MO417 – Ata do Exercício 8-2

Ewerton Almeida Silva

30 de abril de 2010

Enunciado: Vamos supor que temos um arranjo de n registros de dados para ordenar e que a chave de cada registro tem valor 0 ou 1. Um algoritmo para ordenar tal conjunto de registros poderia ter algum subconjunto das três características desejáveis a seguir:

1. O algoritmo é executado no tempo $O(n)$.
2. O algoritmo é estável.
3. O algoritmo ordena localmente, sem utilizar mais do que uma quantidade constante de espaço de armazenamento além do arranjo original.

a. Dê um algoritmo que satisfaça os critérios 1 e 2 anteriores.

Resposta: O algoritmo Counting-Sort pode ordenar tal entrada em tempo linear (no pior caso), além de preservar a ordenação inicial dos campos com a mesma chave, ou seja, é estável.

Observação: Para este item, um algoritmo diferente foi proposto pelo colega Alexandre. A idéia consistia em se criar um arranjo auxiliar do mesmo tamanho que o de entrada. Em seguida, percorrer-se-ia o arranjo original do menor para o maior índice e, para cada chave com valor 0 encontrada, o registro correspondente seria copiado para o arranjo auxiliar. Depois, efetuar-se-ia a mesma tarefa para os registros com chaves de valor 1.

b. Dê um algoritmo que satisfaça os critérios 1 e 3 anteriores.

Resposta: O algoritmo Partition pode ordenar tal entrada em tempo linear (no pior caso), além de realizar esta ordenação localmente. Ao particionar o arranjo de entrada em dois (maiores e menores elementos que um pivô), o algoritmo terá efetuado a ordenação dos registros, já que há apenas dois valores possíveis de chave para os mesmos (0 ou 1).

c. Dê um algoritmo que satisfaça os critérios 2 e 3 anteriores.

Resposta: O algoritmo Insertion-Sort pode ordenar tal entrada localmente, além de ser um algoritmo estável, visto que preservará a ordem inicial de registros com chave de mesmo valor.

d. Algum dos seus algoritmos de ordenação das partes (a)-(c) pode ser usado para ordenar n registros com chaves de b bits usando radix sort em tempo $O(bn)$? Explique como ou por que não.

Resposta: O algoritmo Counting-Sort foi o único entre os algoritmos dos itens (a)-(c) a se encaixar nos requisitos exigidos pela questão, já que o Radix-Sort requer um algoritmo de ordenação estável (Partition já pode ser desconsiderado) e que seja executado no tempo *O(bn)* (Insertion-Sort não pode ser usado, já que sua complexidade é $O(n^2)$).

Observação: Uma prova de que o algoritmo Radix-Sort ordenará em tempo *O(bn)* um arranjo com 0's e 1's baseia-se na idéia de que:

(1) O Radix-Sort tem complexidade $\Theta(d(n+k))$, em que d é a quantidade de dígitos e cada dígito pode assumir k valores possíveis.

(2) Temos, portanto, que $d = b$ (chaves de b bits) e $k = 2$ (dois dígitos distintos possíveis ou bits: 0 e 1).

Substituindo esses valores em $\Theta(d(n+k))$, concluímos que:

$$\Theta(d(n+k)) = \Theta(b(n+2)) = \Theta(bn+2n) = \Theta(bn)$$

e. Suponha que os n registros tenham chaves no intervalo de 1 a k. Mostre como modificar a ordenação por contagem de tal forma que os registros possam ser ordenados localmente no tempo $O(n+k)$. Você pode usar o espaço de armazenamento $O(k)$ fora do arranjo de entrada. Seu algoritmo é estável? (*Sugestão*: Como você faria isso para $k = 3$?)

Resposta: O algoritmo proposto pelo colega Daniel Cason foi o escolhido como uma das soluções possíveis desta questão. A solução usa duas estruturas de armazenamento de tamanho $O(k)$: B e C.

(1) $B[1..k]$ armazena o número de vezes que cada valor de chave i ocorre na entrada.

(2) $C[0..k]$ guarda o número de elementos menores ou iguais a uma chave i, tal que $C[0] = 0$ e para $0 < i \leq n$, $C[i] = C[i-1] + B[i]$.

Ainda, o algoritmo não é estável.

Algoritmo 1 COUNTING-SORT-LOCAL(A, k)

```
for i ← 1 to k do
    B[i] ← 0
end for
for i ← 1 to comprimento[A] do
    B[A[i]] ← B[A[i]] + 1
end for
C[0] ← 0
for i ← 1 to k do
    C[i] ← C[i − 1] + B[i]
end for
for i ← n downto 1 do
    while i > C[A[i]] do
        B[A[i]] ← B[A[i]] − 1
        swap(i, C[A[i] − 1] + B[A[i]] + 1)
    end while
end for
```

Observação: O procedimento *swap(i, j)* realiza a troca de elementos (registros) entre duas posições do vetor A, como mostrado no código a seguir.

Algoritmo 2 *swap*(i, j)

```
aux ← A[i]
A[i] ← A[j]
A[j] ← aux
```