POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Gli Algoritmi di Ordinamento iterativi lineari

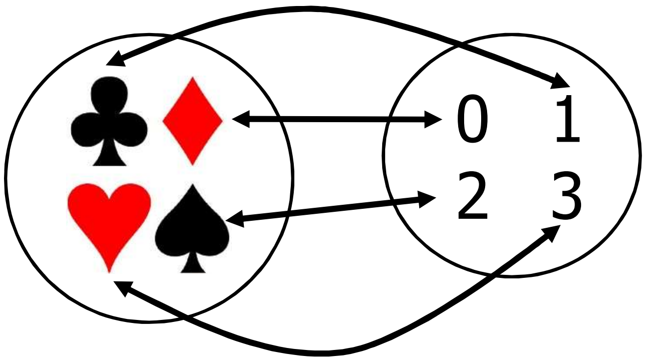
Paolo Camurati

Caratteristiche generali

- La posizione di un elemento nell'ordinamento non si determina tramite confronti, bensì calcolandola
- Non vale più il limite inferiore linearitmico alla complessità di caso peggiore
- La complessità è lineare T(N) = O(N)
- Ci sono limitazioni all'uso
- Algoritmi:
  - Counting sort
  - Radix sort
  - Bin/bucket sort: richiede le liste, trattate nel Corso del II anno

Counting sort

- scopo: ordinare un vettore di N interi appartenenti all'intervallo 0 …k-1
- ogni insieme finito di k elementi può essere messo in corrispondenza biunivoca con gli interi nell'intervallo 0 … k-1



- l'input potrebbe contenere ripetizioni oppure
- l'input potrebbe non contenere alcuni dei dati nell'intervallo 0 … k-1

Approccio

- Si procede per calcolo e non per confronto
- per ciascun elemento da ordinare x si calcolano quanti elementi lo precedono nell’ordinamento
  - prima si calcolano le occorrenze semplici di x, cioè quante istanze di x compaiono nell’input
  - a partire dalle occorrenze semplici si calcolano le occorrenze multiple, cioè quanti elementi sono ≤ x
- scorrendo il vettore da ordinare da destra a sinistra, si dispone l’elemento corrente nella posizione finale corretta

Strutture dati

Si usano 3 vettori:

- Vettore di input: A[0..N-1] di N interi
- Vettore risultato: B [0..N-1] di N interi
- Vettore delle occorrenze semplici/multiple C di k interi se i dati sono nell'intervallo [0..k-1]

