớ trong (main memory). Nhưng điều gì sẽ xảy ra nếu ta muốn xử lý phiếu điều tra dân số toàn quốc hay thông tin về quản lý đất đai cả nước chẳng hạn? Trong các bài toán như vậy, số lượng dữ liệu vượt quá khả năng lưu trữ của bộ nhớ trong. Để có thể giải quyết các bài toán đó chúng ta phải dùng bộ nhớ ngoài để lưu trữ và xử lý. Các thiết bị lưu trữ ngoài như băng từ, đĩa từ đều có khả năng lưu trữ lớn nhưng đặc điểm truy nhập hoàn toàn khác với bộ nhớ trong. Chúng ta cần tìm các cấu trúc dữ liệu và giải thuật thích hợp cho việc xử lý dữ liệu lưu trữ trên bộ nhớ ngoài.

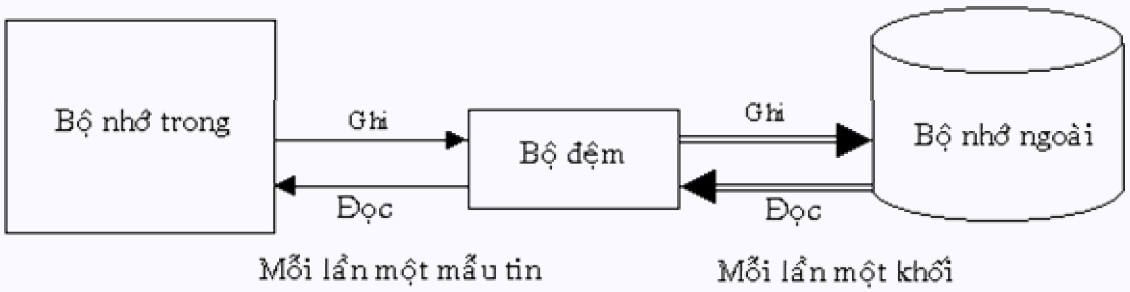
Kiểu dữ liệu tập tin là kiểu thích hợp nhất cho việc biểu diễn dữ liệu được lưu trong bộ nhớ ngoài. Hệ điều hành chia bộ nhớ ngoài thành các khối (block) có kích thước bằng nhau, kích thước này thay đổi tùy thuộc vào hệ điều hành nhưng nói chung là từ 512 bytes đến 4096 bytes.

Trong quá trình xử lý, việc chuyển giao dữ liệu giữa bộ nhớ trong và bộ nhớ ngoài được tiến hành thông qua vùng nhớ đệm (buffer). Bộ đệm là một vùng dành riêng của bộ nhớ trong mà kích thước bằng với kích thước của một khối của bộ nhớ ngoài.

Có thể xem một tập tin bao gồm nhiều mẩu tin được lưu trong các khối . Mỗi khối lưu một số nguyên vẹn các mẩu tin, không có mẩu tin nào bị chia cắt để lưu trên hai khối khác nhau.

Trong thao tác đọc, nguyên một khối của tập tin được chuyển vào trong bộ đệm và lần lượt đọc các mẩu tin có trong bộ đệm cho tới khi bộ đệm rỗng thì lại chuyển một khối từ bộ nhớ ngoài vào bộ đệm.

Để ghi thông tin ra bộ nhớ ngoài, các mẩu tin lần lượt được xếp vào trong bộ đệm cho đến khi đầy bộ đệm thì nguyên một khối được chuyển ra bộ nhớ ngoài. Khi đó bộ đệm trở nên rỗng và lại có thể xếp tiếp các mẩu tin vào trong đó.



Hình 4-1: Mô hình giao tiếp giữa bộ nhớ trong, bộ nhớ ngoài và vùng nhớ đệm

Như vậy đơn vị giao tiếp giữa bộ nhớ trong và bộ đệm là mẩu tin còn giữa bộ đệm và bộ nhớ ngoài là khối.

Hình 4-1 mô tả hoạt động của bộ nhớ trong, bộ đệm và bộ nhớ ngoài trong thao tác đọc và ghi tập tin

IV.2- ĐÁNH GIÁ CÁC GIẢI THUẬT XỬ LÝ NGOÀI

Đối với bộ nhớ ngoài thì thời gian tìm một khối để đọc vào bộ nhớ trong là rất lớn so với thời gian thao tác trên dữ liệu trong khối đó. Ví dụ giả sử ta có một khối có thể lưu 1000 số nguyên được lưu trên đĩa quay với vận tốc 1000 vòng/ phút thì thời gian để đưa đầu từ vào rãnh chứa khối và quay đĩa để đưa khối đến chỗ đầu từ hết khoảng 100 mili giây. Với thời gian này máy có thể thực hiện 100000 lệnh, tức là đủ để sắp xếp các số nguyên này theo giải thuật QuickSort. Vì vậy khi đánh giá các giải thuật thao tác trên bộ nhớ ngoài, chúng ta tập trung vào việc xét số lần đọc khối vào bộ nhớ trong và số lần ghi khối ra bộ nhớ ngoài ta gọi chung là phép truy xuất khối (block access). Vì kích thước các khối là cố định nên ta không thể tìm cách tăng kích thước một khối mà chúng ta phải tìm cách giảm số lần truy xuất khối.

IV.3- SẮP XẾP NGOÀI

Sắp xếp dữ liệu được tổ chức như một tập tin hoặc tổng quát hơn, sắp xếp dữ liệu được lưu trên bộ nhớ ngoài gọi là sắp xếp ngoài.

IV.3.1- Sắp xếp trộn (merge sorting)

Khái niệm về đường

Đường độ dài k là một tập hợp k mẩu tin đã đươc sắp thứ tự theo khoá tức là, nếu các mẩu tin $r_1, r_2, ..., r_k$ có khoá lần lượt là $k_1, k_2, ..., k_k$ tạo thành một đường thì $k_1 \le k_2 \le ... \le k_k$.

Cho tập tin chứa các mẩu tin $r_1, r_2, ..., r_n$, ta nói tập tin được tổ chức thành đường có độ dài k nếu ta chia tập tin thành các đoạn k mẩu tin liên tiếp và mỗi đoạn là một đường, đoạn cuối có thể không có đủ k mẩu tin, trong trường hợp này ta gọi đoạn ấy là đuôi (tail).

Ví dụ 4-1: Tập tin gồm 14 mẩu tin có khóa là các số nguyên được tổ chức thành 4 đường độ dài 3 và một đuôi có độ dài 2

5	6	9	13	26	27	1	5	8	12	14	17	23	25

Giải thuật

Để sắp xếp tập tin F có n mẩu tin ta sử dụng 4 tập tin F1, F2, G1 và G2.

Khởi đầu ta phân phối các mẩu tin của tập tin đã cho F luân phiên vào trong hai tập tin F1 F2. Như vậy hai tập tin này được xem như được tổ chức thành các đường độ dài 1.

Bước 1: Đọc 2 đường, mỗi đường độ dài 1 từ hai tập tin F1, F2 và trộn hai đường này thành đường độ dài 2 và ghi luân phiên vào trong hai tập tin G1, G2. Đổi vai trò của F1 cho G1, F2 cho G2.

Bước 2: Đọc 2 đường, mỗi đường độ dài 2 từ hai tập tin F1, F2 và trộn hai đường này thành đường độ dài 4 và ghi luân phiên vào trong hai tập tin G1, G2. Đổi vai trò của F1 cho G1, F2 cho G2.

Quá trình trên cứ tiếp tục và sau i bước thì độ dài của một đường là 2^i. Nếu $2^i \geq n$ thì giải thuật kết thúc, lúc đó tập tin G2 sẽ rỗng và tập tin G1 chứa các mẩu tin đã được sắp.

Đánh giá giải thuật sắp xếp trộn

Ta thấy giải thuật kết thúc sau i bước với $i \geq \log n$. Mỗi bước phải đọc từ 2 tập tin và ghi vào 2 tập tin, mỗi tập tin có trung bình n/2 mẩu tin. Giả sử mỗi một khối lưu trữ được b mẩu tin thì mỗi bước cần đọc và ghi $\frac{2*2*n}{2*b} = \frac{2n}{b}$ khối mà chúng ta cần logn bước vậy tổng cộng chúng ta cần $\frac{2n\log n}{b}$ phép truy xuất khối.

