Massimo Benerecetti

Tabelle Hash

Lezione n....

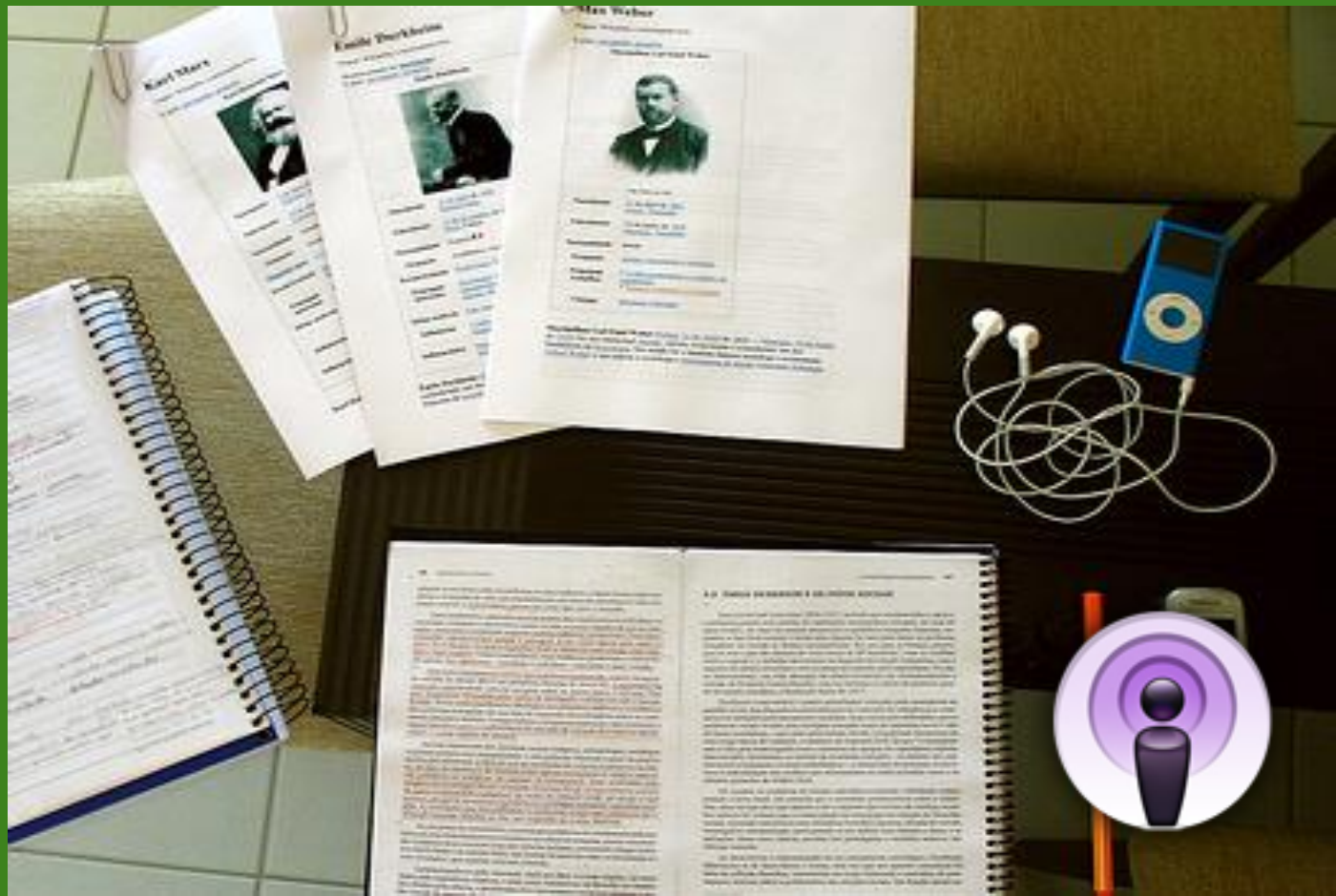
Parole chiave:
Inserire testo

Corso di Laurea:
Informatica

Insegnamento:
Algoritmi e
Strutture Dati I

Email Docente:
bene@na.infn.it

A.A. 2009-2010



Rappresentazione di insiemi dinamici

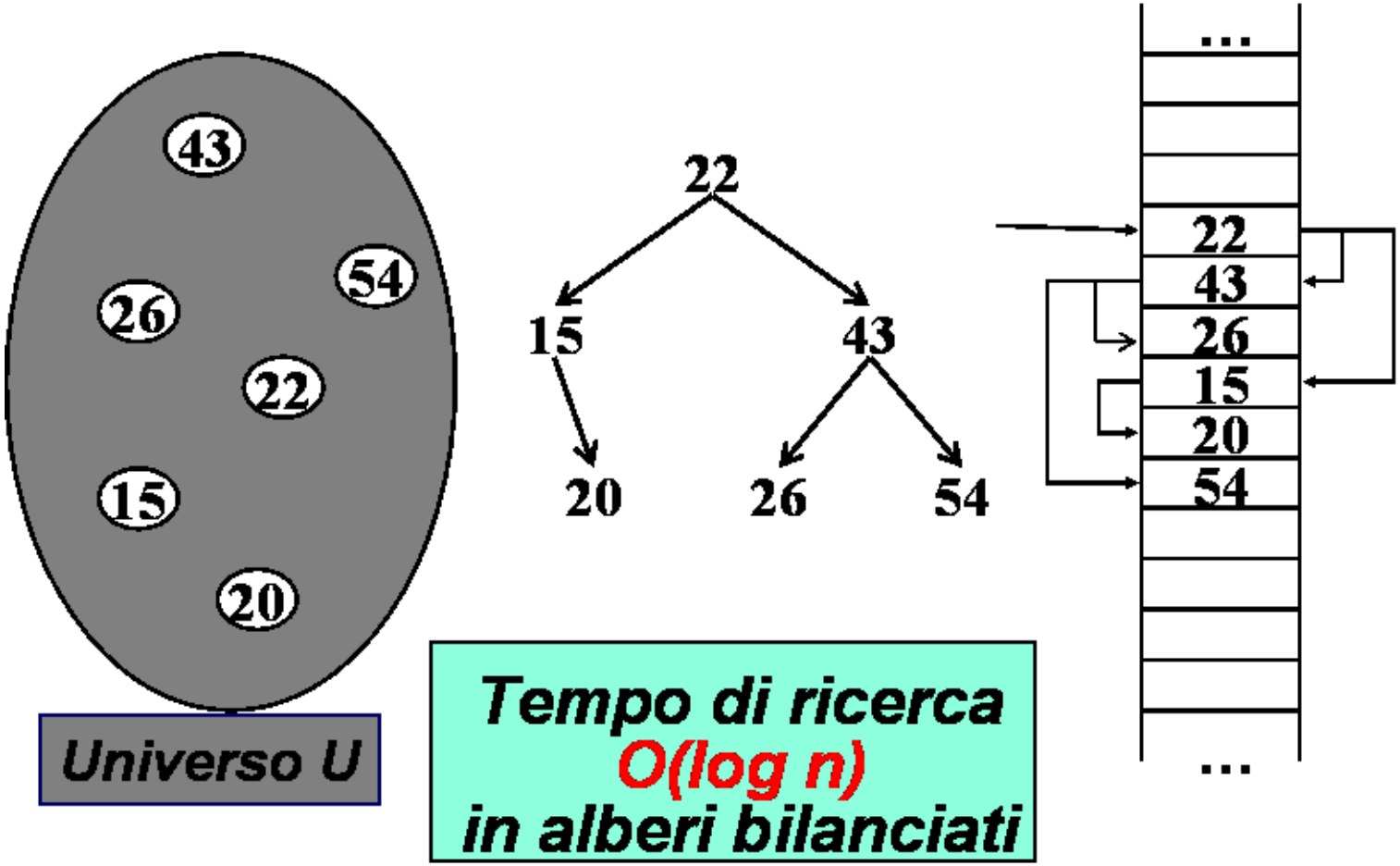
- Gli insiemi dinamici possono essere rappresentati con varie strutture dati, ciascuna con caratteristiche di flessibilità e di prestazioni differenti.

- Array, liste ed alberi sono tra le rappresentazioni più diffuse.

- Gli alberi binari di ricerca bilanciati offrono un buon compromesso tra flessibilità e prestazioni, garantendo tempi di ricerca logaritmici rispetto al numero di elementi.

- Rinunciando ad un po' della flessibilità degli alberi, è possibile però ottenere strutture dati con migliori prestazioni per la ricerca degli elementi.

Insiemi dinamici come alberi



Rappresentazione ad ***albero*** di un insieme dinamico di chiavi prese da un universo **U**.

