JBE. CONCETTO E APPLICAZIONE

Lidia Mitkovets
Dottorato di ricerca presso l'Università statale bielorussa di informatica e radioelettronica in Lituania
Bielorussia

Daniele Sidorov
Dottorato di ricerca presso l'Università statale bielorussa di informatica e radioelettronica in Lituania
Bielorussia

Alevtina Gourinovitch
Dottorato di ricerca presso l'Università statale bielorussa di informatica e radioelettronica in Lituania
Bielorussia

Annotazione

Per salvare le informazioni all'interno dell'archiviazione, 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 della 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: compressione, codifica, codifica sorgente.

Introduzione

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

La compressione si basa sull'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 correlato al fatto che alcuni valori nei dati compressi sono più comuni di altri. La riduzione del volume di dati è ottenuta sostituendo i dati che si verificano di frequente con parole di codice brevi 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 ipotesi sulla trasformazione che ridurrebbe il volume del messaggio.

Il modello di ridondanza può essere statico, immutabile per l'intero messaggio compresso, oppure costruito o parametrizzato nella fase di compressione (e ripristino).

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

- Compressione senza perdita di dati
- Compressione con perdita

Quando si utilizza la compressione lossless, è possibile ripristinare completamente i dati originali, la compressione lossy 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 lossy, che è significativamente più efficiente della 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, lossy e lossless. La compressione lossy riduce 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 capire e

la decodifica del flusso di output avviene sapendo 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 logica è utile ed efficace (raggiunge il miglior rapporto di compressione) su determinati tipi di dati [1].

ALGORITMI CORRELATI

A. Codifica run-length

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 [1]. RLE è utilizzata principalmente per comprimere sequenze 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 trasformazione di Burrows-Wheeler (BWT) funziona in modalità blocco mentre altri 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 possono quindi essere utilizzati come input per un altro algoritmo per ottenere buoni rapporti di compressione. Poiché la BWT opera su dati in memoria, potresti imbatterti in file troppo grandi da elaborare in un colpo solo.

In questi casi, il file deve essere suddiviso ed elaborato un blocco alla volta [2]. 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 [4]. 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 è pensata per essere utilizzata come ottimizzazione per un altro algoritmo come la trasformazione 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 progettato in modo tale che un simbolo ad alta probabilità restringa 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 può ottenere Huffman (un altro algoritmo di metodo statistico) [1]. Questa tecnica è utile per la sequenza finale dell'algoritmo di combinazione di compressione dei dati e fornisce il massimo per il rapporto di compressione.

ALGORITMO PROPOSTO

La codifica J-bit (JBE) [7] 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 passo dopo passo può essere descritto come di seguito:

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 bit '0' in dati byte temporanei per byte di input 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 nei dati II.
6. Cancellare i dati byte temporanei.
7. Ripetere i passaggi da 1 a 6 fino a raggiungere la fine del file.
8. Scrivere i 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 visuale passo dopo passo per la codifica J-bit. La lunghezza di input originale inserita all'inizio dell'output verrà utilizzata come informazione per la dimensione dei dati I e II.

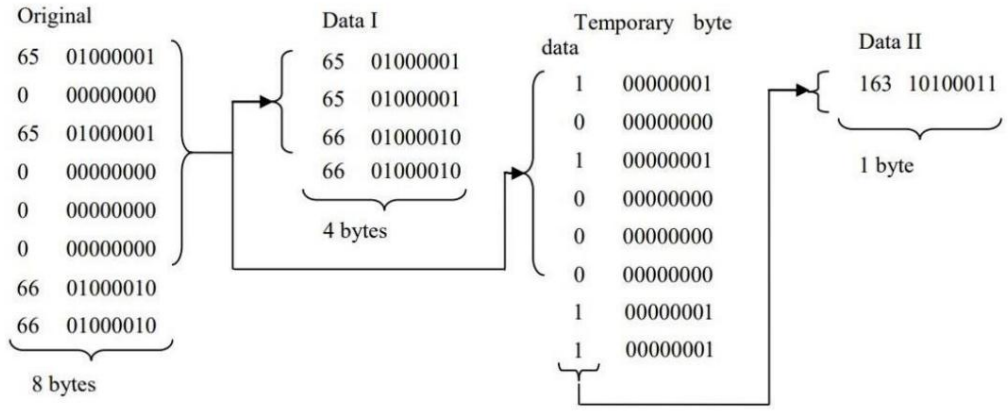

Figura 1. Processo di compressione passo dopo passo per la codifica J-bit

Di seguito è possibile descrivere passo dopo passo il processo di decompressione:

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.

