Codifica binaria delle informazioni

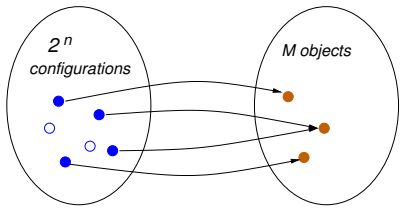
M. Favalli

Engineering Department in Ferrara

Sommario

Codici binari

- Come codice binario (blockcode) si intende una funzione $f : \{0,1\}^n \rightarrow J$ dove J rappresenta un insieme di M informazioni
  - alcune configurazioni possono restare inutilizzate
  - la stessa informazione puó essere rappresentata da piú di una configurazione
  - l'utilizzo di n bit é una restrizione in quanto esistono codici in cui a informazioni diverse possono essere associate parole di lunghezza diversa (es. codici di Huffman)



Selezione del valore di n e del tipo di codice

- Il numero di configurazioni binarie deve essere maggiore o uguale al numero di informazioni (M) da codificare

$$2^n \geq M \tag{1}$$

- Risolvendo tale disequazione rispetto n si ottiene

$$n \geq \lceil \log_2 M \rceil \tag{2}$$

- Fissato n il numero di codici possibili C é dato dal numero di disposizioni delle configurazioni binarie 2^n prese a M alla volta

$$C = 2^n(2^n - 1)....(2^n - M + 1) = \frac{2^n!}{(2^n - M)!} \tag{3}$$

Dimostrazione

- Il numero di possibili codici puó essere calcolato considerando:
  - il numero di diverse parole di codice effettivamente utilizzate che ́dato dal numero di combinazioni di 2^n elementi presi a M alla volta

$$\binom{2^n}{M}$$

  - per ciascuna di esse bisogna considerare le possibili permutazioni delle M parole effettivamente utilizzate ($M!$)
- Quindi:

$$C = \binom{2^n}{M} M! = \frac{2^n!}{(2^n - M)!M!} M! = \frac{2^n!}{(2^n - M)!}$$

Esempio

- Si consideri il problema di codificare le cifre decimali $\{0, ..., 9\}$ utilizzando il minimo numero di bit
- Da Eq. 2 si ha $n = \lceil \log_2 10 \rceil = 4$, cui corrispondono $C = \frac{2^4!}{(2^4-10)!} = 29059430400$ possibili codici
- Si decide di codificare ciascuna cifra con l'equivalente binario del numero corrispondente
- Codice BCD (Binary Coded Decimal)

cifra dec.	codice BCD	cifra dec.	codice BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Esercizi

Esercizi per verificare la comprensione dell'argomento

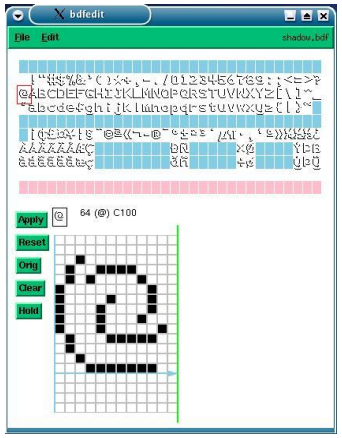
1. Si determini il numero minimo di bit necessario per codificare i tasti di una semplice calcolatrice `0,1,....,9,+,-,*,/,=`
2. Si determini il numero minimo di bit necessario per codificare `red, green, blue` e il numero di codici possibili

Esempi di codici

- Nella codifica dell'informazione all'interno di CPU e altri sistemi digitali, si utilizzano le seguenti rappresentazioni (codifiche):
  - complemento a 2 per gli interi
  - virgola mobile o fissa per i reali
  - ASCII (8 bit) o ISO (16 bit) per i caratteri
- Al di fuori di questo ambito si utilizzano diversi tipi di codice che rispondono ad esigenze di efficienza, affidabilitá e semplicitá di conversione e di utilizzo
- L'operazione di passaggio da un codice all'altro viene detta di conversione (o transcodifica)

