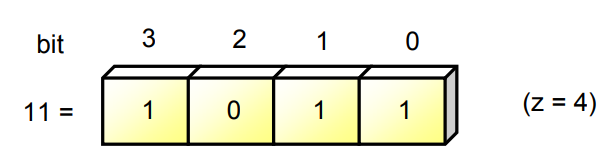
**8.10. THUẬT TOÁN SẮP XẾP BẰNG CƠ SỐ (RADIXSORT)**

Bài toán đặt ra là: Cho dãy khoá là các số tự nhiên k1, k2, …, kn hãy sắp xếp chúng theo thứ tự không giảm. (Trong trường hợp ta đang xét, TKey là kiểu số tự nhiên)

**8.10.1. Sắp xếp cơ số theo kiểu hoán vị các khoá (Exchange RadixSort)**

Hãy xem lại thuật toán QuickSort, tại bước phân đoạn nó phân đoạn đang xét thành hai đoạn thoả mãn mỗi khoá trong đoạn đầu  mọi khoá trong đoạn sau và thực hiện tương tự trên hai đoạn mới tạo ra, việc phân đoạn được tiến hành với sự so sánh các khoá với giá trị một khoá chốt.

Đối với các số nguyên thì ta có thể coi mỗi số nguyên là một dãy z bit đánh số từ bit 0 (bit ở hàng đơn vị) tới bit z - 1 (bit cao nhất). Ví dụ:



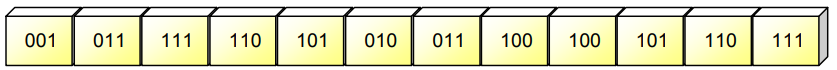
**Đánh số các bit**

Vậy thì tại bước phân đoạn dãy khoá từ k1 tới kn, ta có thể đưa những khoá có bit cao nhất là 0 về đầu dãy, những khoá có bit cao nhất là 1 về cuối dãy. Dễ thấy rằng những khoá bắt đầu bằng bit 0 sẽ phải nhỏ hơn những khoá bắt đầu bằng bit 1. Tiếp tục quá trình phân đoạn với hai đoạn dãy khoá: Đoạn gồm các khoá có bit cao nhất là 0 và đoạn gồm các khoá có bit cao nhất là 1. Với những khoá thuộc cùng một đoạn thì có bit cao nhất giống nhau, nên ta có thể áp dụng quá trình phân đoạn tương tự trên theo bit thứ z - 2 và cứ tiếp tục như vậy …

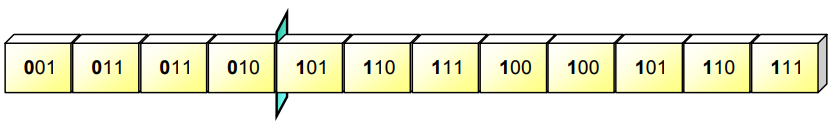
Quá trình phân đoạn kết thúc nếu như đoạn đang xét là rỗng hay ta đã tiến hành phân đoạn đến tận bit đơn vị, tức là tất cả các khoá thuộc một trong hai đoạn mới tạo ra đều có bit đơn vị bằng nhau (điều này đồng nghĩa với sự bằng nhau ở tất cả những bit khác, tức là bằng nhau về giá trị khoá).

Ví dụ:

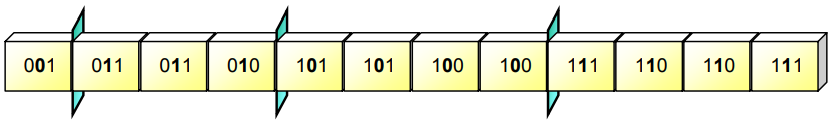
Xét dãy khoá: 1, 3, 7, 6, 5, 2, 3, 4, 4, 5, 6, 7. Tương ứng với các dãy 3 bit:



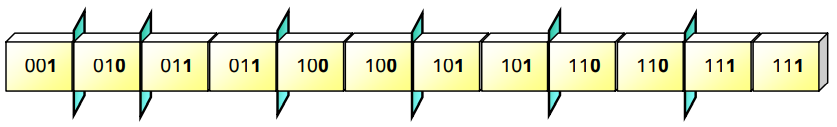
Trước hết ta chia đoạn dựa vào bit 2 (bit cao nhất):



Sau đó chia tiếp hai đoạn tạo ra dựa vào bit 1:



Cuối cùng, chia tiếp những đoạn tạo ra dựa vào bit 0:



Ta được dãy khoá tương ứng: 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7 là dãy khoá sắp xếp.