Esempio: N=8 k=6

A

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

Vettore da ordinare

C

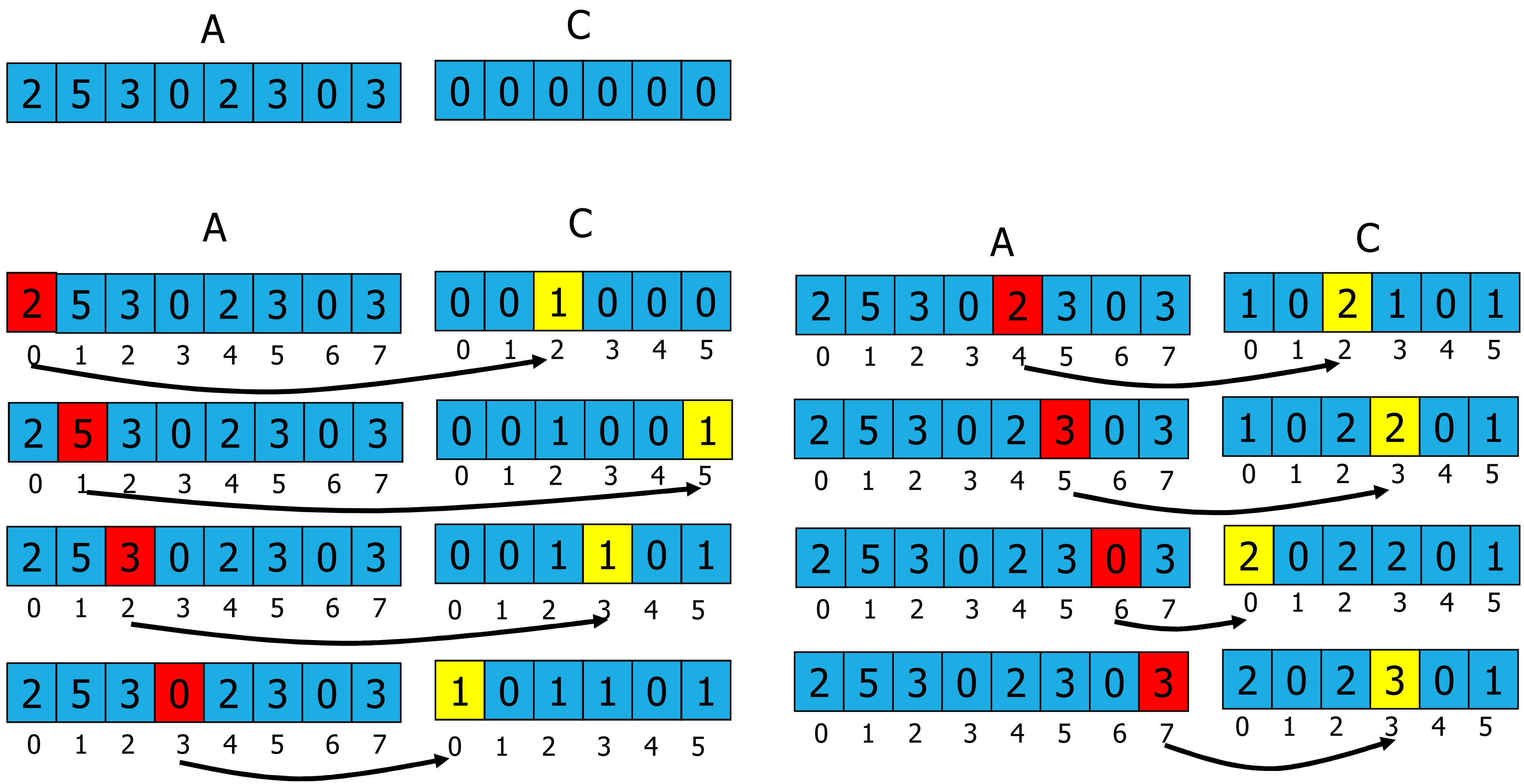
0	1	2	3	4	5

Vettore delle occorrenze semplici/multiple

Calcolo delle occorrenze semplici

- Inizializzazione a tutti 0 del vettore C
- Scansione del vettore di input
  - A[i] è un’occorrenza di quel valore, compreso nell’intervallo 0… k-1
  - A[i] viene usato come indice in C per incrementare di 1 il valore di quella cella

```
for (i = l; i <= r; i++)
  C[A[i]]++;
```


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

Calcolo delle occorrenze multiple

- Scansione del vettore C delle occorrenze semplici
  - in C[0] sono memorizzate le occorrenze di 0 e di tutti i dati che lo precedono (nessuno per definizione!)
  - le occorrenze di dati che precedono i ($1 \leq i < k$) sono memorizzate in C[i-1]
  - le occorrenze di dati che precedono o sono uguali a i si calcolano come C[i] = C[i-1] + C[i]

```
for (i = 1; i < k; i++)
  C[i] += C[i-1];
```

