

**Università di Pisa**

**Pietro Ducange**

**Algoritmi e strutture dati**  
**Ricerca in un insieme: Metodo Hash**

**a.a. 2020/2021**

**Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione  
la maggior parte delle slide utilizzate nella presente lezione**

## Ricerca Lineare in Array

```
int linearSearch (InfoType *A , int n, InfoType x) {  
    for (int i=0;i<n;i++) if (A[i]==x) return 1;  
    return 0;  
}
```

Limite Inferiore?

Complessità ?

## Ricerca Binaria

Algoritmo applicabile se gli elementi sono mantenuti ordinati

```
int binSearch (InfoType *A ,InfoType x, int i=0, int j=n-1) {  
    if (i > j) return 0;  
    int k=(i+j)/2;  
    if (x == A[k]) return 1;  
    if (x < A[k]) return binSearch(A, x, i, k-1);  
    else return binSearch(A, x, k+1, j)  
}
```

Limite Inferiore?

Complessità ?

## **Metodo di ricerca hash**

- **metodo di ricerca in array**
- **non basato su confronti**
- **molto efficiente**

## Metodo hash

$h$  (funzione hash) : InfoType  $\rightarrow$  indici

$x \rightarrow h(x)$

$$n \leq k$$

$n$  = numero massimo di elementi  
 $k$  = dimensione dell'array

0

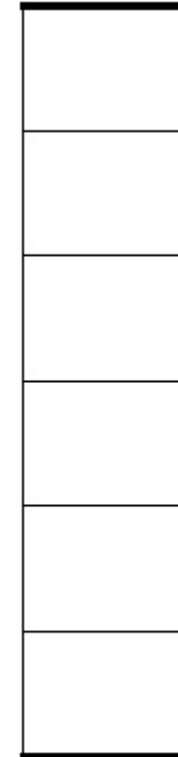
1

■

■

■

$k-1$



## Metodo hash: accesso diretto

**h iniettiva:**

**$h(x)$  : indirizzo hash dell'elemento che contiene x**

```
bool hashSearch (InfoType *A , int n, InfoType x) {  
    int i=h(x); if (A[i]==x) return true;  
    else return false;  
}
```

**Problema:** memoria (k può essere molto grande)

Complessità?

## insieme di al massimo 5 cifre : accesso diretto

Insieme = {0, 1, 2, 7, 9} (non noto apriori)

**n = 5**

**k = 10**

**Inserimenti: { 0, 2, 7 }**

**$h(x) = x$**

**$h(0) = 0$**

**$h(2) = 2$**

**$h(7) = 7$**

0	1
1	0
2	1
3	0
4	0
5	0
6	0
7	1
8	0
9	0

**NB: non è necessario memorizzare l'elemento**



## Domanda

Quale dovrebbe essere il valore di  $k$  per memorizzare, ad accesso diretto le possibili parole della lingua italiana di lunghezza minore o uguale a 10 caratteri?

## Metodo hash senza accesso diretto

**Si rilascia l'iniettività e si permette che due elementi diversi abbiano lo stesso indirizzo hash:**

$$h(x1) = h(x2) \text{ collisione}$$

**Bisogna gestire le seguenti situazioni:**

- **Come si cerca un elemento se si trova il suo posto occupato da un altro**
- **come si inseriscono gli elementi**

## Metodo hash ad indirizzamento aperto

**Una prima soluzione:**

**funzione hash modulare:**  $h(x) = (x \% k)$  dove  $k$  è la dimensione dell'array

(siamo sicuri che vengono generati tutti e soli gli indici dell'array)

**Legge di scansione lineare:** se non si trova l'elemento al suo posto, lo si cerca nelle posizioni successive fino a trovarlo o ad incontrare una posizione vuota

- L'inserimento è fatto con lo stesso criterio
- Una volta inseriti nella struttura dati, non possano più essere cancellati.

## Esempio: insieme di al massimo 5 cifre

Insieme = {0, 1, 2, 7, 9} (non noto apriori)

