**8.6. THUẬT TOÁN SẮP XẾP KIỂU PHÂN ĐOẠN (QUICKSORT)**

**8.6.1. Tư tưởng của QuickSort**

QuickSort là một phương pháp sắp xếp tốt nhất, nghĩa là dù dãy khoá thuộc kiểu dữ liệu có thứ tự nào, QuickSort cũng có thể sắp xếp được và không có một thuật toán sắp xếp nào nhanh hơn QuickSort về mặt tốc độ trung bình (theo tôi biết). Người sáng lập ra nó là C.A.R. Hoare đã mạnh dạn đặt tên cho nó là sắp xếp "NHANH".

Ý tưởng chủ đạo của phương pháp có thể tóm tắt như sau: Sắp xếp dãy khoá k1, k2, …, kn thì có thể coi là sắp xếp đoạn từ chỉ số 1 tới chỉ số n trong dãy khoá đó. Để sắp xếp một đoạn trong dãy khoá, nếu đoạn đó có  1 phần tử thì không cần phải làm gì cả, còn nếu đoạn đó có ít nhất 2 phần tử, ta chọn một khoá ngẫu nhiên nào đó của đoạn làm "chốt" (pivot). Mọi khoá nhỏ hơn khoá chốt được xếp vào vị trí đứng trước chốt, mọi khoá lớn hơn khoá chốt được xếp vào vị trí đứng sau chốt. Sau phép hoán chuyển như vậy thì đoạn đang xét được chia làm hai đoạn khác rỗng mà mọi khoá trong đoạn đầu đều  chốt và mọi khoá trong đoạn sau đều  chốt. Hay nói cách khác: Mỗi khoá trong đoạn đầu đều  mọi khoá trong đoạn sau. Và vấn đề trở thành sắp xếp hai đoạn mới tạo ra (có độ dài ngắn hơn đoạn ban đầu) bằng phương pháp tương tự.

procedure QuickSort;

procedure Partition(L, H: Integer); {Sắp xếp đoạn từ kL, kL+1, …, kH}

var

i, j: Integer;

Pivot: TKey; {Biến lưu giá trị khoá chốt}

begin

if L  H then Exit; {Nếu đoạn chỉ có  1 phần tử thì không phải làm gì cả}

Pivot := kRandom(H-L+1)+L; {Chọn một khoá ngẫu nhiên trong đoạn làm khoá chốt} i := L;

j := H; {i := vị trí đầu đoạn; j := vị trí cuối đoạn}

repeat

while ki < Pivot do i := i + 1; {Tìm từ đầu đoạn khoá  khoá chốt}

while kj > Pivot do j := j - 1; {Tìm từ cuối đoạn khoá  khoá chốt}

{Đến đây ta tìm được hai khoá ki và kj mà ki  key  kj}

if i  j then

begin

if i < j then {Nếu chỉ số i đứng trước chỉ số j thì đảo giá trị hai khoá ki và kj}

<Đảo giá trị ki và kj> {Sau phép đảo này ta có: ki  key  kj }

i := i + 1;

j := j - 1;

end;

until i > j;

Partition(L, j);

Partition(i, H); {Sắp xếp hai đoạn con mới tạo ra}

end;

begin

Partition(1, n);

end;

Ta thử phân tích xem tại sao đoạn chương trình trên hoạt động đúng: Xét vòng lặp repeat…until trong lần lặp đầu tiên, **vòng lặp while thứ nhất chắc chắn sẽ tìm được khoá ki khoá chốt** bởi chắc chắn tồn tại trong đoạn một khoá bằng khóa chốt. Tương tự như vậy, **vòng lặp while thứ hai chắc chắn tìm được khoá kj  khoá chốt**. Nếu như khoá ki đứng trước khoá kj thì ta đảo giá trị hai khoá, cho i tiến và j lùi. Khi đó ta có nhận xét rằng mọi khoá đứng trước vị trí i sẽ phải **** khoá chốt và mọi khoá đứng sau vị trí j sẽ phải **** khoá chốt.

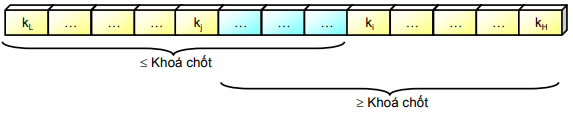


**Vòng lặp trong của QuickSort**

Điều này đảm bảo cho vòng lặp repeat…until tại bước sau, hai vòng lặp while…do bên trong chắc chắn lại tìm được hai khoá ki và kj mà ki **** khoá chốt **** kj, nếu khoá ki đứng trước khoá kj thì lại đảo giá trị của chúng, cho i tiến về cuối một bước và j lùi về đầu một bước. Vậy thì quá trình hoán chuyển phần tử trong vòng lặp repeat…until sẽ đảm bảo tại mỗi bước:

* Hai vòng lặp while…do bên trong luôn tìm được hai khoá ki, kj mà ki **** khoá chốt **** kj. Không có trường hợp hai chỉ số i, j chạy ra ngoài đoạn (luôn luôn có L **** i, j **** H).
* Sau mỗi phép hoán chuyển, mọi khoá đứng trước vị trí i luôn **** khoá chốt và mọi khoá đứng sau vị trí j luôn **** khoá chốt.

Vòng lặp repeat …until sẽ kết thúc khi mà chỉ số i đứng phía sau chỉ số j.