C

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

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

⇒

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

ci sono 2 occorrenze di valori ≤1

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

⇒

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

ci sono 4 occorrenze di valori ≤2

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

⇒

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

ci sono 7 occorrenze di valori ≤3

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

⇒

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

ci sono 7 occorrenze di valori ≤4

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

⇒

2	2	4	7	7	8
0	1	2	3	4	5

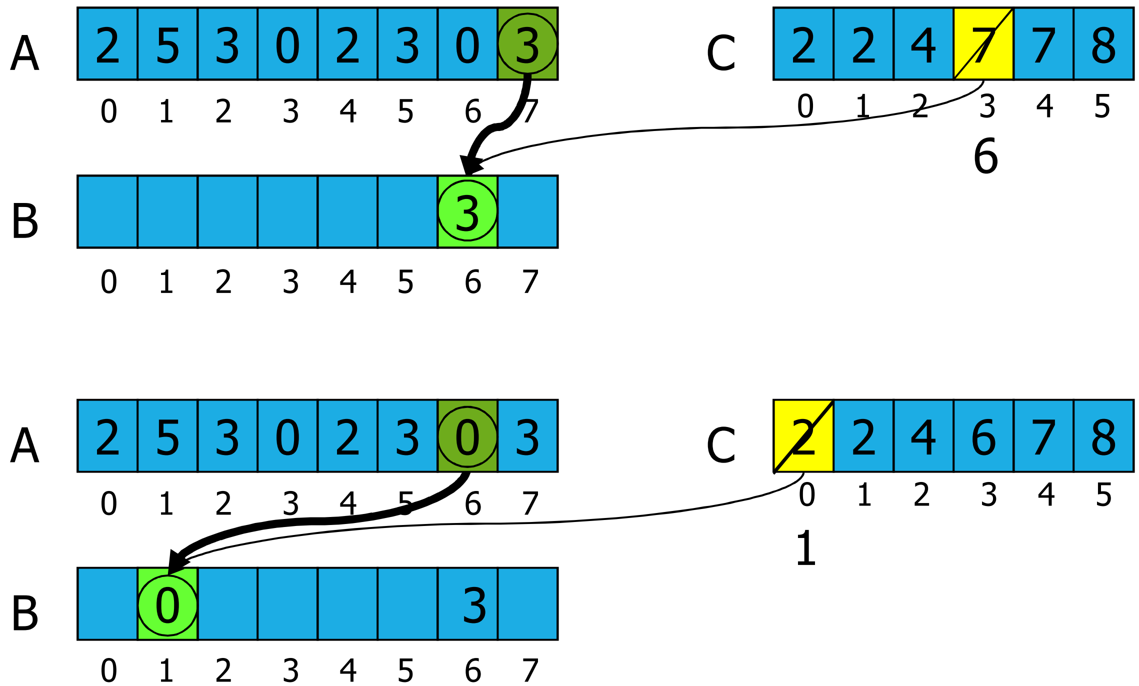
ci sono 8 occorrenze di valori ≤5

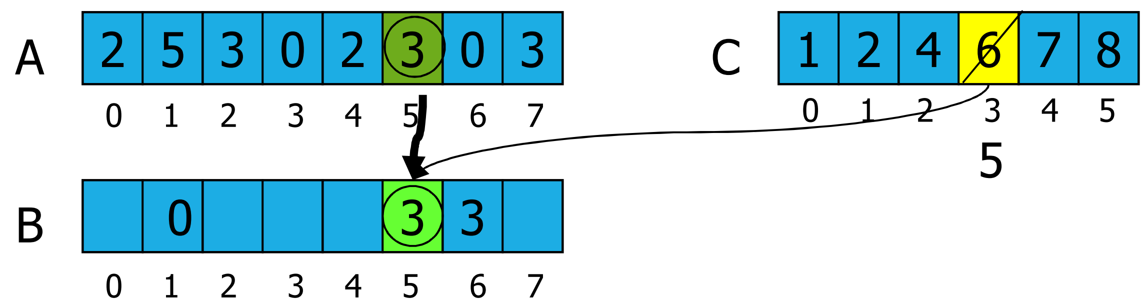
Calcolo delle posizioni finali

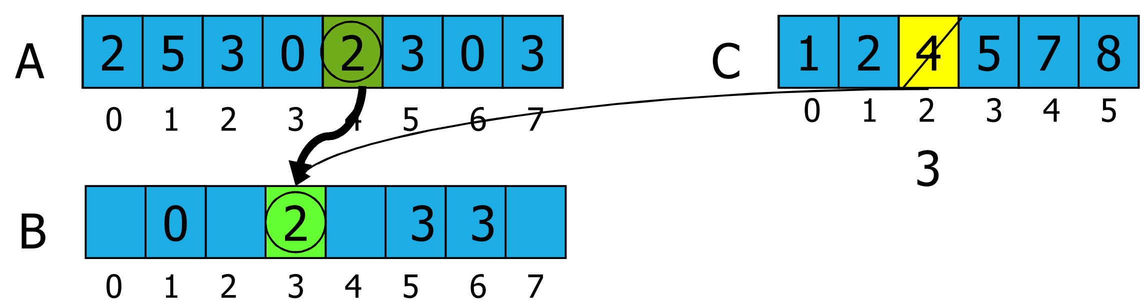
- Scansione del vettore A di input da destra a sinistra
  - in C[A[i]] sono memorizzate le occorrenze multiple di A[i] e di tutti i dati che lo precedono
  - la posizione finale nel vettore B di A[i] è all’indice C[A[i]]-1. Il -1 tiene conto del fatto che nel linguaggio C i vettori iniziano dall’indice 0
  - una volta memorizzato Ai] nella posizione finale si deve aggiornare il vettore delle occorrenze multiple C all’indice A[i] decrementandolo di 1

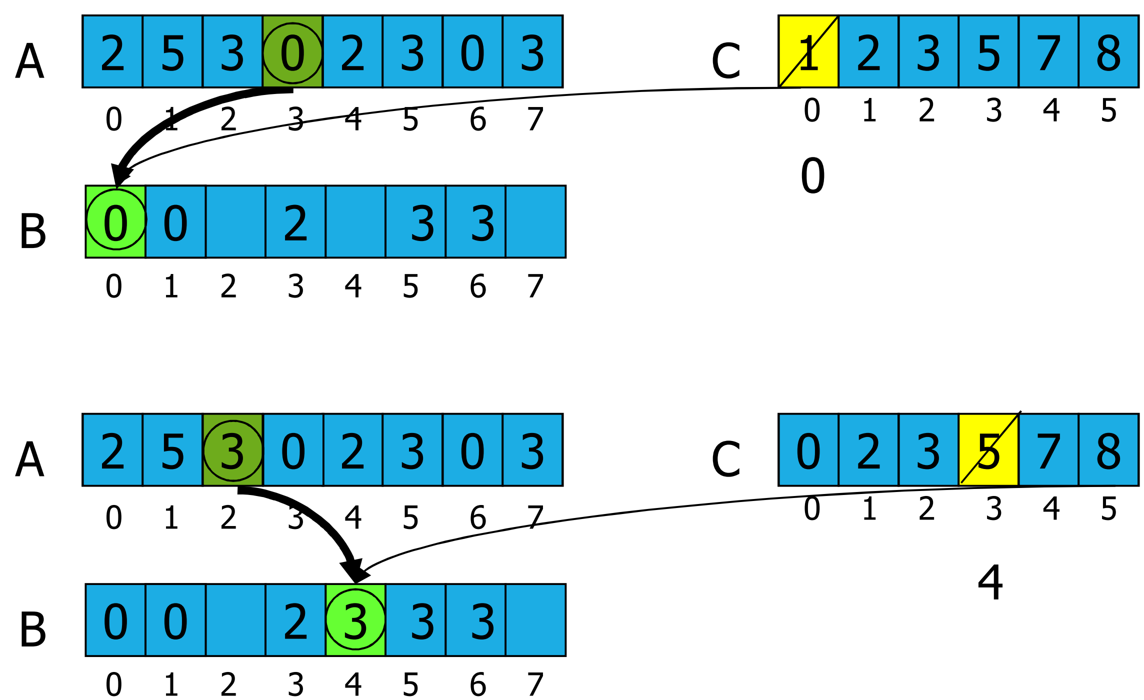
```
for (i = r; i >= l; i--) {
  B[C[A[i]]-1] = A[i];
  C[A[i]]--;
}
```

