6. HASHING E TABELLE HASH

Un'altra struttura dati di tipo INSIEME sono i DIZIONARI, usati per memorizzare INSIEMI DINAMICI di coppie <chiave, valore>.

Le coppie sono indicizzate in base alla chiave.

Le chiavi appartengono ad un UNIVERSO U, che è TOTALMENTE ORDINATO.

Supportano le seguenti operazioni:

(1) INSERT (key, element)

(2) DELETE (key)

(3) SEARCH (key) ⟶ element

Idealmente, vorremmo che il costo di ognuna di queste 3 operazioni sia $O(1)$; tuttavia, né con array non ordinati né ordinati, né con una lista e né con un Red-Black Tree riusciamo ad ottenere ciò.

Ma, può essere reso possibile tramite l' HASHING.

• FUNZIONI HASH:

Una FUNZIONE HASH h mappa le chiavi dell'UNIVERSO U negli interi $\mathbb{N}$.

$$h : U \longmapsto \mathbb{N}$$

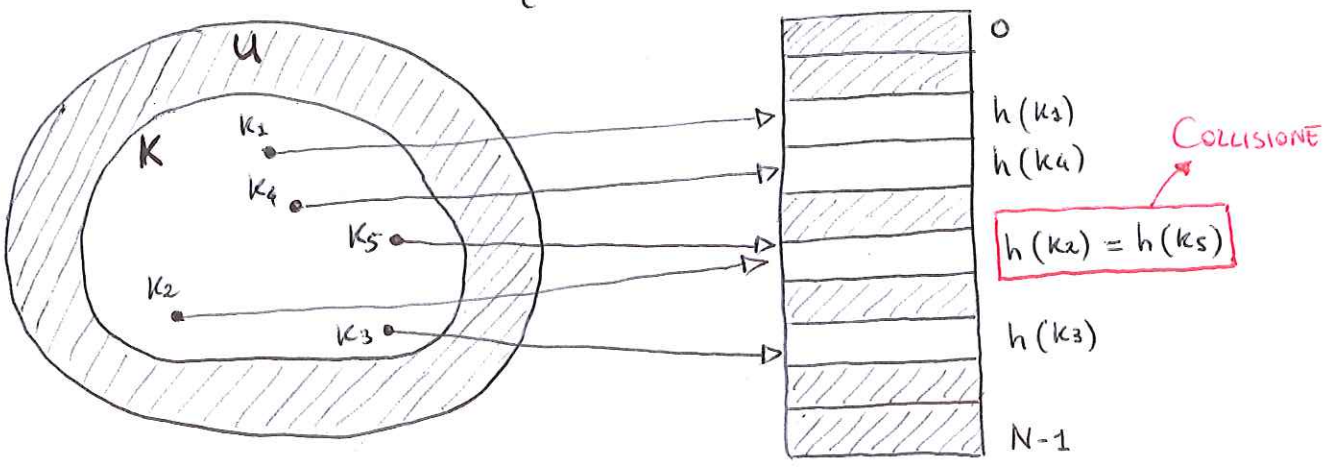
La coppia <chiave, valore> viene memorizzata in un array in POSIZIONE h(chiave).

In altre parole, sia A l'array $\Rightarrow$ A[h(chiave)] ⟶ valore

Tale vettore viene denominato TABELLA HASH.

La tabella hash ha dimensione limitata e prefissata ad N. Le funzioni hash convertono le chiavi dell'universo U in un intero dell' INSIEME DELLE PSEUDOCHIAVI $\{0, \dots, N\}$

$$h : U \longmapsto \{0, 1, \dots, N-1\}$$



Collisioni:

Si verificano quando 2 o più chiavi appartenenti ad U hanno lo stesso valore hash; cioé se: $k_i \neq k_j$, MA $h(k_i) = h(k_j)$.

- **Funzione Hash Banale (Identità): Tabelle ad Accesso Diretto**

Prendiamo come funzione hash la funzione IDENTITÀ: $h(k) = k$.

$\Rightarrow$ La tabella di hash ha dimensione pari alla cardinalità dell'Universo delle chiavi: $N = |U|$.

La chiave k diventa l'indice dell'array del corrispettivo valore. In questo modo si evitano le COLLISIONI, ma per dimensioni già mediamente grandi di U la TABELLA DI HASH esplode $\longrightarrow$ NON EFFICIENTE!