Ví dụ 4-2: Cho tập tin F có 23 mẩu tin với khóa là các số nguyên như sau:

F: 28 31 3 5 93 96 10 40 54 85 65 9 30 39 90 13 10 8 69 77 8 10 22

Để bắt đầu ta phân phối các mẩu tin của F luân phiên vào hai tập tin F1 và F2 được tổ chức thành các đường có độ dài 1

28	3	93	10	54	65	30	90	10	69	8	22

F1

31	5	96	40	85	9	39	13	8	77	10

F2

Bước 1: Trộn các đường độ dài 1 của F1 và F2 được các đường độ dài 2 và ghi luân phiên vào trong hai tập tin G1, G2:

G1:	28	31	93	96	54	85	30	39	8	10	8	10

F1

G2:	3	5	10	40	9	65	13	90	69	77	22

F2

Bước 2: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 2 trong hai tập tin F1 và F2 được các đường độ dài 4 rồi ghi luân phiên vào trong hai tập tin G1 và G2:

G1:	3	5	28	31	9	54	65	85	8	10	69	77

F1

G2:	10	40	93	96	13	30	39	90	8	10	22

F2

Bước 3: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 4 trong hai tập tin F1 và F2 được các đường độ dài 8 rồi ghi luân phiên vào trong hai tập tin G1 và G2:

G1: | 3 5 10 28 | 31 40 93 96 | 8 8 10 10 | 22 69 77 | F1

G2: | 9 13 30 39 | 54 65 85 90 | F2

Bước 4: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 8 trong hai tập tin F1 và F2 được các đường độ dài 16 rồi ghi luân phiên vào trong 2 tập tin G1 và G2.

G1: | 3 5 9 10 13 28 30 31 | 39 40 54 65 85 90 93 96 | F1

G2: | 8 8 10 10 22 69 77 | F2

Bước 5: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 16 trong hai tập tin F1 và F2 được 1 đường độ dài 23 rồi ghi vào trong tập tin G1.

G1: | 3 5 8 8 9 10 10 10 13 22 28 30 31 39 40 54 65 77 85 90 93 96 |

Tập tin G1 chứa các mẩu tin đã được sắp còn tập tin G2 rỗng.

Chương trình

```
procedure Merge(k:integer; f1,f2,g1,g2: File of RecordType);
{Thủ tục này trộn các đường độ dài k và trong hai tập tin f1 và f2 thành các đường đọ dài 2k và
ghi luân phiên vào trong hai tập tin g1 và g2}

var
OutSwithh : boolean; {Nếu OutSwitch = TRUE thì ghi vào tập tin g1, ngược lại ghi vào g2}
Winner: integer; {Để chỉ định mẩu tin hiện hành nào trong hai tập tin f1 và f2 sẽ được ghi ra tập
tin g1 hoặc g2}
Used: array[1..2] of integer; { Used[ij] ghi số mẩu tin đã được đọc trong đường hiện tại của tập
tin fj }
Fin : array[1..2] Of boolean; {Fin[j] sẽ có giá trị TRUE nếu đã đọc hết các mẩu tin trong đường
hiện hành của fj hoặc đã đến cuối tập tin fj }

Current: array[1..2] Of RecordType; { Current[j] để lưu mẩu tin hiện hành của tập tin f[j]}

procedure GetRecord(i:integer);
{Nếu đã đọc hết các mẩu tin trong đường hiện hành của tập tin fi hoặc đã đến cuối tập tin fi thì
đặt fin[i] = TRUE nếu không thì đọc một mẩu tin của tập tin fi vào trong current[i]}

begin
        Used[i] := Used[i] + 1;
        if (Used[i] = k+1 ) or (i = 1) and ( eof(f1)) or
        (i = 12 and ( eof(f2)) then fin[i] := TRUE
```

```
        else if i=1 then Read(f1, current[1])
                else read(f2, current[2]);
end;
begin
        { Khởi tạo }
        OutSwitch := TRUE;
        ReSet(f1);
        ReSet(f2);
        ReWrite(g1);
        ReWrite(g2);
        while (not eof(f1)) or (not eof(f2)) do begin
                {Bắt đầu đọc các mẩu tin từ trong hai đường hiện hành của hai tập tin f1,f2 }
                Used[1] := 0; Used[2] := 0;
                Fin[1] := FALSE ; Fin[2] := FALSE ;
                GetRecord(1) ; GetRecord(2);
                while ( not fin[1] ) or (not fin[2]) do begin
                        {Trộn hai đường }
                        { Chọn Winner }
                        if Fin[1] then Winner := 2
                        else if Fin[2] then Winner := 1
                                else if current[1].key < Current[2].key then
                                        Winner := 1
                                else Winner := 2;
                        if OutSwitch then Write(g1, Current[winner] )
                        else Write(g2, current[winner] );
                        GetRecord(Winner);
                end;
                OutSwitch := Not OutSwitch;
        end;
end;
```

IV.3.2.Cải tiến sắp xếp trộn

Ta thấy quá trình sắp xếp trộn nói trên bắt đầu từ các đường độ dài 1 cho nên phải sau logn bước giải thuật mới kết thúc. Chúng ta có thể tiết kiệm thời gian bằng cách chọn một số k thích hợp sao cho k mẩu tin có thể đủ chứa trong bộ nhớ trong. Mỗi lần đọc vào bộ nhớ trong k mẩu tin, dùng sắp xếp trong (chẳng hạn dùng QuickSort) để sắp xếp k mẩu tin này và ghi luân phiên vào hai tập tin F1 và F2. Như vậy chúng ta bắt đầu sắp xếp trộn với các tập tin được tổ chức thành các đường độ dài k.

Sau i bước thì độ dài mỗi đường là $k2^i$. Giải thuật sẽ kết thúc khi $k2^i \geq n$ hay $i \geq \log \frac{n}{k}$. Do đó số phép truy xuất khối sẽ là $\frac{2n\log\frac{n}{k}}{b}$. Dễ thấy $\frac{2n\log\frac{n}{k}}{b} < \frac{2n\log n}{b}$ tức là ta tăng được tốc độ sắp xếp trộn.

Ví dụ 4-3: Lấy tập tin F có 23 mẩu tin với khóa là các số nguyên như trong ví dụ 4-2

F: 28 31 3 5 93 96 10 40 54 85 65 9 30 39 90 13 10 8 69 77 8 10 22

Ta giả sử bộ nhớ trong có thể chứa được 3 mẩu tin, ta đọc lần lượt 3 mẩu tin của F vào bộ nhớ trong , dùng một sắp xếp trong để sắp xếp chúng và ghi phiên vào 2 tập tin F1 và F2.

3 28 31	10 40 54	30 39 90	8 69 77

F1

5 93 96	9 65 85	8 10 13	10 22

F2

Bước 1: Trộn các đường độ dài 3 của F1 và F2 được các đường độ dài 6 và ghi luân phiên vào trong hai tập tin G1, G2:

G1:

3 5 28 31 93 96	8 10 13 30 39 90

F1

G2:

9 10 40 54 65 85	8 10 22 69 77

F2

Bước 2: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 6 trong 2 tập tin F1 và F2 được các đường độ dài 12 rồi ghi luân phiên vào trong 2 tập tin G1 và G2:

G1:

3 5 9 10 28 31 40 54 65 85 93 96

F1

G2:

8 8 10 10 13 22 30 39 69 77 90

F2

Bước 3: Đổi vai trò của F1 và G1, F2 và G2 cho nhau. Trộn các đường độ dài 12 trong 2 tập tin F1 và F2 được 1 đường ghi vào trong tập tin G1, còn G2 rỗng

G1:

3 5 8 8 9 10 10 10 13 22 28 30 31 39 40 54 65 77 85 90 93 96

Tập tin G1 chứa các mẩu tin đã được sắp còn tập tin G2 rỗng.

