

# Tabelle hash

Vittorio Maniezzo - Università di Bologna

1

1

## Look-up table, caso generale

Anche *memoria associativa*, *associative array*, *map*, *symbol table*, *dictionary*: collezione di elementi nella forma coppia attributo / valore (key, value), quindi dati generici, caratterizzati da un campo chiave, con associate le operazioni:

- Inserimento di una coppia nella collezione
- Cancellazione di una coppia dalla collezione
- Ricerca di una chiave
- Modifica di un valore corrispondente ad una chiave

Vittorio Maniezzo - Università di Bologna

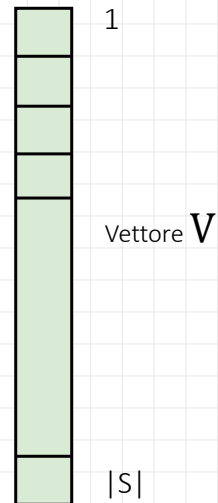
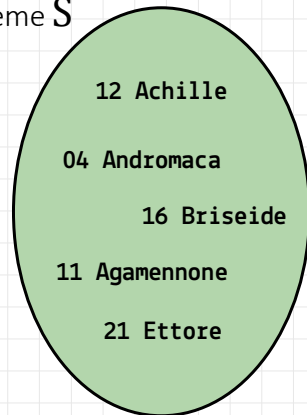
2

2

## Memorizzazione dati strutturati

Ogni elemento ha una chiave  $K_i$  tale che  $1 \leq K_i \leq n$

Insieme  $S$



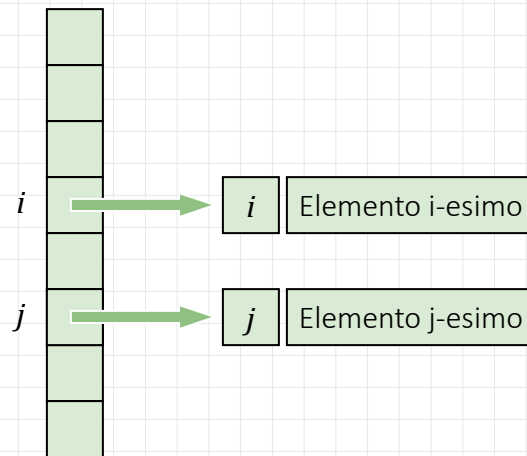
Vittorio Maniezzo - Università di Bologna

3

3

## Tabella a indirizzamento diretto

Posizione nel vettore = valore della chiave



Vittorio Maniezzo - Università di Bologna

4

4

## Tabella a indirizzamento diretto

**Search**(V,k)  
return V[k]

**Insert**(V,x)  
V[key[x]] = x

**Delete**(V,x)  
V[key[x]] = nil

Costo  
computazionale =  $\Theta(1)$

Vittorio Maniezzo - Università di Bologna

5

5


## Problemi...

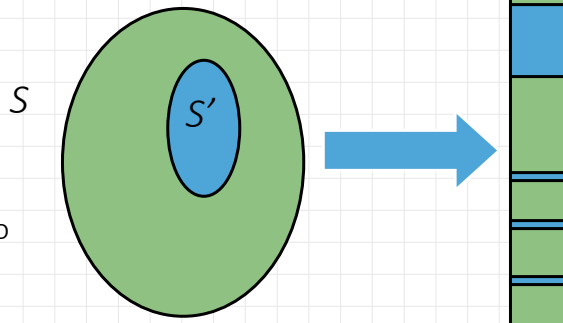
Supponiamo che solo una parte  $S'$  dello spazio  $S$  delle chiavi sia utilizzata/attiva.

Cosa succede quando  $|S'| \ll |S|$  ?

Si spreca molto spazio di memoria !!!

Soluzioni?

 = Spazio sprecato



Vittorio Maniezzo - Università di Bologna

6

6

## Una soluzione ...

Possiamo ridurre l'occupazione di spazio da  $\Theta(|S|)$  a  $\Theta(|S'|)$  usando liste.

PROBLEMA (non finiscono mai):  
Inserimento, Cancellazione e Ricerca costano  $\Theta(|S'|)$  invece di  $\Theta(1)$ .

Cala la memoria e cresce il tempo ...

## Compromesso TEMPO / SPAZIO

Si possono ottenere tutti e due i risultati assieme!

