JBE. CONCETTO E APPLICAZIONE

Lydia Mitkovets, Daniel Sidorov, studenti, Alevtina Gourinovitch,

Dottorato di ricerca presso l'Università statale bielorussa di informatica e radioelettronica, Minsk, via P. Brovki 6.

Annotazione. Per salvare le informazioni all'interno dell'archivio, gli utenti cercano di ridurre al minimo le dimensioni dei file utilizzando un software di compressione dati. Si tratta di un nuovo algoritmo per la compressione dati in questo articolo. Si tratta di codifica j-bit (JBE). Questo algoritmo manipola ogni bit di dati all'interno del file per ridurre al minimo le dimensioni senza perdere dati dopo la decodifica. È una compressione lossless classificata. Questo algoritmo di base si combina con altri algoritmi di compressione dati per ottimizzare il rapporto di compressione. L'implementazione di questo algoritmo consiste in una combinazione di vari algoritmi di compressione dati.

Parole chiave: confezionamento dei dati, compressione, codifica di compressione, codifica sorgente.

Introduzione

La compressione dei dati è una trasformazione algoritmica dei dati per ridurre la quantità di dati che contiene. occupa. Questo algoritmo è applicato per un uso efficiente dei dispositivi di archiviazione e trasferimento dati.

La compressione è l'eliminazione della ridondanza contenuta nei dati sorgente. L'esempio più semplice di ridondanza è la ripetizione di frammenti nel testo (ad esempio, parole del linguaggio naturale o della macchina). Tale ridondanza viene solitamente eliminata sostituendo la sequenza ripetuta con un riferimento al frammento già codificato con un'indicazione della sua lunghezza. Un altro tipo di ridondanza è il fatto che alcuni valori nei dati compressi sono più comuni di altri.

La riduzione del volume di dati si ottiene sostituendo i dati che si verificano di frequente con parole di codice corte e i dati rari con parole di codice lunghe (codifica dell'entropia). La compressione di dati che non hanno la proprietà di ridondanza (ad esempio, segnale casuale o rumore bianco, messaggi crittografati) è fondamentalmente impossibile senza perdite.

Al centro di qualsiasi metodo di compressione c'è il modello di origine dati, o più precisamente, il modello di ridondanza. In altre parole, la compressione dati utilizza alcune informazioni a priori sul tipo di dati compressi. Senza tali informazioni sull'origine, è impossibile fare qualsiasi ipotesi sulla trasformazione che ridurrebbe il volume del messaggio. Il modello di ridondanza può essere statico, immutabile per l'intero messaggio compresso, o costruito o parametrizzato nella fase di compressione (e ripristino).

Tutti i metodi di compressione dei dati si differenziano in due classi principali:

- Compressione senza perdita di dati
- Compressione delle perdite

Quando si utilizza la compressione lossless, è possibile ripristinare completamente i dati originali, la compressione lossless consente di ripristinare i dati con distorsioni che sono solitamente insignificanti dal punto di vista dell'ulteriore utilizzo dei dati ripristinati. La compressione lossless è solitamente utilizzata per la trasmissione e l'archiviazione di dati di testo, programmi per computer, meno spesso per ridurre il volume di dati audio e video, foto digitali, ecc., nei casi in cui la distorsione è inaccettabile o indesiderata. La compressione lossless è significativamente più efficiente della compressione lossless. La compressione lossless è solitamente utilizzata per ridurre il volume di dati audio e video e foto digitali nei casi in cui tale riduzione è una priorità e non è richiesta la piena conformità dei dati originali e ripristinati.

La compressione dei dati è un modo per ridurre i costi di archiviazione eliminando le ridondanze che si verificano nella maggior parte dei file. Esistono due tipi di compressione, loss e lossless. La compressione loss ha ridotto le dimensioni del file eliminando alcuni dati non necessari che non saranno riconosciuti dall'uomo dopo la decodifica, spesso utilizzata dalla compressione video e audio. La compressione lossless, d'altro canto, manipola ogni bit di dati all'interno del file per ridurre al minimo le dimensioni senza perdere alcun dato dopo la decodifica. Questo è importante perché se il file perde anche un solo bit dopo la decodifica, significa che il file è corrotto.