IV.3.3- Trộn nhiều đường (multiway merge)

Giải thuật

Để sắp xếp tập tin F có n mẩu tin ta sử dụng m tập tin (m là một số chẵn) F[1], F[2], ..., F[m]. Trong trường hợp m=4 ta có giải thuật sắp xếp trộn bình thường.

Gọi h = m/2, ta có nội dung của phương pháp như sau (ta vẫn giả sử bộ nhớ trong có thể chứa k mẩu tin).

Khởi đầu: Mỗi lần đọc từ tập tin F vào bộ nhớ trong k mẩu tin, sử dụng một sắp xếp trong để sắp xếp k mẩu tin này thành một đường rồi ghi luân phiên vào các tập tin F[1], F[2], ... , F[h].

Bước 1: Trộn các đường độ dài k của h tập tin F[1], F[2], ..., F[h] thành một đường độ dài kh và ghi luân phiên vào trong h tập tin F[h+1], F[h+2], ... , F[m]. Đổi vai trò của F[i] và F[h+i]] cho nhau (với $1 \le i \le h$).

Bước 2: Trộn các đường độ dài kh của h tập tin F[1], F[2], ..., F[h] thành một đường độ dài kh^2 và ghi luân phiên vào trong h tập tin F[h+1], F[h+2], ... , F[m]. Đổi vai trò của F[i] và F[h+i]] cho nhau (với $1 \le i \le h$).

Sau i bước thì độ dài mỗi đường là khi và giải thuật kết thúc khi $kh^i \geq n$ và khi đó tập tin đã được sắp chính là một đường ghi trong F[h+1].

Đánh giá giải thuật sắp xếp trộn nhiều đường

Theo trên thì giải thuật kết thúc sau i bước, với $kh^i \geq n$ hay $i \geq \log_{hk}\frac{n}{k}$. Mỗi bước ta phải đọc từ h tập tin và ghi vào trong h tập tin, trung bình mỗi tập tin có n/h mẩu tin. Ta vẫn giả sử mỗi khối lưu được b mẩu tin thì mỗi bước phải truy xuất $\frac{2*2*n}{2*b} = \frac{2n}{b}$ khối. Do chúng ta cần $\log_{hk}\frac{n}{k}$ bước nên tổng cộng ta chỉ cần $\frac{2n\log_{hk}\frac{n}{k}}{b}$ phép truy xuất khối.Ta thấy rõ ràng $\log_{hk}\frac{n}{k} < \log\frac{n}{k}$ và thủ tục mergeSort nói trên là một trường hợp đặc biệt khi h = 2.

Ví dụ 4-4: Lấy tập tin F có 23 mẩu tin với khóa là các số nguyên như trong ví dụ 4-2

F: 28 31 3 5 93 96 10 40 54 85 65 9 30 39 90 13 10 8 69 77 8 10 22.

Sử dụng 6 tập tin để sắp xếp tập tin F. Ta giả sử bộ nhớ trong có thể chứa được 3 mẩu tin, ta đọc lần lượt 3 mẩu tin của F vào bộ nhớ trong , dùng một sắp xếp trong để sắp xếp chúng và ghi phiên vào 3 tập tin F[1], F[2] và F[3] như sau:

F[1]:	3	28	31	9	65	85	8	69	77
F[2]:	5	93	96	30	39	90	10	22	
F[3]:	10	40	54	8	10	13			

Bước 1: Trộn các đường độü dài 3 trong các tập tin F[1], F[2], F[3] thành các đường độ dài 9 và ghi vào trong các tập tin F[4], F[5] và F[6].

F[4]:	3	5	10	28	31	40	54	93	96	F[1]
F[5]:	8	9	10	13	30	39	65	85	90	F[2]
F[6]:	8	10	22	69	77					F[3]

Bước 2: Đổi vai trò của F[1] cho F[4], F[2] cho F[5] và F[3] cho F[6]. Trộn các đường độ dài 9 trong các tập tin F[1], F[2], F[3] thành 1 đường độ dài 23 và ghi vào trong tập tin F[4].

F[4]:	3	5	8	8	9	10	10	10	13	22	28	30	31	39	40	54	65	77	85	90	93	96

Tập tin F[4] chứa các mẩu tin đã được sắp còn F[5] và F[6] rỗng.

IV.4- LƯU TRỮ THÔNG TIN TRONG TẬP TIN

IV.4.1- Tập tin tuần tự

IV.4.2- Tăng tốc độ cho các thao tác tập tin
IV.4.3- Tập tin băm
IV.4.4- Tập tin chỉ mục
IV.4.5- Cấu trúc B-cây

Trong phần này ta sẽ nghiên cứu các cấu trúc dữ liệu và giải thuật cho lưu trữ (storing) và lấy thông tin (retrieving) trong các tập tin được lưu trữ ngoài. Chúng ta sẽ coi một tập tin như là một chuỗi tuần tự các mẩu tin, mỗi mẩu tin bao gồm nhiều trường (field). Một trường có thể có độ dài cố định hoặc độ dài thay đổi. Ở đây ta sẽ xét các mẩu tin có độ dài cố định và khảo sát các thao tác trên tập tin là:

· Insert: Thêm một mẩu tin vào trong một tập tin,

· Delete: Xoá một mẩu tin từ trong tập tin,

· Modify: Sửa đổi thông tin trong các mẩu tin của tập tin, và

· Retrieve: Tìm lại thông tin được lưu trong tập tin.

Sau đây ta sẽ nghiên cứu một số cấu trúc dữ liệu dùng để lưu trữ tập tin. Với mỗi cấu trúc chúng ta sẽ trình bày tổ chức, cách thức tiến hành các thao tác tìm, thêm, xoá mẩu tin và có đánh giá về cách tổ chức đó. Sự đánh giá ở đây chủ yếu là đánh giá xem để tìm một mẩu tin thì phải đọc bao nhiêu khối vì các thao tác khác đều phải sử dụng thao tác tìm.

IV.4.1- Tập tin tuần tự

Tổ chức: Tập tin tuần tự là một danh sách liên kết của các khối, các mẩu tin được lưu trữ trong các khối theo một thứ tự bất kỳ.

Tìm mẩu tin: Việc tìm kiếm một mẩu tin có giá trị xác định được thực hiện bằng cách đọc từng khối, với mỗi khối ta tìm mẩu tin cần tìm trong khối, nếu không tìm thấy ta lại đọc tiếp một khối khác. Quá trình cứ tiếp tục cho đến khi tìm thấy mẩu tin hoặc duyệt qua toàn bộ các khối của tập tin và trong trường hợp đó thì mẩu tin không tồn tại trong tập tin.

Thêm mẩu tin mới: Việc thêm một mẩu tin có thể thực hiện đơn giản bằng cách đưa mẩu tin này vào khối cuối cùng của tập tin nếu như khối đó còn chỗ trống. Ngược lại nếu khối cuối cùng đã hết chỗ thì xin cấp thêm một khối mới, thêm mẩu tin vào khối mới và nối khối mới vào cuối danh sách.

Sửa đổi mẩu tin: Để sửa đổi một mẩu tin có giá trị cho trước, ta tìm mẩu tin cần sửa đổi rồi thực hiện các sửa đổi cần thiết sau đó ghi lại mẩu tin vào vị trí cũ trong tập tin.

Xoá mẩu tin: Để xoá một mẩu tin, trước hết ta cũng cần tìm mẩu tin đó, nếu tìm thấy ta có thể thực hiện một trong các cách xoá sau đây:

Một là xoá mẩu tin cần xoá trong khối lưu trữ nó, nếu sau khi xoá, khối trở nên rỗng thì xoá khối khỏi danh sách (giải phóng bộ nhớ).

Hai là đánh dấu xoá mẩu tin bằng một cách nào đó. Nghĩa là chỉ xoá mẩu tin một cách logic, vùng không gian nhớ vẫn còn dành cho mẩu tin. Việc đánh dấu có thể được thực hiện bằng một trong

· Thay thế mẩu tin bằng một giá trị nào đó mà giá trị này không bao giờ là giá trị thật của bất kỳ một mẩu tin nào.

