Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Il problema dell'ordinamento: algoritmi lineari

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

- Abbiamo dimostrato il teorema che asserisce che ogni algoritmo di ordinamento *che opera per confronti* ha un costo computazionale di $\Theta(n \log n)$.

- Com'è possibile, allora, avere degli algoritmi di ordinamento di costo computazionale lineare?

- Basta rimuovere l'ipotesi che l'algoritmo sia basato unicamente sui confronti.

Counting Sort

- Ipotesi: ciascuno degli n elementi da ordinare è un intero di valore compreso in un intervallo $[0,\ldots,k]$

- Idea: fare in modo che il valore di ogni elemento della sequenza determini direttamente la sua posizione nella sequenza ordinata.

- Il costo computazionale è di $\Theta(n+k)$.

Se $k = O(n)$ allora l'algoritmo ordina *n* elementi in tempo lineare, cioè $\Theta(n)$.

Counting Sort - Esempio

A:	8	6	7	2	5	6	1	8	4	4	1	6

n=12, k=8

C:	0	0	0	0	0	0	0	0	0

1) Scorrendo A conteggiamo in C il numero di occorrenze di ciascun valore

	0	1	2	3	4	5	6	7	8
C:	0	2	1	0	2	1	3	1	2

Nota: $\sum_{i=0}^{k} C[i] = n$

2) Scorrendo C ricopiamo in A ciascun indice di C tante volte quanto è il valore in C di quell'indice

A:	1	1	2	4	4	5	6	6	6	7	8	8

Counting Sort

```
def  Counting_sort (A):
    k=max(A)                                                    Θ(n)
    n=len(A)                                                    Θ(1)
    #Crea una lista C per registrare le occorrenze degli
    #elementi presenti in lista
    C=[0 for _in range(k+1)]                                    Θ(k)
    for i in range(n):                                          Θ(n)
          C[A[i]]+=1                                              Θ(1)
    #C[i] ora contiene il numero di occorrenze di i in A
    t = 0                                                       Θ(1)
    for i in range(k+1):                                        iterato k+1 volte
        for _ in range(C[i]):                                   iterato C[i]+1 volte
            A[t] = i                                              Θ(1)
            t+=1                                                  Θ(1)
```

$$
\begin{aligned}
T(n) &= \Theta(n) + \Theta(k) + \sum_{i=0}^{k} (C[i]+1)\cdot\Theta(1) \\
&= \Theta(n) + \Theta(k) + \Theta(1)\left(\sum_{i=0}^{k} C[i] + \sum_{i=0}^{k} 1\right) \\
&= \Theta(n) + \Theta(k) + \Theta(1)(n+k+1) \\
&= \Theta(n) + \Theta(k)
\end{aligned}
$$

ricorda che $\sum_{i=0}^{k} C[i] = n$

Bucket sort

- **Ipotesi:** gli n elementi da ordinare sono distribuiti in modo uniforme nell'intervallo $[1,\ldots,k]$, senza alcuna ipotesi su k.