La maggior parte dei metodi di compressione sono fisici e logici. Sono fisici perché guardano solo i bit nel flusso di input e ignorano il significato dei contenuti nell'input. Un metodo del genere traduce un flusso di bit in un altro, più breve. L'unico modo per comprendere e decodificare il flusso di output è sapere come è stato codificato. Sono logici perché guardano solo i singoli contenuti nel flusso sorgente e sostituiscono i contenuti comuni con codici brevi. Il metodo di compressione logico è utile ed efficace (raggiunge il miglior rapporto di compressione) su determinati tipi di dati [2].

Algoritmi correlati

A. Codifica della lunghezza di esecuzione

La codifica run-length (RLE) è una delle tecniche di base per la compressione dei dati. L'idea alla base di questo approccio è questa: se un elemento di dati *d* si verifica *n* volte consecutive nel flusso di input, sostituire le n occorrenze con la singola coppia *nd* [2]. RLE consiste nel comprimere esecuzioni dello stesso byte. Questo approccio è utile quando la ripetizione si verifica spesso all'interno dei dati. Ecco perché RLE è una buona scelta per comprimere un'immagine bitmap, in particolare quella a basso bit (ad esempio un'immagine bitmap a 8 bit).

B. Trasformazione di Burrows-Wheeler

La trasformata di Burrows-Wheeler (BWT) funziona in modalità blocco mentre altre funzionano principalmente in modalità streaming. Questo algoritmo è stato classificato come algoritmo di trasformazione perché l'idea principale è quella di riorganizzare (aggiungendo e ordinando) e concentrare i simboli. Questi simboli concentrati vengono utilizzati per un altro algoritmo per ottenere buoni rapporti di compressione. Poiché la BWT opera su dati in memoria, potrebbe incontrare file troppo grandi per essere elaborati in un colpo solo. In questi casi, il file deve essere suddiviso ed elaborato un blocco alla volta [3]. Per accelerare il processo di ordinamento, è possibile effettuare l'ordinamento parallelo o utilizzare un blocco di input più grande se è disponibile più memoria.

C. Spostarsi in avanti per trasformare

La trasformazione Move to Front (MTF) è un'altra tecnica di base per la compressione dei dati. MTF è un algoritmo di trasformazione che non comprime i dati ma può aiutare a ridurre la ridondanza a volte [5].
L'idea principale è di spostare in primo piano i simboli che si verificano di più, in modo che tali simboli abbiano un numero di output inferiore. Questa tecnica serve per implementare l'ottimizzazione per un altro algoritmo come la trasformata di Burrows-Wheeler.

D. Codifica aritmetica

La codifica aritmetica (ARI) utilizza un metodo statistico per comprimere i dati. Il metodo inizia con un certo intervallo, legge il file di input simbolo per simbolo e utilizza la probabilità di ogni simbolo per restringere l'intervallo. Specificare un intervallo più stretto richiede più bit, quindi il numero costruito dall'algoritmo cresce continuamente. Per ottenere la compressione, l'algoritmo è il seguente: un simbolo ad alta probabilità restringe l'intervallo meno di un simbolo a bassa probabilità, con il risultato che i simboli ad alta probabilità contribuiscono con meno bit all'output. La codifica aritmetica è un codificatore entropico ampiamente utilizzato, l'unico problema è la sua velocità, ma la compressione tende a essere migliore di quella che Huffman (altro algoritmo di metodo statistico) può ottenere [2]. Questa tecnica è utile per la sequenza finale della compressione dei dati tramite l'algoritmo di combinazione e fornisce il massimo per il rapporto di compressione.