$n = k = 5$

$n/k = 1$

$$h(x) = x \% k$$

$$h(0) = 0$$

$$h(2) = 2$$

$$h(7) = 2$$

{ 0, 2, 7 }

0

0

1

-1

2

2

3

7

4

-1

## esempio

Esempio:  $h(x) = x \% k$

$K=99$

$h(x)$

$h(99) = h(198) = h(297) = 0$

$h(199) = h(100) = 1$

99, 198, 199, 297, 100

0	99
1	198
2	199
3	297
	100
98	

## Conseguenze

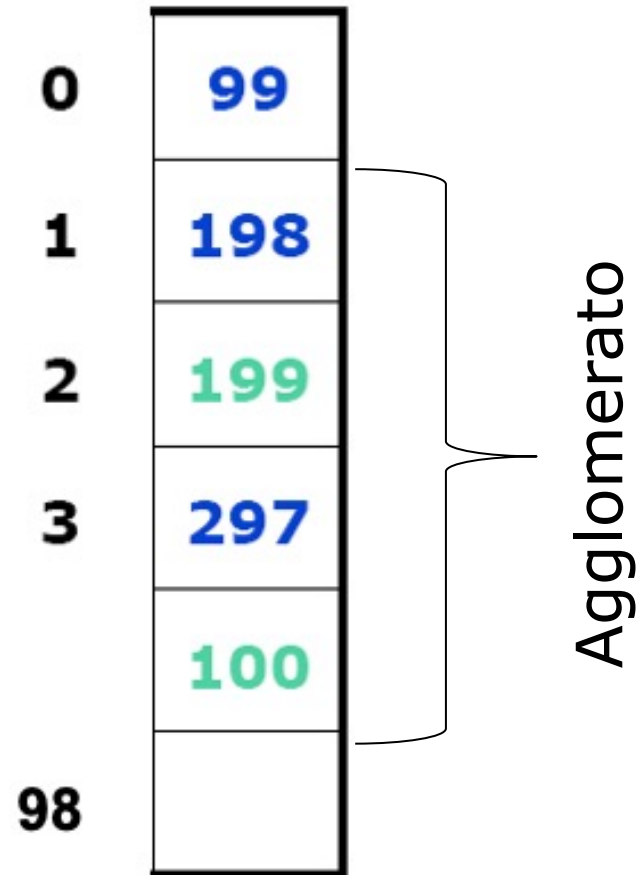
**Agglomerato:** gruppo di elementi  
adiacenti con indirizzi hash diversi

La presenza di collisioni ed  
agglomerati aumenta il tempo di  
ricerca

$$h(99)=h(198)=h(297)=0$$

$$H(199)=h(100)=1$$

99, 198, 199, 297, 100



## Metodo hash: ricerca con scansione lineare

```
bool hashSearch (int *A , int k, int x) {  
    int i=h(x);  
    for (int j=0; j<k; j++) {  
        int pos = (i+j)%k;      // somma in modulo  
        if (A[pos ]== -1) return false ;  
        if (A[pos ] == x) return true ;  
    }  
    return false ;  
}
```

**-1: posizione vuota**

## Metodo hash : inserimento

```
int hashInsert (int *A , int k, int x) {  
    int i=h(x); int b=0;  
    for (int j=0; !b & j<k; j++) {  
        int pos = (i+j)%k;  
        if (A[pos] == -1) {  
            A[pos] = x; b=1;  
        }  
    }  
    return b;  
}
```



## Metodo hash: inserimento in presenza di cancellazioni

Consideriamo un array di dimensione 10 che dovrà contenere numeri naturali, con funzione hash  $h(x) = x \% 10$ .

Supponiamo che nell'array siano stati inseriti, nell'ordine, i numeri 94, 52, e 174. La situazione sarà quella di Figura (a):

	0	1	2	3	4	5	6	7	8	9
(a)	-1	-1	52	-1	94	174	-1	-1	-1	-1
	0	1	2	3	4	5	6	7	8	9
(b)	-1	-1	52	-1	-2	174	-1	-1	-1	-1

Se effettuassimo la cancellazione dell'elemento 94, allora dovremmo passare alla situazione della Figura (b). Perché?

## Metodo hash: inserimento in presenza di cancellazioni

```
int hashInsert (int *A , int k, int x) {  
    int i=h(x); int b=0;  
    for (int j=0; !b && j<k; j++) {  
        int pos = (i+j)%k;  
        if ((A[pos] == -1) || (A[pos] == -2)) {  
            A[pos] = x;  
            b=1;  
        }  
    }  
    return b;  
}
```

**-1: posizione vuota**  
**-2: posizione disponibile**

## Metodo hash: ricerca con scansione lineare

```
bool hashSearch (int *A , int k, int x) {  
    int i=h(x);  
    for (int j=0; j<k; j++) {  
        int pos = (i+j)%k;      // somma in modulo  
        if (A[pos ]== -1) return false ;  
        if (A[pos ] == x) return true ;  
    }  
    return false ;  
}
```

-1: posizione vuota  
-2: posizione liberata

## Scansioni

**scansione\_lineare**(x;j) =  $(h(x) + \text{cost} * j) \bmod k$

Es:  $(h(x) + j) \bmod k, j=1, 2, \dots$

**scansione\_quadratica**(x; j) =  $(h(x) + \text{cost} * j^2) \bmod k$

Es:  $(h(x) + j^2) \bmod k, j=1, 2, \dots$

**La diversa lunghezza del passo di scansione riduce gli agglomerati, ma è necessario controllare che la scansione visiti tutte le possibili celle vuote dell'array, per evitare che l'inserimento fallisca anche in presenza di array non pieno.**