Usando tabelle Hash possiamo raggiungere:

- Tempo di accesso:  $\Theta(1)$
- Spazio di memoria:  $\Theta(|S'|)$

Ma ... **in media** e non nel caso pessimo.

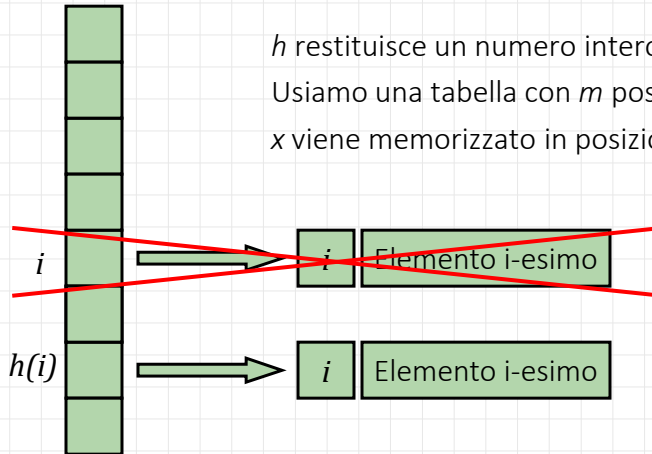
## $h$ , funzione di hash

$h$  = funzione hash

$h$  restituisce un numero intero da 1 a  $m$ .

Usiamo una tabella con  $m$  posizioni

$x$  viene memorizzato in posizione  $h(key[x])$



Vittorio Maniezzo - Università di Bologna

9

9

## Proprietà per una “buona” $h$

Deterministica ma deve sembrare “random” in modo da **minimizzare le collisioni**.

$x$  e  $y$  generano una **collisione** se

$$x \neq y \quad \text{e} \quad h(x) = h(y)$$

$h$  deve minimizzare il numero di collisioni

Vittorio Maniezzo - Università di Bologna

10

10

## Frequenza delle collisioni

Tabella di dimensione  $m$  e  $p$  elementi da inserire:

Funzione hash,  $m^p$  diverse possibilità di allocazione dei  $p$  elementi

(Esempio: con  $p = 8$  e  $m = 10 \rightarrow 10^8$  possibili allocazioni diverse)

Ci sono  $\frac{m!}{(m-p)!}$  possibilità di un hashing senza collisioni

(Esempio: se  $p = 8$  e  $m = 10$  ci sono  $3 \cdot 4 \cdot \dots \cdot 10$  possibilità di hashing senza collisioni)

### Paradosso del compleanno

Con 23 persone, la probabilità che tutti abbiano il compleanno in un giorno diverso è  $< 1/2$

Cioè: con  $p = 23$  e  $m = 365$ , la probabilità di collisione è  $\geq 1/2$

(prob. cond. ogni compl. diverso =  $\left(\frac{1}{365}\right)^{23} \cdot (365 \cdot 364 \cdot \dots \cdot 343) \approx 0.49$ )

La prob. di collisione cresce in modo sorprendente!

Vittorio Maniezzo - Università di Bologna

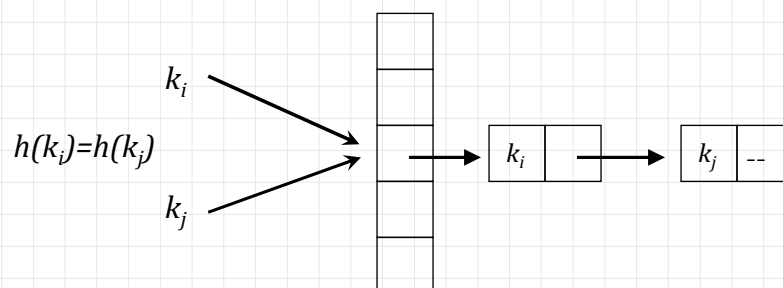
11

11

## Chaining

Risoluzione di collisioni con "chaining"

- Si crea una lista per ogni cella della tabella (valore assumibile dalla  $h$ )
- Si collegano i record afferenti a una stessa cella nella sua lista
- Le celle della tabella sono le teste delle liste (*nil* se liste vuote)



Vittorio Maniezzo - Università di Bologna

12

12

## Chaining

```
Chained-hash-insert(T,x)
  // inserisci x in testa alla lista
  T[h(key[x])]
```

```
Chained-hash-search(T,k)
  // ricerca l'elemento con chiave k nella lista
  T[h(k)]
```

```
Chained-hash-delete(T,x)
  // cancella x dalla lista
  T[h(key[x])]
```

Complessità di T()?

