Modulo: Algoritmi di ordinamento di tipo “Divide et Impera” [P2_12]

Unità didattica: Approfondimento sulla scelta del partizionatore nel Quick Sort [2-AT]

Titolo: Calcolo del valore mediano

Argomenti trattati:

- ✓ Algoritmo “Counting Sort” O(N) per il calcolo del valore mediano

Prerequisiti richiesti: array, ordinamento, Quick Sort, complessità computazionale asintotica, ricorsione

La *Complessità di Tempo* dell'algoritmo Quicksort dipende dal fatto che il partizionamento, ad ogni passo, sia più o meno bilanciato e ciò a sua volta dipende da quali elementi partizionatori siano stati scelti.

Come scegliere l'elemento partizionatore?

In modo random
(distribuzione uniforme)

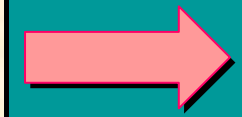
oppure

Sfruttando la distribuzione di probabilità dei dati se è nota

migliore scelta per il partizionatore: valore mediano dei dati

Come calcolare il valore mediano dei dati?

1. Si ordinano i dati
2. Si calcola l'elemento centrale



$T(N) = O(N \log_2 N)$

```
A = 'CBDEICBBBFGEGHI'; % sequenza di caratteri
N=numel(A);   A=sort(A);   disp(A)
```

MATLAB

B B B B C C D **E** E F G G H I I *A ordinato*

```
disp( [ A(round(N/2))   median(A)])
```

E **E**

Per ordinare, si può ridurre la complessità a O(N)!

Esempio: Counting Sort

Il Counting Sort è uno degli algoritmi di ordinamento non basati sui confronti e di complessità lineare.
Richiede che le chiavi possano essere associate a numeri.

Altri algoritmi O(n):

- Bucket Sort
- Radix Sort
- ...

Counting Sort: idea

L'algoritmo conta il numero di occorrenze di ciascun valore presente nell'array *A* da ordinare, memorizzando questa informazione in un array temporaneo *C* di dimensione pari al range dei valori in *A* (range di interi).
Il numero di ripetizioni dei valori precedenti indica la posizione del valore immediatamente successivo.

1. Si calcolano i valori min(*A*) e max(*A*). Il range è
   {min(*A*), ..., max(*A*)}
2. Si prepara un array ausiliario *C* (di dimensione pari al range dei valori) dove *C*[i] rappresenta la frequenza nell'array di partenza *A* dell'elemento di valore
   - i+min(*A*) (se i=0,1,... come in C/C++,Java)
   - i+min(*A*)+1 (se i=1,... come in MATLAB, FORTRAN)
3. Si scorre l'array *A* incrementando di 1 la componente di *C* corrispondente.
4. Per ottenere il vettore *A* ordinato, si scorre l'array *C* e si scrivono su *A*, *C*[i] copie del valore i+min(*A*).

Counting sort: algoritmo di ordinamento O(N)

idea

[Min='B', Max='I']
D=8

MATLAB

```
A='CBDEICBBBFGEGHI'; % array di caratteri
N=numel(A);      % 15: numero elementi di A
Max=max(A); Min=min(A);
B = A - Min+1    % fa partire da 1 gli indici
B  =  2 1 3 4 8 2 1 1 1 5 6 4 6 7 8
D = Max - Min+1; % range dei valori in A
C = zeros(1,D)
C = 0 0 0 0 0 0 0 0 0
```

O(N)

≈O(D)

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

B=A-min(A)+1

$B_i \in \{1,\ldots,D\}$

C
0
0
0
0
0
0
0
0

D

complessità computazionale

vantaggio: D ≈ N ⇒ O(N)

svantaggio: D >> N ⇒ O(D) >> O(N)

Counting sort: algoritmo di ordinamento O(N)

idea

m=2, M=9
D=8

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento   O(N)
end
```

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

c
0
0
0
0
0
0
0
0

c
0
1
0
0
0
0
0
0

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento
end
```

O(N)

