Corso di laurea in Informatica
Introduzione agli Algoritmi

Il problema dell'ordinamento: algoritmi lineari

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

- 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 nell'intervallo $[0, \cdots, k]$

- 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

- trova k, l'elemento massimo dell'array A da ordinare.
- inizializza l'array C con i k contatori delle occorrenze in A.
- scorri A e per ogni indice i incrementa il contatore $C[A[i]]$ delle occorrenze di $A[i]$.
- scorri C e per ogni indice i inserisci $C[i]$ occorrenze dell'elemento i in A.

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)
    n=len(A)
    # crea una lista C di contatori per registrare il numero
    # di occorrenze degli elementi in A
    C=[0]*(k+1)
    # calcola le occorrenze degli elementi in A nel range[0,k]
    for i in range(n):
        C[A[i]]+= 1
    # reinserisce in A i suoi elementi in modo ordinato
    j = 0
    for i in range(len(C)):
        for _ in range(C[i]):
            A[j] = i
            j+=1
```

$\Theta(n)$

$\Theta(k)$

$\Theta(n)$

iterato $k+1$ volte

iterato $C[i]$ volte

$\Theta(n+k)$

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

$$T(n) = \Theta(n+k)$$

Bucket sort (con k buckets)

- **Ipotesi:** gli n elementi da ordinare appartengono all'intervallo $[0,\ldots,M]$ non sono necessariamente interi

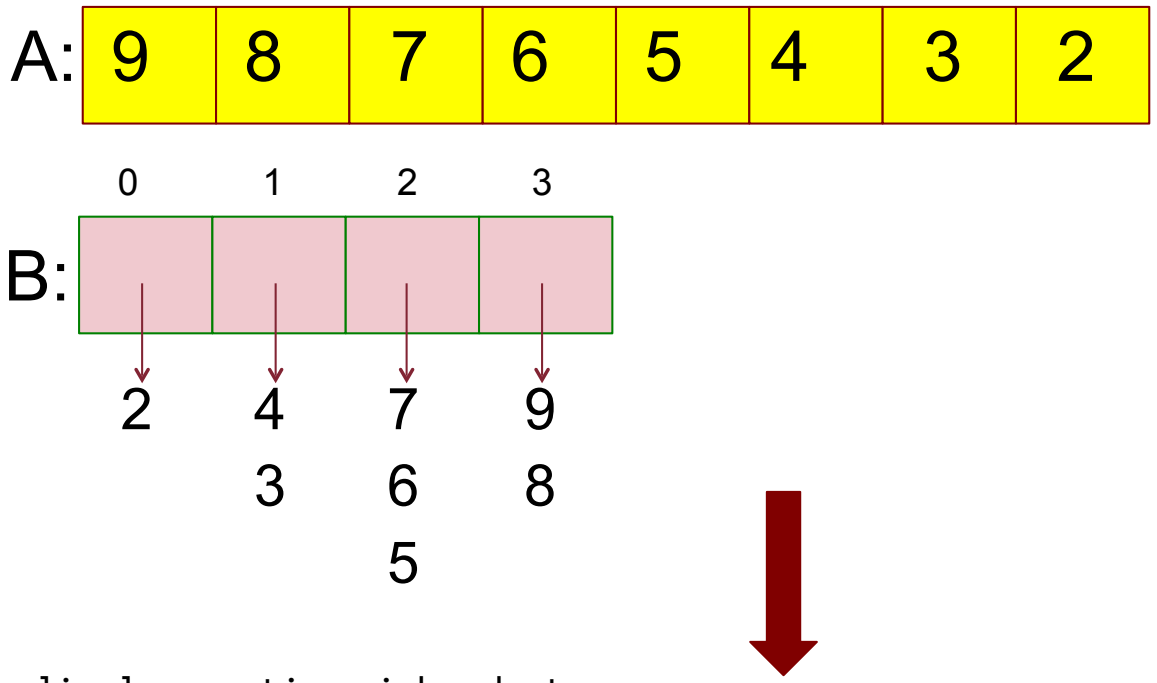
- **Idea**: dividere l'intervallo $[0,\ldots,M]$ in k *sottointervalli* detti ***bucket***, e distribuire i valori nei loro bucket (il generico elemento x finisce nel bucket $\left\lfloor k \cdot \frac{x}{M+1} \right\rfloor$) per poi ordinarli separatamente.

- il costo computazionale dipenderà dall'algoritmo utilizzato per ordinare i vari buckets ma se gli elementi in input sono uniformemente distribuiti non ci si aspetta che molti elementi cadano nello stesso bucket (i.e. in ogni bucket ci saranno circa $\frac{n}{k}$ elementi).
- Se prendiamo $k = \frac{n}{c}$, dove c è una opportuna costante, allora in ogni bucket ci saranno $\frac{n}{n/c}$ = c elementi) e quindi il costo dell'ordinamento all'interno del Bucket sarà $\Theta(1)$.

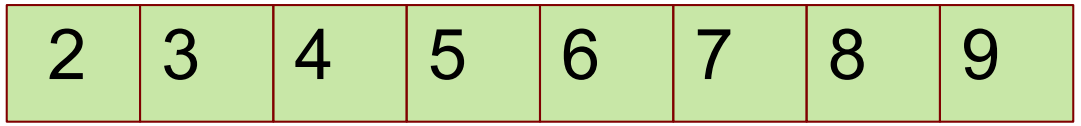
Bucket_sort(A, k)

- crea una lista di k buckets inizialmente vuoti.
- trova M, l'elemento massimo dell'array A da ordinare.
- scorri A e per ogni valore x inserisci x nel bucket $B\left[\left\lfloor k\frac{x}{M+1}\right\rfloor\right]$.
- Ordina gli elementi di ciascun bucket.
- Concatena gli elementi ordinati dei vari bucket.

Esempio BucketSort con n=8 e k=4



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



Bucket sort

