Esempio di codice ciclico.

Costruire un codice ciclico binario di tipo $C(7,4)$ avente polinomio generatore $m(x) = x^3 + x + 1$.

$\rightarrow \mathbb{F}_2^7$ $GF(2)$

$n = 7$, $k = 4$

$r = n - k = 3$

$$G = \begin{matrix} k=4 \\ \{ \end{matrix} \left[\begin{array}{ccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \quad (n=7)$$

$$H = \begin{matrix} n-k \\ = \\ 3 \end{matrix} \left[\begin{array}{ccccccc} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \quad (n)$$

$m(x) = 1 \cdot x^0 + 1 \cdot x^1 + 0 \cdot x^2 + 1 \cdot x^3 + 0 \cdot x^4 + 0 \cdot x^5 + 0 \cdot x^6$

$\pmod{(x^7 - 1)}$

$x^7 \equiv 1$

$1101000 \in C$

$1\ 0\ 0\ 0\ 1\ 1\ 0$ (+)

$h(x)$: $m(x)\,h(x) \equiv 0 \equiv x^7 - 1$

Dividere $x^7 - 1$ per $m(x)$: $x^7 - 1 \;:\; x^3 + x + 1$

$m(x)$ $1\ 1\ 0\ 1\ 0\ 0\ 0$

$x\,m(x)$ $1\ 0\ 0\ 0\ 1\ 1\ 0$

$1\ 1\ 1\ 0\ 1$

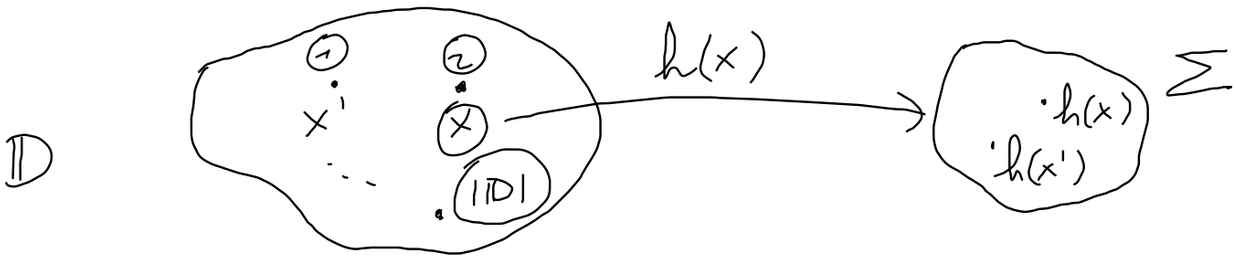
$1\ 0\ 1\ 1\ 1$

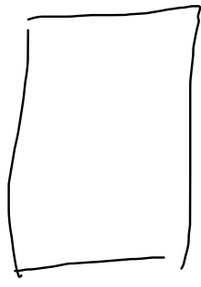
$h(x) = x^4 + x^2 + x + 1$

$h^*(x) = 1 + x^2 + x^3 + x^4$

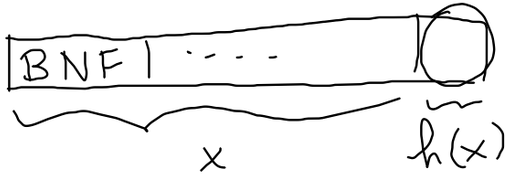
$m(x)\,h(x) = \cancel{1 + x} + \cancel{x^2} + \cancel{x^4} + \cancel{x} + \cancel{x^2} + \cancel{x^3} + \cancel{x^5} + \cancel{x^3} + \cancel{x^4} + \cancel{x^5} + \cancel{x^7} \equiv 0$

Codici hash ; funzione hash





Codice fiscale — carattere di controllo

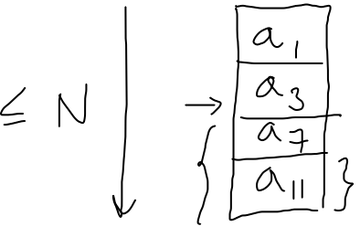


Problema del dizionario

Struttura dati per memorizzare fino a N elementi di un dominio D $a_1, a_2, \ldots, a_N$ e che supporti un insieme di operazioni:

- inserisci(x), $x \in D$: aggiungi x agli elementi memorizzati
- rimuovi(x), $x \in D$: rimuovi x dagli elementi memorizzati
- ricerca(x), $x \in D$: esiste uno degli elementi memorizzati $a_1, a_2, \ldots$ $\exists i = 1 \ldots N : a_i = x$?