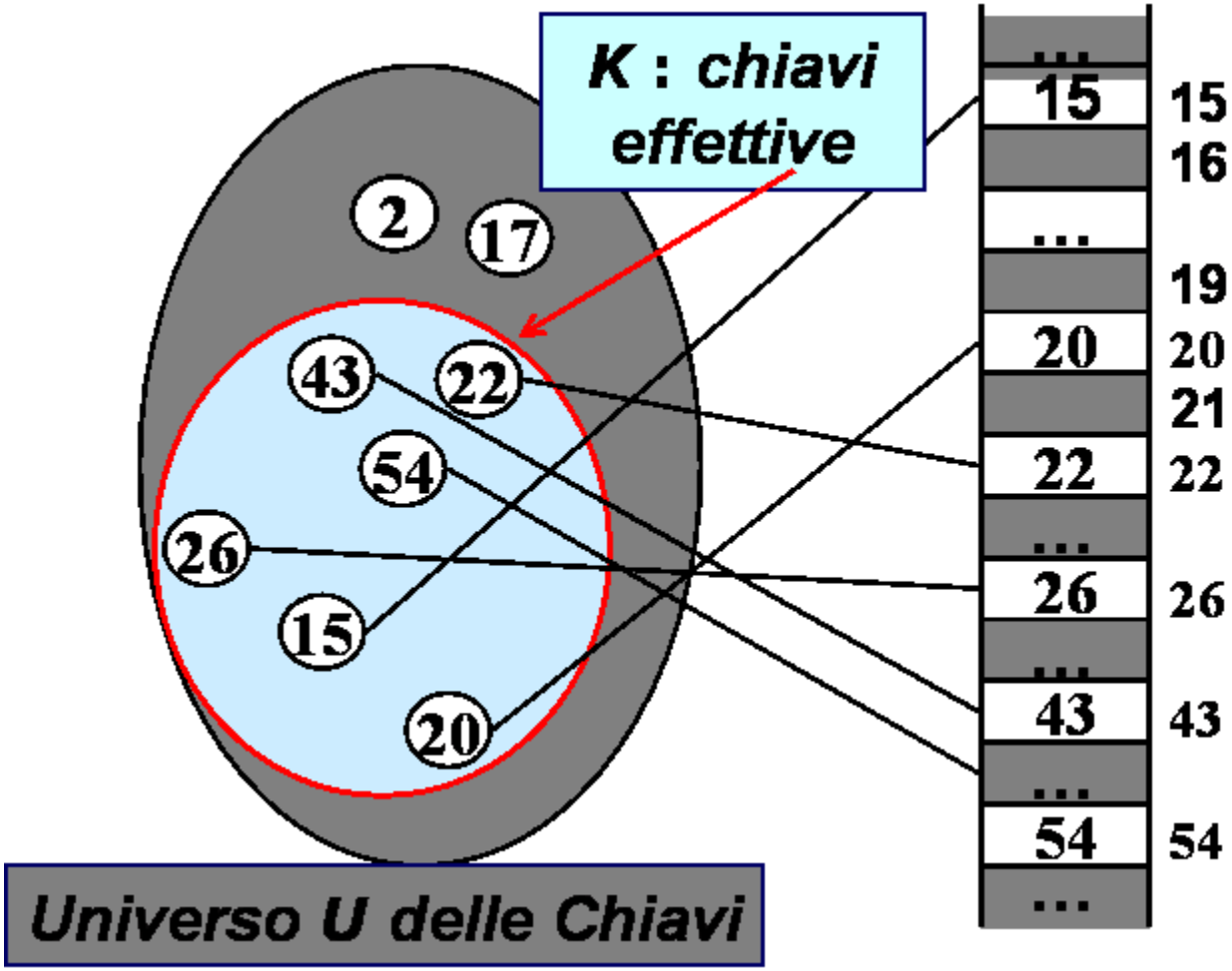
Tabelle ad accesso diretto

- Una ***tabella ad accesso diretto*** è una struttura dati che *suppotra SOLO* le operazioni di:
  - inserimento
  - ricerca
  - cancellazione

- in tempo che è ***O(1)***

- *Non supporta direttamente Minimo, Massimo, Successore, Predecessore (cioè gli Ordinamenti)*

Insiemi dinamici come tabelle ad accesso diretto



Rappresentazione con ***tabella ad accesso diretto*** per un insieme dinamico di chiavi prese da un universo **U**.

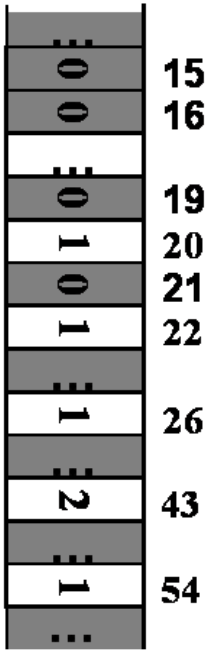
Inserire titolo slide

- La più *semplice* implementazione di una ***tabella ad accesso diretto*** è un ***array***.

- Per memorizzare gli interi a 16-bit possiamo utilizzare un array ***A*** di dimensione $\mathbf{2^{16}}$.

- Le operazioni potrebbero essere definite come segue:

– `inserisci(i):  A[i] = A[i] + 1`
– `ricerca(i)  :  (A[i] > 0)?`
– `cancella(i) :  A[i] = A[i] - 1`



Inserisci: 20, 22, 26, 43, 54, 43

Esempio di funzione indice e tabella ad accesso diretto.