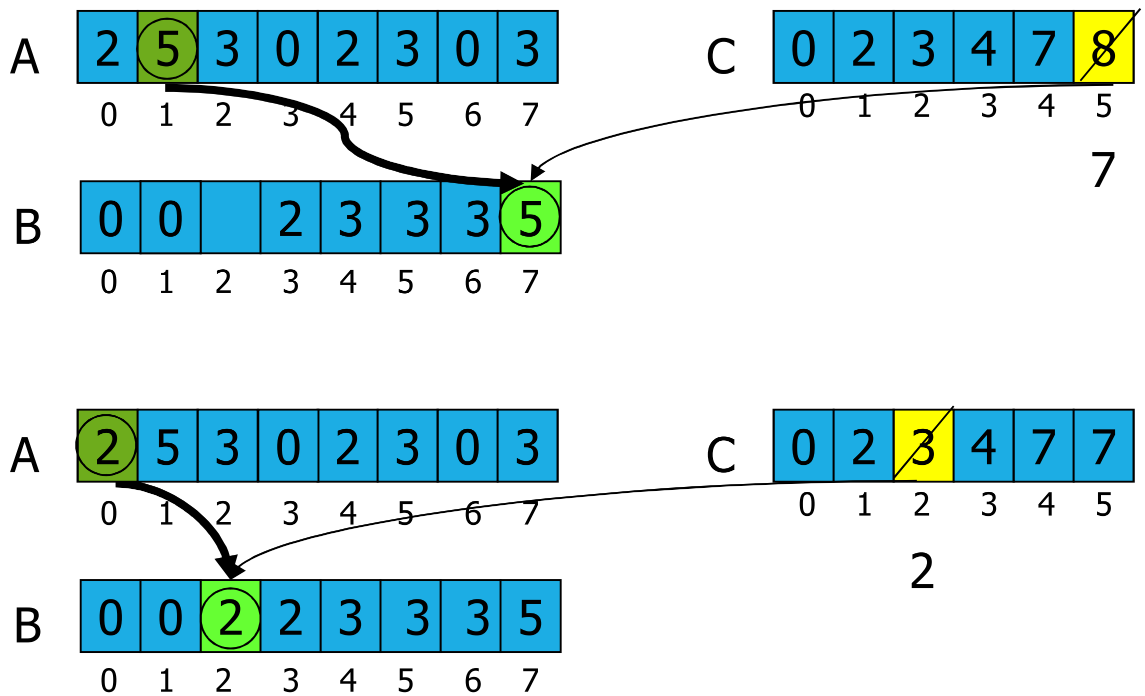
Esempio


A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
2 2 4 7 7 8
0 1 2 3 4 5
6
B
3
0 1 2 3 4 5 6 7
A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
2 2 4 6 7 8
0 1 2 3 4 5
1
B
0 3
0 1 2 3 4 5 6 7


A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
1 2 4 6 7 8
0 1 2 3 4 5
5
B
0 3 3
0 1 2 3 4 5 6 7


A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
1 2 4 5 7 8
0 1 2 3 4 5
3
B
0 2 3 3
0 1 2 3 4 5 6 7


A 2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C 1 2 3 5 7 8
0 1 2 3 4 5
0
B 0 0 2 3 3
0 1 2 3 4 5 6 7
A 2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C 0 2 3 5 7 8
0 1 2 3 4 5
4
B 0 0 2 3 3 3
0 1 2 3 4 5 6 7


A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
0 2 3 4 7 8
0 1 2 3 4 5
7
B
0 0 2 3 3 3 5
0 1 2 3 4 5 6 7
A
2 5 3 0 2 3 0 3
0 1 2 3 4 5 6 7
C
0 2 3 4 7 7
0 1 2 3 4 5
2
B
0 0 2 2 3 3 3 5
0 1 2 3 4 5 6 7

```
void CountingSort(int A[],int B[],int C[],int N,int k){
  int i, l=0, r=N-1;
  for (i = 0; i < k; i++)
    C[i] = 0;
  for (i = l; i <= r; i++)
    C[A[i]]++;
  for (i = 1; i < k; i++)
    C[i] += C[i-1];
  for (i = r; i >= l; i--) {
    B[C[A[i]]-1] = A[i];
    C[A[i]]--;
  }
  for (i = l; i <= r; i++)
    A[i] = B[i];
}
```