## Chaining, analisi

Load factor  $\alpha = n/m$ ,

$n$  = numero di elementi memorizzati in tabella

$m$  = dimensione della tabella

*Caso pessimo*: tutte le  $n$  chiavi finiscono nella stessa posizione. Ricerca =  $\Theta(n)$

*Caso medio*:

Simple uniform hashing:  $Pr(h(k)=i) = Pr(h(k)=j)$

## Simple uniform hashing, esempio

Supponiamo di dover inserire le chiavi

2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24     ( $n = 12$ )

In una tabella T con 10 celle     ( $m=10$ )

Usando la funzione  $h(x) = x \bmod(10)$

Le chiavi sono tutte pari! Tutte le celle di indice dispari non verranno utilizzate.

Le celle di indice dispari hanno probabilità di utilizzo  $p(i) = 0$

Le celle di indice pari hanno probabilità  $p(i) = 2 n/m$

La funzione  $h(x)$  non è SUH

Vittorio Maniezzo - Università di Bologna

15

15

## Simple uniform hashing

Una funzione hash si dice **semplice uniforme** quando rende uniforme il riempimento della tabella.

**Non** quando la distribuzione delle chiavi è uniforme.

Vittorio Maniezzo - Università di Bologna

18

18



## Complessità SUH

Teorema:

Ipotesi:

- collisioni gestite con chaining
- simple uniform hashing
- caso medio

Tesi:

una ricerca ha costo computazionale  $\Theta(1+\alpha)$

**Dimostrazione:** prima caso ricerca senza successo, poi con successo

## Complessità SUH

Dimostrazione: caso di ricerca **senza successo**.

Il load factor  $\alpha$  è la lunghezza media di una catena.

In una ricerca senza successo il numero di elementi esaminati è uguale alla lunghezza media delle catene, cioè  $\frac{n}{m} = \alpha$ .

Calcolare  $h()$  costa  $\Theta(1)$ .

La ricerca costerà  $\Theta(1) + \Theta(\alpha) = \Theta(1 + \alpha)$

## Complessità SUH

*Dimostrazione:* caso di ricerca **con successo**.

Assumiamo di inserire elementi *in coda* alla catena.

Simple uniform hashing  $\rightarrow$  numero medio di elementi in una catena dopo  $i$  inserimenti  $= i/m$

L'elemento  $i$  verrà inserito mediamente nella posizione  $1+(i-1)/m$  all'interno di una catena.

Un elemento generico finirà in media nella posizione data dalla formula:

$$\begin{aligned} 1/n \sum (1 + (i-1)/m) &= 1/n (n + [n(n+1)]/[2m] - n/m) = \\ &= 1 + \alpha/2 - 1/(2m) = \\ &= \Theta(1+\alpha) \end{aligned}$$

Vittorio Maniezzo - Università di Bologna

21

21

## Complessità SUH

Supponiamo che  $n=O(m)$ . Ovvero che il numero di elementi inseriti nella tabella sia proporzionale alla dimensione della tabella. Avremo:

$$\alpha = n/m = O(m)/m = O(1)$$

In questo caso la ricerca impiega **tempo costante !!!**

Cosa succede se gli elementi vengono inseriti all'inizio delle liste?

Vittorio Maniezzo - Università di Bologna

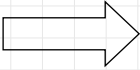
22

22

## Riepiloghiamo...

Se usiamo liste doppiamente linkate per le catene e se inseriamo i nuovi elementi in testa alle liste abbiamo

Ricerca  
Cancellazione  
Inserimento



$O(1)$  operazioni  
in media

## Funzioni hash: progettazione

$Pr(k)$  = probabilità della chiave  $k$

$S_j = \{ k \in U \text{ tali che } h(k) = j \}$

Vogliamo uniform hashing ovvero

$$\sum_{k \in S_j} Pr(k) = 1/m \quad (m = \text{dimensione della tabella})$$

## Funzioni hash: progettazione

Se  $\text{Pr}(\bullet)$  è sconosciuta  $\Rightarrow$  Usiamo euristiche

Supponiamo per semplicità che le chiavi siano numeri naturali.