Soluzione 1: mantenere un array ordinato con gli elementi inseriti fino a quel momento.

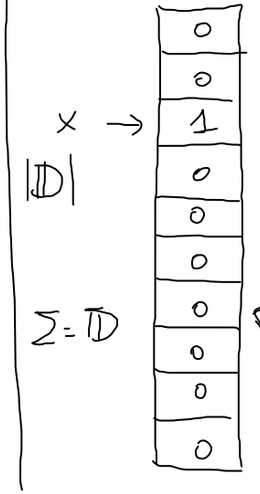


- ricerca(x): applicare un metodo di ricerca binaria; ☺ richiede tempo $O(\log N)$
- inserimento(x): richiede tempo $O(N)$ ☹
- rimuovi(x): " " " ☹

$(a_1 < a_3 < a_7 < a_{11})$ Lo spazio di memoria utilizzato è $O(N)$. ☺

Cerca a_8

Soluzione 2



- ricerca(x): esaminare la cella corrispondente; tempo costante
- inserimento(x): scrivi 1 nella cella corrispondente a x
- rimuovi(x): scrivi 0 " " " " x

→ Tempo costante; lo spazio di memoria utilizzato è $O(|\mathbb{D}|)$.

cella j: bit 1 ⇔ è stato memorizzato il j-esimo elemento del dominio

Def. Una famiglia di funzioni hash $\{h_1, h_2, \ldots, h_m\} = \mathcal{H}$

dove $h_i : \mathbb{D} \longrightarrow \Sigma$ (per ogni $i = 1 \ldots m$) è

detta ε – universale se: $(0 < \varepsilon \leq 1)$

$$\Pr_i [\, h_i(x) = h_i(y) \,] \leq \varepsilon \qquad \forall x, y \in \mathbb{D} \text{ con } x \neq y$$

↖ rispetto alla scelta casuale di $i = 1 \ldots m$

Soluzione 3: simile alla soluzione 2 ma invece di mantenere nell'array

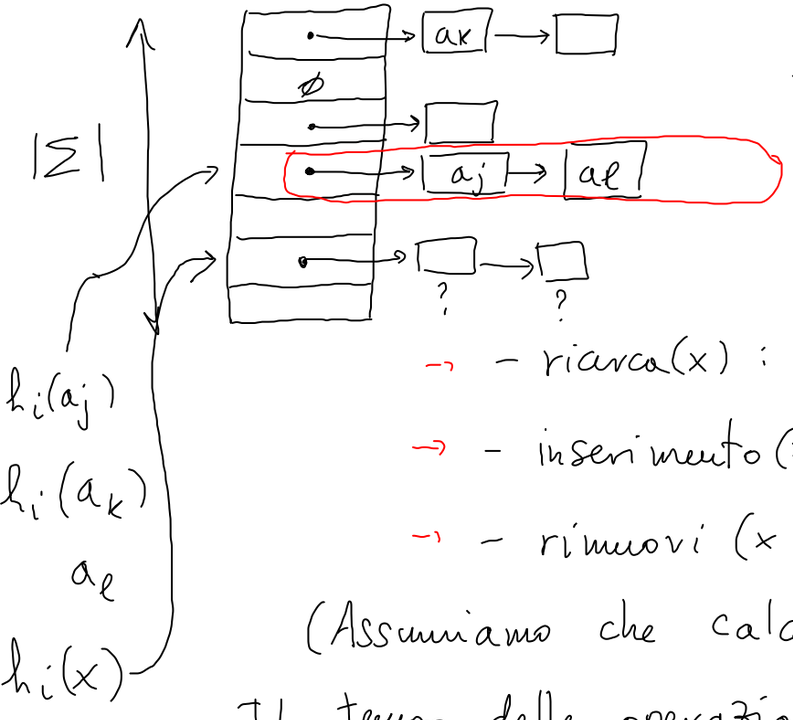
una cella per ogni elemento di $\mathbb{D}$, ho una cella per ogni elemento di Σ.

Sia $\mathcal{H} = \{h_1, \dots, h_m\}$ una famiglia hash ε-universale.

Costruisco un array di liste collegate, una per ogni elemento di Σ.

