
Set 3 di esercizi

MICHAEL CAVICCHIOLI

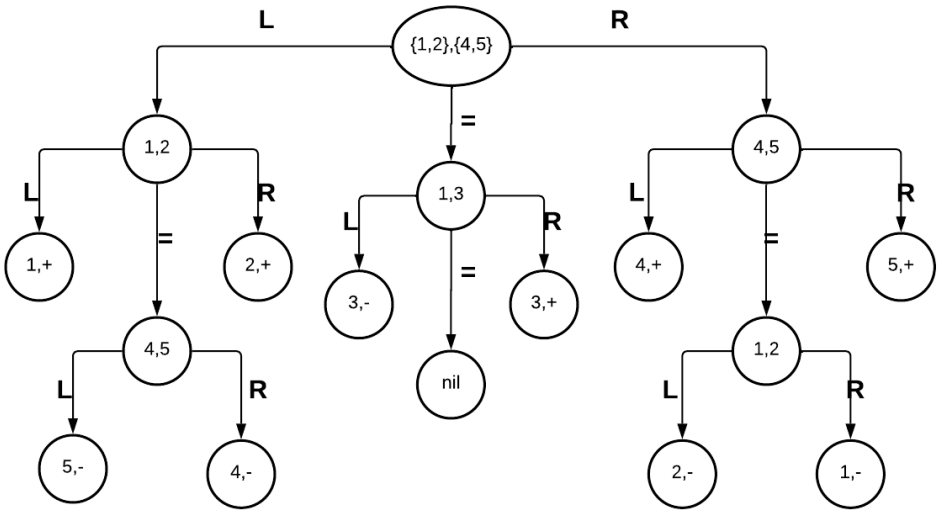
7149344

Anno Accademico 2023-2024

# ESERCIZI DI APPROFONDIMENTO

## 2.2 STRATEGIE E CODICI

a): La soluzione è riportata nella seguente figura sottostante.




b):

- Lunghezza massima della strategia: dato che la lunghezza massima della strategia corrisponde all'altezza dell'albero e quest'ultima coincide con 3, allora la lunghezza massima è uguale a 3.
- Lunghezza media della strategia: questa è pari all'altezza media dell'albero.
  Si osserva che ogni foglia ha probabilità uniforme di essere raggiunta, la quale è pari a $\frac{1}{11}$. Quindi, considerando la seguente formula: $\sum_{x \in X} p(x)l(x)$, dove $p(x)$ è la probabilità di raggiungere una foglia, mentre $l(x)$ è la lunghezza dalla radice ad una foglia, e svolgendo i calcoli, è possibile ricavarsi l'altezza media dell'albero, e di conseguenza anche la lunghezza media della strategia, che corrisponde a $\frac{26}{11}$.

c): Denotata con k l'altezza dell'albero e sapendo che quest'ultimo è di tipo ternario, allora il numero di foglie è al più: $3^k \geqslant 2n+1$, pertanto, l'altezza minima dell'albero, che corrisponde al limite inferiore, in funzione di n, al numero *massimo* di pesate in una qualsiasi strategia è dato da $k \geqslant \lceil \log_3(2n+1) \rceil$.

d): In generale, nota la cardinalità dell'alfabeto X, per esempio $|X| = D$, per il teorema di Shannon è noto che $L_p(C) \geqslant H_D(p)$, dove $L_p(C)$ indica la lunghezza media del codice che dipende dalla distribuzione di probabilità associata, mentre $H_D(p)$ rappresenta l'entropia della distribuzione di probabilità[1].
Il risultato del teorema di Shannon indica che tramite l'entropia della distribuzione di probabilità è possibile conoscere l'altezza media dell'albero, e quindi dare un limite inferiore, in funzione di n, al numero *medio* di pesate in una qualsiasi strategia.
In questo caso, la cardinalità dell'alfabeto è pari a 3 e la distribuzione di probabilità è uniforme, con valori corrispondenti a $\frac{1}{2n+1}$. Pertanto, sostituendo i valori nella formula dell'entropia, si ottiene che:
$H_3(\frac{1}{2n+1}, \frac{1}{2n+1}, \ldots, \frac{1}{2n+1}) = \sum_{i=1}^{2n+1} \frac{1}{2n+1} \log_3(2n+1) = \log_3(2n+1)$,
che è il limite inferiore richiesto.

