Strutture per la compressione dei dati

In informatica la compressione di dati indica l’insieme di tecniche atte a rappresentare un’informazione digitale nel minor numero di bit possibile, in modo da ridurre sensibilmente lo spazio impiegato per rappresentare l’informazione.   
L’ informazione che viene trasmessa può essere di tre tipi:  
- ridondante, ovvero ripetitiva e prevedibile  
- irrilevante, ovvero le informazioni che non possiamo apprezzare e la cui eliminazione non influenza il contenuto del messaggio. Ad esempio visto che l’udito umano è in grado di rilevare i suoni con frequenza tra 16/20 Hz e 16 000/20 000 Hz, sarebbe inutile rappresentare le frequenze che si trovano al di fuori di questo intervallo.  
- fondamentale, ovvero i dati che non sono né ridondanti né irrilevanti e che quindi devono essere obbligatoriamente codificati per ricostruire l’oggetto originale.

La compressione di dati può essere di tipo lossy o loseless a seconda che si verifichi una parziale perdita di dati. Nelle tecniche di compressione lossy le informazioni irrilevanti vengono trascurate, e quindi, a differenza delle compressioni loseless, è impossibile una ricostruzione esatta dei dati originali a partire da quelli compressi. Formati come JPEG, mp3 e Flash sono di tipo lossy mentre PNG, RAR e ZIP sono ti tipo loseless.

L’entropia in una sorgente di messaggi è l’informazione media contenuta in tali messaggi. Può anche essere definita come la differenza tra la quantità totale di dati di un messaggio e la sua ridondanza (l'uso di parole la cui omissione non costituisce una sostanziale perdita di significato). Siccome l’obbiettivo delle tecniche di compressione è rappresentare le informazioni più frequenti con il minore numero di dati possibili, la lunghezza dei messaggi sarà inversamente proporzionale alla loro probabilità.

Normalmente una stringa di caratteri è rappresentata utilizzando un numero fisso di bit (ad esempio 8 per la codifica ACII). L’obbiettivo delle tecniche di compressione è quello di minimizzare il numero di bit che servono a rappresentare univocamente un carattere, tenendo anche d’occhio l’efficienza di codifica e decodifica del testo.

La **codifica di Huffman** usa un metodo specifico per scegliere la rappresentazione di ciascun simbolo, originando un codice prefisso, le cui parole non sono prefisso di nessun’altra parola del codice. L’idea è quindi quella di assegnare un codice più breve ai caratteri che verranno utilizzati più spesso.  
La tecnica utilizzata per ottenere questa struttura dati è l’algoritmo di Huffman, che consiste nella creazione di un albero binario dove i nodi foglia sono etichettati con il carattere e la corrispettiva frequenza. Si inseriscono quindi i figli in una coda con priorità e si ripetono i seguenti passi fino all’esaurimento della coda

1. Si prelevano dalla coda i simboli con frequenza minore e si crea un nodo padre con una frequenza data dalla somma dei due figli.
2. Si aggiunge il nodo genitore dalla lista, dalla quale vengono invece tolti i nodi figli.
3. Si ripetono le operazioni fino all’esaurimento della lista

Assumiamo di avere un testo d lunghezza n e composto da un alfabeto di k simboli.

Per la decodifica di ogni simbolo bisogna attraversare l’albero da radice a foglia, operazione che richiede mediamente la visita di O(log k) nodi. La complessità temporale complessiva sarà dunque di O (n log k).  
La complessità spaziale è invece O(k) per l’albero ed O(n) per il testo codificato

La **codifica aritmetica** si differenzia dalle altre forme di codifica entropica in quanto codifica il messaggio intero in un solo numero decimale S compreso tra 0 e 1. Data una stringa con alfabeto |∑|=n si suddivide quindi l’intervallo [0,1) in n sottointervalli, la cui ampiezza corrisponde alla stima della probabilità che il carattere corrente sia uguale al simbolo corrispondente all’intervallo. Il numero S cadrà in uno degli n intervalli permettendo così di decodificare il primo carattere. Terminato il primo passo si suddividerà l’intervallo preso precedentemente in considerazione in altri n sottointervalli utilizzando il modello già usato in precedenza. L’algoritmo continuerà ricorsivamente in questa maniera. Per terminare la ricorsione è necessario o conoscere in anticipo il numero totale di caratteri della stringa in considerazione o utilizzare un ulteriore intervallo che denoti la fine della stringa.