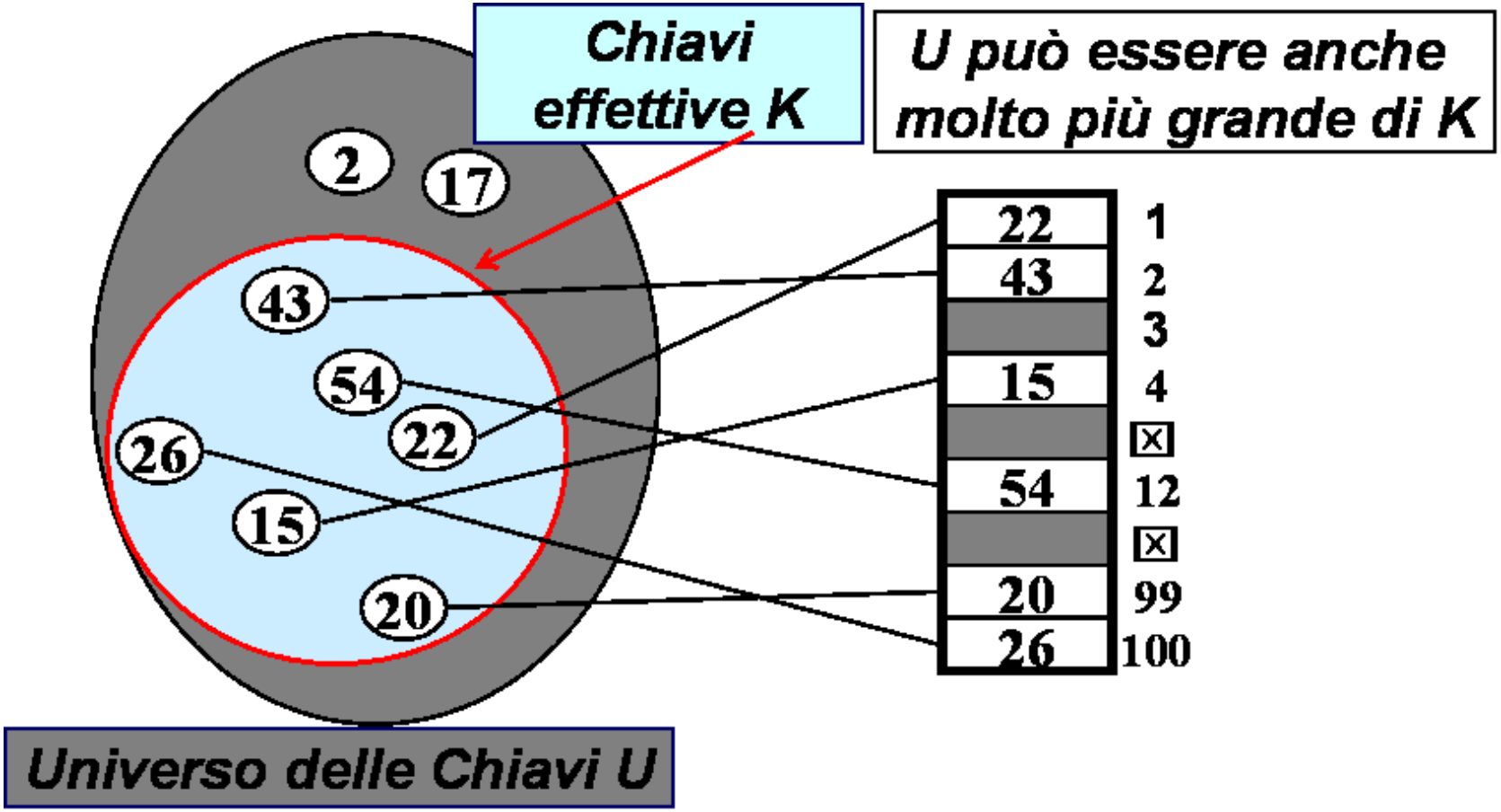
Tabelle Sparse

- Se le ***chiavi*** sono ***stringhe di 8 lettere*** alfabetiche, ci sono **26^8** (o circa 200 miliadri) di possibili chiavi [circa 200 'giga' di chiavi].

- Quasi sempre solo una ***piccola frazione*** di queste chiavi verrà effetticamente ***impiegata***.

- Ne risulterebbe la necessità di un ***array molto grande***, ma con ***pochissime celle occupate***.

- Ci serve, quindi, una ***soluzione migliore!***



Rappresentazione di una ***tabella hash*** per un insieme dinamico di chiavi prese da un universo **U**.