Algoritmo modificato

La codifica J-bit (JBE) [8] funziona manipolando bit di dati per ridurre le dimensioni e ottimizzare l'input per un altro algoritmo. L'idea principale di questo algoritmo è di dividere i dati di input in due dati in cui il primo dato conterrà il byte originale diverso da zero e il secondo dato conterrà il valore di bit che spiega la posizione dei byte diversi da zero e zero. Entrambi i dati possono quindi essere compressi separatamente con un altro algoritmo di compressione dati per ottenere il massimo rapporto di compressione. Il processo di compressione può essere descritto come segue:

1. Legge l'input per byte, può essere qualsiasi tipo di file.
2. Determinare il byte letto come byte diverso da zero o pari a zero.
3. Scrivere un byte diverso da zero nei dati I e scrivere il bit '1' nei dati byte temporanei, oppure scrivere solo il bit '0' nei dati byte temporanei per un byte di input pari a zero.
4. Ripetere i passaggi da 1 a 3 finché i dati byte temporanei non vengono riempiti con 8 bit di dati.
5. Se i dati byte temporanei sono riempiti con 8 bit, scrivere il valore byte dei dati byte temporanei in data II.
6. Cancellare i dati byte temporanei.
7. Ripetere i passaggi da 1 a 6 fino a raggiungere la fine del file.
8. Per scrivere dati di output combinati:
   a) Scrivere dati di output combinati;
   ÿ) Scrivere i dati I.
   ÿ) Scrivere i dati II.
9. Se seguiti da un altro algoritmo di compressione, i dati I e i dati II possono essere compressi separatamente prima di essere combinati (facoltativo).

La figura 1 mostra il processo di compressione visiva per la codifica J-bit passo dopo passo. L'inserito la lunghezza di input originale viene utilizzata per le informazioni relative alla dimensione dei dati I e II nell'inizio dell'output.

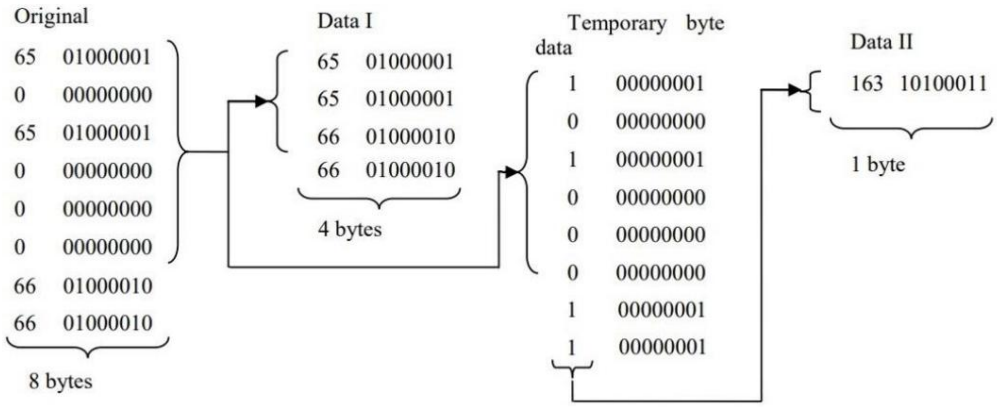

Figura 1. Processo di compressione passo dopo passo per la codifica J-bit

Il processo di decompressione può essere descritto come segue:

1. Leggere la lunghezza dell'input originale.
2. Se sono stati compressi separatamente, decomprimere i dati I e i dati II (facoltativo).
3. Leggere i dati II per bit.
4. Determinare se il bit letto è '0' o '1'.
5. Scrivere sull'output, se il bit letto è '1' allora leggere e scrivere i dati I sull'output, se il bit letto è '0' allora scrivere zero byte sull'output.
6. Ripetere i passaggi da 2 a 5 fino a raggiungere la lunghezza di input originale.