· Mỗi một mẩu tin có một bít xóa, bình thường bit xóa của mẩu tin có giá trị 0, muốn xóa mẩu tin ta đặt cho bit xóa giá trị 1. Với phương pháp này thì một mẩu tin sau khi bị đánh dấu xoá cũng có thể phục hồi được bằng cách đặt bit xoá của mẩu tin giá trị 0.

Đánh giá: Đây là một phương pháp tổ chức tập tin đơn giản nhất nhưng kém hiệu quả nhất. Ta thấy tập tin là một danh sách liên kết của các khối nên các thao tác trên tập tin đều đòi hỏi phải truy xuất hầu như tất cả các khối, từ khối đầu tiên đến khối cuối cùng.

I **IV.4.2- Tăng tốc độ cho các thao tác tập tin**

Nhược điểm của cách tổ chức tập tin tuần tự ở trên là các thao tác trên tập tin rất chậm. Để cải thiện tốc độ thao tác trên tập tin, chúng ta phải tìm cách giảm số phép truy xuất khối. Muốn vậy phải tìm các cấu trúc sao cho khi tìm một mẩu tin chỉ cần phép truy xuất một số nhỏ các khối của tập tin.

Để tạo ra các tổ chức tập tin như vậy chúng ta phải giả sử rằng mỗi mẩu tin có một khoá (key), đó là một tập hợp các trường mà căn cứ vào đó ta có thể phân biệt các mẩu tin với nhau. Chẳng hạn mã sinh viên trong mẩu tin về sinh viên, biển số xe trong quản lí các phương tiện vận tải đường bộ.

Sau đây ta sẽ xét một số cấu trúc như thế.

IV.4.3- Tập tin băm (hash files)

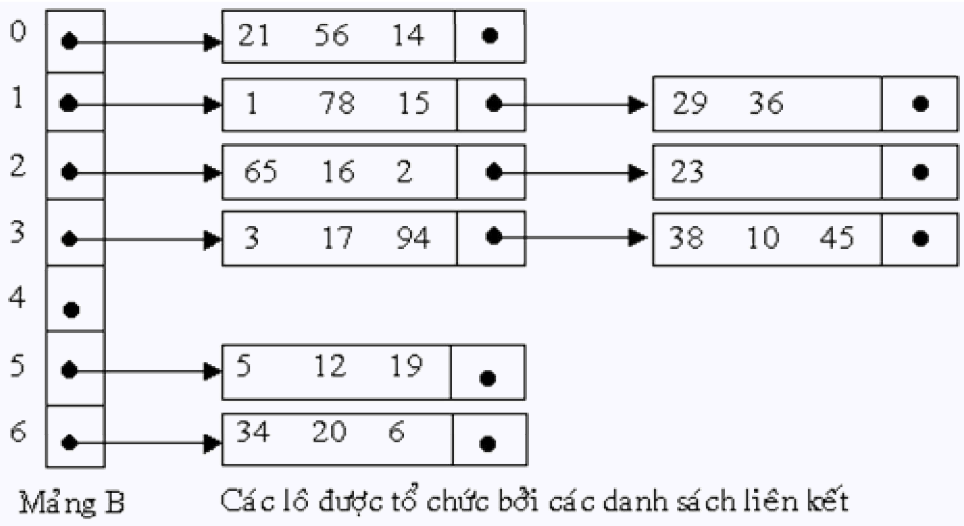
Tổ chức: Ta sẽ sử dụng bảng băm mở để lưu trữ tập tin. Bảng băm là một bảng có m phần tử, mỗi phần tử được đánh số từ 0 đến m-1 (đơn giản nhất là mảng một chiều B gồm m phần tử B[0], B[1], ..., B[m-1]). Mỗi phần tử là một con trỏ, trỏ tới phần tử đầu tiên của danh sách liên kết các khối.

Để phân phối các mẩu tin có khóa x vào trong các danh sách liên kết, ta dùng hàm băm (hash function). Hàm băm h(x) ánh xạ mỗi giá trị khoá x với một số nguyên từ 0 đến m-1. Nếu h(x) = i thì mẩu tin r có khóa x sẽ được đưa vào một khối nào đó trong danh sách liên kết được trỏ bởi B[i].

Có nhiều phương pháp để xác định hàm băm. Cách đơn giản nhất là “nguyên hóa” giá trị khóa x (nếu x không phải là một số nguyên) sau đó ta cho h(x) = x MOD m.

Ví dụ 4-5: Một tập tin có 24 mẩu tin với giá trị khóa là các số nguyên: 3, 5, 12, ,65, 34, 20, 21, 17, 56, 1, 16, 2, 78, ,94, 38 ,15 ,23, 14, 10, 29, 19, 6, 45, 36

Giả sử chúng ta có thể tổ chức tập tin này vào trong bảng băm gồm 7 phần tử và giả sử mỗi khối có thể chứa được tối đa 3 mẩu tin. Với mỗi mẩu tin r có khóa là x ta xác định h(x) = x MOD 7 và đưa mẩu tin r vào trong một khối của danh sách liên kết được trỏ bởi B[h(x)].



Hình 4-2: Tập tin được tổ chức bởi bảng băm

Tìm mẩu tin: Để tìm một mẩu tin r có khóa là x, chúng ta xác định h(x) chẳng hạn h(x) = i khi đó ta chỉ cần tìm r trong danh sách liên kết được trỏ bởi B[i]. Chẳng hạn để tìm mẩu tin r có khóa là 36, ta tính h(36) = 36 MOD 7 = 1. Như vậy nếu mẩu tin r tồn tại trong tập tin thì nó phải thuộc một khối nào đó được trỏ bởi B[1].

Thêm mẩu tin: Để thêm mẩu tin r có khoá x, trước hết ta phải tìm xem đã có mẩu tin nào trong tập tin có khóa x chưa. Nếu có ta cho một thông báo lỗi vì theo giả thiết các mẩu tin không có khoá trùng nhau. Ngược lại ta sẽ tìm một khối (trong danh sách các khối của lô được trỏ bởi B[h(x)]) còn đủ chỗ trống cho mẩu tin r và thêm nó vào khối này. Nếu không còn khối nào đủ chổ cho mẩu tin mới ta yêu cầu hệ thống cấp phát một khối mới và đặt mẩu tin r vào khối này rồi nối khối mới này vào cuối danh sách liên kết của lô.

Xoá mẩu tin: Để xoá mẩu tin r có khoá x, trước hết ta tìm mẩu tin này và đặt bít xoá của nó. Ta cũng có thể xoá hẳn mẩu tin r và nếu việc xoá này làm khối trở nên rỗng thì ta giải phóng khối này (xoá khối khỏi danh sách liên kết các khối).

Đánh giá: Giả sử tập tin có n mẩu tin và mỗi khối lưu trữ được k mẩu tin thì tập tin cần n/k khối. Trung bình mỗi danh sách liên kết có n/km khối, mà chúng ta chỉ tìm trong một danh sách liên kết nên ta chỉ phải truy xuất n/km khối. Số này nhỏ hơn m lần so với cách tổ chức tập tin tuần tự (trong tập tin tuần tự ta cần truy xuất tất cả các khối, tức là n/k khối). Chẳng hạn với 24 mẩu tin như trong ví dụ trên, với cách tổ chức tập tin tuần tự ta cần đúng 8 khối để lưu trữ (vì mỗi khối chứa tối đa 3 mẩu tin). Như vậy để tìm một mẩu tin, chẳng hạn mẩu tin có khóa 36 chúng ta phải đọc đúng 8 khối (do mẩu tin có khóa 36 nằm trong khối cuối cùng). Nhưng với cách tổ chức tập tin bảng băm chúng ta chỉ cần trung bình 24/(3*7) lần đọc khối . Trong thực tế ta chỉ cần 2 lần đọc khối (vì mẩu tin có khóa 36 nằm trong khối thứ 2 của lô được trỏ bởi B[1]).

IV.4.4- Tập tin chỉ mục (index file)

Tổ chức: Một cách khác thường gặp là tập tin được sắp xếp theo khoá, rồi chúng ta tiến hành tìm kiếm như là tìm một từ trong từ điển, tức là tìm kiếm theo từ đầu tiên trên mỗi trang.

