

C++: ordered  $\leftarrow \begin{matrix} \text{map} \\ \text{set} \end{matrix}$  unordered  $\leftarrow \begin{matrix} \text{map} \\ \text{set} \end{matrix}$  & non include il costo di ricerca

Strutture di dati per il DIZIONARIO	Appartenente (RICERCA)	Inserimento	Cancellazione * (siamo già lì)
array non ordinato	$O(n)$	$O(1)$	$O(1)$
array ordinato	$O(\lg n)$	$O(n)$	$O(n)$ [logico]
liste non ordinate	$O(n)$	$O(1)$	$O(n)$   $O(1)$ conoscendo il prec. <del>For</del>
liste ordinate	$O(n)$	$O(1)$ siamo già lì	// vedo sopra
albero binario di ricerca	$O(h)$	$O(h)$	$O(h)$ $1 \leq h \leq n-1$ ordine casuale ok
albero AVL come sopra ma $h = O(\lg n)$	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$   $O(1)$ se logico
tabella HASH (non è ordinata)	$O(1)$ medio	$O(1)$ medio	$O(1)$ medio

→ prossima lezione  $h = \text{altezza}$

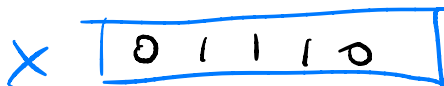
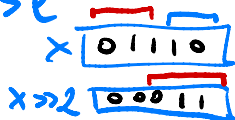
HASH chiavi possono essere "macinate"

↳ non usando più soltanto confronti, non vale più il limite inferiore di  $\Omega(\log n)$  applicato alla ricerca binaria

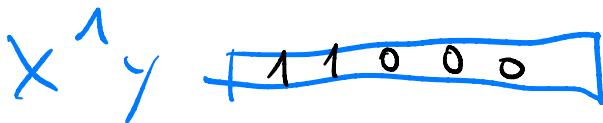
Modello:  $U$  = universo delle chiavi possibili, per esempio  $U = [2^w]$   $w = 64 \text{ bit}$   
oltre ai confronti possiamo manipolare i bit di  $x \in U$

$x+y$ ,  $x \wedge y$ ,  $x \gg l$

↑  
OR esclusivo



$x/2^2$



$$(x \wedge y) \wedge x = y$$

funzione hash :  $h: U \rightarrow [m]$  ,  $m$  è scelto da noi

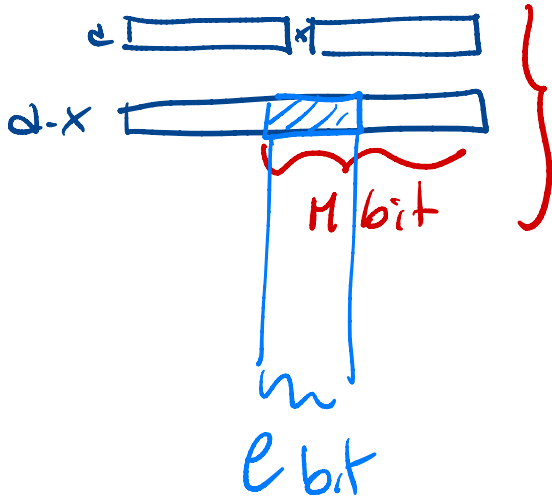
$h(x) = x \% m$  esempio semplice

$h(x) = ((a \cdot x + b) \% p) \% m$  hash universale  $p$  primo  $\in [m+1, 2m]$

$a, b \in \mathbb{Z}_p$  ,  $a \neq 0$  scelti in modo random, uniforme e indipendente

$h(x) = ((a \cdot x) \% 2^H) / 2^{H-l}$  quando  $m = 2^l$

$a$  dispari



$a * x \gg H - l$   $\odot x$   
C++

Per chiavi lunghe  $X$ :

$$X = x_1 \cdot x_2 \cdot \dots \cdot x_s$$

$$h'(x) = h(x_1) \wedge h(x_2) \wedge \dots \wedge h(x_s)$$

FOLDING

$|x_i|$  = quella su cui potete applicare  
le  $h()$  precedenti

```
uint64_t myhash(char *x, int64_t len) {  
    auto y = (uint64_t *) x;  
    uint64_t hash = 0;
```

```
    for (i = 0; i < len/8; i++)
```

```
        // hash = hash ^
```

```
        hash ^= (2 * y[i] >> 64 - 6);
```

```
    return hash;
```