Combinazione di varianti

Sono state utilizzate quattro combinazioni di algoritmi di compressione dei dati per scoprire quale combinazione offre il miglior rapporto di compressione.

Le combinazioni sono:

1. BWT+MTF+GIORNO.
2. BWT+RLE+ARE.
3. RLE+BWT+MTF+RLE+ARI (come utilizzato in [3]).
4. RLE+BWT+MTF+JBE+ARI.

Tali combinazioni sono state testate con sei tipi di file. Ogni tipo è composto da 80 campioni. Ogni campione ha dimensioni diverse per mostrare le reali condizioni del file system. Tutti i campioni sono non compressi, questo include immagini bitmap raw e audio raw senza perdita di compressione.

No	Name	Qty	Type	Spec.
1	Image	80	Bitmap Image	Raw 8 bit
2	Image	80	Bitmap Image	Raw 24 bit
3	Text	80	Text Document	
4	Binary	80	Executable, library	
5	Audio	80	Wave Audio	Raw
6	Video	80	Windows Media Video	VBR

Figura 2. Campioni di esperimenti

Applicazione JBE

Il sistema di compressione dei dati strutturali si presenta in questo modo:

Dati sorgente –> Codificatore –> Dati compressi –> Decodificatore –> Dati recuperati

In questo schema: i dati generati dalla sorgente sono i dati sorgente e la loro rappresentazione compatta sono i dati compressi. Il sistema di compressione dei dati è costituito da un codificatore e un decodificatore sorgente. Il codificatore converte i dati sorgente in dati compressi e il decodificatore deve recuperare i dati sorgente dai dati compressi. I dati recuperati generati dal decodificatore possono corrispondere esattamente ai dati originali della sorgente o differirne leggermente.

Nei sistemi di compressione senza perdita di dati, il decoder recupera i dati sorgente in modo assolutamente accurato, quindi la struttura del sistema di compressione è la seguente:

Vettore dati X –> Codificatore –> B (X) –> Decodificatore - > X

Il vettore dei dati sorgente X da comprimere è una sequenza X = (1, 2, … ,) di lunghezza finita. Il campione (i componenti del vettore X) è stato selezionato dall'alfabeto finito di dati A. In questo caso, la dimensione del vettore di dati n è limitata, ma può essere arbitrariamente grande. Quindi, la sorgente al suo output forma come dati X una sequenza di lunghezza n dall'alfabeto A.

Il vettore dei dati sorgente X da comprimere è una sequenza *B(X)* = (1, 2, … ,), misurare che *k* riceve l'istruzione X. Chiama *B(X)* la parola di codice assegnata al vettore X dal codificatore (o dal codice parola in cui il vettore X è stato trasformato dal codificatore). Poiché il sistema di compressione è non distruttivo, lo stesso vettore = deve corrispondere alle stesse parole di codice *B( )= B( ).*

Implementare BWT+MTF+ARI

Sia la stringa di input "ABACABAÿ".

1. Velocità di crociera

La conversione è stata implementata in tre fasi. Nella prima fase, è stata compilata una tabella di tutti gli spostamenti ciclici della stringa di input. Nella seconda fase, è stato eseguito l'ordinamento lessicografico (in ordine alfabetico) delle righe della tabella. Nella terza fase, è stata selezionata l'ultima colonna della tabella di conversione in la riga di output. Il seguente esempio illustra l'algoritmo descritto:

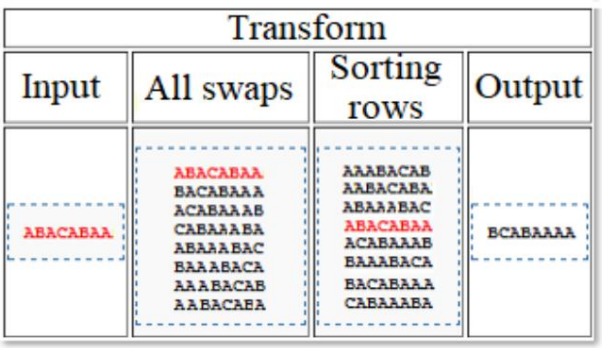

Figura 3. Algoritmo di conversione BWT

Pertanto, il risultato dell'algoritmo BWT(s) è "BCABAAAA".

