Ordenação em tempo linear

CLRS cap 8

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações é uma "árvore de decisão".

Todo algoritmo de ordenação baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Devolve um vetor $B[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

```
CountingSort(A, n)
para i ← 1 até k faça
    C[i] ← 0
para j ← 1 até n faça
    C[A[j]] ← C[A[j]] + 1
        /* C[j] é o número de ítens j em A */
para i ← 2 até k faça
    C[i] ← C[i] + C[i − 1]
        /* C[j] é o número de ítens ≤ j em A */
para j ← n decrescendo até 1 faça
    B[C[A[j]]] ← A[j]
    C[A[j]] ← C[A[j]] − 1
devolva B
```

Consumo de tempo

linha	consumo na linha
1	$\Theta(k)$
2	$\mathrm{O}(k)$
3	$\Theta(n)$
4	$\mathrm{O}(n)$
5	$\Theta(k)$
6	$\mathrm{O}(k)$
7	$\Theta(n)$
8	$\mathrm{O}(n)$
9	$\mathrm{O}(n)$
10	$\Theta(1)$
total	????

linha	consumo na linha
1	$\Theta(k)$
2	$\mathrm{O}(k)$

Counting Sort

CountingSort(A, n)
1 **para** $i \leftarrow 1$ **até** k **faça**
2 $\quad C[i] \leftarrow 0$
3 **para** $j \leftarrow 1$ **até** n **faça**
4 $\quad C[A[j]] \leftarrow C[A[j]] + 1$
5 **para** $i \leftarrow 2$ **até** k **faça**
6 $\quad C[i] \leftarrow C[i] + C[i-1]$
7 **para** $j \leftarrow n$ **decrescendo até** 1 **faça**
8 $\quad B[C[A[j]]] \leftarrow A[j]$
9 $\quad C[A[j]] \leftarrow C[A[j]] - 1$
10 **devolva** B

Consumo de tempo: $\Theta(k+n)$

Se $k = \mathrm{O}(n)$, o consumo de tempo é $\Theta(n)$.

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados (Hollerith!)
- registros cuja chave tem vários campos

campo 1: menos significativo
campo d: mais significativo

RadixSort(A, n, d)
para $i \leftarrow 1$ **até** d **faça**
$\quad$ Ordene (A, n, i)

Ordene (A, n, i): ordena $A[1\,..\,n]$ pelo i-ésimo campo dos registros em A por meio de um algoritmo estável.

Estabilidade

Um algoritmo de ordenação é estável se sempre que, inicialmente, $A[i] = A[j]$ para $i < j$, a cópia $A[i]$ termina em uma posição menor do vetor que a cópia $A[j]$.

Isso só é relevante quando temos informação satélite.

Quais dos algoritmos que vimos são estáveis?

- inserção direta? seleção direta? bubblesort?
- mergesort?
- quicksort?
- heapsort?
- countingsort?

Consumo de tempo do Radixsort

Depende do algoritmo Ordene.

Se cada campo é um inteiro de 1 a k,
então podemos usar o CountingSort.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Se d é limitado por uma constante (ou seja, se $d = \mathrm{O}(1)$)
e $k = \mathrm{O}(n)$, então o consumo de tempo é $\Theta(n)$.

Bucketsort

CLRS sec 8.4

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde
cada elemento é um número no intervalo $[0,1)$.

A										
	.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

Devolve um vetor $C[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

C										
	.03	.12	.38	.42	.47	.62	.77	.82	.91	.93

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.47	.42
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.93	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.42	.47
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.91	.93

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.42	.47
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.91	.93

.03	.12	.38	.42	.47	.62	.77	.82	.91	.93

Bucket Sort

Recebe um inteiro n e um vetor $A[1..n]$ onde cada elemento é um número no intervalo $[0,1)$.

Devolve um vetor $C[1..n]$ com os elementos de $A[1..n]$ em ordem crescente.

BucketSort(A,n)
para $i \leftarrow 0$ **até** $n-1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\,A[i]\rfloor], A[i])$
para $i \leftarrow 0$ **até** $n-1$ **faça**
    OrdeneLista$(B[i])$
$C \leftarrow$ Concatene(B,n)
devolva C

Bucket Sort

BucketSort(A,n)
para $i \leftarrow 0$ **até** $n-1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\,A[i]\rfloor], A[i])$
para $i \leftarrow 0$ **até** $n-1$ **faça**
    OrdeneLista$(B[i])$
$C \leftarrow$ Concatene(B,n)
devolva C

INSIRA(p,x): insere x na lista apontada por p

OrdeneLista(p): ordena a lista apontada por p

Concatene(B,n): devolve a lista obtida da concatenação das listas apontadas por $B[0],\ldots,B[n-1]$.

Consumo de tempo

Suponha que os números em $A[1..n]$ são uniformemente distribuídos no intervalo $[0,1)$.

Suponha que o OrdeneLista seja o InsertionSort.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

Y_i: número de comparações para ordenar a lista $B[i]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \le {X_i}^2$.

Logo $\mathrm{E}[Y_i] \le \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2 + \sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

$$\mathrm{E}[(\sum_j X_{ij})^2] = \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}]$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \le {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \le \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Observe que ${X_{ij}}^2$ é uma variável aleatória binária. Vamos calcular sua esperança:

$$\mathrm{E}[{X_{ij}}^2] = \Pr[{X_{ij}}^2 = 1] = \Pr[X_{ij} = 1] = \frac{1}{n}.$$

Consumo de tempo

Para calcular $\mathrm{E}[X_{ij} X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij} X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Logo,

$$\begin{aligned} \mathrm{E}[Y_i] &\le \sum_j \frac{1}{n} + \sum_j \sum_{k \neq j} \frac{1}{n^2} \\ &= \frac{n}{n} + n(n-1)\frac{1}{n^2} \\ &= 1 + (n-1)\frac{1}{n} \\ &= 2 - \frac{1}{n}. \end{aligned}$$

Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que Y é o número de comparações realizadas pelo BUCKETSORT no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BUCKETSORT.

$$\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \le 2n - 1 = \mathrm{O}(n).$$

O consumo de tempo esperado do BUCKETSORT quando os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$ é $\mathrm{O}(n)$.