Idealmente vorremmo funzioni hash senza collisioni, cioé perfette.
Una funzione HASH è PERFETTA se è INIETTIVA:

$$\forall k_1, k_2 \in U : k_1 \neq k_2 \Rightarrow h(k_1) \neq h(k_2)$$

Purtroppo però, l'UNIVERSO DELLE CHIAVI è spesso molto VASTO, SPARSO e SCONOSCIUTO, ed è impraticabile ottenere una funzione hash perfette.
Quindi, bisogna cercare di LIMITARE LA PROBABILITÀ DI COLLISIONE.

- **Uniformità Semplice:**

Si cerca di avere una DISTRIBUZIONE UNIFORME delle chiavi negli indici $[0, \dots, N-1]$.

* *Principio di Uniformità Semplice:*

Sia $P(k)$ la probabilità che la chiave k venga inserita nella tabella
Sia $Q(i)$ la probabilità che una chiave finisca nell'elemento i-esimo della tabella

$$\Rightarrow Q(i) = \sum_{k \in U : h(k) = i} P(k)$$

Si ha UNIFORMITÀ SEMPLICE se: $\boxed{\forall i \in [0, \dots, N-1] : Q(i) = \frac{1}{N}}$

! Molto difficile da realizzare, poiché deve essere nota a priori la distribuzione di probabilità P.

$\longrightarrow$ STRADA POCO PRATICABILE!

Come Realizzare una Funzione di Hash

Le chiavi possono essere di qualsiasi tipo → occorre INTERPRETARE la loro RAPPRESENTAZIONE come un NUMERO (di solito binario)

Per esempio, come interpretare le STRINGHE:

- ord(c): valore BINARIO del char c, in qualche codifica (ASCII, UNICODE, ...)
- bin(k): RAPPRESENTAZIONE BINARIA della chiave k, concatenando i valori binari (ord(c)) di ogni char della stringa
- int(b): valore numerico DECIMALE associato al numero binario b
- int(k) = int(bin(k))

• Funzione Hash: Estrazione

L'ESTRAZIONE consiste nel prendere una certa parte di bit (di solito i meno significativi) della rappresentazione binaria della chiave come risultato della funzione hash:

Sia $N = 2^p$, sia b un sottoinsieme di p bit presi da bin(k)

$$\Rightarrow \boxed{h(k) = int(b)}$$

Esempio: $N = 2^{16} = 65536$; sia K = "Alessandro"

Prendiamo i 16 bit meno significativi della RAPPR. BINARIA di "Alessandro"

→ 01110010 01101111 ⇒ h("Alessandro") = int(01110010 01101111) = 29295

* Problema: probabilità di COLLISIONE molto elevata!

• Funzione Hash: XOR

$N = 2^p$; $h(k) = int(b)$, dove b è lo XOR effettuato a gruppi di p bit di bin(k). Può essere necessario il PADDING, ovvero riempire con degli zeri gli spazi vuoti.

* Ma la Probabilità di Collisione è ancora TROPPO ALTA!

Per esempio: h("accettare") = h("Cercarote") = 29696

• Funzione Hash : Metodo della Divisione

Si prende N DISPARI, preferibilmente NUMERO PRIMO e distante da potenze di 2.

$$h(k) = int(b) \mod N$$

→ Resto della divisione per N

ESEMPIO : $N = 571$ ⇒

$h("rome") = 416$
$h("romo") = 523$
$h("orme") = 318$
$h("orme") = 322$

NON VANNO BENE : • $N = 2^p$ → significherebbe utilizzare solo i p bit meno significativi

• $N = 2^p - 1$ → permutazioni di sottostringhe di lunghezza 2^p hanno lo stesso valore di hash.

Come Gestire le Collisioni

Cosa fare in caso di COLLISIONE ?