**Trạng thái trước khi gọi đệ quy**

Theo những nhận xét trên, nếu có một khoá nằm giữa kj và ki thì khoá đó phải đúng bằng khoá chốt và nó đã được đặt ở vị trí đúng của nó, nên có thể bỏ qua khoá này mà chỉ xét hai đoạn ở hai đầu. Công việc còn lại là gọi đệ quy để làm tiếp với đoạn từ kL tới kj và đoạn từ ki tới kH. Hai đoạn này ngắn hơn đoạn đang xét bởi vì L **** j < i **** H. Vậy thuật toán không bao giờ bị rơi vào quá trình vô hạn mà sẽ dừng và cho kết quả đúng đắn.

Xét về độ phức tạp tính toán:

Trường hợp tồi tệ nhất, là khi chọn khoá chốt, ta chọn phải khoá nhỏ nhất hay lớn nhất trong đoạn, khi đó phép phân đoạn sẽ chia thành một đoạn gồm n - 1 phần tử và đoạn còn lại chỉ có 1 phần tử. Có thể chứng minh trong trường hợp này, thời gian thực hiện giải thuật T(n) = O(n2) Trường hợp tốt nhất, phép phân đoạn tại mỗi bước sẽ chia được thành hai đoạn bằng nhau. Tức là khi chọn khoá chốt, ta chọn đúng trung vị của dãy khoá. Có thể chứng minh trong trường hợp này, thời gian thực hiện giải thuật T(n) = O(nlog2n)

Trường hợp các khoá được phân bố ngẫu nhiên, thì trung bình thời gian thực hiện giải thuật cũng là **T(n) = O(nlog2n).**

Việc tính toán chi tiết, đặc biệt là khi xác định T(n) trung bình, phải dùng các công cụ toán phức tạp, ta chỉ công nhận những kết quả trên.

**8.6.2. Vài cải tiến của QuickSort**

Việc chọn chốt cho phép phân đoạn quyết định hiệu quả của QuickSort, nếu chọn chốt không tốt, rất có thể việc phân đoạn bị suy biến thành trường hợp xấu khiến QuickSort hoạt động chậm và tràn ngăn xếp chương trình con khi gặp phải dây chuyền đệ qui quá dài. Một cải tiến sau có thể khắc phục được hiện tượng tràn ngăn xếp nhưng cũng hết sức chậm trong trường hợp xấu, kỹ thuật này khi đã phân được [L, H] được hai đoạn con [L, j] và [i, H] thì chỉ gọi đệ quy để tiếp tục đối với đoạn ngắn, và lặp lại quá trình phân đoạn đối với đoạn dài.

procedure QuickSort;

procedure Partition(L, H: Integer); {Sắp xếp đoạn từ kL, kL+1, …, kH}

var

i, j: Integer;

begin

repeat

if L **** H then Exit;

<Phân đoạn [L, H] được hai đoạn con [L, j] và [i, H]>

if <đoạn [L, j] ngắn hơn đoạn [i, H]> then

begin

Partition(L, j);

L := i;

end

else

begin

Partition(i, H);

H := j;

end;

until False;

end;

begin

Partition(1, n);

end;

Cải tiến thứ hai đối với QuickSort là quá trình phân đoạn nên chỉ làm đến một mức nào đó, đến khi đoạn đang xét có độ dài  M (M là một số nguyên tự chọn nằm trong khoảng từ 9 tới 25) thì không phân đoạn tiếp mà nên áp dụng thuật toán sắp xếp kiểu chèn.

Cải tiến thứ ba của QuickSort là: Nên lấy trung vị của một dãy con trong đoạn để làm chốt, (trung vị của một dãy n phần tử là phần tử đứng thứ n / 2 khi sắp thứ tự). Cách chọn được đánh giá cao nhất là chọn trung vị của ba phần tử đầu, giữa và cuối đoạn.

Cuối cùng, ta có nhận xét: QuickSort là một công cụ sắp xếp mạnh, chỉ có điều khó chịu gặp phải là trường hợp suy biến của QuickSort (quá trình phân đoạn chia thành một dãy rất ngắn và một dãy rất dài). Và điều này trên phương diện lý thuyết là không thể khắc phục được: Ví dụ với n = 10000.

Nếu như chọn chốt là khoá đầu đoạn (Thay dòng chọn khoá chốt bằng Pivot := kL) hay chọn chốt là khoá cuối đoạn (Thay bằng Pivot := kH) thì với dãy sau, chương trình hoạt động rất chậm:

(1, 2, 3, 4, 5, …, 9999, 10000)

Nếu như chọn chốt là khoá giữa đoạn (Thay dòng chọn khoá chốt bằng Pivot := k(L+H) div 2) thì với dãy sau, chương trình cũng rất chậm:

(1, 2, …, 4999, 5000, 5000, 4999, …, 2, 1)

Trong trường hợp chọn chốt là trung vị dãy con hay chọn chốt ngẫu nhiên, thật khó có thể tìm ra một bộ dữ liệu khiến cho QuickSort hoạt động chậm. Nhưng ta cũng cần hiểu rằng với mọi chiến lược chọn chốt, trong 10000! dãy hoán vị của dãy (1, 2, … 10000) thế nào cũng có một dãy làm QuickSort bị suy biến, tuy nhiên trong trường hợp chọn chốt ngẫu nhiên, xác suất xảy ra dãy này quá nhỏ tới mức ta không cần phải tính đến, như vậy khi đã chọn chốt ngẫu nhiên thì ta không cần phải quan tâm tới ngăn xếp đệ quy, không cần quan tâm tới kỹ thuật khử đệ quy và vấn đề suy biến của QuickSort.