## Tempo medio di ricerca per l'indirizzamento aperto

**Il tempo medio di ricerca (numero medio di confronti) dipende da**

- **Fattore di carico:** rapporto  $\alpha = n/k$  (sempre  $\leq 1$ ) :  
numero medio di elementi per ogni posizione
- **Legge di scansione** (migliore con la scansione quadratica e altre più sofisticate)
- **Uniformità della funzione hash** (genera gli indici con uguale probabilità)

## Stima del numero medio di accessi

**numero medio di accessi con la scansione lineare**

$$\leq 1/(1 - \alpha)$$

**Per esempio per  $\alpha = 0,5$ , abbiamo 2 accessi, per  $\alpha = 0,9$  ne abbiamo 10**

## Esempio di tempi di scansione

$\frac{n}{M}$	scansione lineare	scansione quadratica
10%	1.06	1.06
50%	1.50	1.44
70%	2.16	1.84
80%	3.02	2.20
90%	5.64	2.87

$$M=k$$

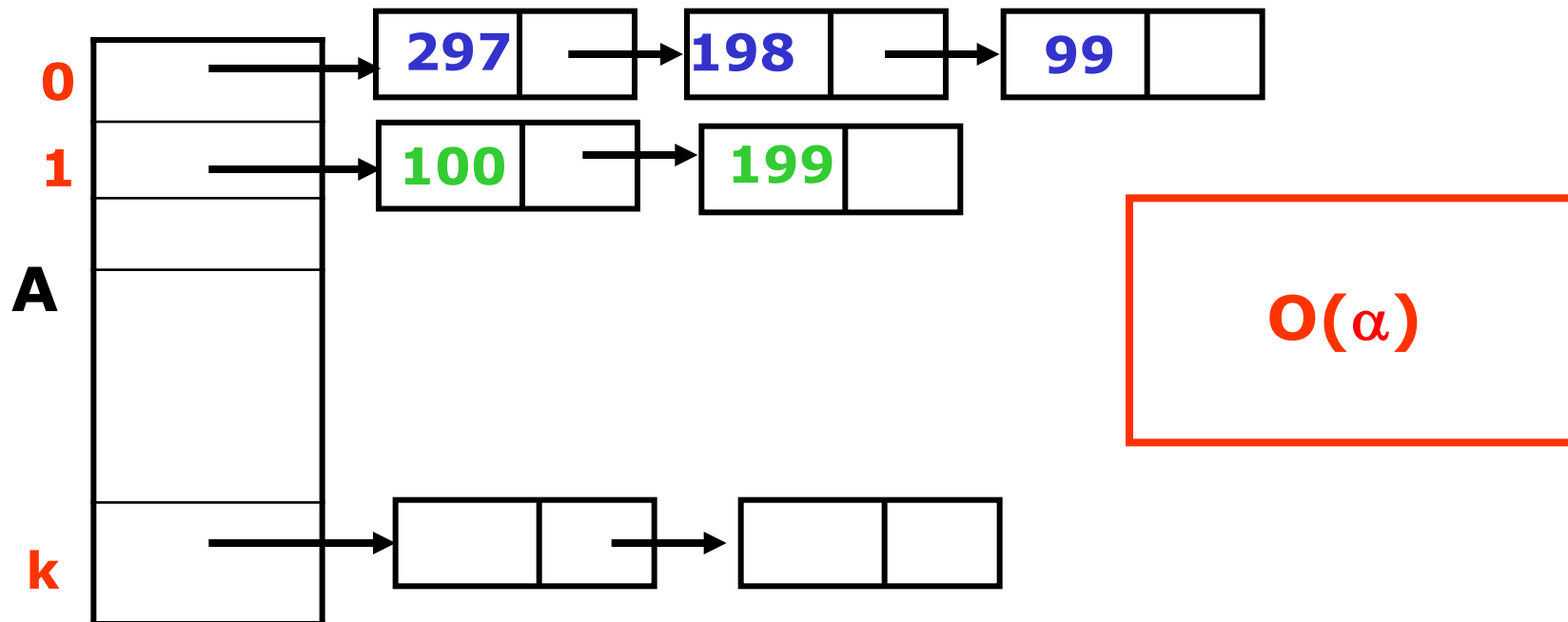
### **Problemi con l'indirizzamento aperto**

**Molti inserimenti e cancellazioni degradano il tempo di ricerca a causa degli agglomerati. E' necessario periodicamente "risistemare" l'array.**



## Metodo di concatenazione

- Array A di  $k \leq n$  puntatori (  $n/k \geq 1$  )
- elementi che collidono su  $i$  nella lista di puntatore  $A[i]$
- evita del tutto gli agglomerati



## Qualche domanda

Quale è la complessità dell'inserimento con il metodo di concatenazione?