vettori allocati nel `main` e passati come parametri

inizializzazione di C

occorrenze semplici

occorrenze multiple

posizionamento corretto elementi

ricopiatura risultato

Caratteristiche del Counting sort

- non in loco: oltre al vettore A si usano anche i vettori B e C
- stabile: la stabilità è garantita dalla scansione da destra a sinistra del vettore A quando si posizionano gli elementi: in caso di chiavi duplicate, la prima a essere posizionata è l'ultima e finisce il più a destra possibile. Le altre chiavi duplicate non potranno mai «scavalcarla» visto che si decrementa la cella corrispondente del vettore delle occorrenze multiple
- se la scansione fosse stata da sinistra verso destra non si sarebbe garantita la stabilità dell'algoritmo. Il risultato sarebbe comunque ordinato.

Analisi di complessità del Counting sort

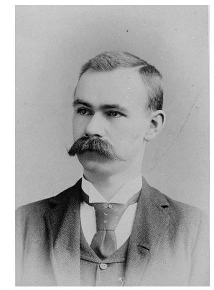
- Ciclo di inizializzazione di C: $\Theta(k)$
- Ciclo di calcolo delle occorrenze semplici: $\Theta(N)$
- Ciclo di calcolo delle occorrenze multiple: $\Theta(k)$
- Ciclo di posizionamento in B: $\Theta(N)$
- Ciclo di ricopiatura di B in A: $\Theta(N)$

$$T(N) = \Theta(N+k).$$

Se $k = \Theta(N)$, $T(N) = \Theta(N)$.

Applicabilità: k ed N devono essere “ragionevolmente” della stessa dimensione. Se $k=10^6$, $N=3$ e A= 999999, 1, 1000, non ha senso allocare un vettore di dimensione $k=10^6$ per ordinare 3 dati!

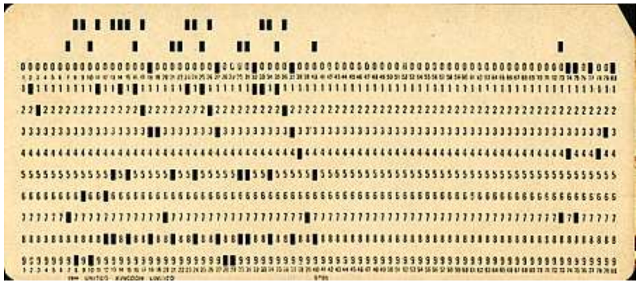
Radix sort



- 1890: primo censimento «moderno» negli Stati Uniti: grandi quantità di dati complessi
- Herman Hollerith introduce:
  - la scheda perforata per registrare informazioni in forma binaria
  - la «tabulating machine» per ordinare meccanicamente i dati

La scheda perforata (punched card)

- foglio di carta rigida organizzato per righe e colonne
- perforazioni per indicare in una certa riga/colonna la presenza/assenza di un’informazione
- caratteristiche:
  - informazioni in forma binaria
  - informazioni formate da più campi



La «tabulating machine»

Dispositivo elettromeccanico in grado di

- «leggere» le schede perforate
- contare le informazioni in base alla presenza/assenza di una perforazione in una colonna

La Tabulating Machine Company di Hollerith nel 1924 diventa la International Business Machines (IBM).

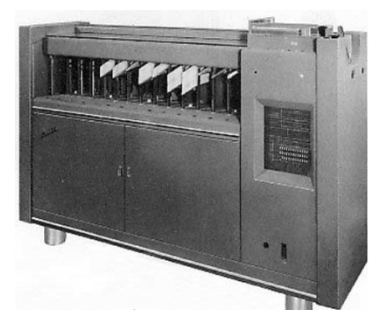
Ordinamento di schede perforate (anni '60)

- Partendo dalla colonna più a destra, una macchina distribuiva le schede in contenitori diversi a seconda dell'informazione registrata nella colonna
- Le schede contenute nei contenitori vengono prelevate mantenendo l'ordine in cui si trovano
- La distribuzione per contenitori prosegue sulle colonna successiva
- Si termina quando si raggiunge ed elabora la colonna più a sinistra.