$h(y[i])$  \*

//  $y[0]$  = ai primi 8 byte di  $x$



## Rolling hash

$\alpha$  = número primo

$$h'(x) = \left( \sum_{i=1}^s h(x_i) \cdot \sigma^{i-1} \right) \% m$$

$$1357 = 1 \cdot 10^3 + 3 \cdot 10^2 + 5 \cdot 10 + 7 \cdot 1$$

$$1 \underbrace{351} \rightarrow \underbrace{3572}$$

$$(1357 - 1 \cdot 10^3) \times 10 + 2$$

$$P(x) = \sum_{i=0}^d a_i \cdot x^i \quad \left| \quad \begin{array}{l} y = a_d \\ \text{for } i = 1 \dots d : \\ \quad y = (y \cdot x) + a_{d-i} \end{array} \right.$$

A C A

$$x = x_1 \ x_2 \ x_3$$
$$h(x_1) \quad h(x_2) \quad h(x_3)$$

65 67 65

$A = 65$

$B = 66$

$$C = 67$$
$$0 = 131$$
$$1 = 257$$

$$h'(x) = \underbrace{65 \cdot 1 + 67 \cdot 257 + 65 \cdot 257^2}_{\text{m}} \% m$$

ACA  $\rightarrow$  CAC

$$\overline{h'(CAC)} = (\overbrace{h'(ACA)} - \underbrace{65.257^2}_{\uparrow}) \times \underbrace{257}_{\uparrow} + \underbrace{67}_{\uparrow}$$

$$\boxed{3x^2 - 2x + 4}$$

$$\begin{aligned} y &= 3 \\ y &= 3x - 2 \\ y &= 3x^2 - 2x + 4 \end{aligned}$$

$$a_d X^d + a_{d-1} X^{d-1} + \dots + a_1 X + a_0$$

$$\underbrace{(a_d X^{d-1} + a_{d-1} X^{d-2} + \dots + a_1)}_Y \cdot X + a_0$$

MD5, SHA-1, SHA-256, PGP, - -

funzioni crittografiche difficili da invertire  
(sconsigliate per il dizionario)

funzione hash  $h: U \rightarrow [m]$  ,  $|U| \gg m$

PROBLEMA INEVITABILE: COLLISIONI

$\forall h \exists x, y \in U ; x \neq y \text{ ma } h(x) = h(y)$

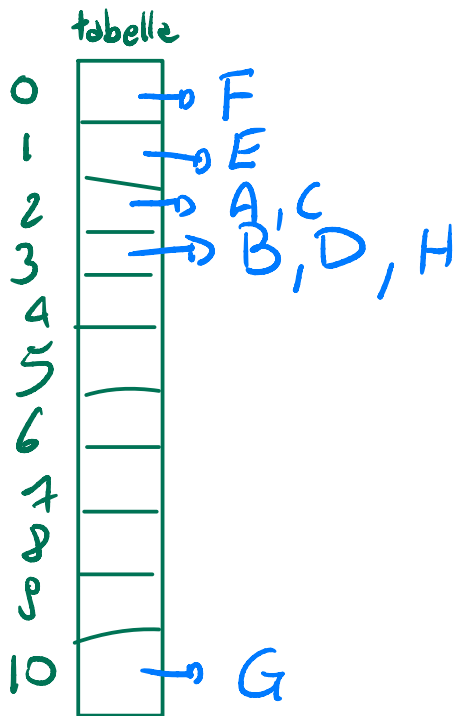
Tabella hash  $\rightarrow$  funzione hash  $h$   
 $\rightarrow$  gestione delle collisioni

LISTE  
DI TRABOCCHI  
(chaining)

INDIRIZZAMENTO  
APERTO  
(open addressing)

(esiste  
hash perfetto)

# LISTE DI TRABOCCO



$x \in S$	B	A	D	C	F	G	E	H	$n = 8$
$h(x)$	3	2	3	2	0	10	1	3	$n = 11$
	.	.	.	.	.	.	.	.	