2. MTF.

Inizialmente, ogni possibile valore di byte viene scritto in un elenco (alfabeto), in una cella con un numero uguale al valore di byte, ovvero (0, 1, 2, 3,..., 255). Questo elenco cambia man mano che i dati vengono elaborati. Quando arriva il carattere successivo, il numero dell'elemento contenente il suo valore è stato inviato all'output. Dopodiché, questo simbolo si sposta all'inizio dell'elenco, spostando gli elementi rimanenti verso destra.

Gli algoritmi moderni (ad esempio, bzip2) utilizzano l'algoritmo BWT prima dell'algoritmo MTF, quindi, ad esempio, considera la stringa S = "BCABAAAA" ottenuta dalla stringa "ABACABAA" come risultato della trasformazione Burroughs-Wheeler (ne parleremo più avanti). Il primo carattere della stringa S = "B" è il secondo elemento dell'alfabeto "ABC", quindi l'output è uno. Dopo aver spostato 'B' all'inizio dell'alfabeto, assume la forma "BAC". Ulteriori lavori dell'algoritmo:

Tabella 1 — Algoritmo di conversione MTF:

Simbolo	Lista	Produzione
B	ABC	1
C	BAC	2
UN	Contratto collettivo	2
B	ACB	2
UN	BAC	1
UN	ABC	0
UN	ABC	0
UN	ABC	0

Pertanto, il risultato dell'algoritmo MTF(S) è "12221000".

3. ARI.

Provando la codifica aritmetica otteniamo:

ARI =10111010011110100100100 0

Pertanto, se abbiamo a che fare con caratteri a otto bit, l'input è 8*8=64 bit e l'output è 24, ovvero il rapporto di compressione è del 62,5%.

Consideriamo lo stesso esempio, ma con l'aggiunta JBE - BWT+MTF+JBE+ARI:
I punti 1 e 2 sono uguali.

4. JBE.

Tabella 2-Algoritmo per l'applicazione della codifica JBE

Dati originali 1 1				Dati byte temporanei Dati 2			
	00000001 1		00000001 1 0	0000001 248			11111000
2	00000010 2		00000010 1		00000001 -		-
2	00000010 2		00000010 1		00000001 -		-
2	00000001 2		00000010 1		00000001 -		-
1	00000000 1		00000001 1 0	0	00000001 -		-
0	00000000 -		-	0	00000000 -		-
0	00000000 -		-		00000000 -		-
0	00000000 -		-		00000000 - In		-

output, abbiamo un record della lunghezza di input originale + record di dati I + record di dati II =24812221.

5. ARI.

Provando la codifica aritmetica otteniamo:

ARI =10111011101100110011011 0

Pertanto, se abbiamo a che fare con caratteri a otto bit, l'input è 8*8=64 bit e l'output è 24, ovvero il rapporto di compressione è del 62,5%.

Risultato

La figura 4 mostra che le immagini bitmap a 8 bit sono state compresse con un buon rapporto di compressione da algoritmi combinati con la codifica J-bit.