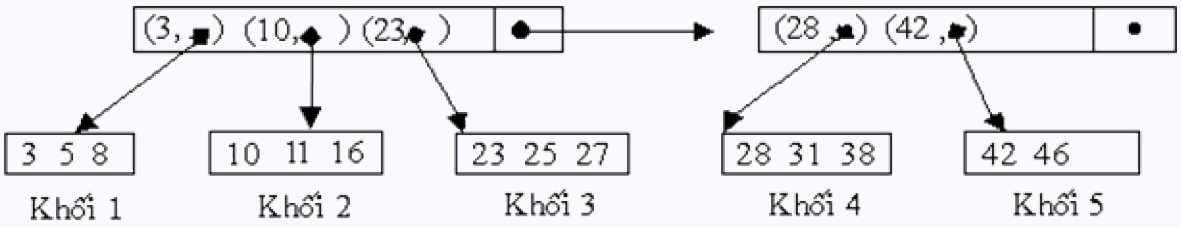
Để thực hiện được điều đó ta sử dụng hai tập tin: Tập tin chính và tập tin chỉ mục thưa (sparse index). Tập tin chính bao gồm các khối lưu các mẩu tin đã được sắp thứ tự theo giá trị khóa. Tập tin chỉ mục thưa bao gồm các khối chứa các cặp (x, p) trong đó x là khoá của mẩu tin đầu tiên trong một khối của tập tin chính, còn p là con trỏ, trỏ đến khối đó.

Để xây dựng được một cấu trúc như vậy, trước hết ta sắp xếp các mẩu tin theo giá trị khóa và phân phối chúng vào trong các khối của tập tin chính. Có hai cách phân phối là: Lấp đầy các khối bởi các mẩu tin. Hoặc chỉ phân phối vào các khối một số ít mẩu tin để còn những chỗ trống của khối dành cho việc xen các mẩu tin sau này. Sau khi phân phối các mẩu tin, ta bắt đầu duyệt qua từng khối để lập chỉ mục bằng cách lấy khoá của mẩu tin đầu tiên của mỗi khối cùng với địa chỉ khối để tạo thành một cặp trong tập tin chỉ mục. Cách phân phối các cặp (khóa, địa chỉ) vào cho các khối của tập tin chỉ mục cũng được tiến hành bằng một trong hai cách như đối với tập tin chính.

Ví dụ 4-6: Ta có tập tin gồm các mẩu tin với khoá là các số nguyên: 5, 8, 3, 11, 23, 10, 28, 25, 27, 38, 16, 46, 42, 31.

Đầu tiên ta sắp thứ tự các mẩu tin theo giá trị khóa: 3, 5, 8, 10, 11, 16, 23, 25, 27, 28, 31, 38, 42, 46.

Giả sử mỗi khối chứa được 3 mẩu tin, ta có thể phân phối các mẩu tin vào các khối bằng cách lấp đầy các khối bởi 3 mẩu tin. Sau đó tạo tập tin chỉ mục thưa cho tập tin chính theo cách đã trình bày ở trên. Kết quả ta được cấu trúc như trong hình sau:



Hình 4-3: Tập tin chỉ mục

Tìm kiếm: Để tìm mẩu tin r có khoá x, ta phải tìm cặp (z,p) với z là giá trị lớn nhất và

$z \leq x$. Mẩu tin r có khoá x nếu tồn tại thì sẽ nằm trong khối được trỏ bởi p.

Chẳng hạn để tìm mẩu tin r có khoá 11, ta tìm trong tập tin chỉ mục được cặp (10, ·), chứa địa chỉ khối 2, Ta chỉ cần tìm mẩu tin có khoá 11 trong khối này.

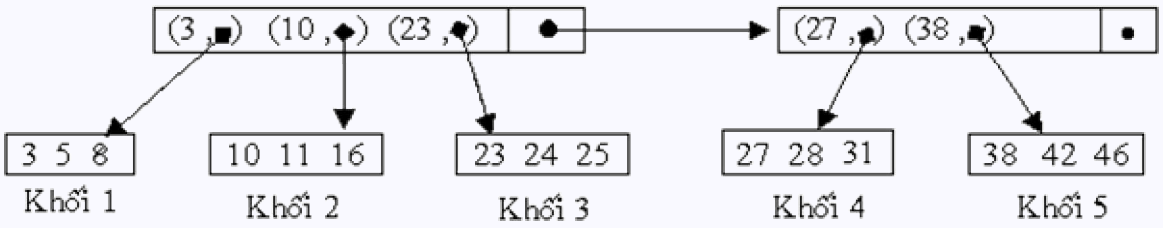
Tìm một mẩu tin trong một khối của tập tin chính có thể tiến hành bằng tìm kiếm tuần tự hoặc bằng tìm kiếm nhị phân bởi lẽ các mẩu tin trong một khối đã được sắp thứ tự.

Thêm mẩu tin: Giả sử tập tin chính được lưu trong các khối B1, B2, ..., Bm. Để xen một mẩun tin r với khóa x vào trong tập tin, ta phải dùng thủ tục tìm kiếm để xác định một khối Bi nào đó có thể chứa mẩu tin r. Nếu Bi còn chỗ trống thì xen r vào đúng vị trí của nó trong Bi. Ta phải chỉnh lại tập tin chỉ mục nếu mẩu tin mới trở thành mẩu tin đầu tiên trong khối Bi. Nếu Bi không còn chỗ trống để xen thì ta phải xét khối Bi+1 để có thể chuyển mẩu tin cuối cùng trong khối Bi thành mẩu tin đầu tiên của khối Bi+1 và xen mẩu tin r vào đúng vị trí của nó trong khối Bi . Điều chỉnh lại tập tin chỉ mục cho phù hợp với trạng thái mới của Bi+1. Quá trình này có thể dẫn đến việc ta phải xét khối Bm, nếu Bm đã hết chỗ thì yêu cầu hệ thống cấp thêm một khối mới Bm+1, chuyển mẩu tin cuối cùng của Bm sang Bm+1, mẩu tin cuối cùng của Bm-1 sang Bm….xen mẩu tin r vào khối Bi và cập nhật lại tập tin chỉ mục.

Đến đây chúng ta sẽ thấy rằng cách phân phối các mẩu tin vào trong các khối của tập tin

chính, nếu còn trừ lại một số chỗ trống tuy có hao tốn bộ nhớ ngoài nhưng lại rất thuận tiện cho việc xen thêm các mẩu tin.

Ví dụ 4-7: Chẳng hạn ta cần xen mẩu tin r với khóa x=24 vào trong tập tin được biểu diễn trong hình 4-3. Thủ tục tìm x trong tập tin chỉ mục xác định được khối cần xen r là khối 3. Vì khối 3 có đủ 3 mẩu tin nên phải chuyển mẩu tin có khóa 27 sang khối 4 và xen r vào khối 3. Vì khối 4 cũng đã đủ 3 mẩu tin nên xét khối 5 đểi chuyển mẩu tin có khóa 38 thành mẩu tin đầu tiên của khối 5. Vì mẩu tin đầu tiên của khối 4 bây giờ có khóa 27 nên ta phải sửa lại giá trị này trong cặp của tập tin chỉ mục tương ứng với khối 4. Ta cũng phải làm tương tự đối với khối 5. Cấu trúc của tập tin sau khi thêm mẩu tin r có khóa 24 như sau:



Hình 4-4: Xen mẩu tin vào tập tin chỉ mục

Xoá mẩu tin: Để xoá mẩu tin r có khoá x, trước hết ta cần tìm r, nếu không tìm thấy thì thông báo lỗi, ngược lại ta xoá mẩu tin r trong khối chứa nó, nếu mẩu tin bị xoá là mẩu tin đầu tiên của khối thì phải cập nhật lại giá trị khoá trong tập tin chỉ mục. Trong trường hợp khối trở nên rỗng sau khi xoá mẩu tin thì giải phóng khối đó và xoá cặp (khoá, con trỏ) của khối trong tập tin chỉ mục. Việc xoá trong tập tin chỉ mục cũng có thể dẫn đến việc giải phóng khối trong tập tin này.