```
def  Bucket_sort( A, k):
    #Crea una lista di k buckets inizialmente vuoti
    n = len(A)
    B = [ [] for _ in range(k) ]
    #distribuisci gli  elementi di  A nei buckets
    m = max(A)
    for x in A :
        i = n * x//(M+1)
        B[i].append(x)
    #ordina gli elementi in ciascuno dei k buckets
    for i in range( k ):
        B[i].sort()
    #concatena  gli elementi dei k buckets ordinati
    C = [ ]
    for i in range( k):
        C.extend(B[i])
    return R
```

$\Theta(k)$

$\Theta(n)$

$\left.\begin{array}{l}\\\\\\\\\\\\\end{array}\right\} \sum_{i=0}^{k-1} \Theta(\ costo\ per\ ordinare\ |B[i]|\ elementi)$

$\left.\begin{array}{l}\\\\\\\\\\\\\end{array}\right\} \sum_{i=0}^{k-1} \Theta(|B[i]|) = \Theta(k+n)$

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} \Theta(\ \textbf{costo per ordinare}\ |B[i]|\ \textbf{elementi}\ )$$

$$T(n) = \Theta(n) + \sum_{i=0}^{k-1} \Theta(\text{ costo per ordinare } |B[i]| \text{ elementi })$$

Assumiamo che il numero di Bucket scelto sia una frazione di n vale a dire $k = \left\lceil \frac{n}{c} \right\rceil$

Caso pessimo: tutti gli elementi finiscono nello stesso bucket si ha

$T(n) = \Theta(n) + \Theta(\text{ costo per ordinare } n \text{ elementi })$

e la complessità dipenderà quindi dall'algoritmo di sort utilizzato.

Caso ottimo: gli elementi sono equidistribuiti nell'intervallo $[0,\ldots,m]$ allora in ogni bucket finiranno circa $\frac{n}{k} = \Theta(1)$ elementi da cui segue $T(n) = \Theta(n) + n \cdot \Theta(\text{ costo per ordinare } \Theta(1) \text{ elementi })$ indipendentemente dall'algoritmo di ordinamento usato. Si ha in questo caso: $T(n) = \Theta(n) + \Theta(n) \cdot \Theta(1) = \Theta(n)$

ESERCIZIO risolto (esame dell' appello Marzo 2023).

Data una lista A di n interi non negativi distinti, si vuole determinare se esistono almeno tre numeri tra loro consecutivi di valore inferiore a 100.

Ad esempio se
$A = [101, 5, 9, 31, 33, 10, 100, 4, 8, 32, 500, 11, 99]$,
gli elementi $8, 9$ e 10 così come gli elementi $31, 32$ e 33 rispettano la proprietà mentre $99, 100$ e 101 no.

Progettare un algoritmo che, data la lista A, in tempo $\Theta(n)$ restituisce il valore dell'elemento centrale della terna se questa è presente, -1 altrimenti. Se esistono più terne allora bisogna restituire l'elemento centrale di valore massimo (nell'esempio sopra l'algoritmo dovrebbe restituire 32).

l'algoritmo che consiste nello scorrere l'array per ciascun elemento alla ricerca dei due elementi che possano far parte della terna ha tempo quadratico, $\Theta(n^2)$ mentre ordinare l'array e poi scorrerlo una sola volta alla ricerca di una eventuale terna richiederebbe tempo $\Theta(n \log n)$.

Una prima soluzione corretta consiste nel selezionare i soli elementi di valore al più 100 (questo costerà $\Theta(n)$) ordinarli (questo richiederà tempo costante poiché sono tutti distinti non saranno più di 100) e scorrerli partendo da destra alla ricerca eventuale della prima terna di consecutivi.

Altro possibile algoritmo richiede di creare una lista C di 100 contatori dove in $C[i]$ inserisco il numero di occorrenze di i in A (questa prima parte richiede tempo $\Theta(n)$). Poi scorro i contatori a partire da destra alla ricerca eventuale di 3 locazioni consecutive a valore superiore a 0 (questa seconda parte richiede tempo $\Theta(1)$).

Ecco di seguito possibili codici python dei due algoritmi proposti:

```
def terna1(A):
    B = []
    for x in A:
        if x < 100:
            B.append(x)
    B.sort()
    i = len(B)-2
    while i > 0:
        if B[i]==B[i+1]-1 and B[i]==B[i-1]+1:
            return B[i]
        i -= 1
    return -1
```

```
def terna2(A):
    C = [0]*100
    for i in range (len(A)):
        if A[i] < 100:
            C[ A[i] ] += 1
    i=98
    while i > 0:
        if C[i+1] and C[i] and C[i-1]:
            return i
        i -= 1
    return -1
```

Corso di laurea in Informatica
Introduzione agli Algoritmi

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?

- Generalizzare il Counting sort in modo che funzioni anche quando sono presenti interi negativi.

- 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 di Bucket sort 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?