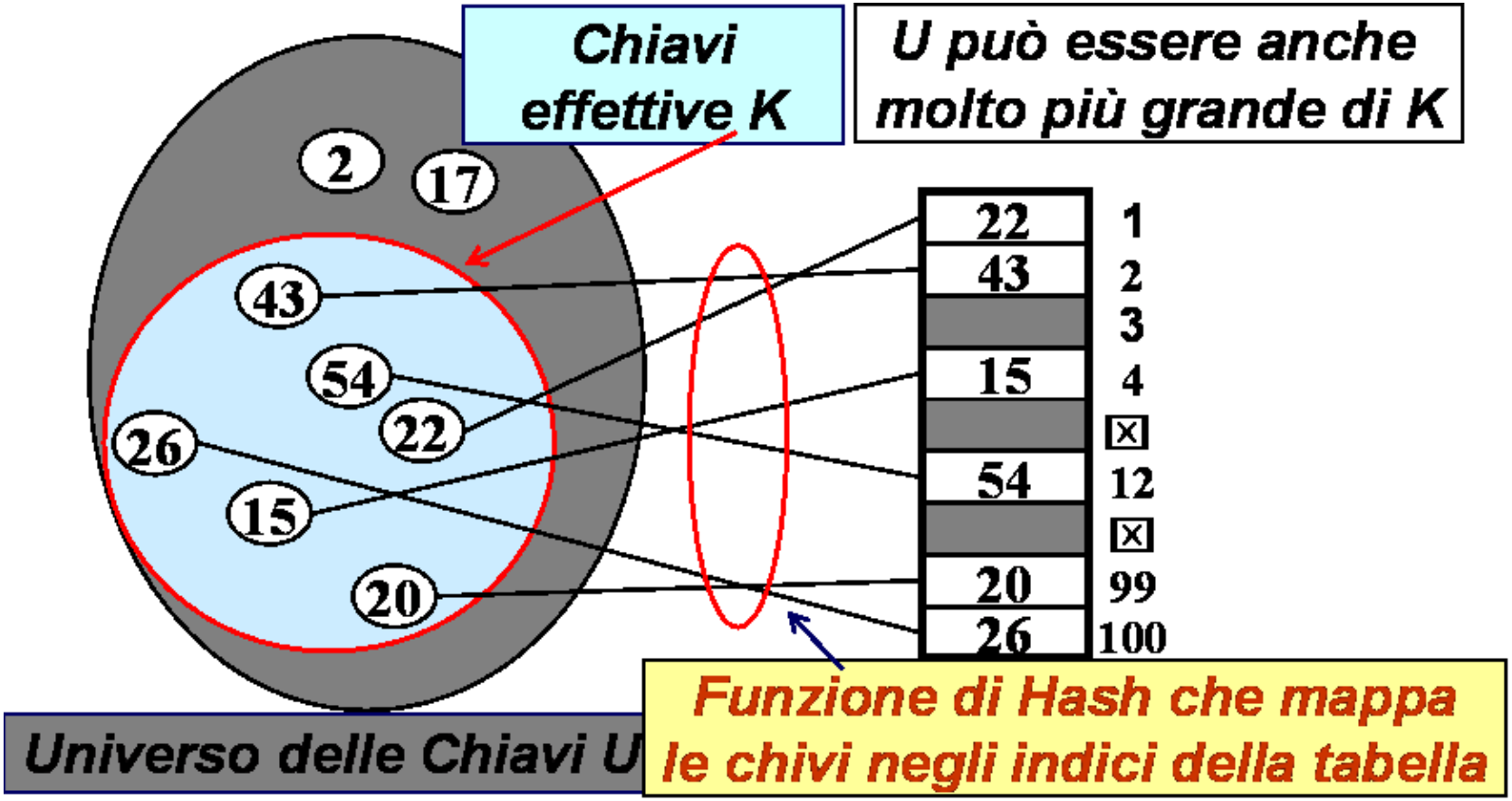


Illustrazione di ina ***tabella hash*** e della ***funzione di hashing***.

Tabelle Hash

- Uno ***schema di hashing*** consiste di una ***tabella ad accesso diretto*** (la tabella hash) e di una ***funzione di hashing*** con dominio l'universo delle chiavi e codominio l'insieme degli indici della tabella.

- Una ***funzione hash*** prende in input una chiave e la "***mappa***" su qualche ***indice*** all'interno della tabella.

- Uno schema di hashing ammette che differenti chiavi possibili vengano "***mappate***" nella stessa locazione. Quando ciò avviene, si parla di ***collisione*** tra chiavi.

- È quindi necessario definire dei meccanismi opportuni per la gestione delle ***collisioni***.

Operzioni su Tabelle Hash

Data una funzione di hash ***hash(key)*** che ritorna un intero, l'approccio semplicistico potrebbe essere il seguente

- `inserisci(key):  A[hash(key)] = key`
- `ricerca(key)  :  (A[hash(key)] == key)?`
- `cancella(key) :  A[hash(key)] = NULL`

Progettazione di uno schema di hashing

Nella definizione di una ***tabella hash*** adeguata alle necessità applicative si deve scegliere:

- una opportuna *funzione di hash* che abbia buone proprietà di distribuzione uniforme delle chiavi sugli indici
- la *dimensione della tabella* che spesso dipende dal tipo di funzione hash scelto
- la *politica* di *gestione* *e soluzione* delle *collisioni*