Simboli grafici

- La rappresentazione dei caratteri all'interno di una CPU non é chiaramente adatta per l'output
- Un carattere o simbolo viene rappresentato con una matrice $w \times l$ di bit il cui valore 0/1 determina lo stato bianco/nero di un pixel su display o la stampa di un dot su carta
- Esempio di editor di matrici per X11: matrice corrispondente al carattere @



Codifica delle informazioni di tipo numerico

- In un sistema di calcolo la rappresentazione di informazioni numeriche riveste particolare importanza
- I codici utilizzati per rappresentarle determinano in parte la complessitá e le prestazioni delle unitá che elaborano i dati di tipo numerico
- Conviene notare la distinzione fra numeri e loro rappresentazione

Codici numerici posizionali

- Codici posizionali (numeri razionali positivi):
  - $\mathcal{A} = (a_1, a_2, ..., a_b)$: insieme ordinato di simboli;
  - $b = |\mathcal{A}|$: base;
- Parola di codice (lunghezza $n + k$):

$$X = (\overbrace{\alpha_{n-1}\alpha_{n-2}....\alpha_1\alpha_0}^{\textit{parte intera}}, \overbrace{\alpha_{-1}....\alpha_{-k}}^{\textit{parte frazionaria}})$$

- Valore numerico (informazione):

$$v(X) = \alpha_{n-1}b^{n-1} + \alpha_{n-2}b^{n-2} + ... + \alpha_1 b^1 + \alpha_0 b^0 + \alpha_{-1}b^{-1} + ... + \alpha_{-k}b^{-k}$$

Esempi di basi

Codice	Base	Alfabeto
binario	2	$\{0, 1\}$
ottale	8	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
decimale	10	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
esadecimale	16	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Notazione, poiché alcuni simboli sono condivisi, occorre denotare i numeri con riferimento alla base. Esempio, $10_2 = (2_{10})$, 10_{10}, ...

Rappresentazione dei numeri naturali

- Ciascuna parola di codice abbia n simboli
- Non si ha la parte frazionaria $X = (\alpha_{n-1}, ..., \alpha_0)$

$$v(X) = \sum_{i=0}^{n-1} \alpha_i b^i \tag{4}$$

- Si possono rappresentare numeri naturali $X \in [0, b^n - 1] \subset \mathcal{N}$

Conversione di base

- Dato X in base b, si vuole Y in base c in modo che $v(X) = v(Y)$
- Si puó utilizzare l'eq. 4, in cui si convertono i coefficienti di X e b nella nuova base c e si effettuano somme e prodotti nella nuova base
- Esempio: si vuole convertire 10100_2 in base 10, $1_{10} \cdot 2^4_{10} + 0_{10} \cdot 2^3_{10} + 1_{10} \cdot 2^2_{10} + 0_{10} \cdot 2^1_{10} + 0_{10} \cdot 2^0_{10} = 20_{10}$
- Per motivi di praticitá, questo algoritmo ha senso quando la nuova base é 10

Conversione di base per divisioni ripetute

- Per la conversione da base 10 verso altre basi si puó utilizzare un altro algoritmo che consente di effettuare i calcoli in base 10
- Metodo delle divisioni ripetute
- Sia X la parola di codice in base b che si vuole convertire in Y in base c
- Se $Y = [\beta_{m-1}, \beta_{m-2}, ..., \beta_1, \beta_0]$ il suo valore é: $v(Y) = \beta_{m-1} c^{m-1} + \beta_{m-2} c^{m-2} + ... + \beta_1 c + \beta_0$
- Dividendo (divisione intera) per c si ha: $v(Y) = Qc + R$ (dove Q é il quoziente e R é il resto) dove:

$$Q = \beta_{m-1} c^{m-2} + \beta_{m-2} c^{m-3} + ... + \beta_1$$

$$R = \beta_0$$

- Si nota che questo risultato é indipendente dal tipo di base utilizzato per eseguire le operazioni

Conversione di base per divisioni ripetute

- Si noti che essendo $R < c$, esso ha la stessa rappresentazione in base b e c
- Questa operazione puó essere ripetuta in maniera iterattiva fino a calcolare tutti i coefficienti β_i
- Le operazioni possono essere svolte utilizzando l'aritmetica nella base di partenza b rappresentando quindi c in base b
- Questo algoritmo riveste particolare interesse nella conversione da base 10 a base 2