CONFRONTO COMBINAZIONE

Vengono utilizzate quattro combinazioni di algoritmi di compressione dei dati per scoprire quale combinazione con 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 [2]).
4. RLE+BWT+MTF+JBE+ARI.

Tali combinazioni vengono testate con 6 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 compressione lossy.

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 per l'esperimento

APPLICAZIONE PRATICA DI 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 definiti come dati sorgente e la loro rappresentazione compatta è definita come dati compressi. Il sistema di compressione dei dati è costituito da un codificatore e da un decodificatore sorgente. Il codificatore converte i dati sorgente in dati compressi

dati, e il decoder è progettato per recuperare i dati sorgente dai dati compressi. I dati recuperati generati dal decoder possono corrispondere esattamente ai dati originali della sorgente, o differire leggermente da essi.

Nei sistemi di compressione lossless, il decodificatore 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 di dati sorgente X da comprimere è una sequenza X = ( ) di lunghezza finita. I campioni, ovvero i $x_1, x_2, ..., x_n$ componenti del vettore X, sono x_i selezionati dall'alfabeto finito di dati A. In questo caso, la dimensione del vettore di dati n è limitata, ma può essere arbitrariamente grande.
Pertanto, la sorgente in uscita 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) = ($b_1, b_2, ..., b_n$), il parametro k viene assegnato a X. Chiamiamo B(X) la parola di codice assegnata al vettore X dal codificatore (o la parola di codice in cui il vettore X viene trasformato dal codificatore). Poiché il sistema di compressione è non distruttivo, gli stessi vettori devono corrispondere alle stesse parole di codice B( ) = B( ). $X_l = X_m$ X_l X_m

ESEMPIO DI UTILIZZO DI BWT+MTF+ARI

Sia la stringa di input "ABACABAÿ".

1. Velocità di crociera

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

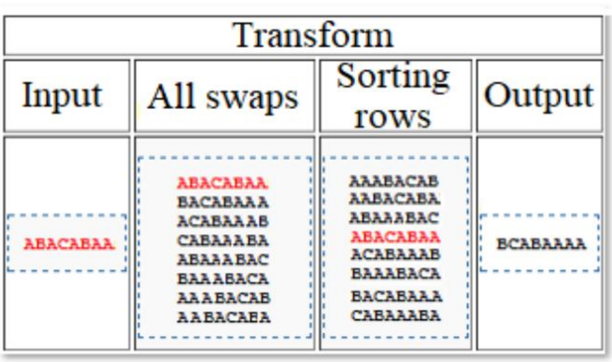

Figura 3. Algoritmo di conversione BWT

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 viene inviato all'output.
Dopodiché, il 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 1. 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(I)=101110100111101001001000

Pertanto, se abbiamo a che fare con caratteri a otto bit, l'input è 8*8=64 bit e l' la potenza è 24, cioè il rapporto di compressione è 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

Originale		Dati 1		Dati byte temporanei		Dati 2	
1	00000001	1	00000001	1	00000001	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	00000001	-	-
0	00000000	-	-	0	00000000	-	-
0	00000000	-	-	0	00000000	-	-
0	00000000	-	-	0	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(I)=101110111011001100110110

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

RISULTATO

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

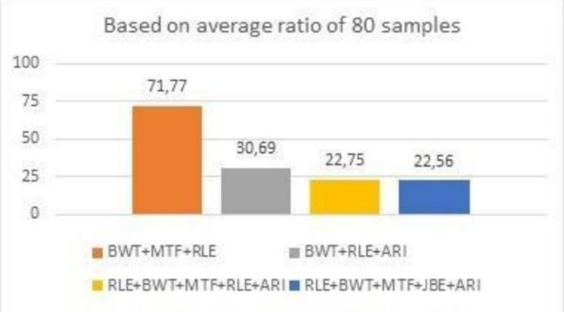

Figure 4. Diagram of the ratio of compression of 8-bit bitmaps between different algorithms

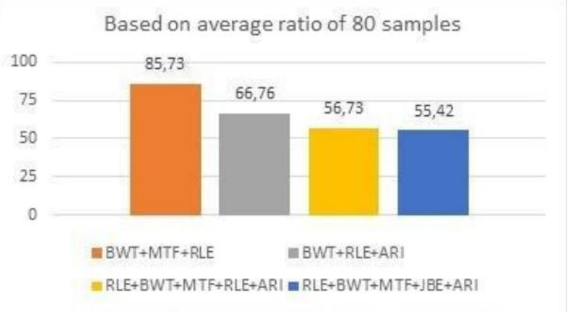

Figure 5. Diagram of the ratio of compression of 24-bit bitmaps between different algorithms