$$tabella[i] = \{x \in S : i = h(x)\} \quad \text{lista di trabocco}$$

$$\alpha = \text{fattore di carico} = \frac{n}{m} = \text{lunghezza media di } tabella[i]$$

$$\left( \sum_{i=0}^{m-1} |tabella[i]| = n \right)$$

$$\text{caso pessimo : } \max_i |tabella[i]| \leq n$$

$$\text{caso medio} = O(1 + \alpha)$$

↑ calcolo  
↑ scorrimento lista

Per esempio :  $tabella[i] = \text{array non ordinato}$

$\underbrace{ins(x) \text{ in tab.hash}}_{\text{tab. hash}} \Leftrightarrow \underbrace{ins(x) \text{ in } tabella[h(x)]}_{\text{array ordinato}}$

Idea:  $h()$  è un "router" verso l'array non ordinato corrispondente (lista)

tabella hash:  $\text{array}[m]$  of vectors :  $\text{vector} \langle \text{int} \rangle \text{ tab}[m];$

ric(x):  
auto v = tab[h(x)];  
return v.contains(x)

ins(x):  
auto v = tab[h(x)]  
if !ric(x) then v.push-back(x);

del(x):  
esercizio 😊

occorre una buona funzione hash  $h$   
per garantire che  $d = \frac{n}{m}$

$$\Pr(h(x)=h(y) \wedge x \neq y) \sim \frac{1}{m}$$

$$d \sim \frac{1}{2}$$

$$O(1 \pm d) = O(1)$$

# INDIRIZZAMENTO APERTO

$n < m$

$x \in S$	B	A	D	C	F	G	E	H	I	$n=8$
$h(x)$	3	2	3	2	0	10	1	3	4	$n=11$
	.	.	.	.	.	.	.	.	.	

0	1	2	3	4	5	6	7	8	9	10 = m-1
F	E	A	B	D	C	H	I			G

Scansione  
lineare  
(modulo m)

cluster = porzione massimale (non si può estendere ulteriormente, modulo m) di <sup>occupate</sup> posizioni  
per esempio: ric/ius della chiave L t.c.  $h(L) = 10$  scandisce tutto il cluster

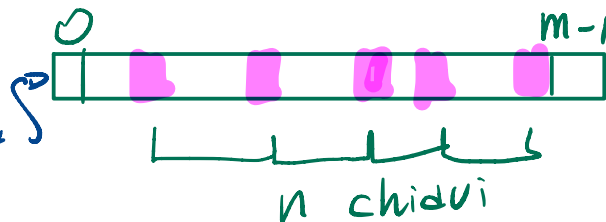
caso pessimo =  $\max |\text{cluster}| \leq n$

oss cluster = unione delle liste di trabocco di alcune chiavi

caso medio =  $O((1-\alpha)^{-1})$  tempo  $\alpha = \frac{n}{m}$  fattore di carico :  $\alpha = \frac{1}{2} \Rightarrow O(2)$  medio

$T(n, m) = \#$  medio di posizioni esaminate (ric) in una tabella di  $m$  posizioni contenente  $n$  chiavi

$\alpha = \frac{n}{m} =$  probabilità di trovare una posizione occupata  
( $h$  è una buona funzione hash)



relazione di ricorrenza:

$$T(n, m) = \begin{cases} 1 & \text{se } n=0 \\ 1 + \alpha \cdot T(n-1, m-1) & \text{altrimenti} \end{cases} \leftarrow 1 \cdot \frac{m-n}{n} + (1 + T(n-1, m-1)) \frac{n}{m}$$

$\uparrow$   
 posizione esaminata

$\underbrace{1 \cdot \frac{m-n}{n}}_{\text{prob. che posizione è libera}} + \underbrace{(1 + T(n-1, m-1))}_{\substack{\text{la chiave esaminata ci dice che ci sono } n-1 \text{ chiavi nelle rimanenti } m-1 \text{ posizioni}}} \underbrace{\frac{n}{m}}_{\text{prob. che posizione è occupata}}$

Per induzione e sostituzione:

$$T(n, m) \leq \frac{m}{m-n} = \frac{1}{1-\alpha} = (1-\alpha)^{-1}$$

$h_1, h_2$

Per chi vuole approfondire: cuckoo hash

ric / conc  
ins

$O(1)$  tempo caso pessimo

$O(1)$  tempo caso medio ammort.