## 2.3 LA CAPACITÀ DEL PICCIONE VIAGGIATORE

a): Dato che la matrice del canale soddisfa le seguenti propritetà:

- Tutte le righe sono uguali a meno di permutazioni.
- $\exists\, c \mid \forall y$: $\sum_{x \in X} P(y|x) = c$, cioè che la somma degli elementi di ogni colonna è pari a c.

questo permette di affermare che il canale è *debolmente simmetrico*.
Pertanto, vale la proprietà che $C = \log_2 |Y| - H(r)$, dove C è la capacità del canale, Y è l'alfabeto di uscita e H(r) è l'entropia della r-esima riga.
Dato che ogni piccione riesce ad arrivare a destinazione, l'entropia associata ad una qualsiasi riga della matrice è 0, questo perchè si ha un solo messaggio possibile, cioè quello corretto, che viene restituito con probabilità pari ad 1.
Inoltre, è noto che $|Y| = 2^8 = 256$, che sono i possibili messaggi codificati con 8 bit.

1 Il pedice D indica che i logaritmi, con cui si calcola l'entropia, sono in base D, con D intero.

Dunque, la capacità del canale è pari a $C = \log_2 |Y| = 8$ e quindi, dato che in un'ora è possibile inviare al più 12 piccioni, la capacità bit/ora di questo canale è pari a 12*8 = 96.

b): È noto che i nemici abbattono una frazione di $\alpha$ piccioni, sostituendo il messaggio con uno qualsiasi, pertanto è ovvio che una parte dei piccioni che arrivano con il messaggio corretto è pari a $1 - \alpha$. Si fa notare che i nemici sostituiscono il messaggio con uno qualsiasi, pertanto è possibile che reinseriscano uno corretto.
Dunque, la probabilità che arrivi il messaggio corretto è pari a $1 - \alpha + \frac{\alpha}{256} = 1 - \alpha\frac{255}{256}$, mentre la probabilità di un messaggio non corretto è pari a $\frac{\alpha}{256}$.
Quindi, la matrice del canale in questione, la quale è formata da 256x256 elementi, ha valori:

- sulla diagonale pari a $1 - \alpha\frac{255}{256}$
- non sulla diagonale pari a $\frac{\alpha}{256}$

Si osserva che la matrice soddisfa le prorietà riportate nel punto a), pertanto essa è *debolmente simmetrica*, quindi la capacità del canale è data dalla formula: $C = \log_2 |Y| - H(r)$.
Per trovare H(r), si deve prima dimostrare che:
$H(p_1, p_2, \ldots, p_n) = H(p_1) + (1 - p_1)H(\frac{p_2}{1-p_1}, \frac{p_3}{1-p_1}, \ldots, \frac{p_n}{1-p_1})$.
Sfruttando il suggerimento presente nella traccia dell'esercizio, definendo X v.a. distribuita come $\mathbf{p}$ e Y v.a. ausiliaria dove Y = 1, se $X = x_1$, Y = 0 altrimenti, si ricorda che, per la proprietà Chain-Rule si ha:
$H(X, Y) = H(X) + H(X|Y) = H(Y) + H(X|Y)$, dove però il termine $H(Y|X) = 0$, poichè si conosce già il valore di X, il quale da certezza del valore di Y.
Pertanto si ottiene la seguente formula:
$H(X) = H(Y) + H(X|Y)$ e sfruttando la definizione di *entropia differenziale*, si ottiene:
$H(X) = H(Y) + p_1 H(X|Y = 1) + (1 - p_1)H(X|Y = 0)$, ma, ancora una volta, osservando che se $Y = 1$, allora $X = x_1$, di conseguenza $H(X|Y = 1) = 0$, e quindi si giunge a:
$H(X) = H(Y) + (1 - p_1)H(X|Y = 0)$.
Il risultato ottenuto può essere visto come un modo diverso di riscrivere l'equazione che si è dovuto dimostrare, difatti: considerando X distribuita come $\mathbf{p}$, l'entropia di Y come l'entropia binaria di $p_1$ e $H(X|Y = 0) = H(\frac{p_2}{1-p_1}, \frac{p_3}{1-p_1}, \ldots, \frac{p_n}{1-p_1})$, si ottiene quanto richiesto.
Dunque, si può procedere con il calcolo dell'entropia della prima

riga[2] della matrice:
$H(r = 0) = B(1 - \alpha\frac{255}{256}) + (\alpha\frac{255}{256})H(\frac{1}{255}, \frac{1}{255}, \ldots, \frac{1}{255})$.
Si osserva che per la definizione di entropia, $H(\frac{1}{255}, \frac{1}{255}, \ldots, \frac{1}{255}) = \log_2 255 \approx 7.994$.
Pertanto, moltiplicando tutti i valori per 12[3], si otterrà la capacità bit/ora richiesta:
$C = 96 - 12 * B(1 - \alpha\frac{255}{256}) - 12 * (\alpha\frac{255}{256}) * 7.994$