IDEA:

- $h$  deve dipendere da tutti i bit di  $k$
- deve essere indipendente da eventuali pattern che possono essere presenti nelle chiavi

## Metodo della divisione

$$h(k) = k \bmod m$$

Esempio:

$m=12, k=100,$

$$h(100) = 100 \bmod 12 = 4$$

Per controllare se uno ha scelto un buon  $m$  è consigliabile usare un “benchmark” reale.

## Metodo della divisione

Metodo della divisione, possibili cattive scelte di  $m$

Se  $m$  e tutte le chiavi sono pari  $\rightarrow$  si usano solo la metà degli slot

Es.,  $m = 100$  e i dati sono 50, 100, 150, 200, 250

50	$\rightarrow$	50
100	$\rightarrow$	100
150	$\rightarrow$	50
200	$\rightarrow$	100
250	$\rightarrow$	50

Es.,  $m = 2^4 = 16$  e i dati sono 0, 32, 64, 96

0000000 = 0	$\rightarrow$	16
0010000 = 32	$\rightarrow$	16
0100000 = 64	$\rightarrow$	16
0110000 = 96	$\rightarrow$	16

**Euristica:** usare  $m$  primo e non troppo vicino a una potenza di 2 o di 10 (le basi più comuni)

## Metodo della moltiplicazione

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

- Scegli una costante  $A$  con  $0 < A < 1$
- Calcola  $k \cdot A$
- Prendi la parte frazionaria:  $k \cdot A - \lfloor k \cdot A \rfloor$
- Moltiplica il risultato per  $m$
- Prendi la parte intera
- La chiave  $k$  è mappata in  $\lfloor (m \cdot (k \cdot A - \lfloor k \cdot A \rfloor)) \rfloor$

Esempio:

$$A = (5^{1/2} - 1)/2 = 0.618\dots,$$

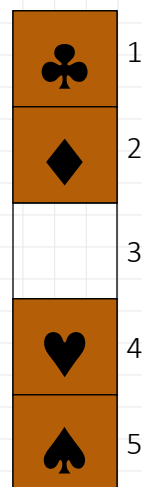
$$k = 123456,$$

$$m = 10000$$

$$h(123456) = \text{conti conti conti} = 41$$

## Risoluzione collisioni: open addressing

$$h(\spadesuit, 2) = 5$$



Vittorio Maniezzo - Università di Bologna

29

29

## Open addressing

- Nessun puntatore: spazio risparmiato!
- $\alpha \leq 1$  sempre. Nessuna lista per gestire le collisioni
- Hash function più complessa:  $\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$  deve essere una permutazione di  $\langle 1, \dots, m \rangle$

$h(k,i)$  = posizione della tabella in cui inserire la chiave  $k$  quando tutte le posizioni  $h(k,0), \dots, h(k,i-1)$  sono già occupate.

Vittorio Maniezzo - Università di Bologna

30

30

## Open addressing, uniform hashing

Se gestiamo le collisioni con il metodo open addressing, la funzione hash restituisce una permutazione degli indici  $\langle 1, \dots, m \rangle$ .

Invece di simple uniform hashing parliamo di **uniform hashing**.

Uniform hashing: tutte le permutazioni devono apparire con la stessa probabilità

## Open addressing, inserimento

```
Hash-insert(T,k)
  i=0
  repeat j=h(k,i)
    if T[j]=nil
      then T[j]=k
      return j
    else i=i+1
  until i=m
  error "hash table overflow"
```

## Open addressing, ricerca

Hash-search( $T, k$ )

$i=0$

repeat  $j=h(k, i)$

if  $T[j]=k$

then return  $j$

else  $i=i+1$

until  $(T[j]==\text{nil})$  or  $(i==m)$

return nil

## Open addressing, cancellazione

Hash-delete( $T, k$ )

$i=\text{Hash-search}(T, k)$

if  $i \neq \text{nil}$  then  $T[i]=\text{nil}$

**NON FUNZIONA**



## Open addressing, cancellazione

1	3
2	7
3	8
4	

Inseriamo 6

$h(6,0)=1$   
 $h(6,1)=2$   
 $h(6,2)=3$   
 $h(6,3)=4$

1	3
2	7
3	8
4	6

Cancelliamo 8

1	3
2	7
3	
4	6

Ricerchiamo 6

Risposta:  
6 non c'è

Vittorio Maniezzo - Università di Bologna

35

35

## Open addressing, cancellazione

### Esercizio:

Modificare Hash-search e Hash-delete per risolvere il problema illustrato nel lucido precedente.