$$hash(Key) = Key \bmod TSIZE$$

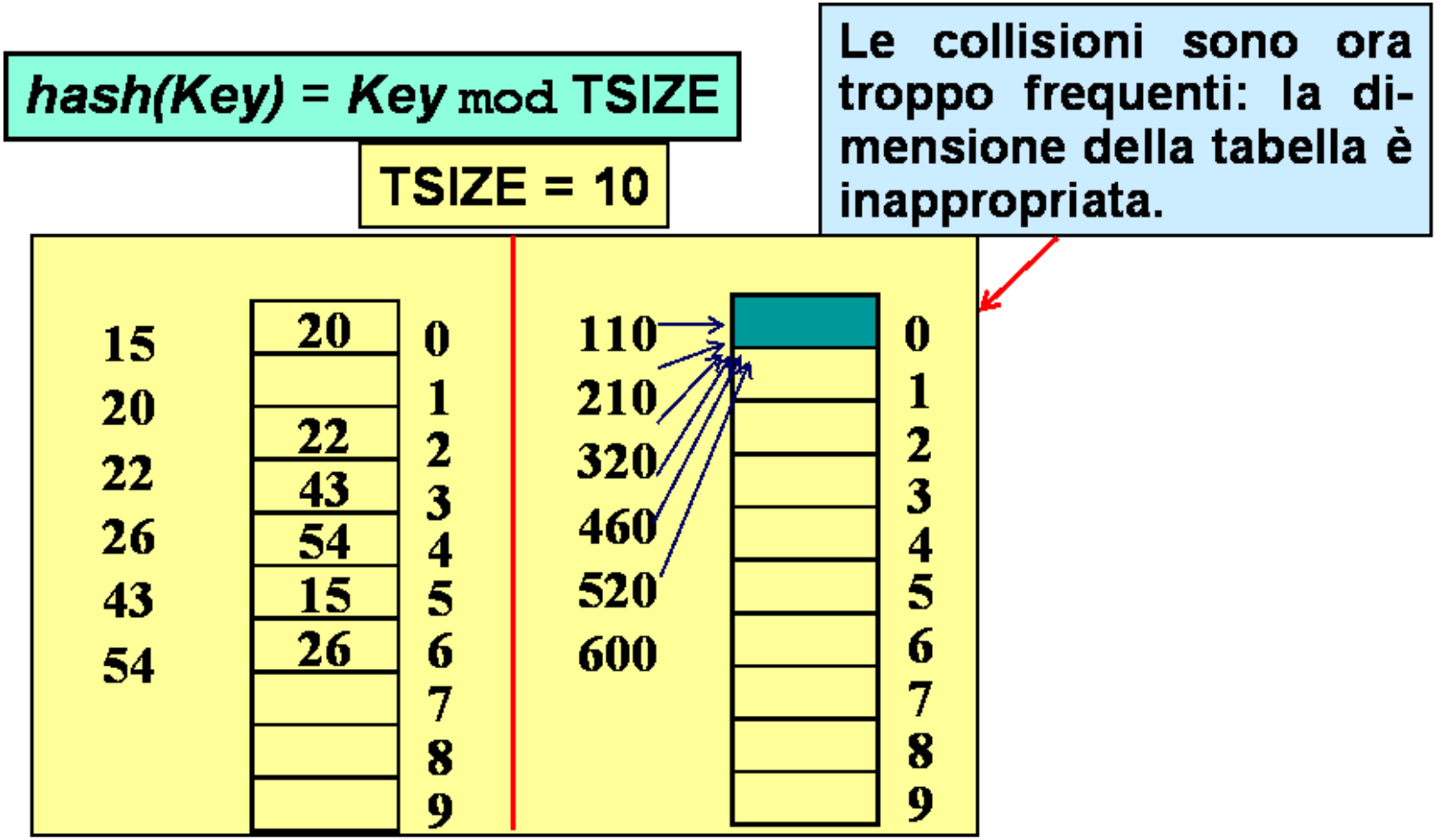
TSIZE = 10

15	20	0
20		1
22	22	2
26	43	3
43	54	4
54	15	5
	26	6
		7
		8
		9

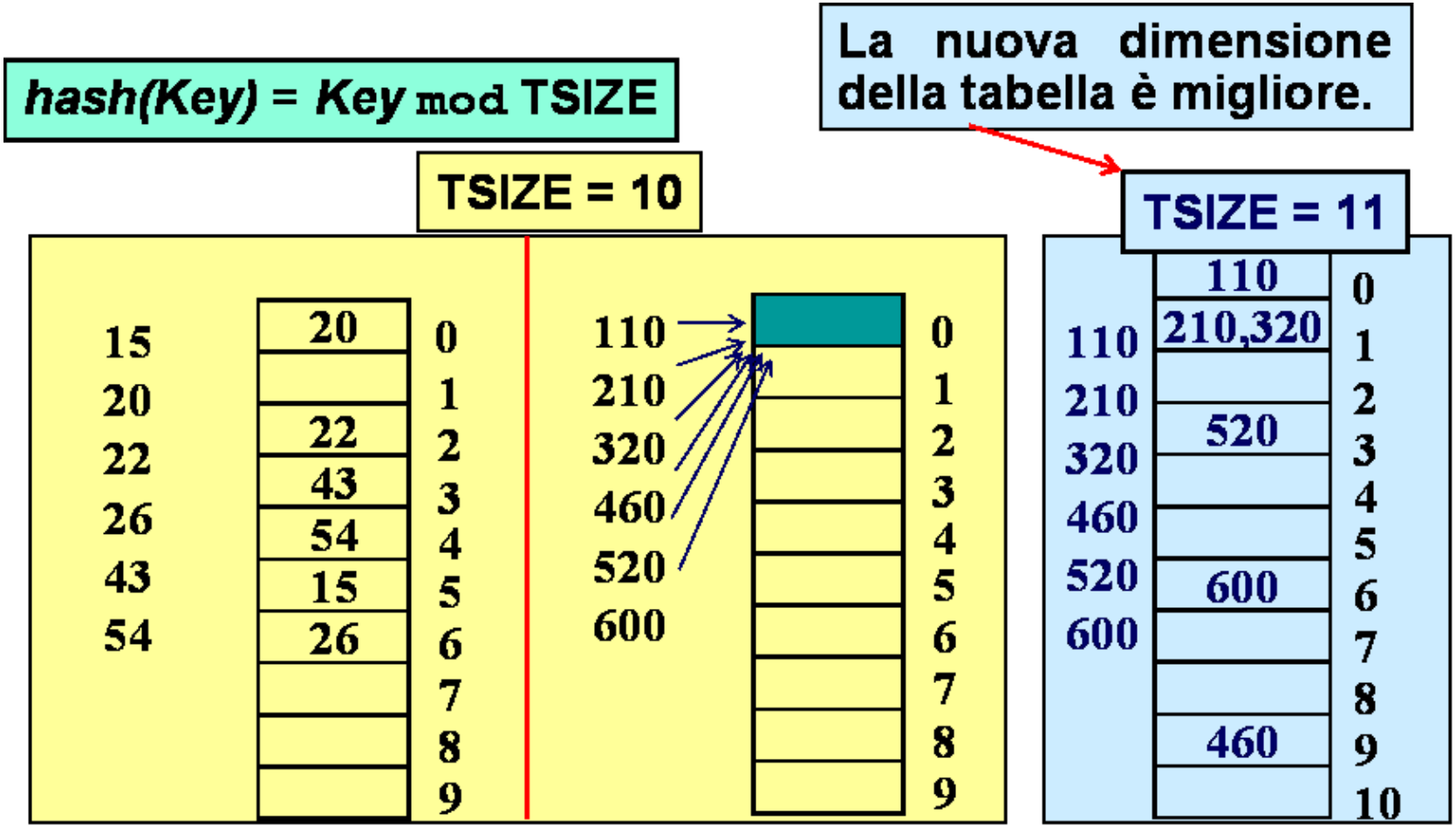
Semplice funzione hash su interi:

- TSIZE è la dime'nsione della tabella
- mod è l'operazione di modulo

Esempio di tabella hash e funzione hash.