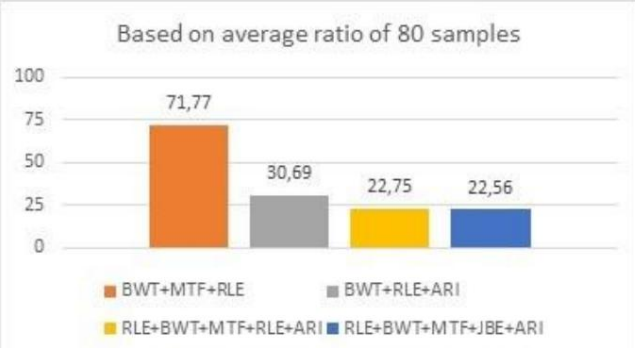

Figure 4. Diagram of the ratio of compression of 8-bit bitmaps between different algorithms

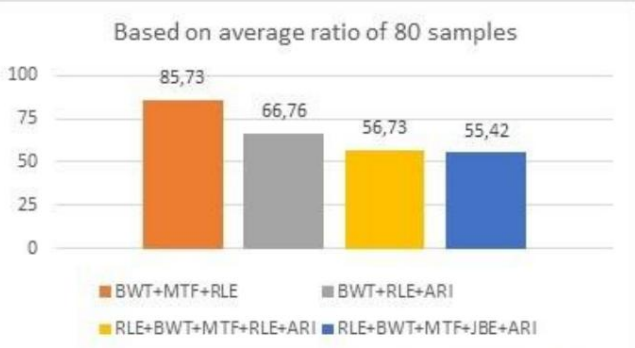

Figure 5. Diagram of the ratio of compression of 24-bit bitmaps between different algorithms

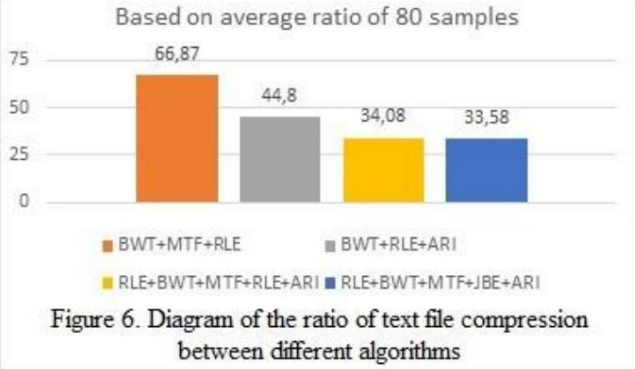

Figure 6. Diagram of the ratio of text file compression between different algorithms

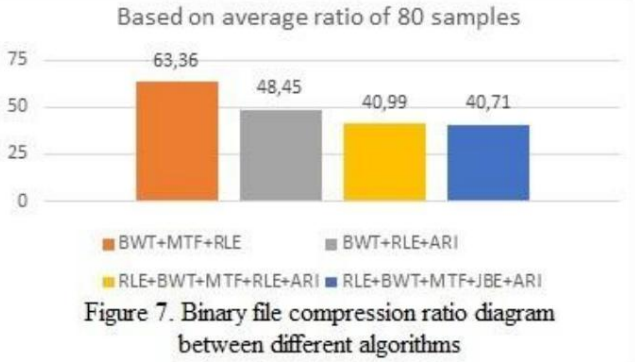

Figure 7. Binary file compression ratio diagram between different algorithms

La Figura 5 mostra che le immagini bitmap a 24 bit sono state compresse con un rapporto di compressione migliore tramite algoritmi combinati con la codifica J-bit. Un'immagine bitmap a 24 bit ha dati più complessi di 8 bit poiché memorizza più colore. La compressione a perdita per l'immagine sarebbe più appropriata per 24 bit

immagine bitmap per ottenere il miglior rapporto di compressione, anche se ciò diminuirà la qualità dell'immagine originale.

La figura 6 mostra: i file di testo sono stati compressi con un rapporto di compressione migliore grazie ad algoritmi combinati con la codifica J-bit.

La figura 7 mostra: i file binari sono stati compressi con un rapporto di compressione migliore grazie ad algoritmi combinati con la codifica J-bit.