Card puncher

Punched cards

Card reader



Card sorter

Radix sort

- Finora i dati da ordinare sono sempre stati «monolitici» (ad esempio 1234, VCDF, etc.)
- Nel Radix sort i dati da ordinare sono composti da d campi, i cui valori appartengono a un insieme di cardinalità n

Esempio:
numeri decimali di 3 cifre
d=3, n=10, valori=0,1,2…9
329
457
657
839
436
720
355

Esempio:
Targhe: 3 campi: 2 lettere 3 cifre 2 lettere
d=3,
lettere n=22, valori=A,…,Z escluse I, O, Q e U
cifre n=10, valori=0,1,2…9
FA 457 AA
GC 657 SD
AB 839 MN
ZZ 000 AA

Ordinamento «intuitivo» per campi

- Si ordina secondo la colonna più a sinistra, poi secondo la colonna immediatamente a destra, fino ad ordinare secondo la colonna più a destra
- Se si tratta di numeri sembra intuitivo, in quanto essi sono rappresentati posizionalmente

329	329	329	720
457	355	720	355
657	436	436	436
839	457	355	457
436	657	457	657
720	720	657	329
355	859	859	859

il risultato non è ordinato! Per un risultato corretto si deve utilizzare la ricorsione, argomento del Corso del II anno.

Ordinamento «controintuitivo» per campi

- Si ordina secondo la colonna più a destra, poi secondo la colonna immediatamente a sinistra, fino ad ordinare secondo la colonna più a sinistra
- Se si tratta di numeri sembra controintuitivo, in quanto non tiene conto del fatto che essi sono rappresentati posizionalmente

329	720	720	329
457	355	329	355
657	436	436	436
839	457	355	457
436	657	457	657
720	329	657	720
355	859	859	859

il risultato è ordinato!

- Vincolo sull'algoritmo di ordinamento usato per ogni colonna: deve essere STABILE!
- Il Counting sort è un'ottima scelta:
- è stabile
- è applicabile: la dimensione k del vettore C è fissa e dipende dalla base del sistema di numerazione (radix, di qui il nome Radix sort) delle cifre che compaiono in ciascuna colonna. Stiamo ordinando:
  - numeri in base 10: k = 10
  - stringhe di lettere A…Z: C k = 26
  - stringhe di caratteri ASCII: k = 128

Ordinamento di interi (in base 10)

- Sono dati n interi con un numero non necessariamente fisso di cifre memorizzati in un vettore A
- Si determina il numero massimo di cifre d, è come se i dati con meno di d cifre fossero riempiti con 0 da sinistra (padding)

170		0170
45		0045
2375		2375
90	è come se fosse	0090
802		0802
24		0024
2		0002
66		0066

- Si applicano d passi di Counting sort a partire dalla colonna più a destra, quella con peso 10^0 fino a quella a sinistra con peso 10^{d-1}

```
void radixSort(int A[], int B[], int C[], int D[], int n) {
  int largest, d=1, i;
  largest = getMax(A, n);

  while (largest/10 > 0){
    d++;
    largest /= 10;
  }
  for (i = 0; i < d; i++)
    CountingSort(A, B, C, D, n , i);
}
```

identificazione del massimo di A

calcolo del numero di cifre d

d iterazioni di Counting sort

Esempio

Passo 0
colonna delle unità

A dopo il Counting sort sulla colonne delle unità

A	D	
170	0	170
45	5	90
2375	5	802
90	0	2
802	2	24
24	4	45
2	2	2375
66	6	66

Passo 1
colonna delle decine

A dopo il
Counting sort
sulla colonne
delle decine

A	D	A dopo il Counting sort
170	7	802
90	9	2
802	0	24
2	0	45
24	2	66
45	4	170
2375	7	2375
66	6	90