Quale è la complessità della cancellazione con il metodo di concatenazione?

Quale è il principale problema del metodo di concatenazione?

## Dizionari (tabelle)

chiave	informazione

**Ricerca**  
**Inserimento**  
**Cancellazione**

Con  **$h(\text{chiave})$**  si raggiunge l'informazione

**Es: rubrica telefonica, studenti (chiave: matricola)**

## Da Data Base Relazionali a Dizionari K-V

<b>employee_id</b>	<b>first_name</b>	<b>last_name</b>	<b>address</b>
1	John	Doe	New York
2	Benjamin	Button	Chicago
3	Mycroft	Holmes	London

**FOREIGN KEY**

<b>payment_id</b>	<b>employee_id</b>	<b>amount</b>	<b>date</b>
1	1	50,000	01/12/2017
2	1	20,000	01/13/2017
3	2	75,000	01/14/2017
4	3	40,000	01/15/2017
5	3	20,000	01/17/2017
6	3	25,000	01/18/2017

*Image extracted from:*

*<https://medium.com/@wishmithasmendis/from-rdbms-to-key-value-store-data-modeling-techniques-a2874906bc46>*

# Da Data Base Relazionali a Dizionari K-V

employee_id	first_name	last_name	address
1	John	Doe	New York
2	Benjamin	Button	Chicago
3	Mycroft	Holmes	London



```
employee:$employee_id:$attribute_name = $value
```

```
employee:1:first_name = "John"  
employee:1:last_name = "Doe"  
employee:1:address = "New York"
```

```
employee:2:first_name = "Benjamin"  
employee:2:last_name = "Button"  
employee:2:address = "Chicago"
```

```
employee:3:first_name = "Mycroft"  
employee:3:last_name = "Holmes"  
employee:3:address = "London"
```

# Da Data Base Relazionali a Dizionari K-V

:

**payment:\$payment\_id:\$employee\_id:\$attribute\_name = \$value**

payment_id	employee_id	amount	date
1	1	50,000	01/12/2017
2	1	20,000	01/13/2017
3	2	75,000	01/14/2017
4	3	40,000	01/15/2017
5	3	20,000	01/17/2017
6	3	25,000	01/18/2017



```
payment:1:1:amount = "50000"  
payment:1:1:date = "01/12/2017"
```

```
payment:2:1:amount = "20000"  
payment:2:1:date = "01/13/2017"
```

```
payment:3:2:amount = "75000"  
payment:3:2:date = "01/14/2017"
```

```
payment:4:3:amount = "40000"  
payment:4:3:date = "01/15/2017"
```

```
payment:5:3:amount = "20000"  
payment:5:3:date = "01/17/2017"
```

```
payment:6:3:amount = "25000"  
payment:6:3:date = "01/18/2017"
```

## Da Data Base Relazionali a Dizionari K-V

```
employee:1:first_name = "John"  
employee:1:last_name = "Doe"  
employee:1:address = "New York"  
  
employee:2:first_name = "Benjamin"  
employee:2:last_name = "Button"  
employee:2:address = "Chicago"  
  
employee:3:first_name = "Mycroft"  
employee:3:last_name = "Holmes"  
employee:3:address = "London"  
  
payment:1:1:amount = "50000"  
payment:1:1:date = "01/12/2017"  
  
payment:2:1:amount = "20000"  
payment:2:1:date = "01/13/2017"  
  
payment:3:2:amount = "75000"  
payment:3:2:date = "01/14/2017"  
  
payment:4:3:amount = "40000"  
payment:4:3:date = "01/15/2017"  
  
payment:5:3:amount = "20000"  
payment:5:3:date = "01/17/2017"  
  
payment:6:3:amount = "25000"  
payment:6:3:date = "01/18/2017"
```

## Esercizio

Dato un array di 101 posizioni che memorizza un insieme di al massimo **80 elementi**, indicare il contenuto delle prime **4 celle** dell'array, inizialmente vuoto, dopo le operazioni seguenti (indirizzamento con il metodo del resto, scansione lineare unitaria):

- a) Inserimento dell'elemento 101
- b) Inserimento dell'elemento 202
- c) Inserimento dell'elemento 204
- d) Cancellazione dell'elemento 202
- e) Inserimento dell'elemento 304

Indicare il risultato se l'azione d) non viene eseguita

**Qual è il fattore di carico?**



## Bibliografia

Demetrescu:

Paragrafo 7.1 e 7.2

Cormen:

Capitolo 11