Conversione di base per divisioni ripetute

Conversione di 27_{10} in base 2 (si divide per 2):

iterazione	Q	$R = \beta_i$	β_i
0	27	1	β_0
1	13	1	β_1
2	6	0	β_2
3	3	1	β_3
4	1	1	β_4
5	0		

Quindi $27_{10} = 11011_2$

Esercizi

1. Si calcoli la rappresentazione in base 10 dei seguenti numeri naturali in base 2: 1010_2, 110010_2, 100100111_2
2. Si calcoli la rappresentazione in base 2 dei seguenti numeri naturali in base 10: 1010_{10}, 87_{10}, 747_{10}
3. Si eseguano le seguenti conversioni $102_3 \rightarrow Y_{10}$, $643_7 \rightarrow Y_{10}$, $67_{10} \rightarrow Y_8$, $1419_{10} \rightarrow Y_3$
4. Si converta $145_6 \rightarrow Y_7$
5. Conversioni fra le basi 2, 8 e 16

Conversione base 2 $\Leftrightarrow$ base 16

- Conversione diretta da binario a esadecimale:
  - suddividere i bit in gruppi di 4 (da destra) aggiundendo eventuali zeri a sinistra
  - sostituire ogni gruppo di 4 bit con il corrispondente simbolo esadecimale
  - $X_2 = 1110011110 = 0011|1001|1110$ da cui $X_{16} = 39E$
- Nel processo inverso si tratta semplicemente di sostituire i simboli esadecimali con gli equivalenti binari

Conversione base 10 $\Leftrightarrow$ base 16

- Si puó sempre passare per la base 2
- Conversione da base 16 a base 10:

$$X_{16} = AB4 \rightarrow X_{10} = 10 \cdot 16^2 + 11 \cdot 16^1 + 4 \cdot 16^0 = 2740$$

- Conversione da base 10 a base 16 (divisioni ripetute):
- Esempio $X_{10} = 1410$

iterazione	Q	$R = \beta_i$	β_i
0	1410	2	β_0
1	88	8	β_1
2	5	5	β_2
3	0	0	

- Da cui $X_{16} = 582$

Aritmetica base 16

- Le somme possono essere fatte convertendo gli operandi in base 10 o 2, oppure direttamente in base 16
- In particolare, se A e B sono i due numeri da sommare, é sufficiente usare $s_i = (a_i + b_i)_{mod16}$ e $r_i = (a_i + b_i)/16$ (riporto)
- Esempio:

r_i	1	1	0	0	
A	A	4	7	B	+
B	1	E	F	2	=
$A+B$	C	3	6	D	

Guasti nei sistemi digitali

- Nonostante l'attenzione rivolta verso lo sviluppo di tecnologie affidabili, i sistemi digitali sono soggetti a guasti
- I guasti si possono produrre durante la fabbricazione dei circuiti integrati o durante il loro normale funzionamento a causa di rotture di materiale o disturbi esterni
- Possono essere permanenti o transitori
- I guasti possono dare luogo ad errori nell'elaborazione, memorizzazione e trasmissione dei dati
- Molte applicazioni dei sistemi digitali sono critiche dal punto di vista della sicurezza e vanno quindi protette dai malfunzionamenti indotti dai guasti

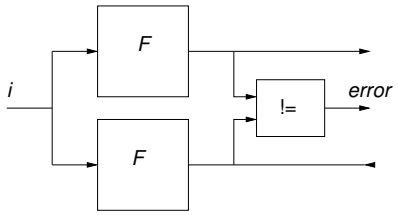
Sommario

Il ruolo della ridondanza

- Per proteggersi dai guasti, si adottano tipicamente forme di ridondanza, ovvero il sistema utilizza un numero di "risorse" maggiore di quello strettamente necessario
- Tali risorse vengono utilizzate per **rivelare** la presenza di errori, per **correggerli** o per **mascherare** la presenza di guasti
- Tipi di ridondanza:
  1. Funzionale (i risultati vengono calcolati da piú di un unitá funzionale consentendo cosí rivelazione o mascheramento di errori)
  2. Temporale (ripetizione di elaborazioni in modo proteggersi da guasti transitori)
  3. Delle informazioni (il codice utilizzato non é quello minimo e presenta informazioni aggiuntive tali da consentire la rivelazione o correzione di errori)