(1) In caso di INSERIMENTO → trovare una posizione alternative

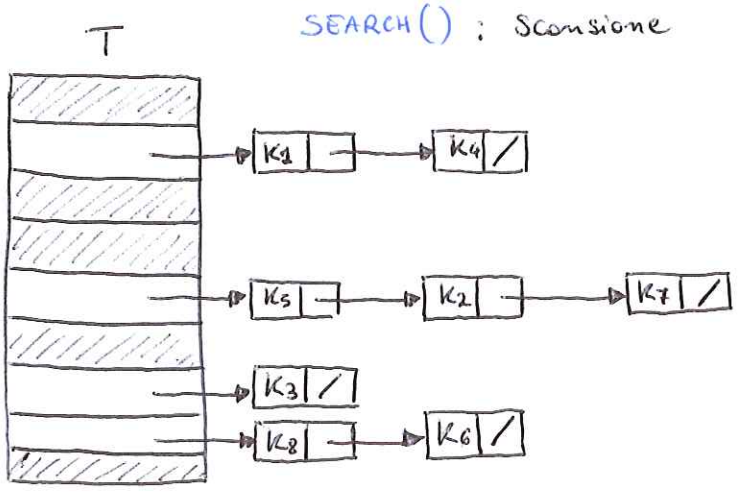
(2) In caso di RICERCA → se non troviamo la chiave cercata bisogna cercarla altrove

Comunque sia, queste operazioni dovrebbero avere nel caso medio un costo $O(1)$

• Liste di Trabocco :

Le chiavi con lo stesso valore hash vengono organizzate in una LISTA SINGOLARMENTE COLLEGATA (oppure ARRAY DINAMICO).

Le operazioni fondamentali sono: INSERT() : inserimento in testa
DELETE() : scansione
SEARCH() : scansione



Analisi Complessità : Liste di Trabocco

Consideriamo il FATTORE DI CARICO $\alpha = \frac{n}{N}$, dove n è il numero di elementi inseriti nella tabella hash. Ci dice la percentuale di riempimento della tabella.

Assumiamo che il COSTO DI CALCOLO di $h(\cdot)$ sia $\Theta(1)$.

- CASO PESSIMO: tutte le chiavi in un'unica lista di trabocco

  INSERT() : $\Theta(1)$, poiché in testa

  DELETE() / SEARCH() : $\Theta(n)$

- CASO MEDIO: Se assumiamo l'hashing con DISTRIBUZIONE UNIFORME delle chiavi, la lunghezza di una lista è $\alpha = \frac{n}{N}$

  INSERT() : $\Theta(1)$

  DELETE() / SEARCH() :
  - se con SUCCESSO : $\Theta(1) + \frac{\alpha}{2}$
  - se con INSUCCESSO : $\Theta(1) + \alpha$

$\Rightarrow$ Se $n = O(N) \Rightarrow \boxed{\alpha = O(1)}$

Con un appropriato dimensionamento della Tabella Hash, è possibile avere nel caso medio costo $O(1)$ per tutte le operazioni.

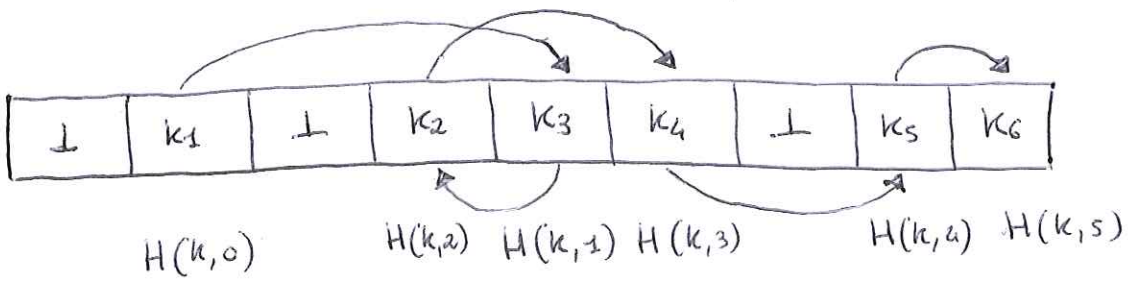
Intuitivamente, di media dev'esserci 1 elemento per lista

Gestione delle Collisioni : Indirizzamento Aperto

Se si vuole evitare l'utilizzo di STRUTTURE DATI collegate come le liste, le collisioni devono essere RISOLTE ALL'INTERNO della TABELLA HASH.

Si utilizza una FUNZIONE HASH ESPANSA $H(k, i)$ che calcola la posizione dell'elemento di chiave k all'i-esimo tentativo.