Si può usare un carattere con il quale contrassegnare gli elementi cancellati.

1	3
2	7
3	D
4	6

Vittorio Maniezzo - Università di Bologna

36

36

## Open addressing, linear probing

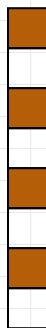
Sia  $h'$  una funzione hash "ordinaria".

Definiamo  $h(k,i) = (h'(k) + i) \bmod m$

Esempio di linear probing:  $m=5, k=3, h'(3)=4$

$$\left. \begin{array}{l} h(k,0) = 4 \\ h(k,1) = 0 \\ h(k,2) = 1 \\ h(k,3) = 2 \\ h(k,4) = 3 \end{array} \right\} = \text{probe}$$

## Linear probing, primary clustering



Tempo medio di  
accesso per una  
ricerca  
senza successo: 1.5  
Perche'?



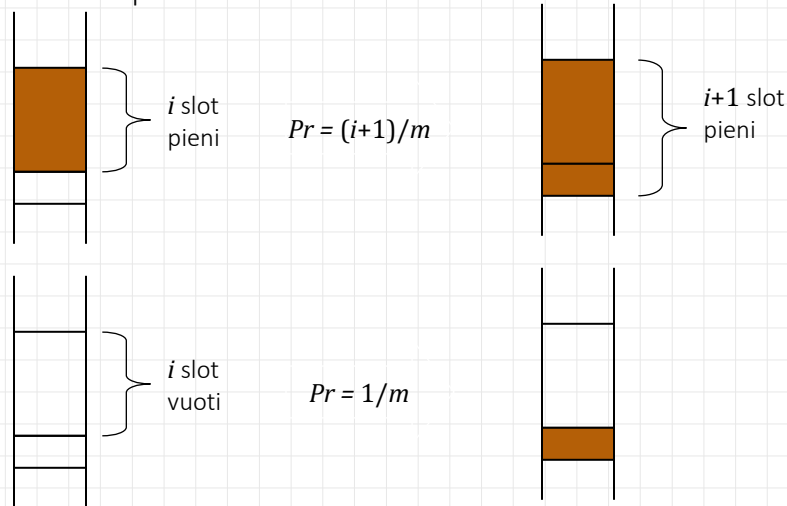
Tempo medio di  
accesso per una ricerca  
senza successo: 2.5  
Perche'?



Clustering primario

## Linear probing, primary clustering

Usando linear probing il clustering primario si forma con alta probabilità.



Vittorio Maniezzo - Università di Bologna

39

39

## Quadratic probing

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$$

con  $c_2 \neq 0$

Cosa si può dire sul clustering primario ?

Vittorio Maniezzo - Università di Bologna

40

40

## Double hashing

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$

Cosa succede se  $\text{MCD}(m, h_2(k)) = d > 1$  ?

Es. se  $m$  pari e  $h_2(k) = 2$ .

(conviene sempre scegliere per  $m$  un numero primo)

## Double hashing, esempio

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod (m - 2))$$

$$m = 13, h_1(k) = k \bmod 13, h_2(k) = 1 + (k \bmod 11)$$

$k$	$h_1(k)$	$h_2(k)$	$h(k,i)$
18	5	8	5
41	2	9	2
22	9	1	9
44	5	1	5,6
59	7	5	7
32	6	11	6,4
31	5	10	5,2,7,11
73	8	8	8

## Open addressing, ricerca

### Teorema:

Data una hash table con open addressing e load factor  $\alpha = n/m < 1$ , la **lunghezza media di una "probe" in una ricerca senza successo** è  $1/(1 - \alpha)$ .

(Ipotesi: uniform hashing)

$$\alpha = (m-1)/m \quad \Rightarrow \quad 1/(1 - \alpha) = m$$

(valore massimo di  $\alpha$ )

$$\alpha = 1/m \quad \Rightarrow \quad 1/(1 - \alpha) = m/(m-1)$$

(valore minimo di  $\alpha$ )

$$\alpha = 1/2 \quad \Rightarrow \quad 1/(1 - \alpha) = 2$$

Vittorio Maniezzo - Università di Bologna

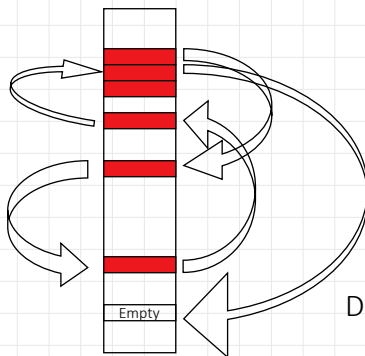
43

43

## Open addressing, ricerca

### Dimostrazione:

IDEA: cosa succede quando facciamo una ricerca senza successo ???