Đánh giá: Ta thấy việc tìm một mẩu tin chỉ đòi hỏi phải đọc chỉ một số nhỏ các khối (một khối trong tập tin chính và một số khối trong tập tin chỉ mục). Tuy nhiên trong việc xen thêm mẩu tin, như trên đã nói, có thể phải đọc và ghi tất cả các khối trong tập tin chính. Đây chính là nhược điểm của tập tin chỉ mục.

IV.4.5- Cấu trúc B-cây

Cây tìm kiếm m-phân

Cây tìm kiếm m-phân (m-ary tree) là sự tổng quát hoá của cây tìm kiếm nhị phân trong đó mỗi nút có thể có m nút con. Giả sử n1 và n2 là hai con của một nút nào đó, n1 bên trái n2 thì tất cả các con của n1 có giá trị nhỏ hơn giá trị của các nút con của n2.

Chúng ta sẽ sử dụng cây m-phân để lưu trữ các mẩu tin trong tập tin trên bộ nhớ ngoài. Mỗi một nút biểu diễn cho một khối vật lý trong bộ nhớ ngoài. Trong đó các nút lá lưu trữ các mẩu tin của tập tin chính. Các nút trong lưu trữ m con trỏ, trỏ tới m nút con, nó cũng lưu trữ m-1 khoá để phân chia các giá trị trong các nút con cháu.

Nếu ta dùng cây tìm kiếm nhị phân n nút để lưu trữ một tập tin thì cần trung bình logn phép truy xuất khối để tìm kiếm một mẩu tin. Nếu ta dùng cây tìm kiếm m-phân để lưu trữî một tập tin thì chỉ cần logmn phép truy xuất khối để tìm kiếm một mẩu tin. Sau đây chúng ta sẽ nghiên cứu một trường hợp đặc biệt của cây tìm kiếm m-phân là B-cây.

B-cây (B-tree)

B-cây bậc m là cây tìm kiếm m-phân cân bằng có các tính chất sau:

· Nút gốc hoặc là lá hoặc có ít nhất hai nút con,

· Mỗi nút, trừ nút gốc và nút lá, có từ **[m/2] đến m** nút con và

· Các đường đi từ gốc tới lá có cùng độ dài.

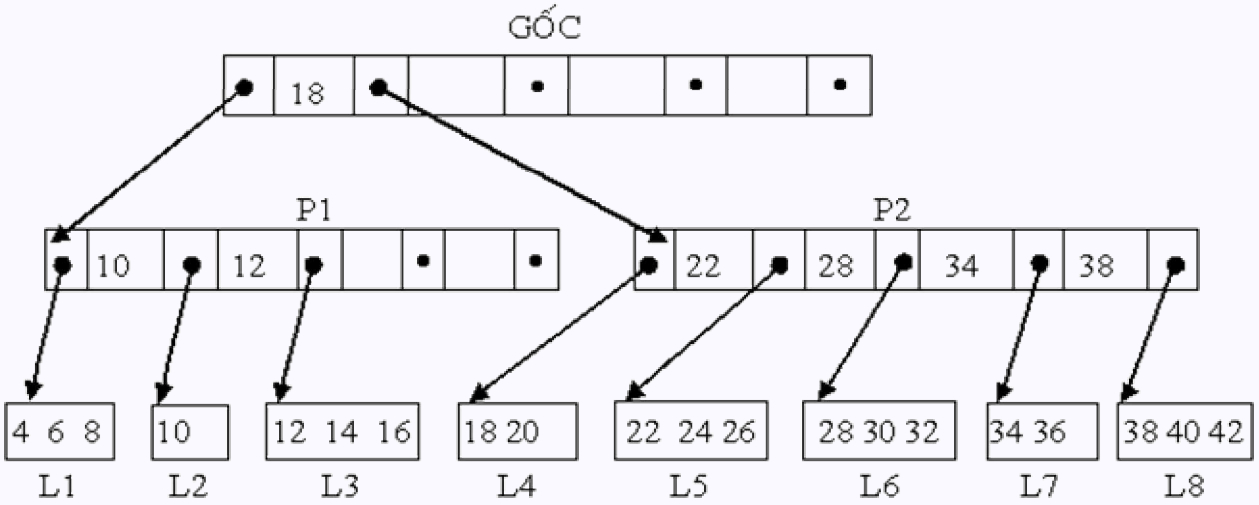
Tổ chức: Ta có thể sử dụng B-cây bậc m để lưu trữ tập tin như sau:

Mỗi nút trên cây là một khối trên đĩa, các mẩu tin của tập tin được sắp thứ tự theo khoá và được lưu trữ trong các nút lá trên B-cây. Giả sử mỗi nút lá lưu trữ được nhiều nhất b mẩu tin.

Mỗi nút không phải là nút lá có dạng (p0, k1, p1, k2, p2, ..., kn, pn), với pi ($0 \le i \le n$) lài con trỏ, trỏ tới nút con thứ i của nút đó và ki là các giá trị khóa. Các khoá trong một nút được sắp thứ tự, tức là k1 < k2 < ... < kn.

Tất cả các khoá trong cây con được trỏ bởi p0 đều nhỏ hơn k1. Tất cả các khoá nằm trong cây con được trỏ bởi pi ($1 \le i \le n$) đều lớn hơn hoặc bằng ki và nhỏ hơn ki+1. Tất cả các khoá nằm trong cây con được trỏ bởi pn đều lớn hơn hoặc bằng kn.

Ví dụ 4-8: Cho tập tin bao gồm 20 mẩu tin với giá trị khóa là các số nguyên được tổ chức thành B-cây **bậc 5** với các nút lá chứa được nhiều nhất **3 mẩu tin**.



Hình 4-5: Tập tin B-cây bậc 5

Tìm kiếm: Để tìm kiếm một mẩu tin r có khoá là x chúng ta sẽ lần từ nút gốc đến nút lá chứa r (nếu r tồn tại trong tập tin). Tại mỗi bước ta đưa nút trong (p0, k1, p1, ..., kn, pn) vào bộ nhớ trong và xác định mối quan hệ giữa x với các giá trị khóa ki.

· Nếu $ki \le x < ki+1$ ($1 \le i < n$) chúng ta sẽ xét tiếp nút được trỏ bởi pi,

· Nếu x < k1 ta sẽ xét tiếp nút được trỏ bởi p0 và

· Nếu $x \ge kn$ ta sẽ xét tiếp nút được trỏ bởi pn.

Quá trình trên sẽ dẫn đến việc xét một nút lá. Trong nút lá này ta sẽ tìm mẩu tin r với khóa x bằng tìm kiếm tuần tự hoặc tìm kiếm nhị phân.

