Counting-Sort

Ordinamento in tempo lineare NON basato su confronti

Ipotesi

La lunghezza n dell'array è dello stesso ordine di grandezza del range dei valori da ordinare k

$$\forall A[i] \quad (i = 1 \ldots n) \quad A[i] \in [1 \ldots k]$$

Se $k = O(n)$ il counting sort è $O(n)$

Esempio

1	1000	3	47	915	194	15231	12003	4	5
1	2	3	4	5	6	7	8	9	10

$n = 10$, $k = 15231$, quindi $k \notin O(n)$

Esempio

5	7	15	11	12	9	6	4	12	15
1	2	3	4	5	6	7	8	9	10

$n = 10$, $k = 15$, quindi $k \in O(n)$

Algoritmo...

Input A: array da ordinare;

Output B: array ordinato;

Idea Si posiziona $A[i]$ in $B[j]$ sulla base del numero di elementi di A più piccoli di $A[i]$ (se due elementi hanno lo stesso valore non si possono mettere nella stessa posizione).

Fissato un $A[i]$, per determinare quanti elementi minori di $A[i]$ sono presenti in A serve un array di appoggio C.
La dimensione di C dipende dal massimo valore degli elementi di A.
Inizialmente in C vengono contate le occorrenze di ciascun elemento di A:
for $j = 1$ to $length[A]$ do $C[A[j]] \longleftarrow C[A[j]] + 1$
Esempio

Array A

3	6	4	1	3	4	1	4
1	2	3	4	5	6	7	8

$n = 8$, $k = 6$, quindi $length[C] = 6$

Array C

2	0	2	3	0	1
1	2	3	4	5	6

Gli indici di C corrispondono ai valori di A
Per esempio: l'elemento 3 nell'array A occorre 2 volte, quindi $C[3] = 2$

... Algoritmo: come contare il numero di elementi di $A \leq i$

A pratire dal numero di occorrenze di ciascun elemento di A, si determina il numero di elementi di A che sono minori o uguali di i

Viene modificato C

Quanti elementi di A sono ≤ 1?

2	0	2	3	0	1
1	2	3	4	5	6

Quanti elementi di A sono ≤ 2?

2	2	2	3	0	1
1	2	3	4	5	6

Quanti elementi di A sono ≤ 3?

2	2	4	3	0	1
1	2	3	4	5	6

Quanti elementi di A sono ≤ 4?

2	2	4	7	0	1
1	2	3	4	5	6

Quanti elementi di A sono ≤ 5?

2	2	4	7	7	1
1	2	3	4	5	6

Quanti elementi di A sono ≤ 6?

2	2	4	7	7	8
1	2	3	4	5	6

for $i = 2$ to k do $C[i] \longleftarrow C[i] + C[i-1]$

Come si costruisce l'array ordinato

Array A

3	6	4	1	3	4	1	4
1	2	3	4	5	6	7	8

Array C

2	2	4	7	7	8
1	2	3	4	5	6

$\forall$ **j = n downto 1**

- si calcola quanti elementi vengono prima di $A[j]$ (se $A[1] = 3$ e $C[3] = C[A[1]] = 4$ ci sono 4 elementi ≤ 3):
  - si memorizza $A[j]$ in posizione $C[A[j]]$ nell'array B
    $B[C[A[j]]] \longleftarrow A[j]$
  - si decrementa $C[A[j]]$ ora c'è un elemento in meno
    $C[A[j]] \longleftarrow C[A[j]] - 1$

Array B ...inizio

						4	
1	2	3	4	5	6	7	8

Con il ciclo downto l'algoritmo è *stabile*

A parità di valore, nel nuovo array vengono rispettate le precedenze presenti nell'array originario

Stabilità - Esempio

Graduatoria per un esame scritto: a parità di voto vince chi consegna prima

Inizialmente i compiti sono ordinati secondo il tempo di consegna:

rossi 30 min	bianchi 40 min	verdi 45 min	neri 50 min	rosa 60 min	celeste 62 min	bruni 65 min	dorato 67 min
1	2	3	4	5	6	7	8

Array A

Poi vengono assegnati i voti (max. 1 , min. 6)

rossi 30 min **3**	bianchi 40 min **6**	verdi 45 min **4**	neri 50 min **1**	rosa 60 min **3**	celeste 62 min **4**	bruni 65 min **1**	dorato 67 min **4**
1	2	3	4	5	6	7	8

Array A

Infine viene ordinato l'array

neri 50 min **1**	bruni 65 min **1**	rossi 30 min **3**	rosa 60 min **3**	verdi 45 min **4**	celeste 62 min **4**	dorato 67 min **4**	bianchi 40 min **6**
1	2	3	4	5	6	7	8

Array B

;; inizializzo l'array di appoggio C
for $i = 1$ to k $O(k)$
. do $C[i] \longleftarrow 0$

$+$

;; conto le occorrenze di i in A : $C[i]$ è il numero di occorrenze di i in A
for $j = 1$ to $length[A]$ $O(n)$
. do $C[A[j]] \longleftarrow C[A[j]] + 1$

$+$

;; conto quanti elementi di $C[i]$ sono $\leq i$ e li memorizzo in $C[i]$
for $i = 2$ to k $O(k)$
. do $C[i] \longleftarrow C[i] + C[i-1]$

$+$

;; costruisco l'array B
for $j = length[A]$ downto 1
. do $B[C[A[j]]] \longleftarrow A[j]$ $O(n)$
. $C[A[j]] \longleftarrow C[A[j]] - 1$

$=$
$O(n)$