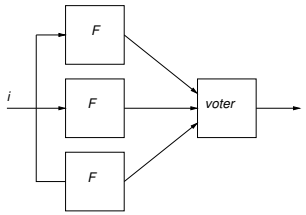
Ridondanza funzionale

Duplicazione (duplex) ⇒ rivelazione di errore



La rete *checker* != confronta le uscite dei due blocchi uguali e determina se si é avuto un errore singolo

Triplicazione (TMR) ⇒ mascheramento di errore



La rete *voter* produce un uscita uguale al valore della maggioranza degli ingressi ⇒ se un solo modulo ha un guasto l'uscita é corretta

Ridondanza dell'informazione

Rivelazione di errore

- Si supponga di voler codificare due informazioni *rosso* e *verde*, secondo Eq. 1 serve un singolo bit: 1 = *rosso*, 0 = *verde* (es.)
- In presenza di un guasto l'errore prodotto non puó essere riconosciuto e si riveceverá il simbolo errato
- Supponiamo di utilizzare una codifica ridondante (2 bit): 11 = *rosso* e 00 = *verde*
  - in presenza di un guasto che cambia un singolo bit (errore singolo), riceverá 01 o 10 che non appartengono al codice e riconosceró quindi l'errore
  - non sono peró in grado di correggerlo
  - cosa succede se ho 2 errori? o se utilizzo la codifica 01 = *rosso* e 00 = *verde*?

Ridondanza di informazione

Correzione di errori

- Supponiamo di aumentare ulteriormente la ridondanza (2 bit): 111 = *rosso* e 000 = *verde*
  - in presenza di un guasto che cambia un singolo bit, supponendo che il valore corretto sia 000, riceverá 001, 010 o 100 dai quali posso dedurre che il simbolo corretto é *verde* (correzione di errore)
  - cosa succede se ho 2 errori? o se utilizzo la codifica 011 = *rosso* e 010 = *verde*?
  - si noti che il meccanismo di codifica utilizzato é simile alla ridondanza modulare tripla

Considerazioni

- La ridondanza delle informazioni puó aiutare a rivelare o correggere errori
- Bisogna scegliere peró il codice in maniera opportuna
- La ridondanza delle informazioni implica una ridondanza hardware (per supportare i bit che vengono aggiunti)
- Il numero di errori che mi aspetto possano essere prodotti da un guasto é particolarmente importante: le capacitá di rivelazione e correzione di errore dei codici sono limitate

Sommario

Molteplicitá degli errori

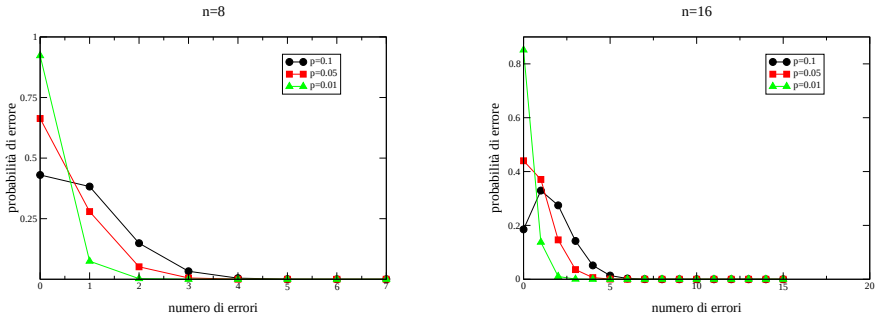
- Ipotesi: parole di n bit con probabilitá di errore su un singolo bit p e errori indipendenti sui singoli bit
- Probabilitá di avere la parola corretta $(1-p)^n$, probabilitá di un singolo errore $np(1-p)^{n-1}$
- Probabilitá di avere k errori

$$P_k = \binom{n}{k} p^k (1-p)^{n-k} \tag{5}$$

  ove il prodotto binomiale rappresenta il numero di combinazioni di n bit a k alla volta
- Se $p \ll 1$ e n non é molto grande il numero di errori con molteplicitá > 1 é trascurabile

Molteplicitá degli errori: esempi

Si osserva che l'ipotesi p piccolo é relativa al valore di n.