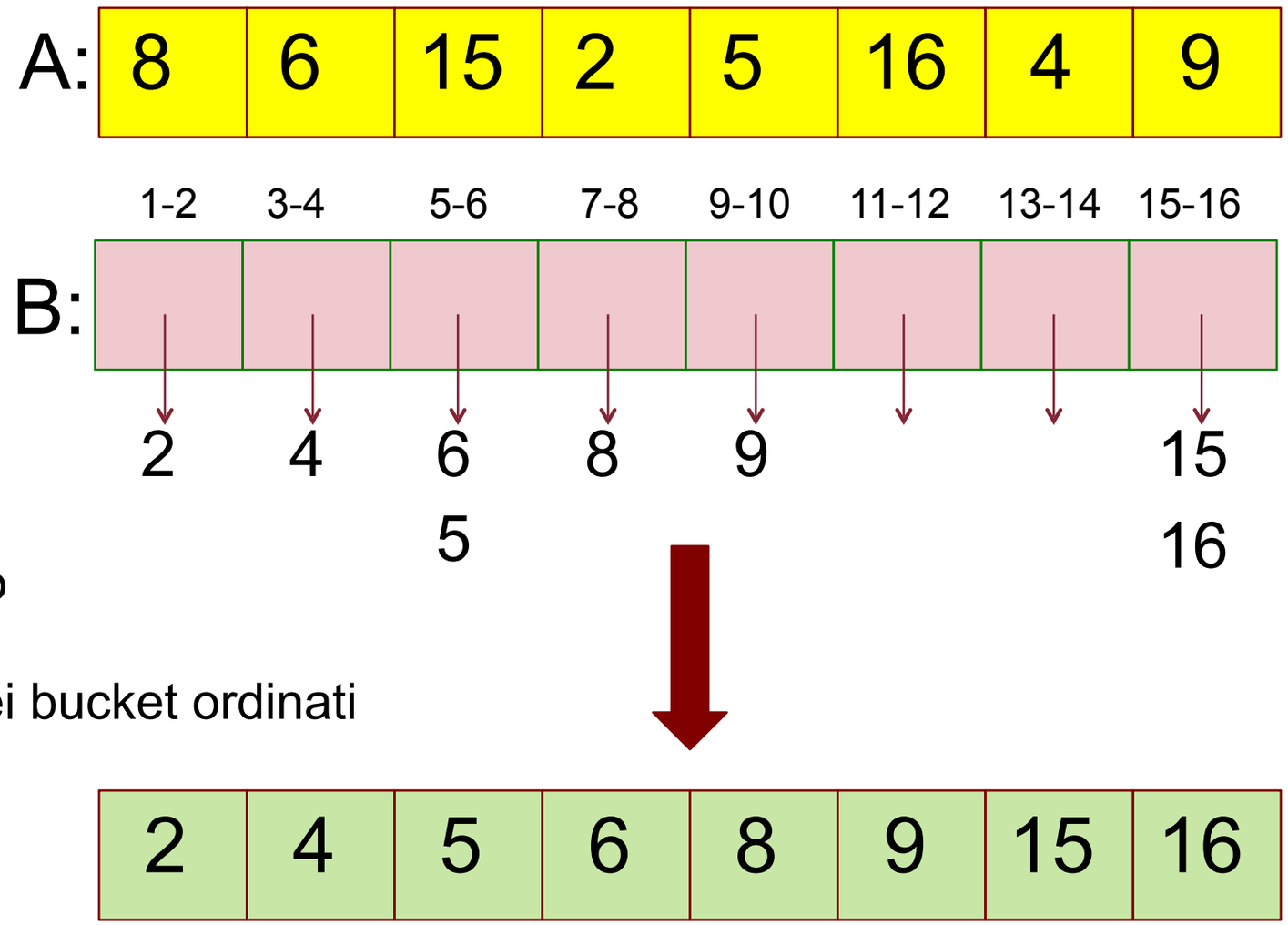
- **Idea**: dividere l'intervallo $[1,\ldots,k]$ in n sottointervalli di uguali dimensioni $\frac{k}{n}$, detti ***bucket***, e distribuire i valori nei bucket (il generico elemento x finisce nel bucket $\left\lfloor x \cdot \frac{n}{k} \right\rfloor$).

- Poiché gli elementi in input sono uniformemente distribuiti, non ci si aspetta che molti elementi cadano nello stesso bucket.

- Il costo computazionale **medio** è di $\Theta(n)$.

Bucket sort - Esempio

n=8, k=16



1) Inserimento degli elementi nei bucket
2) Ordinamento di ogni bucket
3) Copiatura dei bucket ordinati

```python
def bucketSort(A):

    n=len(A)
    #creo i bucket
    B=[[] for i in range(n)]

    #inserisco gli elementi di lista nei bucket
    k=max(A)+1

    for x in A:

        B[int(n*x/k)].append(x)

    #ordino gli elementi dei vari bucket
    for i in range(n):

        B[i].sort()

    #copio gli elementi dei vari bucket in lista
    j=0

    for i in range(n):

        for x in B[i]:

            A[j]=x

            j+=1
```

$\Theta(1)$

$\Theta(n)$

$\Theta(n)$

n **iterazioni**

$\Theta(1)$

n **iterazioni**

costo ordinamento di $B[i]$ $\Big\}$ $\sum_{i=1}^{n} \Theta(\ costo\ per\ ordinare\ |B[i]|\ elementi)$

n **iterazioni**

$|B[i]|$ **iterazioni** $\Big\}$ $\sum_{i=1}^{n} |B[i]|\Theta(1) = \Theta(1)\sum_{i=1}^{n} |B[i]| = \Theta(1)n = \Theta(n)$

$\Theta(1)$

$$T(n) = \Theta(n) + \sum_{i=1}^{n} (\textbf{costo per ordinare } |B[i]| \textbf{ elementi})$$

•Ci sono n bucket ed n valori; per l'ipotesi di equiprobabilità, in media ci sarà un numero costante di valori in ogni bucket, quindi la lista $B[i]$ ha in media lunghezza costante.

Vale quindi: $T_{medio}(n) = \Theta(n) + \sum_{i=1}^{n} \Theta(1) = \Theta(n) + \Theta(n) = \Theta(n)$

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZI

- Mostrare che il Counting sort è un algoritmo di ordinamento stabile.Cosa accade al tempo di esecuzione del Bucket sort quando tutti gli elementi della lista A sono uguali?

- Qual è il tempo di esecuzione del Bucket sort nel caso peggiore se come algoritmo di ordinamento usiamo l'Insertion sort?

- Qual è il tempo di esecuzione del Bucket sort nel caso peggiore se come algoritmo di ordinamento usiamo il Merge sort?

- Quale semplice modifica dell'algoritmo consente di conservare tempo medio lineare e costo $\Theta(n \log n)$ nel caso peggiore?

- Il Bucket sort può essere modificato in modo che l'ordinamento all'interno delle liste sia eseguito tramite Counting sort. Affinché il costo dell'algoritmo sia lineare anche nel caso peggiore, quale ipotesi bisogna fare su k?