2 Si osserva che il valore ottenuto è costante per ogni riga dato che la matrice è *debolmente simmetrica*.

3 Numero massimo di piccioni inviati in un'ora.

# ESERCIZI DI PROGRAMMAZIONE

## 3.2 CODICI DI HUFFMAN

La soluzione di questo esercizio è presente nel file Python *huffman.py* e per eseguire il codice bisogna scrivere *python huffman.py*, o *python3 huffman.py*, nella cartella in cui risiede tale file oppure eseguirlo da Visual Studio Code (ma bisogna aver installato l'estensione Python per tale editor).
Nel file sono presenti due metodi principali:

- codice_huffman
- decodifica_huffman

Il primo riguarda la prima parte dell'esercizio e prende come parametri la lista delle lettere e la lista delle porbabilità ad esse associate.
All'interno del metodo si è controllato, innanzitutto, se i parametri fossero stati inseriti correttamente e, successivamente, si è ordinata, in modo crescente, la lista delle probabilità, aggiornando anche la lista delle lettere, in maniera tale da non perdere l'associazione, tramite il metodo *riordina_dati*.
Dopodichè, si è istanziato un dizionario vuoto, il quale conterrà, come chiavi, le lettere che si sono usate per costruire i nodi dell'albero binario, mentre come valori i bit 0 o 1, i quali indicano quale percorso porta a quella determinata lettera.
In seguito, all'interno di un ciclo, si è rimossa la i-esima coppia di probabilità, per la creazione dell'i-esimo nodo dell'albero, ottenuto le cifre decimali di ognuna di loro e aggiornato la lista delle probabilità con la somma dei valori della coppia, arrotondata alla cifra decimale più grande.
Sempre nello stesso ciclo, successivamente, si è andati a fare i medesimi passi, ma per la lista delle lettere (il passo delle cifre decimali non è stato fatto per ovvie ragioni, dato che le lettere sono codificate come caratteri e non come valori *float*). Poi, si è andati ad aggiornare i valori del dizionario, come spiegato precedentemente, e ordinato nuovamente le due liste, in modo da considerare sempre la coppia di probabilità di valore minore rispetto alle altre.

Finito il ciclo, è possibile che si sia ottenuto un dizionario che abbia avuto come chiavi o foglie oppure tuple di tuple di ... di tuple, quindi, il prossimo passo è stato quello di leggere le informazioni base riportate nelle tuple, cioè i valori delle foglie che le hanno composte, per poi aggiornare i valori di quelle foglie che non erano state salvate come tuple, aggiungendo, in testa, il valore corrispondente alla tupla iterata[4]. Per fare questo si è andati ad implementare il metodo *aggiorna_valori*, che opera come sopra descritto, in maniera ricorsiva, fino ad ottenere la singola lettera per poi associargli il valore corrispondente, questo per simulare un percorso di discesa nell'albero binario.
Infine, si è rimosso dal dizionario tutte le chiavi che non corrispondevano alle foglie e restituire il dizionario stesso.
Invece, il metodo *decodifica_huffman*, prende in ingresso un codice prefix-free, salvato come dizionario, ed una stringa binaria.
Per prima cosa si è creata la variabile temporanea *blocco* e la stringa decodificata, vuota per il momento.
In seguito, si è andati ad iterare ogni carattere della stringa, concatenandolo alla variabile temporanea. Questo è servito per controllare se il blocco fosse stato presente nel dizionario (quindi come chiave del codice): in caso positivo, allora si è andati a concatenare alla stringa decodificata il valore corrispondente a quel blocco e resettando quest'ultimo, cioè assegnandogli la stringa vuota; mentre, in caso negativo, il ciclo ha ripreso l'esecuzione, mantenendo i valori precedenti salvati nella variabile blocco.
Una volta ciclata tutta la stringa presa in ingresso, si è restituita quella decodificata.

4 Esempio: (('a', 'b'), 'c'): '0', significa che le foglie salvate nel dizionario dei percorsi sono 'a', 'b' e 'c', quindi, all'interno del dizionario si è aggiunto il valore 0 in testa al percorso che era già stato definito per tali foglie.