Sommario

Distanza

- La ridondanza di per se stessa puó non aggiungere alcuna capacitá di rivelazione o correzione di errore al codice, bisogna trovare una proprietá (Hamming) che ci aiuti a capire tali capacitá
- Si definisce distanza d fra due configurazioni binarie di n bit il numero di bit omologhi per cui esse differiscono
- Ad esempio, se $a = 001001$ e $b = 010100$, $d(a, b) = 4$
- La distanza minima di un codice é il valore minimo di d per ogni possibile coppia di configurazioni distinte

$$d_{min} = \min_{\forall a,b \in \mathcal{C},\ a \neq b} d(a, b) \tag{6}$$

Esempio

- Si calcoli la distanza minima del seguente codice

w_0	0101
w_1	1100
w_2	1000
w_3	1010
w_4	0001
w_5	0100
w_6	1001
w_7	1110

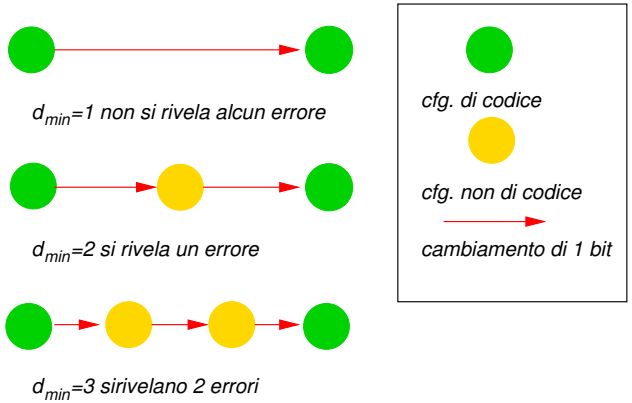
w_1	2						
w_2	3	1					
w_3	3	2	1				
w_4	4	3	2	3			
w_5	1	1	2	3	2		
w_6	2	2	1	2	1	3	
w_7	3	1	2	1	4	2	3
	w_0	w_1	w_2	w_3	w_4	w_5	w_6

- Quindi $d_{min} = 1$

Rivelazione di errori

Sia k il numero di errori rivelabili:

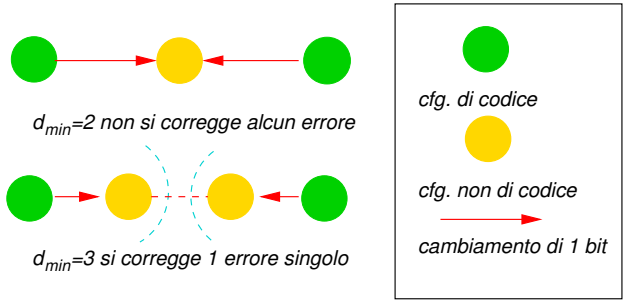
$$d_{min} = k + 1 \tag{7}$$



Correzione di errori

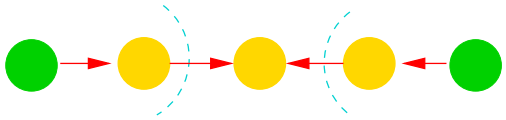
Per correggere k errori, l'insieme di configurazioni non di codice raggiungibili da una configurazione di codice come conseguenza di k errori deve essere separato da quelli raggiungibili da altre configurazioni di codice:

$$d_{min} = 2k + 1 \tag{8}$$



Correzione e rivelazione di errori

- Un codice a correzione di errore implica un partizionamento dell'insieme delle possibili configurazioni di n bit
- Un codice a correzione di errore implica anche la capacitá di riconoscere la presenza di errori
- In particolare, un codice che verifica: $d_{min} = k + j + 1$ con $k > j$ corregge fino a j errori e ne rivela fino a $k - j$
- $d_{min} = 4 = 2 + 1 + 1 \Rightarrow$ consente di correggere un errore e di rivelarne uno