Per fare ciò si genera una SEQUENZA DI ISPEZIONE della tabella hash; tuttavia, la tabella può andare in overflow per $\alpha = \frac{n}{N} = 1$.



Sequenze di Ispezione

Determinano le sequenze con cui vengono "ispezionate" le posizioni successive delle tabelle hash;

- La funzione $H(k,i)$ DEVE PERMETTERE di ISPEZIONARE TUTTE LE POSIZIONI possibili della tabella hash;

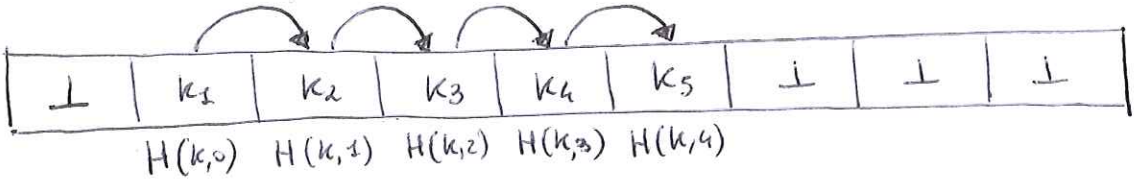
Ci sono tecniche differenti:

- ▶ SCANSIONE LINEARE;
- ▶ SCANSIONE QUADRATICA;
- ▶ HASHING DOPPIO.

- **Scansione Lineare:**

Si cerca nelle posizioni adiacenti ad $h(k)$ linearmente:

$$H(k,i) = h(k) + c \cdot i \mod N$$



Si ha il problema dell' AGGLOMERAZIONE PRIMARIA (primary clustering):
si generano LUNGHE SOTTOSEQUENZE OCCUPATE ed i tempi medi per le operazioni che richiedono ispezione crescono molto.

- **Scansione Quadratica:**

Si cerca quadraticamente nelle posizioni adiacenti ad $h(k)$:

$$H(k,i) = h(k) + c \cdot i^2 \mod N$$

Sono possibili al più N sequenze di ispezione.

Si ha il problema dell' AGGLOMERAZIONE SECONDARIA (secondary clustering):
se $h(k_1) = h(k_2)$, le sequenze di ispezione per le 2 chiavi sono le stesse.

- **HASHING DOPPIO:**

Una funzione hash per determinare la posizione iniziale e un'altra funzione hash per generare la sequenza d'ispezione:

$$H(k,i) = h_1(k) + i \cdot h_2(k) \mod N$$

Sono possibili al più N^2 sequenze di ispezione.

BISOGNA SCEGLIERE $h_2(\cdot)$ in modo da consentire UN'ISPEZIONE COMPLETA:

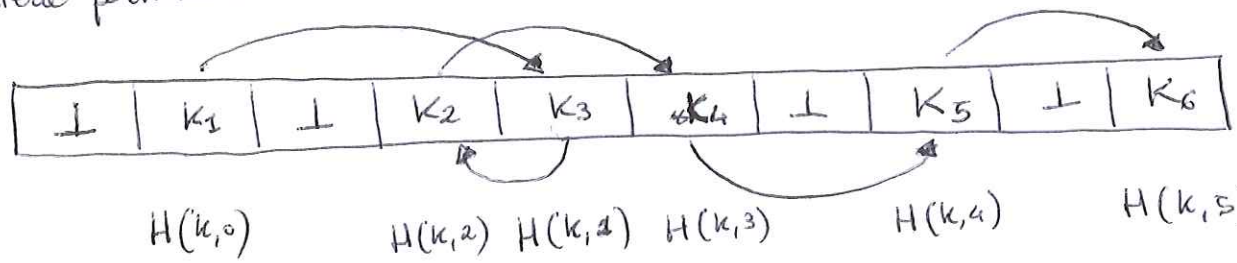
- Si ottiene solo se $h_2(k)$ ed N sono COPRIMI:

(1) $N = 2^p$ e $h_2(k)$ genere solo numeri dispari

(2) N PRIMO e $h_2 : U \longmapsto \{0, \dots, N-1\}$

CANCELLAZIONE NELL'INDIRIZZAMENTO APERTO:

Nel caso di indirizzamento aperto, la cancellazione deve essere eseguita in maniera particolare.