**Prediction by Partial Matching** (PPM) è un algoritmo adattivo di compressione dei dati basato su un modello di previsione statistica. I modelli PPM eseguono le loro predizioni in base al contesto, ovvero fanno un’analisi su un flusso di simboli non compressi che sono già stati tradotti in precedenza.  
Ai vari simboli è assegnato un valore, che sarà direttamente proporzionale alla frequenza con cui tale simbolo appare in un determinato contesto. Il numero di simboli precedenti presi in considerazione, n, determina l’ordine del modello PPM(n). Se n = 0 il valore abbinato ai simboli sarà dato esclusivamente dalla sua frequenza del testo.

**BWT** è un algoritmo usato nelle tecniche di compressione. La sua finalità non è direttamente quella di comprimere i dati, ma di permutarli in modo che un algoritmo di compressione funzioni in modo più efficace. Il suo output, infatti, non è altro che una permutazione della stringa originaria e potrà contenere lunghe sequenze di caratteri uguali posizionati adiacentemente, rendendosi così più adatto alla compressione. Se la stringa originale contiene molte sottostringhe che appaiono spesso, allora la stringa dopo la trasformazione conterrà diverse posizioni nelle quali lo stesso carattere sarà ripetuto varie volte di fila.  
Ecco un esempio di trasformazione con la parola MACACO$ (uso $ come simbolo di fine della parola)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Tutte le permutazioni | Permutazioni ordinate | Output |
| MACACO$ | MACACO$  ACACO$M  CACO$MA  ACO$MAC  CO$MACA  O$MACAC  $MACACO | ACACO$**M**  ACO$MA**C**  CACO$M**A**  CO$MAC**A**  MACACO**$**  O$MACA**C**  $MACAC**O** | MCAA$CO |

A questo punto entra in azione una caratteristica fondamentale della BWT: la reversibilità. Se infatti l’unico scopo dell’algoritmo fosse permutare la stringa originale in modo da comprimere i dati in modo più semplice, basterebbe allora ordinare i caratteri in ordine alfabetico. Con la trasformata di Burrows-Wheeler invece è possibile ricostruire l’input iniziale a partire solamente dai caratteri dell’ultima colonna.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input:  MCAA$CO | A  A  C  C  M  O  $ | AC  AC  CA  CO  MA  O$  $M | ACA  ACO  CAC  CO$  MAC  O$M  $MA | ACAC  ACO$  CACO  CO$M  MACA  O$MA  $MAC | ACACO  ACO$M  CACO$  CO$MA  MACAC  O$MAC  $MACA | ACACO$  ACO$MA  CACO$M  CO$MAC  MACACO  O$MACA  $MACAC | ACACO$M  ACO$MAC  CACO$MA  CO$MACA  MACACO$  O$MACAC  $MACACO | Output:  MACACO$ |

Un **array di suffissi** è l’array ordinato di tutti i suffissi di una stringa.   
Sia S = S1, S2, ... Sn $ una stringa e S [i,j], la sottostringa di S che va dalla i-esima alla j-esima posizione di S. L’array dei suffissi A della Stringa S è costituito da un array di interi che forniscono le posizioni iniziali dei suffissi di S in ordine alfabetico. Quindi A[i] contiene la posizione iniziale dell’i-esimo suffisso più piccolo di S e quindi ꓯi 1 < i ≤ n S[A[i-1], n] < S [A[i], n].  
Consideriamo un esempio con S = MACACO$

|  |  |  |  |
| --- | --- | --- | --- |
| Suffissi | indice | Suffissi ordinati | Indice |
| MACACO$  ACACO$  CACO$  ACO$  CO$  O$  $ | 1  2  3  4  5  6  7 | ACACO$  ACO$  CACO$  CO$  MACACO$  O$  $ | 2  4  3  5  1  6  7 |

I **wavelet tree** sono una struttura dati che serve ad immagazzinare stringhe in uno spazio compresso. L’albero, le cui foglie sono costituite da singoli caratteri, divide ricorsivamente l’alfabeto in una coppia di sottostringhe. Se l’albero è bilanciato, le operazioni di accesso, rank e select possono essere effettuate in tempo O(log n), dove n indica la cardinalità dell’alfabeto.  
Nell’esempio che riporto sotto il criterio per la divisione della stringa è l’ordine alfabetico. Se un carattere appartiene alla prima metà del dato sottoalfabeto, allora si troverà nel nodo figlio sinistro, in caso contrario sarà nel destro.