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
0
1
0
0
0
0
0
0

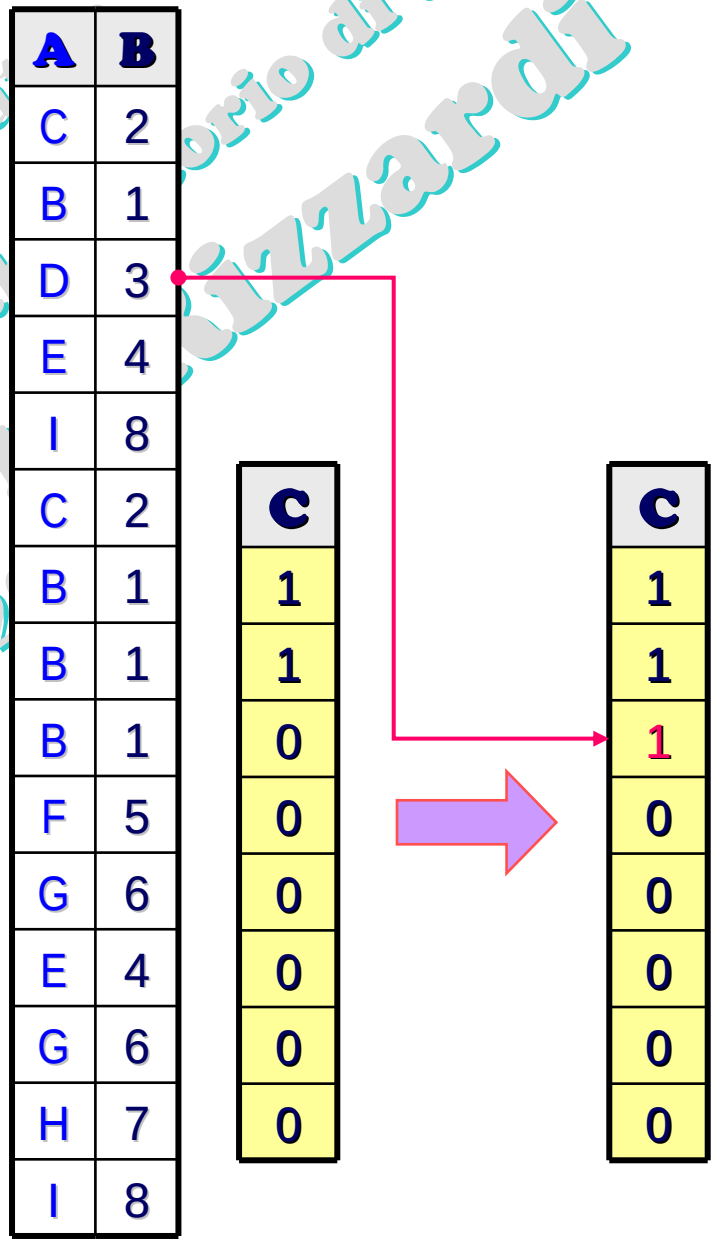
C
1
1
0
0
0
0
0
0

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento
end
```

O(N)



Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento   O(N)
end
```