$X$  = lunghezza probe =  
quante volte devo calcolare  
 $h(k, i)$  prima di trovare uno slot  
vuoto = elemento non trovato

Dobbiamo valutare  $E[X]$  = media di  $X$

Vittorio Maniezzo - Università di Bologna

44

44

## Open addressing, ricerca

Lemma:

$X$  **variabile aleatoria** discreta  
(prob. di trovare i pos. occupate prima di una libera,  $p_i = p(X=i)$ )

$X =$

$0 \rightarrow p_0$   
 $1 \rightarrow p_1$   
.....  
 $i \rightarrow p_i$   
.....

$$E[X] = \sum_{i=0}^{\infty} i p_i \quad (\text{nota: } p_i = 0 \quad \forall i > n)$$

$$= \sum_{i=0}^{\infty} i p(X=i)$$

$$= \sum_{i=0}^{\infty} i (p(X \geq i) - p(X \geq i+1)) = 1P_1 - 1P_2 + 2P_2 - 2P_3 + 3P_3 \dots$$

$$= \sum_{i=1}^{\infty} p(X \geq i) \quad (\text{nota: per } i=0 \text{ gli addendo nullo})$$

(continua ...)

Vittorio Maniezzo - Università di Bologna

45

45

## Open addressing, ricerca

$$E[X] = \sum_{i=1}^{\infty} p(X \geq i)$$

$$\leq \sum_{i=1}^{\infty} (n/m)^i$$

$$\leq \sum_{i=1}^{\infty} \alpha^i$$

Costo per la ricerca:

costo pos. libera +  
costo pos. occupate =

$$1 + E[X] \leq$$

$$1 + \sum_{i=1}^{\infty} \alpha^i =$$

$$1 + \alpha + \alpha^2 + \alpha^3 + \dots \quad (\alpha < 1)$$

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

□

$$q_i = p(X \geq i)$$

$$q_1 = n/m$$

$$q_2 (1 \text{ occupata}) = n/m \cdot n-1/m-1 < (n/m)^2$$

$$q_i (i-1 \text{ occupate}) = n/m \cdot n-1/m-1 \cdot \dots \cdot n-(i-1)/m-(i-1) < (n/m)^i$$

Assunte  
indipendenti

Vittorio Maniezzo - Università di Bologna

46

46

## Open addressing, inserimento

### Teorema:

Data una hash table con open addressing e load factor  $\alpha = n/m < 1$ , la lunghezza media di una "probe" è  $1/(1-\alpha)$ .

(Ipotesi: uniform hashing)

### Dimostrazione:

Nota:  $\alpha$  deve essere  $< 1$ .

Per inserire un elemento abbiamo bisogno di determinare la posizione nella tabella dove inserirlo.

Ricerca: costo  $1/(1-\alpha)$ .

Per inserire nella tabella nella posizione appena determinata:  $\Theta(1)$ .

Se la tabella è piena al 50% , ci aspettiamo 2 probe

Se la tabella è piena al 90% , ci aspettiamo 10 probe

## Hashing per crittografia

Hashing fondamentali anche per crittografia. Usate ad esempio per:

- **Verifica dell'integrità dei dati:** hash genera rappresentazioni univoche di dimensioni fisse dei dati in ingresso. Confrontando i valori hash prima e dopo la trasmissione o la memorizzazione dei dati, è possibile verificare se i dati sono stati alterati o manomessi.
- **Memorizzazione delle password:** La memorizzazione delle password in chiaro è altamente insicura, ma possono essere sottoposte a hashing e vengono memorizzati solo i valori di hash. Quando un utente accede, la sua password viene sottoposta a hash e confrontata con l'hash memorizzato. Se corrispondono, la password viene accettata. Anche se gli hash memorizzati vengono compromessi, le password effettive non vengono rivelate direttamente.