Sommario

Esercizi

Esercizi per verificare la comprensione dell'argomento

Si considerino vari codici proposti per codificare le informazioni `rosso` e `verde` si determini la distanza di Hamming e si verifichino i ragionamenti fatti nei lucidi precedenti con quanto fornito dalle equazioni 7 e 8

Codici a rivelazione di errore: codici separabili

- I codici separabili contengono un certo numero (I) di bit di informazione e un certo numero (C) di bit di check
- I bit di informazione costituiscono spesso un codice non ridondante (interi, caratteri), mentre i bit di check sono aggiunti per ottenere un valore adeguato di distanza minima
- Il successo di questi codici consiste nella facilitá con cui la parte di informazione puó essere estratta dalla parola di codice
- Il codice di paritá e quello Berger sono separabili

Codice di paritá

- L'idea é di aggiungere un bit ai bit di informazione (in numero I) in modo che il numero di 1 nella parola complessiva sia pari
- In pratica, tale bit puó essere calcolato come la somma aritmetica dei bit di informazione modulo 2
- Se $\langle i_1, i_2, ..., i_I \rangle$ sono i bit di informazione, il bit di paritá viene calcolato come:

$$p = \left(\sum_{k=1}^{I} i_k \right) \mod 2$$

- Denotando come $\oplus$ l'operatore (associativo) che realizza la somma modulo 2 di due bit ($a \oplus b = (a + b) \mod 2$) si ha

$$p = i_1 \oplus i_2 \oplus i_3 \oplus ....i_I$$

Codice di paritá

Il codice di paritá é in grado di rivelare un singolo errore

Si tratta di dimostrare che $d_{min} = 2$.
Si considerino due configurazioni dei bit di informazione $a \neq b$ e si supponga che queste differiscano di un solo bit. In tale caso, la paritá del numero di 1 sará diversa da una parola all'altra e quindi $p_a \neq p_b$, per cui la distanza fra tali parole é 2.
Se due parole hanno lo stesso bit di paritá allora, o sono uguali o differiscono di 2 bit. Quindi $d_{min} = 2$ □

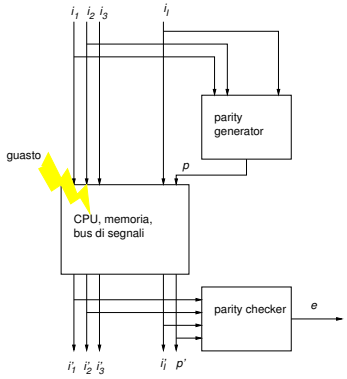
Calcolo del segnale di errore

- Siano $\langle i_1, i_2, ..., i_I \rangle$ i bit di informazione e p il bit di paritá $\Rightarrow$ parola di codice $\langle i_1, i_2, ..., i_I, p \rangle$
- In caso di guasti, la parola che viene ricevuta o letta sará $\langle i'_1, i'_2, ..., i'_I, p' \rangle$
- La sindrome di errore viene calcolata confrontando il bit di paritá ricevuto p' con quello ricalcolato come

$$p'' = i'_1 \oplus i'_2 \oplus i'_3 \oplus ....i'_I$$

- Si ha $e = p' \oplus p''$, infatti se $p' \neq p''$ ho errore ed $e = 1$, altrimenti 0. Il checker esegue:

$$e = i'_1 \oplus i'_2 \oplus i'_3 \oplus ....i'_I \oplus p'$$



Esempi

Svolti in aula

Codice di Berger

Example

- In questo codice si conta il numero di 0 (k) presenti nei bit di informazione e poi si aggiunge la codifica binaria di tale numero
- Se i bit di informazione sono I il numero di bit di check deve soddisfare $2^C - 1 \geq I \Rightarrow C = \lceil \log_2(I+1) \rceil$
- Esempio: $I = 8$, $C = \lceil \log_2 9 \rceil = 4$

informazione	k	checkbit ($c_8 c_4 c_2 c_1$)
00010111	4	0100
00101000	6	0110
00000000	8	1000
....	....	....