Quá trình chia đoạn dựa vào bit b có thể chia thành một đoạn rỗng và một đoạn gồm toàn bộ các phần tử còn lại, nhưng việc chia đoạn không bao giờ bị rơi vào quá trình đệ quy vô hạn bởi những lần đệ quy tiếp theo sẽ phân đoạn dựa vào bit b - 1, b - 2 …và nếu xét đến bit 0 sẽ phải dừng lại. Công việc còn lại là cố gắng hiểu đoạn chương trình sau và phân tích xem tại sao nó hoạt động đúng:

procedure ExchangeRadixSort;

var

z: Integer; {Độ dài dãy bit biểu diễn mỗi khoá}

procedure Partition(L, H, b: Integer); {Phân đoạn [L, H] dựa vào bit b}

var

i, j: Integer;

begin

if L **** H then Exit;

i := L;

j := H;

repeat

{Hai vòng lặp trong dưới đây luôn cầm canh i < j}

while (i < j) and (Bit b của ki = 0) do i := i + 1; {Tìm khoá có bit b = 1 từ đầu đoạn}

while (i < j) and (Bit b của kj = 1) do j := j - 1; {Tìm khoá có bit b = 0 từ cuối đoạn}

<Đảo giá trị ki cho kj>; until i = j;

if <Bit b của kj = 0> then j := j + 1; {j là điểm bắt đầu của đoạn có bit b là 1}

if b > 0 then {Chưa xét tới bit đơn vị}

begin

Partition(L, j - 1, b - 1);

Partition(j, R, b - 1); end;

end;

begin

<Dựa vào giá trị lớn nhất của dãy khoá, xác định z là độ dài dãy bit biểu diễn mỗi khoá>

Partition(1, n, z - 1);

end;

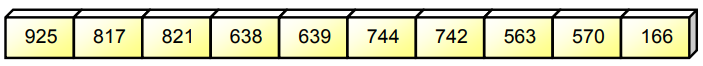
Với RadixSort, ta hoàn toàn có thể làm trên hệ cơ số R khác chứ không nhất thiết phải làm

trên hệ nhị phân (ý tưởng cũng tương tự như trên), tuy nhiên quá trình phân đoạn sẽ không phải chia làm 2 mà chia thành R đoạn. Về độ phức tạp của thuật toán, ta thấy để phân đoạn bằng một bit thì thời gian sẽ là C.n để chia tất cả các đoạn cần chia bằng bit đó (C là hằng số). Vậy tổng thời gian phân đoạn bằng z bit sẽ là C.n.z. **Trong trường hợp xấu nhất, độ phức tạp của RadixSort là O(n.z)**. Và **độ phức tạp trung bình của RadixSort là O(n.min(z, log2n))**.

Nói chung, RadixSort cài đặt như trên chỉ thể hiện tốc độ tối đa trên các hệ thống cho phép xử lý trực tiếp trên các bit: Hệ thống phải cho phép lấy một bit ra dễ dàng và thao tác với thời gian nhanh hơn hẳn so với thao tác trên Byte và Word. Khi đó RadixSort sẽ tốt hơn nhiều QuickSort. (Ta thử lập trình sắp xếp các dãy nhị phân độ dài z theo thứ tự từ điển để khảo sát). Trên các máy tính hiện nay chỉ cho phép xử lý trực tiếp trên Byte (hay Word, DWord v.v…), việc tách một bit ra khỏi Byte đó để xử lý lại rất chậm và làm ảnh hưởng không nhỏ tới tốc độ của RadixSort. Chính vì vậy, tuy đây là một phương pháp hay, nhưng khi cài đặt cụ thể thì tốc độ cũng chỉ ngang ngửa chứ không thể qua mặt QuickSort được.

**8.10.2. Sắp xếp cơ số trực tiếp (Straight RadixSort)**

Ta sẽ trình bày phương pháp sắp xếp cơ số trực tiếp bằng một ví dụ: Sắp xếp dãy khoá:



Trước hết, ta sắp xếp dãy khoá này theo thứ tự tăng dần của chữ số hàng đơn vị bằng một thuật toán sắp xếp khác, được dãy khoá:



Sau đó, ta sắp xếp dãy khoá mới tạo thành theo thứ tự tăng dần của chữ số hàng chục bằng một thuật toán sắp xếp ổn định, được dãy khoá:



Vì thuật toán sắp xếp ta sử dụng là ổn định, nên nếu hai khoá có chữ số hàng chục giống nhau thì khoá nào có chữ số hàng đơn vị nhỏ hơn sẽ đứng trước. Nói như vậy có nghĩa là dãy khoá thu được sẽ có thứ tự tăng dần về giá trị tạo thành từ hai chữ số cuối.

Cuối cùng, ta sắp xếp lại dãy khoá theo thứ tự tăng dần của chữ số hàng trăm cũng bằng một thuật toán sắp xếp **ổn định**, thu được dãy khoá:



Lập luận tương tự như trên dựa vào tính ổn định của phép sắp xếp, dãy khoá thu được sẽ có thứ tự tăng dần về giá trị tạo thành bởi cả ba chữ số, đó là dãy khoá đã sắp.

**Nhận xét**:

Ta hoàn toàn có thể coi số chữ số của mỗi khoá là bằng nhau, như ví dụ trên nếu có số 15 trong dãy khoá thì ta có thể coi nó là 015.