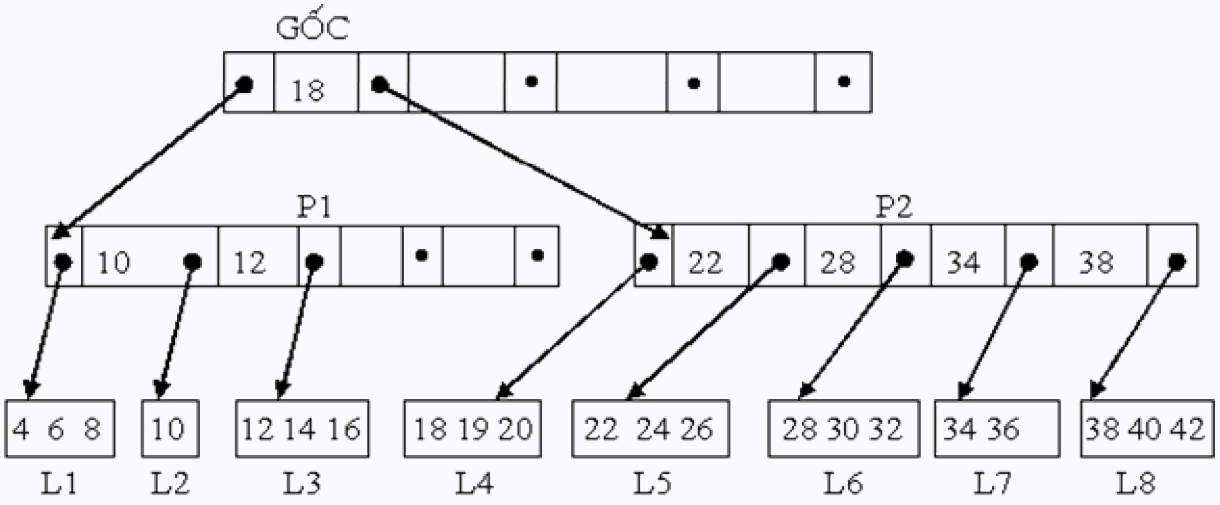
Thêm mẩu tin: Để thêm một mẩu tin r có khoá là x vào trong B-cây, ta áp dụng thủ tục tìm kiếm nói trên để xác định nút lá L mà mẩu tin này phải nằm trong đó. Nếu khối L này còn đủ chổ cho r thì ta thêm r vào sao cho đúng thứ tự của nó trong khối L và giải thuật kết thúc. Nếu L không còn chỗ cho r thì ta yêu cầu hệ thống cấp phát một khối mới L'. Dời [b/2] (b là số mẩu tin nhiều nhất có thể lưu trong một khối) mẩu tin nằm ở phân nửa cuối khối L sang L' rồi xen r vào L hoặc L' sao cho việc xen đảm bảo thứ tự các khoá trong khối. Giả sử nút P là cha của L (P phải được biết vì thủ tục tìm đi từ gốc đến L phải thông qua P). Bây giờ ta áp dụng thủ tục xen đệ quy để xen vào P một khóa k' và con trỏ p' tương ứng của nút lá L' (k' là khoá của mẩu tin đầu tiên trong L'). Trong trường hợp trước khi xen k' và p' , P đã có đủ m con thì ta phải cấp thêm một khối mới P' và chuyển một số con của P sang P' và xen con mới vào P hoặc P' sao cho cả P và P' đều có ít nhất [m/2] con. Việc chia cắt P này đòi hỏi phaií xen một khóa và một con trỏ vào nút cha của P... Quá trình này có thể sẽ dẫn tới nút gốc và cũng có thể phải chia cắt nút gốc, trong trường hợp này phải tạo ra một nút gốc mới mà hai con của nó là hai nửa của nút gốc cũ. Khi đó chiều cao của B-cây sẽ tăng lên 1.

Ví dụ 4-9: Thêm mẩu tin r có khoá 19 vào tập tin được biểu diễn bởi B-cây trong ví dụ 4-8

· Quá trình tìm kiếm sẽ xuất phát từ GỐC đi qua P2 và dẫn tới nút lá L4

· Trong nút lá L4 còn đủ chỗ để xen r vào đúng vị trí và giải thuật kết thúc.

Kết quả việc xen ta được B-cây trong hình 4-6:



Hình 4-6: Xen thêm mẩu tin r có khoá 19 vào trong B-cây hình 4-5

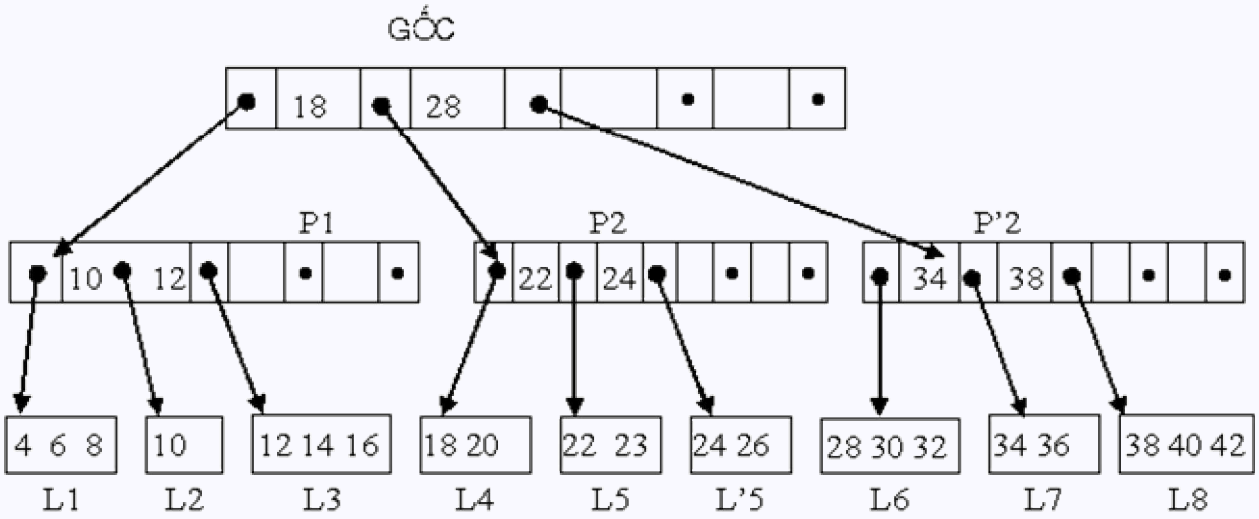
Ví dụ 4-10: Thêm mẩu tin r có khoá 23 vào trong tập tin biểu diễn bởi B-cây trong ví dụ 4-8 (hình 4-5)

· Quá trình tìm kiếm đi từ nút GỐC, qua P2 và tới nút lá L5.

· Vì L5 đã đủ 3 mẩu tin nên phải tạo ra một nút lá mới L'5 và chuyển 2 mẩu tin có khóa 24, 26 sang L'5 sau đó xen r vào L5.

· Giá trị khóa của mẩu tin đầu tiên trong L'5 là 24, ta phải xen 24 và con trỏ của L'5 vào P2, nhưng P2 đã có đuủ 5 con vậy cần tạo ra một nút mới P'2, chuyển các cặp khóa, con trỏ tương ứng với 34 và 38 sang P'2 và xen cặp con trỏ, khóa 24 vào P2.

· Do có một nút mới P'2 nên phải xen vào cha của P2 (Ở đây là nút GỐC) một cặp khóa, con trỏ trỏ tới P'2. Con trỏ p0 của nút P'2 trỏ tới nút lá L6, giá trị khóa đầu tiên của L6 là 28. Giá trị này phải được xen vào nút GỐC cùng với con trỏ của P'2.



Hình 4-7: Xen thêm mẩu tin r có khoá 23 vào trong B-cây hình 4-5

Xóa một mẩu tin: Để xóa mẩu tin r có khóa x, trước hết ta tìm đến nút lá L chứa mẩu tin r, xóa r khỏi L. Nếu r là mẩu tin đầu tiên của L, thì ta phải quay lui lên nút P là cha của L để đặt lại giá trị khóa của L trong P, giá trị mới này bằng giá trị khóa của mẩu tin mới đầu tiên của L. Trong trường hợp L là con đầu tiên của P, thì khóa của L không nằm trong P mà nằm trong tổ tiên của P, chúng ta phải quay lui lên mà sửa đổi.

Nếu sau khi xóa mẩu tin r mà L trở nên rỗng thì giải phóng L và nếu số con của P bây giờ nhỏ hơn [m/2] thì kiểm tra nút P' ngay bên trái hoặc bên phải và cùng mức với P. Nếu P' có ít nhất [m/2] + 1 con, chúng ta chuyển một con P' sang P. Lúc này cả P và P' có ít nhất [m/2] con. Sau đó ta phải cập nhật lại giá trị khóa của P và P' trong cha của chúng, và nếu cần chúng ta phải sửa cả trong tổ tiên của chúng.

Nếu P' có đúng [m/2] con, ta nối P và P' thành một nút có m con. Sau đó ta phải xóa khóa và con trỏ của P' trong nút cha của P'. Việc xóa này có thể phải quay lui lên tổ tiên của P'. Kết quả của quá trình xóa quay lui này có thể dẫn tới việc nối hai con của nút gốc, tạo nên một gốc mới và giải phóng nút gốc cũ, độ cao của cây giảm đi 1.