Codice di Berger

Il codice di Berger é in grado di rivelare un singolo errore

Si considerino due configurazioni dei bit di informazione $a, b \mid d(a,b) = 1$. Se differiscono di 1 bit, $k_a = k_b \pm 1$ e quindi i checkbit corrisponderanno a due numeri binari diversi.
Supponiamo ora che sia $a \neq b$ e $k_a = k_b$, se due parole sono diverse esisterá una posizione i tale che $a_i = 0$ e $b_i = 1$, ma se il numero di 0 é lo stesso ne deve esistere un altra j tale che $a_j = 1$ e $b_j = 0$.
Quindi $d_{min} = 2$ □

Confronto

Il codice di paritá e quello di Berger hanno la stessa capacitá di rivelare errori, ma quello di Berger richiede un numero maggiore di bit di check a paritá di informazione. Per capire come mai é stato introdotto bisogna analizzare le capacitá di rivelazione di errore con maggiore dettaglio:

- Il codice di paritá non é in grado di rivelare qualsiasi errore su un numero pari di bit
- Il codice di Berger non é in grado di rivelare errori che cambiano il valore di un insieme di bit contenente lo stesso numero di 0 e di 1. É in grado di rivelare errori che affliggono insiemi di bit che valgono tutti 1 o 0
- Quest'ultimo tipo di errore (per ragioni fisiche) é piuttosto probabile

Esempi

Svolti in aula

Codici non separabili

- i bit di informazione non sono separabili da quelli di check
- Il codice m su n appartiene a questa categoria: utilizza tutte le possibili configurazioni di n bit che contengono m bit con valore 1
- Ad esempio il codice 2 su 4 utilizza le parole $\{0011, 0110, 0101, 1010, 1001, 1100\}$

Il codice m su n ha la capacitá di rivelare un singolo errore

Infatti, se due configurazioni $a \neq b$ appartengono al codice, esisterá un indice i tale che $a_i = 0(1)$ e $b_i = 1(0)$, poiché il numero di 1 rimane m, ne esisterá un altra j in cui $a_i = 1(0)$ e $b_i = 0(1) \Rightarrow d_{min} = 2$ □

- Il numero di configurazioni appartenente a questo codice é pari al numero di combinazioni di n oggetti a m alla volta: $\binom{n}{m}$

Approfondimenti

Si determini (dato n) il valore m che rende massimo il numero di informazioni codificate dal codice m su n (si risolva prima il caso di n pari).

Si tracci un grafico che sull'asse x contiene il numero totale (M) di informazioni che si vogliono codificare e sull'asse y il numero di bit minimo da utilizzare nei codici di paritá, Berger e m su n (ottimale) per codificare tali informazioni (es. se $M = 5$, servono 4 bit nel codice di paritá, 6 bit nel codice Berger e 4 nel codice di paritá)

Codici a correzione di errore: codice di Hamming

- Si tratta di un codice a correzione di errore singolo
- Utilizzo di un insieme di C checkbit che sia in grado di codificare la posizione di tale errore (o l'assenza di errori)

$$2^C \geq I + C + 1$$

- I checkbit venono calcolati come bit di paritá su sottoinsiemi dei bit di informazione

Sommario

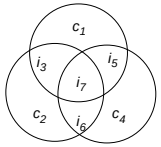
Codice di Hamming

- Consideriamo $I = 4$ da cui $C = 3$ e disponiamo i bit in modo che i bit di check abbiano un indice corrispondente a una potenza intera di 2

$$c_1\, c_2\, i_3\, c_4\, i_5\, i_6\, i_7$$

- I checkbit sono calcolati in questo modo

$$c_1 = i_3 \oplus i_5 \oplus i_7 \qquad (9)$$
$$c_2 = i_3 \oplus i_6 \oplus i_7$$
$$c_4 = i_5 \oplus i_6 \oplus i_7$$



- La regola consistente nell'associare a c_{2^k} tutti gli i_j in cui la codifica binaria dell'indice j ha un 1 nella k-ma posizione

Capacitá di correzione di errore del codice Hamming

Distanza minima del codice Hamming

Supponiamo di avere due configurazioni dei bit di informazione con distanza 1. Siccome il bit per cui differiscono compare in almeno 2 checkbit, tali checkbit differiranno nella parola complessiva che avrá distanza 3.