Cũng từ ví dụ, ta có thể thấy rằng số lượt thao tác sắp xếp phải áp dụng đúng bằng số chữ số tạo thành một khoá. Với một hệ cơ số lớn, biểu diễn một giá trị khoá sẽ phải dùng ít chữ số hơn. Ví dụ số 12345 trong hệ thập phân phải dùng tới 5 chữ số, còn trong hệ cơ số 1000 chỉ cần dùng 2 chữ số AB mà thôi, ở đây A là chữ số mang giá trị 12 còn B là chữ số mang giá trị 345.

Tốc độ của sắp xếp cơ số trực tiếp phụ thuộc rất nhiều vào thuật toán sắp xếp ổn định tại mỗi bước. Không có một lựa chọn nào khác tốt hơn phép đếm phân phối. Tuy nhiên, phép đếm phân phối có thể không cài đặt được hoặc kém hiệu quả nếu như tập giá trị khoá quá rộng, không cho phép dựng ra dãy các biến đếm hoặc phải sử dụng dãy biến đếm quá dài (Điều này xảy ra nếu chọn hệ cơ số quá lớn).

Một lựa chọn khôn ngoan là nên chọn hệ cơ số thích hợp cho từng trường hợp cụ thể để dung hoà tới mức tối ưu nhất ba mục tiêu:

Việc lấy ra một chữ số của một số được thực hiện dễ dàng.

Sử dụng ít lần gọi phép đếm phân phối.

Phép đếm phân phối thực hiện nhanh

procedure StraightRadixSort;

const

radix = …; {Tuỳ chọn hệ cơ số radix cho hợp lý}

var

t: TArray; {Dãy khoá phụ}

p: Integer;

nDigit: Integer; {Số chữ số cho một khoá, đánh số từ chữ số thứ 0 là hàng đơn vị đến chữ số thứ nDigit - 1}

Flag: Boolean; {Flag = True thì sắp dãy k, ghi kết quả vào dãy t; Flag = False thì sắp dãy t, ghi kq vào k}

function GetDigit(Num: TKey; p: Integer): Integer; {Lấy chữ số thứ p của số Num (0p<nDigit)}

begin

GetDigit := Num div radixp mod radix; {Trường hợp cụ thể có thể có cách viết tốt hơn}

end;

{Sắp xếp ổn định dãy số x theo thứ tự tăng dần của chữ số thứ p, kết quả sắp xếp được chứa vào dãy số y}

procedure DCount(var x, y: TArray; p: Integer); {Thuật toán đếm phân phối, sắp từ x sang y}

var

c: array[0..radix - 1] of Integer; {cd là số lần xuất hiện chữ số d tại vị trí p}

i, d: Integer;

begin

for d := 0 to radix - 1 do cd := 0;

for i := 1 to n do

begin

d := GetDigit(xi, p);

cd := cd + 1;

end;

for d := 1 to radix - 1 do cd := cd-1 + cd; {các cd trở thành các mốc cuối đoạn}

for i := n downto 1 do {Điền giá trị vào dãy y}

begin

d := GetDigit(xi, p);

ycd := xi;

cd := cd - 1;

end;

end;

begin {Thuật toán sắp xếp cơ số trực tiếp}

<Dựa vào giá trị lớn nhất trong dãy khoá, xác định nDigit là số chữ số phải dùng cho mỗi khoá trong hệ radix>;

Flag := True;

for p := 0 to nDigit - 1 do {Xét từ chữ số hàng đơn vị lên, sắp xếp ổn định theo chữ số thứ p}

begin

if Flag then DCount(k, t, p)

else DCount(t, k, p);

Flag := not Flag; {Đảo cờ, dùng k tính t rồi lại dùng t tính k …}

end;

if not Flag then k := t; {Nếu kết quả cuối cùng đang ở trong t thì sao chép giá trị từ t sang k}

end;

Xét phép đếm phân phối, ta đã biết độ phức tạp của nó là O(max(radix, n)). Mà radix là một hằng số tự ta chọn từ trước, nên khi n lớn, độ phức tạp của phép đếm phân phối là O(n). Thuật toán sử dụng nDigit lần phép đếm phân phối nên có thể thấy **độ phức tạp của thuật toán là O(n.nDigit)** bất kể dữ liệu đầu vào.

Ta có thể coi sắp xếp cơ số trực tiếp là một mở rộng của phép đếm phân phối, khi dãy số chỉ toàn các số có 1 chữ số (trong hệ radix) thì đó chính là phép đếm phân phối. Sự khác biệt ở đây là: Sắp xếp cơ số trực tiếp có thể thực hiện với các khoá mang giá trị lớn; còn phép đếm phân phối chỉ có thể làm trong trường hợp các khoá mang giá trị nhỏ, bởi nó cần một lượng bộ nhớ đủ rộng để giăng ra dãy biến đếm số lần xuất hiện cho từng giá trị.