Passo 2 colonna delle centinaia

A dopo il Counting sort sulla colonne delle centinaia

A	D	A dopo il Counting sort
802	8	2
2	0	24
24	0	45
45	0	66
66	0	90
170	1	170
2375	3	2375
90	0	802

A	D	A dopo il Counting sort sulla colonne delle migliaia
2	0	2
24	0	24
45	0	45
66	0	66
90	0	90
170	0	170
2375	2	802
802	0	2375

Passo 2
colonna delle
migliaia

A dopo il
Counting sort
sulla colonne
delle migliaia

Identificazione delle cifre

Rappresentazione posizionale dei numeri in base b:

- cifre tra 0 e b-1
  - in base 2 b=2, cifre 0, 1
  - in base 10 b=10, cifre 0,1,2,3,4,5,6,7,8,9
  - in base 8 b=8, cifre 0,1,2,3,4,5,6,7
  - In base 16 b=16, cifre 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

$x_{(\text{in base b, su d cifre})} = a_{d-1}b^{d-1} + a_{d-2}b^{d-2} + a_{d-3}b^{d-3} + \ldots\ldots\ a_1b^1 + a_0b^0$

Esempio

$12345_{(\text{in base 10, su 5 cifre})} = 1 \cdot 10^4 + 2 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$

- unità: $(x / b^0) \% b$
- decine: $(x / b^1) \% b$
- centinaia: $(x / b^2) \% b$
- ....

()%b: resto della divisione intera di () per b

Esempio

$x = 12345_{\text{(in base 10)}}$

- unità: $(x / b^0) \% b$ $(12345/1) \% 10 = 5$
- decine: $(x / b^1) \% b$ $(12345/10) \% 10 = 1234 \% 10 = 4$
- centinaia: $(x / b^2) \% b$ $(12345/100) \% 10 = 123 \% 10 = 3$
- migliaia: $(x / b^3) \% b$ $(12345/1000) \% 10 = 12 \% 10 = 2$
- decine di migliaia: $(x / b^4) \% b$ $(12345/10000) \% 10 = 1 \% 10 = 1$

Il vettore ausiliario D di n interi memorizza ad ogni passo la colonna corrispondente e viene usato nel Counting sort per ordinare A

```
...
int i, ..., weight=1;
for (i=0; i < step; i++)
  weight *= 10;

for (i = l; i <= r; i++)
  D[i] =(A[i]/weight)%10;
...
```

```c
void CountingSort(int A[],int B[],int C[],int D[], int N, int step){
  int i, l=0, r=N-1, weight=1;

  for (i=0; i < step; i++) weight *= 10;

  for (i = 0; i < 10; i++) C[i] = 0;

  for (i = l; i <= r; i++) D[i] =(A[i]/weight)%10;

  for (i = l; i <= r; i++) C[D[i]]++;

  for (i = 1; i < 10; i++) C[i] += C[i-1];

  for (i = r; i >= l; i--) {
    B[C[D[i]]-1] = A[i];
    C[D[i]]--;
  }
  for (i = l; i <= r; i++) A[i] = B[i];
}
```

calcolo di 10^{step}

identificazione colonna

occorrenze semplici

occorrenze multiple

ordinamento

ricopiatura

Caratteristiche del Radix sort

- non in loco: oltre al vettore A si usano anche i vettori B, C e D. Si potrebbe evitare D ricalcolando il valore quando necessario
- stabile: la stabilità è garantita dall'uso di un algoritmo stabile come il Counting sort per ciascuno dei passi.

Analisi di complessità del Radix sort

- La complessità del Counting sort è $T(N) = \Theta(N+k)$, dove i valori da ordinare sono interi nell'intervallo (0… k-1)
- si eseguono d passi di Counting sort
- La complessità è $T(N) = \Theta(d(N+k))$.
- Nel caso di numeri in base 10, k è fisso e vale 10, quindi

$$T(N) = \Theta(dN)$$

- Se anche il numero di cifre d è fisso

$$T(N) = \Theta(N).$$