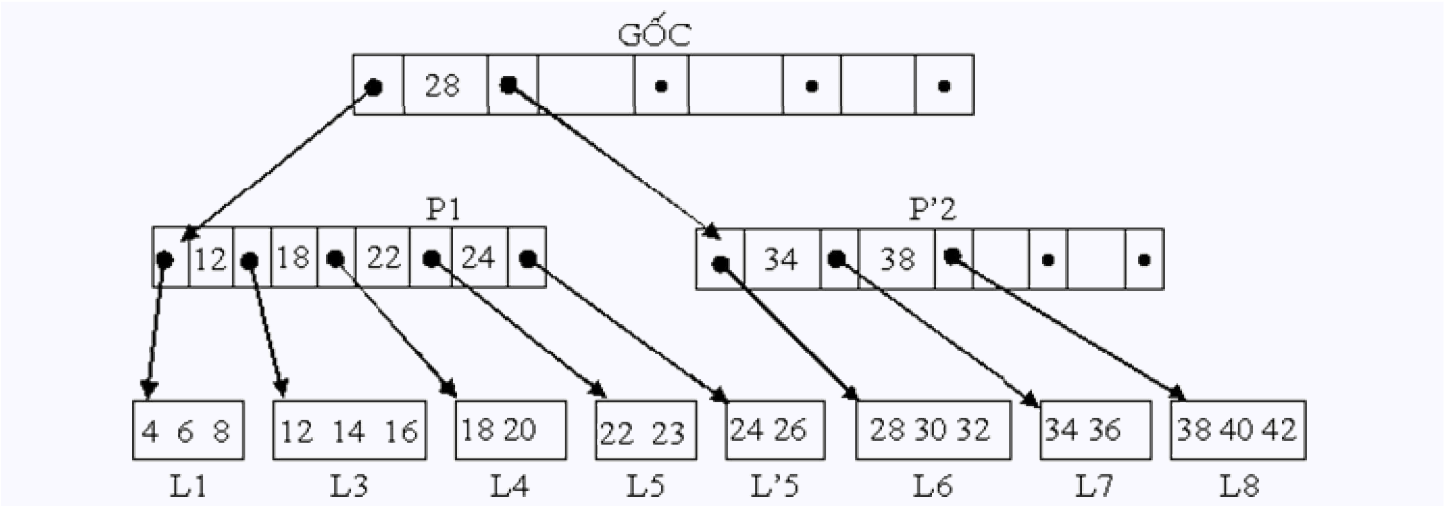
Ví dụ 4-11: Xóa mẩu tin r có khóa 38 trong tập tin biểu diễn bởi B-cây kết quả của ví dụ 4-10 (hình 4-6).

· Quá trình tìm kiếm, xuất phát từ nút GỐC, đi qua P'2 và lần đến nút lá L8,

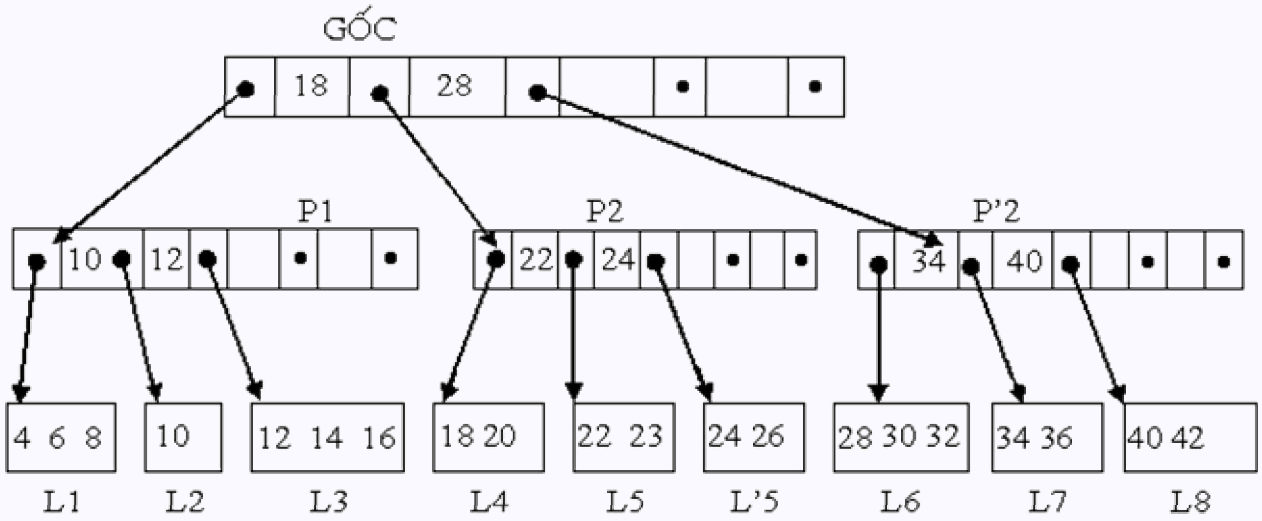
· Xóa mẩu tin r khỏi L8.

· Mẩu tin đầu tiên của L8 bây giờ có khóa 40,

· Sửa lại giá trị khóa của L8 trong P'2 (thay 38 bởi 40) ta được kết quả là B-cây sau:



Hình 4-9: Xoá mẩu tin r có khoá 10 vào trong B-cây hình 4-6



Hình 4-8: Xoá mẩu tin r có khoá 38 vào trong B-cây hình 4-6

Ví dụ 12: Xóa mẩu tin r có khóa 10 trong tập tin biểu diễn bởi B-cây kết quả của ví dụ 10.

· Quá trình tìm kiếm, xuất phát từ nút GỐC, đi qua P1 và lần đến nút lá L2.

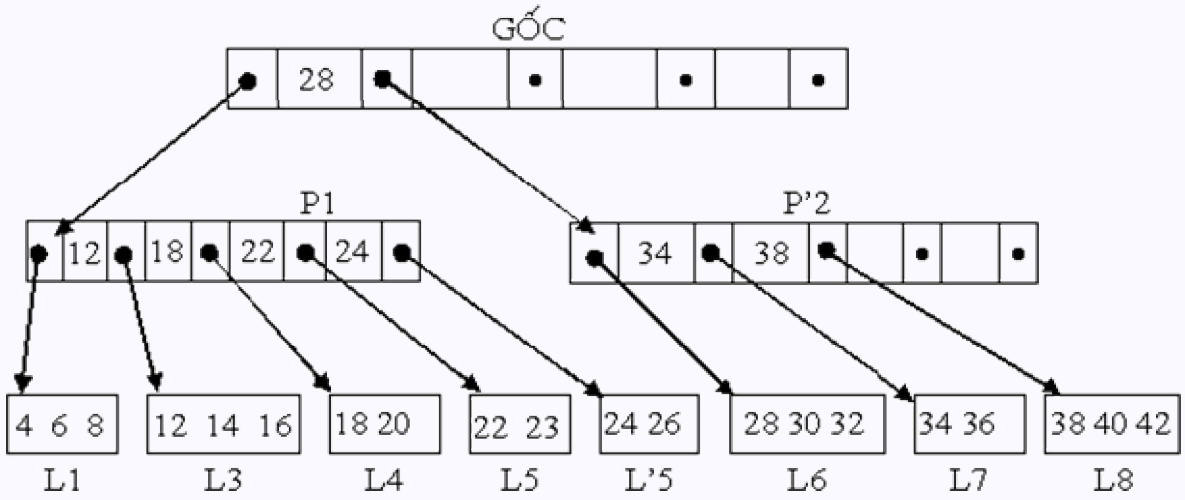
· Xóa mẩu tin r khỏi L2.

· L2 bây giờ trở nên rỗng, giải phóng L2.

· Xóa giá trị khóa 10 và con trỏ của L2 trong P1, P1 bây giờ chỉ có 2 con (2 < [5/2]).

· Xét nút P2, bên phải và cùng cấp với P1, P2 có đúng [5/2] = 3 con nên ta nối P1 và P2 để P1 có đúng 5 con,

· Xóa khóa và con trỏ của P2 trong nút GỐC, ta được B-cây kết quả như sau:



Hình 4-9: Xoá mẩu tin r có khoá 10 vào trong B-cây hình 4-6