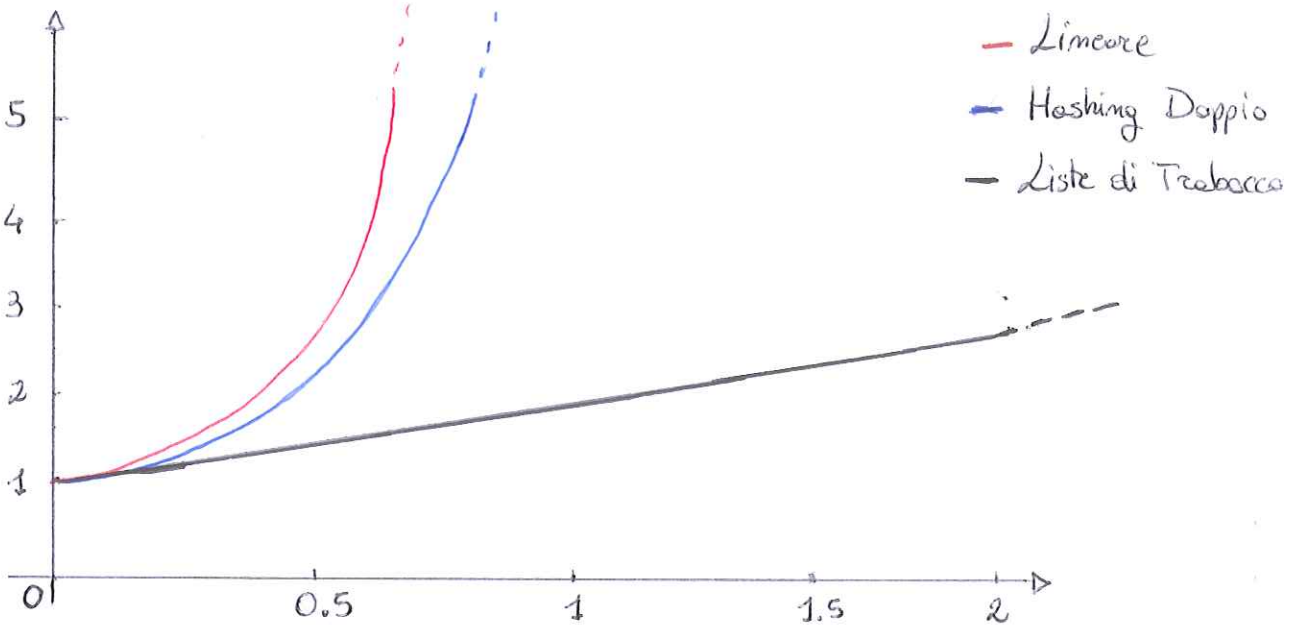


Se facessi DELETE(k_4) $\Rightarrow$ Al posto di K_4 avrei ⊥

Poi, se eseguissi SEARCH(k_6) $\Rightarrow$ La sequenza di ispezione mi porterebbe prima nella cella che conteneva in precedenza K_4 e, trovando ⊥, verrebbe concluso che K_6 non c'è.

- Si introduce l'elemento speciale (DEL), per indicare che un elemento è stato cancellato. In caso di ispezione, DEL viene considerato come un elemento qualsiasi che però non deve mai essere restituito.

COMPLESSITÀ:



Le TABELLE HASH più efficienti sono quelle con l'utilizzo di liste collegate per la gestione delle collisioni, cioè le LISTE DI TRABOCCO !

TABELLA HASH ADATTATIVA

A prescindere dalle strategie per gestire le COLLISIONI, è chiaro che è conveniente non far aumentare il FATTORE DI CARICO $\alpha \left(= \frac{n}{N}\right)$; l'unico modo per farlo è → AUMENTARE N.

Anziché utilizzare come Tabella Hash un semplice array, si può far uso di un ARRAY DINAMICO:

Sopra una certa soglia del fattore di carico, si RADDOPPIANO le dimensioni della tabella hash e si REINSERISCONO TUTTE LE CHIAVI. Nel caso di indirizzamento aperto, si possono rimuovere anche tutti i DEL.

→ COSTO: $O(1)$ AMMORTIZZATO