Esempio di tabella hash e funzione hash con molte collisioni.



Esempio di tabella hash e funzione hash con meno collisioni.

Dimensioni della tabella hash

La ***dimensione della tabella*** può influire sulla *frequenza delle collisioni*

- Numero Composto
  - 10: 2*5
  - 300: 2*2*3*5*5
  - Maggiori possibilità di collisioni
- Numero Primo
  - 11
  - 10007
  - Minori possibilità di collisioni

Scelta della funzione di hash

- Una proprietà importante di un meccanismo di hashing è quella di ***Hashing Uniforme Semplice***: *ogni chiave ha la stessa probabilità di essere mappata in una delle n celle della tabella, indipendentemente dalla cella in cui è mappata ogni altra chiave*.

- Proprietà desiderabili di una "**buona**" ***funzione di hash*** sono:
  - efficienza e facilità di calcolo
  - distribuzione uniforme delle chiavi sul dominio degli indici
  - minimizzazione delle collisioni

Metodi di costruzione di una funzione hash

1. ***Metodo della Divisione***

   Convertire la chiave in un intero e calcolare il modulo (**mod**) rispetto alla dimensione della tabella

2. ***Metodo de Moltiplicazione***

   Convertire la chiave in un intero e calcolare un valore tramite operazioni di moltiplicazione.

3. ***Troncamento***

   Ignorare parte della chiave e usare la porzione che rimane come indice

4. ***Folding***

   Partizionare la chiave in parti differenti e combinare queste parti in modo da ottenere l'indice

- **Metodo della divisione**

  e.g. Semplicemente calcolare il ***modulo*** rispetto alla dimensione della tabella:

$$hash(62538194) = 62538194 \textbf{ mod } 1000 = 194$$

- Il metodo è sensibile al valore scelto per la dimensione della tabella.

  e.g. l'uso di una potenza di due per la dimensione può causare scarsa uniformità della funzione. Se $n = 2^p$, allora vengono considerati solo i p bit meno significativi della chiave.

- Nel caso dell'aritmetica modulare, la migliore scelta della ***dimensione della tabella hash*** è un ***numero primo*** non troppo vicino ad una potenza di 2.

- Nel caso sopra, tabelle di dimensione 997 o 1009 (entrambi numeri primi) darebbero migliori prestazioni dal punto di vista della distribuzione delle chiavi sugli indici.

Metodo della moltiplicazione

- **Metodo della motiplicazione**
  1. Prima si moltiplica la chiave ***k*** per una costante ***a*** nell'intervallo $0 < \boldsymbol{a} < 1$.
  2. Poi si estrae la parte frazionaria del prodotto $\boldsymbol{k \cdot a}$ (cioè $\boldsymbol{k \cdot a} - \lfloor \boldsymbol{k \cdot a} \rfloor$).
  3. Infine si moltiplica il valore ottenuto per la dimensione ***n*** della tabella.

$$hash(\boldsymbol{k}) = \lfloor \boldsymbol{n} \cdot (\boldsymbol{k \cdot a} - \lfloor \boldsymbol{k \cdot a} \rfloor) \rfloor$$

- Questo metodo non è particolarmente sensibile al valore ***n*** della dimensione della tabella.
- Prestazioni molto buone in termini di uniformità si sono ottenute prendendo $a = \frac{\sqrt{5}-1}{2} \approx 0{,}618$

Metodo della moltiplicazione

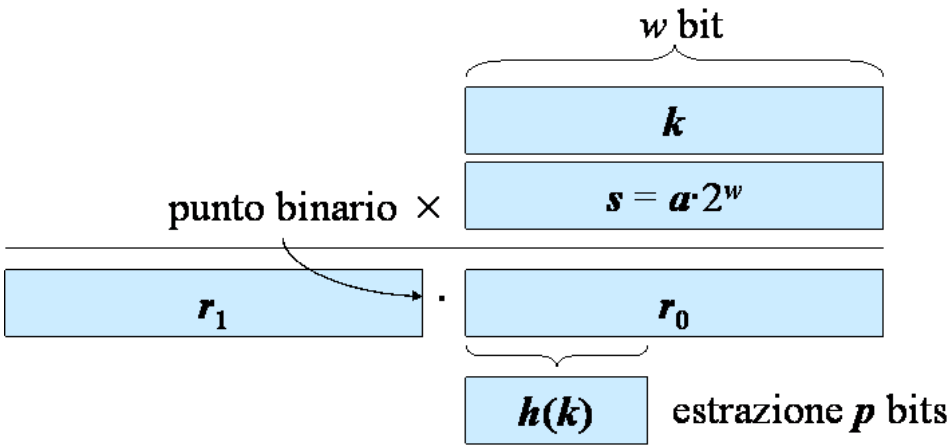
Il metodo della moltiplicazione può essere implementato facilmente ed efficientemente scegliendo il valore n come una potenza di **2**.