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
1
1
1
0
0
0
0
0

C
1
1
1
1
0
0
0
0

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento
end
```

O(N)

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

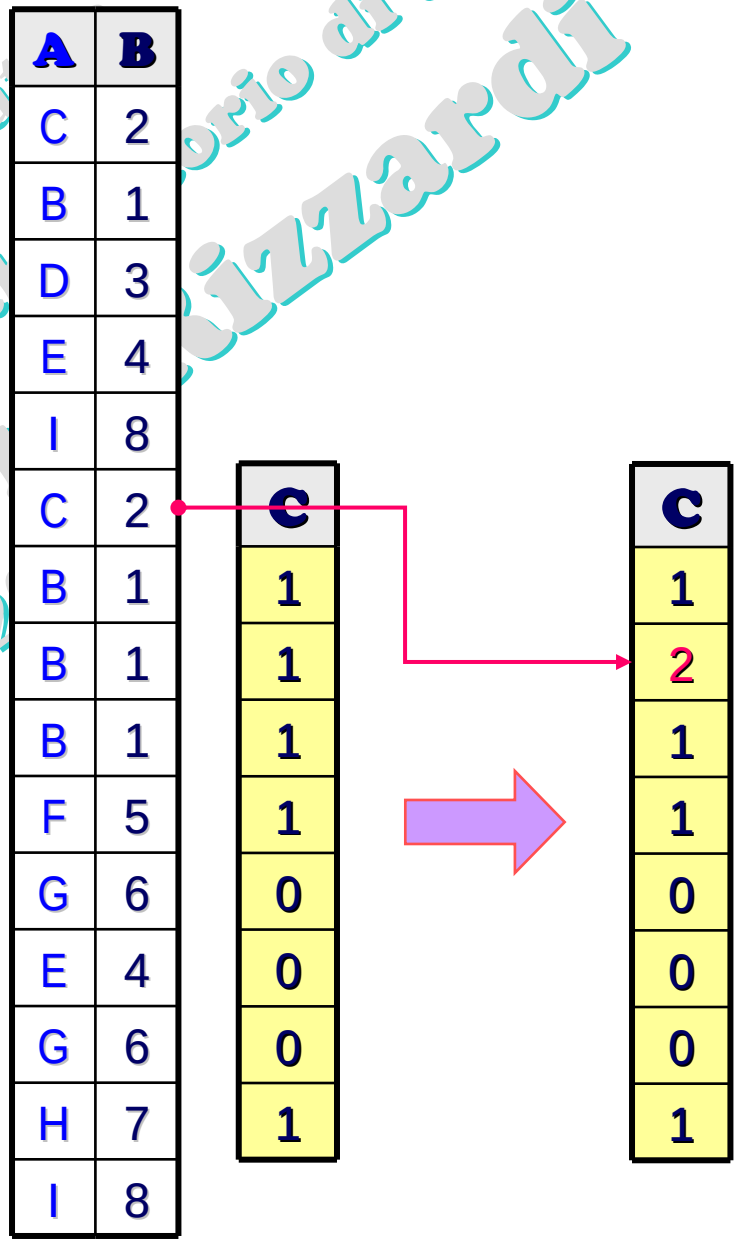
C
1
1
1
1
0
0
0
0

C
1
1
1
1
0
0
0
1

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento   O(N)
end
```



Counting sort: algoritmo di ordinamento O(N)

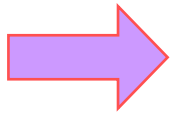
idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento
end
```

O(N)

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
1
2
1
1
0
0
0
1



C
2
2
1
1
0
0
0
1

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento   O(N)
end
```

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
2
2
1
1
0
0
0
1

C
3
2
1
1
0
0
0
1

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento   O(N)
end
```

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
3
2
1
1
0
0
0
1

C
4
2
1
1
0
0
0
1

Counting sort: algoritmo di ordinamento O(N)

idea

```
C = zeros(1,D);
for k=1:N
    C(B(k)) = C(B(k))+1; % incremento
end
```

O(N)

A	B
C	2
B	1
D	3
E	4
I	8
C	2
B	1
B	1
B	1
F	5
G	6
E	4
G	6
H	7
I	8

C
4
2
1
1
0
0
0
1

C
4
2
1
1
1
0
0
1

...

Counting sort: algoritmo di ordinamento O(N)

idea

cumsum(C)

somma cumulativa delle componenti di C

≈O(N)

```
SC = cumsum(C); % somma cumulativa
```

cerca la prima componente in SC il cui valore è > N/2

```
J = find(SC > floor(N/2),1)
```

```
mediano = J+Min-1;
char(mediano)
```

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

	SC
4	4
4+2	6
4+2+1	7
4+...+2	9
4+...+1	10
...	12
...	13
somma	15

Counting sort: algoritmo di ordinamento O(N)

idea

B	2	1	3	4	8	2	1	1	1	5	6	4	6	7	8
A	C	B	D	E	I	C	B	B	B	F	G	E	G	H	I
	B	B	B	B	C	C	D	E	E	F	G	G	H	I	I

A_k	B_k	J	C_k
B	2	1	4
C	3	2	6
D	4	3	7
E	5	4	9
F	6	5	10
G	7	6	12
H	8	7	13
I	9	8	15

char(mediano) = E ⟸ mediano = 4+Min-1 ⟸ char(Min)=B, Min = 66, J=4

```
SC=cumsum(C); % somma cumulativa
J=find(SC > floor(N/2),1)
mediano = J+Min-1;
char(mediano)
```

in **B** numero elementi ≤ 4
in **A** ≤ E

trova l'indice J della 1ª componente di SC > N/2

N/2 = 7.5