Compressione

Algoritmo di Huffman

Come detto prima, questo algoritmo è molto usato nei testi ed è di tipo statico, ovvero si fonda su alcuni studi effettuati a priori proprio sulla frequenza delle lettere nelle parole.

In un libro scritto in italiano, troveremo sicuramente una frequenza molto alta di lettere A rispetto a lettere Z o U che si presentano meno. Partendo da queste semplici considerazioni, si cerca quindi di codificare le lettere con maggior frequenza come la A con dei codici binari più brevi rispetto a quelli utilizzati per la Z. In questo modo le lettere più frequenti in un testo (che sul pc vengono rappresentate sempre con 8 bits) verranno rimpiazzate da codici di bits più corti.

Ecco un esempio: Vogliamo comprimere un file testo contenente la stringa: **CIAO_MAMMA**
In un normale file testo, ogni lettera è rappresentata da 8 bits codificati rispettando il codice ASCII. Le lettere dell'esempio verrebbero quindi codificate nel modo seguente:
A = 01000001
C = 01000011
I = 01001001
O = 01001111
M = 01001101
_ = 00100000

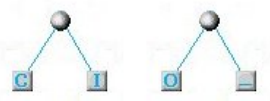
Il nostro file salvato sarà composto così da 80 bits, ovvero 10 lettere * 8 bit.
01000011 01001001 01000001 01001111 00100000 01001101 01000001 01001101 01001101 01000001

Adesso applichiamo l'algoritmo di Huffman per calcolare dei nuovi codici da assegnare alle lettere dell'esempio. Il metodo usato è piuttosto semplice. Si deve costruire un albero in cui le lettere più frequenti siano posizionate più vicino alla radice rispetto a quelle con minore frequenza.

Per prima cosa è necessario contare la frequenza di ogni lettera nella nostra stringa:
C(1) I(1) A(3) M(3) O(1) _(1)
A questo punto vanno individuate le 2 lettere con minore frequenza. Nel nostro caso, le lettere con minore frequenza sono ad esempio la **C** e la **I**, che useremo per costruire un nuovo nodo alla base dell'albero.

Il nodo costituito dalle 2 lettere di minor frequenza: C ed I

Nel nostro schema delle frequenze, la C e la I saranno da ora in avanti rappresentate come un unica lettera e con una frequenza che sarà la somma delle loro frequenze:
C+I(2) A(3) M(3) O(1) _(1)
Troviamo nuovamente le due lettere con minore frequenza (O e _) e facciamo la stessa cosa...



Il nodo costituito dalle 2 lettere di minor frequenza Q e _

C+I(2) A(3) M(3) **O+_(2)**
A questo punto cerchiamo ancora le 2 lettere con minor frequenza e vediamo che si tratta

Indice

1. Introduzione
2. Concetto base
3. Tipi di compressione
4. Algoritmo di Huffman

Argomenti correlati

Bits e Bytes
La memoria di un Computer
Hard Disk
MP3
MP3 Pro
MPEG
La tecnologia RFID

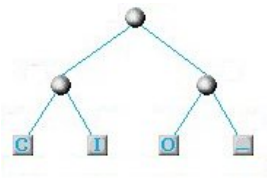
I piu' visitati

- Prendere la patente
- Il Viagra
- Il GPS
- eBay
- La stampante
- Il telescopio
- La tecnologia RFID
- Il Microprocessore
- Un Blog
- Lo standard Bluetooth

In Evidenza

- Ebay
- Porta USB
- Viagra
- DVD
- Stampante Laser
- Memoria Flash
- GPS
- PayPal
- Scheda Grafica
- Bluetooth
- O.N.U.
- Prendere la patente

dell'accoppiata C+I e O+_ per cui il nuovo nodo sarà schematizzato usando i nodi creati in precedenza:

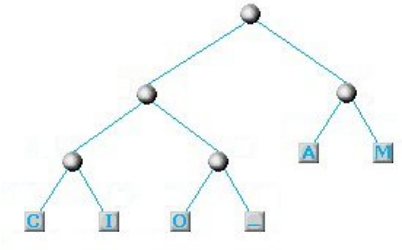


Il nodo costituito dalle 2 lettere di minor frequenza: C+I e O+_

Lo schema diventerà a questo punto il seguente:
C+I+O+_(4) A(3) M(3)
Prendiamo ancora una volte le 2 lettere di minor frequenza che adesso sono la A e la M:

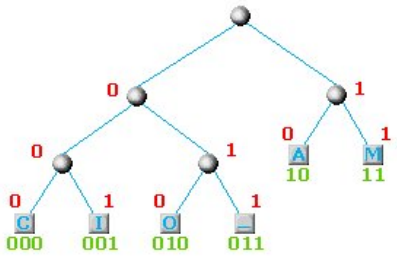
Il nuovo nodo costituito dalle 2 lettere di minor frequenza: A e M

Il nostro schema a questo punto si ridurrà a 2 soli elementi:
C+I+O+_(4) A+M(6)
Dovendo prendere le due lettere con minor frequenza non ci rimane che selezionare le uniche due rimaste e concludere finalmente il nostro albero:



L'albero nella sua versione finale (o quasi)

Una volta creato l'albero, bisogna associare ad ogni nodo un bit. E' possibile associare lo 0 a tutti i nodi di sinistra e 1 a tutti quelli di destra. E' possibile anche fare il contrario senza compromettere il risultato finale. Nel nostro caso scegliamo lo 0 a sinistra e l'1 a destra ed ecco il risultato finale:



L'albero con i relativi bit associati e con i valori (in verde) assegnati ad ogni singola lettera.

Volendo scrivere a questo punto il nostro CIAO_MAMMA con i nuovi codici otterremo la seguente sequenza di bits:

000001100100111110111110
Solamente 24 bits invece degli 80 usati in precedenza con il codice ASCII ! E solamente su una stringa di 10 caratteri. Immaginiamo lo stesso algoritmo applicato ad un file testo di migliaia di parole per capirne la vera potenza.

E' importante sottolineare che il codice ottenuto non è univoco, nel senso che può essere creato un codice altrettanto valido ma differente dal nostro che può dipendere ad esempio dalle lettere con minore frequenza scelte all'inizio oppure dal modo in cui decidiamo di impostare gli 0 e gli 1 sui rami dell'albero.

Una caratteristica particolare di questo algoritmo è che il codice ottenuto per ogni singola lettera NON è un prefisso di un altra lettera. Nel nostro esempio, la A ha il codice 10 e nessuna altra lettera ha una codifica che comincia con 10 (lo stesso ovviamente vale per ogni altra lettere che abbiamo codificato). In questo modo leggendo la stringa di 24 bits che abbiamo ottenuto in precedenza, possiamo riconoscere le singole lettere senza bisogno di un separatore tra loro.