Supponiamo ora che ci siano 2 bit di informazione diversi, in tale caso si deve osservare che esiste almeno un checkbit che non li considera insieme, quindi anche in questo caso la distanza é 3.
Rimane il caso in cui le due parole differiscono per 3 bit di informazione. Quindi $d_{min} = 3$ □

Calcolo delle sindromi di errore

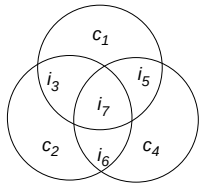
- Siano $\langle c'_1\, c'_2\, i'_3\, c'_4\, i'_5\, i'_6\, i'_7 \rangle$ i bit ricevuti o letti e quindi eventualmente afflitti da errore
- Le sindromi di errore vengono calcolate confrontando i checkbit ricevuti con quelli ricalcolati sui bit ricevuti, rivelando quindi errori singoli nell'ambito dell'insieme di bit relativo a ciascun checkbit

$$s_1 = c'_1 \oplus i'_3 \oplus i'_5 \oplus i'_7 \qquad (10)$$
$$s_2 = c'_2 \oplus i'_3 \oplus i'_6 \oplus i'_7$$
$$s_4 = c'_4 \oplus i'_5 \oplus i'_6 \oplus i'_7$$

Correzione dell'errore

In assenza di errore (singolo) $s_4 s_2 s_1 = 000$

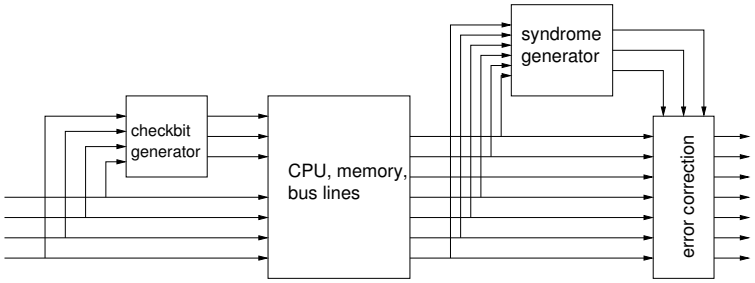
posizione dell'errore	$s_4 s_2 s_1$
c_1	001
c_2	010
i_3	011
c_4	100
i_5	101
i_6	110
i_7	111



In presenza di errore singolo, con le scelte adottate, la configurazione delle sindromi di errore fornisce la codifica binaria dell'indice del bit errato.

Correzione dell'errore

Una volta riconosciuta la posizione dell'errore, questo puó essere corretto semplicemente cambiandone il valore.



Esempio

- Sia $\langle i_3, i_5, i_6, i_7 \rangle = 1010$, da cui vengono calcolati i checkbit

$$c_1 = 1 \oplus 0 \oplus 0 = 1$$
$$c_2 = 1 \oplus 1 \oplus 0 = 0$$
$$c_4 = 0 \oplus 1 \oplus 0 = 1$$

- La parola memorizzata o trasmessa é: $\langle c_1, c_2, i_3, c_4, i_5, i_6, i_7 \rangle = 1011010$
- Ipotesi di errore singolo su i_7, riceveremo $\langle c'_1, c'_2, i'_3, c'_4, i'_5, i'_6, i'_7 \rangle = 1011011$
- Le sindromi di errore risultano:

$$s_1 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$
$$s_2 = 0 \oplus 1 \oplus 1 \oplus 1 = 1 \Rightarrow \qquad \langle s_4 s_2 s_1 \rangle = 111 \text{ ovvero } 7$$
$$s_4 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

Conclusioni

- Come rappresentare l'informazione in un sistema binario
  - Condizioni per avere un codice minimo
  - Rappresentazione dei numeri binari
- Come proteggere l'informazione da guasti ed errori
  - codici a rivelazione e correzione di errore

Esempio di codice Berger

Return