- Sia $n = 2^p$ per qualche intero p.
- Sia w la dimensione della parola della macchina (numero di bit in una parola).
- Scegliamo $a = s/2^w$, con $0 < s < 2^w$ intero.
- Moltiplicando k per $s = a \cdot 2^w$. Il risultato sarà un valore di $2 \cdot w$ bit, della forma

$$k \cdot s = r_1 \cdot 2^w + r_0$$

dove r_1 è la parte più significativa del prodotto, e r_0 quella meno significativa.

- Il valore *hash*(k) desiderato è rappresentato dai p bit più significativi di r_0.



Implementazione del metodo della moltiplicazione.

Si noti che:

$$k \cdot a - \lfloor k \cdot a \rfloor = \mathrm{frac}\left(\frac{k \cdot s}{2^w}\right) = r_0$$

e che posto $n = 2^p$ si ottiene

$$n \cdot r_0 = 2^p \cdot r_0$$

cioè i p bit più significativi di r_0.

Troncamento e folding

Dati interi di 8 cifre e una tabella di dimensione 1000

- **Tromcamento**
  - e.g.: usare congiuntamente solo la 4ª, 7ª e 8ª cifra per formare l'indice *hash*(62538194) = 394
- **Folding**
  - e.g.: suddividere ogni chiave in gruppi di 3, 3, e 2 cifre, sommare le parti e troncatare se necessario

$$hash(62538194) = (625+381+94) \textbf{ mod } 1000 = 1100 \textbf{ mod } 1000 = 100$$

In genere questi ultimi metodi vengono utilizzati insieme a uno dei due precedenti.

Funzioni hash per chiavi a stringa

- Definire una funzione che trasforma una stringa in un intero.
- Ad esempio, sommando il valore ASCII di tutti i caratteri:

$$hash(Key) = \sum_i Key[i]$$

$$hash(Key) = \sum_i k^i \cdot Key[n-i] \quad \text{(per } \mathbf{k} \text{ intero)}$$

- Problema: quando le chiavi sono corte e la tabella è grande

  8 caratteri, TSIZE = 10007

  8*256 = 2048

  La tabella necessaria può risultare sproporzionata, determinando spreco di spazio.
- Una possibile soluzione è quella di usare solo alcuni caratteri e moltiplicare tra loro i valori dei caratteri (*metodo del troncamento*):

  C a p o **V** e **r** d e

  Numero di possibili valori di indice: 27*27*27 **=** 17576 > 10007

  Può essere necessario, quindi, integrarlo con il *metodo della divisione*.
- Problema: le lingue non sono casuali

  molte meno combinazioni effettivamente possibili di quelle permesse

  rischio di spazio sprecato

Funzioni hash per chiavi a stringa: esempio

Le seguenti funzioni di hash pesano diversamente ciascun carattere della stringa e impiegano il metodo della divisione (ipotizzando 27 diversi caratteri alfabetici) :

1. $hash_1(Key) = (\ldots + 27^2\, Key[2] + 27\, Key[1] + Key[0] \ldots) \bmod TSIZE$
   $= ((\ldots + Key[2])*27 + Key[1])*27 + Key[0])\ldots) \bmod TSIZE$

2. $hash_2(Key) = ((\ldots + Key[2])*32 + Key[1])*32 + Key[0])\ldots) \bmod TSIZE$

L'algoritmo sotto riportato calcola la seconda funzione di $hash_2$ nell'esempio sopra:

```
Hash_2(key[])
    i = 1
    WHILE (key[i] ≠ '\0') DO
        hash  = (shift(hash,5)) + key[i]
        i  = i + 1
    return (hash mod TSIZE)
```

Si noti che l'espressione **shift(*hash*,5)** corrisponde alla moltiplicazione del valore contenuto in ***hash*** per **32** (cioè $\mathbf{2^5}$)

Funzioni hash per chiavi a stringa: esempi

$hash_3(0) = 5381$

$hash_3(i) = hash_3(i - 1) * 33 + key[i]$

```
Hash_3(key[])
    hash = 5381
    i = 1
    WHILE (key[i] ≠ '\0') DO
        hash  = ((shift(hash,5)) + hash) + key[i]
        i  = i + 1
    return (hash mod TSIZE)
```

Questa funzione di hash ha mostrato prestazioni di uniformità particolarmente buone in pratica.

Un'altra possibilità è ***usare il folding***: elaborare la stringa 4 byte alla vota, convertendo ogni gruppo di 4 byte in un intero, usando uno dei metodi sopra descritti. I valori interi di ogni gruppo vengono poi sommati tra di loro. Infine, si converte il risultato in un intero tra 0 e TSIZE tramite operazione di modulo.

Funzioni hash per chiavi a stringa

- Semplice somma dei valori numerici dei caratteri della stringa
  - Molto semplice da implementare.
  - Può impiegare molto tempo se le chiavi sono lunghe.
  - I primi caratteri possono non venir considerati.
  - Posso essere spostati (**`shift`**) fuori dal range.
- Una possibile soluzione può essere quella di adottare una variante del folding:
  - usare solo alcuni caratteri
  - e.g. Via Cintia 345, Napoli, I-81100
- Un'altra soluzione può essere quella utilizzare il metodo della divisione insieme alla somma (pesata) dei diversi caratteri.