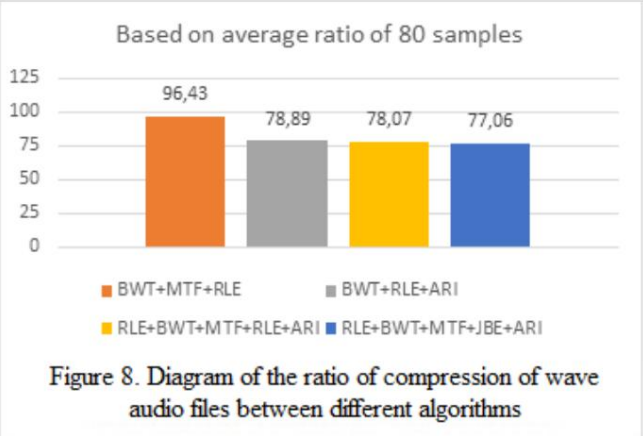

Figure 8. Diagram of the ratio of compression of wave audio files between different algorithms

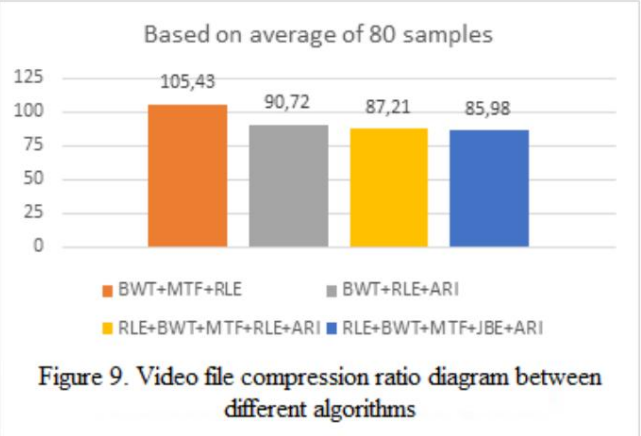

Figure 9. Video file compression ratio diagram between different algorithms

La figura 8 mostra: i file audio Wave sono stati compressi con un rapporto di compressione migliore grazie ad algoritmi combinati con la codifica J-bit.

La figura 9 mostra: i file video sono stati compressi con il miglior rapporto di compressione con gli algoritmi di codifica J-bit combinati.

CONCLUSIONE

È stato proposto l'algoritmo di compressione dati modificato. L'esperimento è stato condotto. Ha utilizzato sei tipi di file con 80 dimensioni diverse per ogni tipo di file. Di conseguenza, sono stati testati e confrontati 4 algoritmi combinatori. L'algoritmo proposto mostra un rapporto di compressione migliore dopo l'inserimento tra "trasformazione del movimento in avanti" (MTF) e codifica aritmetica (ARI). Lo studio fornisce sia la parte teorica che esempi pratici. L'algoritmo considerato ha la prospettiva di introdurre altri algoritmi di compressione dati nella struttura.

RIFERIMENTI

1. Capo-chichi, EP, Guyennet, H. e Friedt, J. K-RLE un nuovo algoritmo di compressione dati per reti di sensori wireless. In Atti della terza conferenza internazionale del 2009 sulle tecnologie e le applicazioni dei sensori.

2. Salomon, D. 2004. Data Compression the Complete References Terza edizione. Springer-Verlag New York, Inc.
3. Nelson, M. 1996. Compressione dei dati con la trasformata di Burrows-Wheeler. Dr. Dobb's Journal.
4. Campos, codifica ASE Run Length. Disponibile: http://www.arturocampos.com/ac_rle.html (ultimo accesso luglio 2012).
5. Campos, ASE si sposta in prima linea. Disponibile: http://www.arturocampos.com/ac_mtf.html (ultimo accesso luglio 2012).
6. Campos, ASE Codifica aritmetica di base. Disponibile: http://www.arturocampos.com/ac_arithmetic.html (ultimo accesso luglio 2012).
7. Springer, Handbook of Data Compression, quinta edizione.
8. Agus Dwi Suarjaya. Dipartimento di tecnologie dell'informazione, Università di Udayana. Bali, Indonesia. Un nuovo algoritmo per l'ottimizzazione della compressione dei dati.