Scegli una funzione h_i a caso in $\mathcal{H}$

- Per ogni elemento $a_j \in \mathcal{D}$ $(j = 1 \dots N)$ da memorizzare, calcolare $h_i(a_j) \in \Sigma$ e aggiungi a_j alla lista corrispondente (quella di indice $h_i(a_j)$).



→ - ricerca(x): calcola $h_i(x)$ e cerca x nella lista corrispondente

→ - inserimento(x): calcola $h_i(x)$ e inserisci x nella lista corrispondente

→ - rimuovi(x): calcola $h_i(x)$ e cancella x dalla lista corrispondente

(Assumiamo che calcolare $h_i(x)$ abbia un costo costante)

Il tempo delle operazioni è proporzionale alla lunghezza della lista corrispondente a $h_i(x)$.

Spazio complessivo utilizzato: proporzionale a $|\Sigma| + N$ ↖ num. max di elementi inseriti

Prop. Sia $\mathcal{H} = \{h_1, \dots, h_m\}$ una famiglia hash ε-universale.
Allora per ogni $x, a_1, a_2, \dots, a_N \in \mathbb{D}$ distinti,

$$\mathbb{E}_{i=1\dots m}\left[\,|\{a_j : h_i(x) = h_i(a_j)\}|\,\right] \le \varepsilon N.$$

(i è scelto a caso $\rightarrow$ $i = 1 \dots m$; $|\{a_j : h_i(x) = h_i(a_j)\}|$ = # collisioni con x)

Dim. Fissiamo j, $1 \le j \le N$. Per definizione di famiglia hash ε-universale

j fissato $\rightarrow$ $\Pr_i[h_i(x) = h_i(a_j)] \le \varepsilon$.

$\mathbb{E}(B) = 1 \cdot \Pr(B=1) + 0 \cdot \Pr(B=0) = \Pr(B=1)$ se $B \in \{0,1\}$

Ma $$\mathbb{E}_i[\#\text{collisioni con } x] = \mathbb{E}_i \sum_{j=1}^{N} \begin{cases} 1 & \text{se } h_i(x) = h_i(a_j) \\ 0 & \text{se } h_i(x) \neq h_i(a_j) \end{cases} = \sum_{j=1}^{N} \Pr_i[h_i(x) = h_i(a_j)] \le \sum_{j=1}^{N} \varepsilon = \varepsilon N$$

($j = 1 \dots N$; $\Pr_i[h_i(x) = h_i(a_j)] \le \varepsilon$)

QED.

Conclusione: se esiste una famiglia hash con dominio $\mathbb{D}$, codominio Σ, ε-universale, e se $\Sigma = O(N)$, allora esiste una struttura dati (randomizzata) che dati N elementi $a_1, a_2, \ldots, a_N \in \mathbb{D}$ supporta ricerca, inserimento e cancellazione in tempo atteso costante e in spazio proporzionale a N ($|\Sigma| + N = O(N)$).

$\frac{N + \varepsilon N}{\Sigma} = 1 + \varepsilon$

$h : \mathbb{D} \to \Sigma$

Data $\mathcal{H} = \{h_1, \ldots, h_m\}$, posso considerare il codice associato:

$$C_{\mathcal{H}} : \mathbb{D} \to \Sigma^n \qquad x \longmapsto (h_1(x), h_2(x), \ldots, h_m(x)) \in \Sigma^n$$

($x \in \mathbb{D}$; $h_1(x), h_2(x), \ldots, h_m(x) \in \Sigma$)

Viceversa dato un codice C sull'alfabeto Σ, con lungh. delle parole di codice pari a n, definisco $h_i(x) \triangleq [\varphi_C(x)]_i$ ← componente i-esima

($h_i(x) \in \Sigma$; φ_C: funz. di codifica; $\varphi_C(x) \in \Sigma^n$)

Prop. Se $\mathcal{H}$ è una famiglia ε-universale, allora il corrispondente $C_{\mathcal{H}}$ ha distanza minima $\geq (1-\varepsilon)n$.

Viceversa se C è un codice con parole di lunghezza n e distanza d_{min}, allora la corrispondente famiglia hash $\mathcal{H}_C$ è una famiglia $\underbrace{(1-\lambda)}_{\varepsilon}$-universale.

con $\lambda = \frac{d_{min}}{n}$

→ Progettare una buona famiglia di funzioni hash equivale a progettare un buon codice correttore di errore.