

Projeto e Análise de Algoritmos

Flávio L. C. de Moura

30 de maio de 2023

Nesta seção veremos uma outra forma de ordenação baseada na ideia de contagem. O que precisamos fazer é contar, para cada elemento a ser ordenado, o número total de elementos que são menores do que este elemento e guardar este resultado em uma tabela. Os valores da tabela indicarão a posição dos elementos na lista ordenada. Por exemplo, se existem 6 elementos menores do que o elemento x , então x deve ser colocado na sétima posição do vetor ordenado. Desta forma podemos ordenar um vetor simplesmente deslocando os elementos para a posição correta. O algoritmo 1 implementa esta ideia.

```
1 let  $C[0..n - 1]$  be a new array;
2 for  $i = 0$  to  $n - 1$  do
3   |  $C[i] \leftarrow 0$ ;
4 end
5 for  $i = 0$  to  $n - 2$  do
6   for  $j = i + 1$  to  $n - 1$  do
7     | if  $A[i] < A[j]$  then
8       | |  $C[j] \leftarrow C[j] + 1$ ;
9       | end
10    | else
11    | |  $C[i] \leftarrow C[i] + 1$ ;
12    | end
13  | end
14 end
15 for  $i = 0$  to  $n - 1$  do
16 |  $B[C[i]] \leftarrow A[i]$ ;
17 end
18 return  $B$ ;
```

Algoritmo 1: comparison-counting-sort($A[0..n - 1]$)

Esta ideia não parece ser muito interessante já que a implementação acima tem complexidade quadrática e ainda utiliza espaço adicional, ou seja a complexidade de espaço é linear no tamanho n da entrada. No entanto, podemos utilizar a ideia da contagem de forma mais eficiente se os elementos a serem ordenados forem "conhecidos". Por exemplo, se o vetor a ser ordenado contém apenas 0s e 1s então podemos utilizar esta informação para fazer a ordenação sem a necessidade de fazer comparações porque com uma única passagem sobre o vetor podemos obter o número k de 0s, e assim retornar o vetor contendo 0s da posição 0 até $k - 1$, e 1s da posição k em diante. De uma forma mais geral, se os elementos a serem ordenados são inteiros entre l e h então podemos computar a frequência de cada um destes valores, e armazená-las em um vetor, digamos $C[0..h - l]$, de forma que as primeiras $C[0]$ posições do vetor ordenado serão preenchidas com l , as $C[1]$ posições seguintes com $l + 1$, e assim por diante. Observe que se os elementos do vetor original não puderem ser sobrescritos então precisaremos de um novo vetor (espaço adicional) para escrever o vetor ordenado.

O algoritmo 2 (counting-sort) a seguir, assume que cada um dos n inteiros a serem ordenados estão no intervalo entre l e h .

```

1 let  $C[0..h-l]$  be a new array;
2 for  $i = 0$  to  $h-l$  do
3   |  $C[i] \leftarrow 0$ ;
4 end
5 for  $i = 0$  to  $n-1$  do
6   |  $C[A[i]-l] \leftarrow C[A[i]-l] + 1$ ;
7 end
8 for  $j = 1$  to  $h-l$  do
9   |  $C[j] \leftarrow C[j] + C[j-1]$ ;
10 end
11 for  $i = n-1$  downto  $0$  do
12   |  $j \leftarrow A[i]-l$ ;
13   |  $B[C[j]-1] \leftarrow A[i]$ ;
14   |  $C[j] \leftarrow C[j]-1$ ;
15 end
16 return  $B$ ;

```

Algoritmo 2: counting-sort($A[0..n-1], l, h$)

Qual é a complexidade deste algoritmo? Por simplicidade denotaremos $k = h - l$, ou seja, k denota o tamanho do intervalo que contém os elementos a serem ordenados. Assim, o laço **for** das linhas 2-4 é executado em tempo $\Theta(k)$, o laço das linhas 5-7 em tempo $\Theta(n)$, o laço das linhas 8-10 em tempo $\Theta(k)$, e por fim o laço das linhas 11-14 em tempo $\Theta(n)$, o que perfaz um total de $\Theta(n+k)$. Assumindo que $k = O(n)$, temos que o tempo de execução de counting-sort é $\Theta(n)$.

O algoritmo radix-sort ordena uma sequência de inteiros com d dígitos cada, em tempo linear. Ele utiliza um algoritmo auxiliar, que precisa ser estável, para ordenar a sequência de inteiros do dígito menos significativo para o mais significativo. Podemos utilizar counting-sort, por exemplo, como algoritmo auxiliar.

```

1 for  $i = 1$  to  $d$  do
2   | use a stable sort algorithm to sort array  $A$  on digit  $i$ ;
3 end

```

Algoritmo 3: radix-sort(A, d)

Teorema 1. *Dados n números com d dígitos, que por sua vez podem assumir até k valores, radix-sort ordena corretamente estes números em tempo $\Theta(d \cdot (n+k))$, se o algoritmo auxiliar estável tem complexidade de tempo $\Theta(n+k)$.*

Definição 2. *Um algoritmo de ordenação é dito estável se não altera a posição relativa dos elementos que têm o mesmo valor.*

Exercício 3. *Prove que o algoritmo counting-sort é estável.*

Exercício 4. *Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a $n^2 - 1$ em tempo linear, ou seja, em $O(n)$.*

Prova. Inicialmente iteramos pela lista dos números, convertendo cada um deles para a base n . Depois aplicamos o algoritmo radix-sort sobre a lista, utilizando o counting-sort como o algoritmo de ordenação dos dígitos da lista de números. Dada a nova base, cada número possuirá 2 dígitos, uma vez que $\log_n(n^2) = 2$, onde cada dígito estará em um intervalo entre 0 e $n - 1$. Sabendo disso, vemos que radix-sort ordenará n números de 2 dígitos, usando o counting-sort em um intervalo de 0 a $n - 1$, resultando em uma complexidade de $O(2(n+n)) = O(n)$. \square

Exercício 5. *Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a $n^3 - 1$ em tempo linear, ou seja, em tempo $O(n)$.*