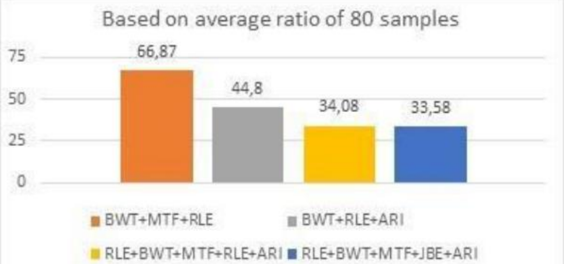

Figure 6. Diagram of the ratio of text file compression between different algorithms

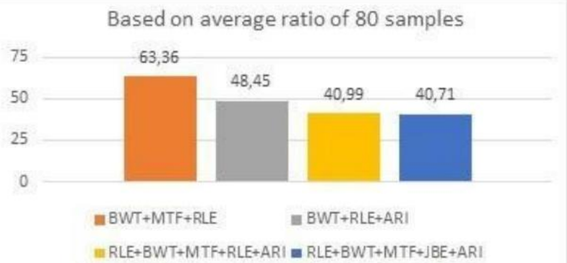

Figure 7. Binary file compression ratio diagram between different algorithms

La Figura 5 mostra che le immagini bitmap a 24 bit sono compresse con un rapporto di compressione migliore tramite algoritmi che si combinano con la codifica J-bit. Un'immagine bitmap a 24 bit ha dati più complessi di una a 8 bit poiché memorizza più colore. La compressione con perdita di dati per l'immagine sarebbe più appropriata per un'immagine bitmap a 24 bit per ottenere il miglior rapporto di compressione, anche se ciò diminuirà la qualità dell'immagine originale.

La figura 6 mostra che i file di testo vengono compressi con un rapporto di compressione migliore tramite algoritmi che combinato con la codifica J-bit.

La figura 7 mostra che i file binari vengono compressi con un rapporto di compressione migliore algoritmi combinati con la codifica J-bit.

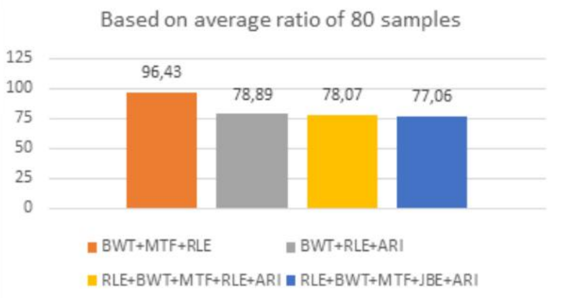

Figure 8. Diagram of the ratio of compression of wave audio files between different algorithms

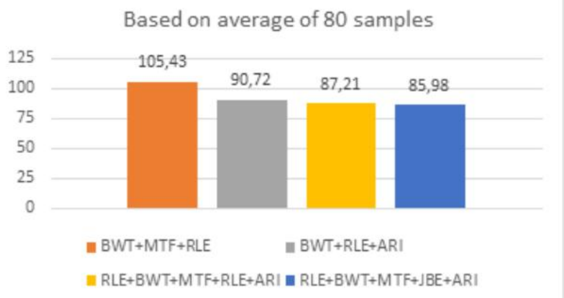

Figure 9. Video file compression ratio diagram between different algorithms

La figura 8 mostra che i file audio wave vengono compressi con un rapporto di compressione migliore algoritmi combinati con la codifica J-bit.

La figura 9 mostra che i file video vengono compressi con il miglior rapporto di compressione utilizzando algoritmi combinati con la codifica J-bit.

Conclusione

Quindi, nel corso dello studio, è stato proposto un algoritmo di compressione dati modificato ed è stato condotto un esperimento utilizzando 6 tipi di file con 80 dimensioni diverse per ogni tipo. Di conseguenza, sono stati testati e confrontati 4 algoritmi combinatori. L'algoritmo proposto fornisce un rapporto di compressione migliore quando inserito 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 dei dati nella struttura.

Riferimenti

1. Salomon, D. 2004. Data Compression: riferimenti completi, terza edizione. Casa editrice Springer-Verlag New York, Inc.

2. Nelson, M. 1996. Compressione dei dati con la trasformata di Burrows-Wheeler. Dr. Dobb's Journal.

3. David Salomon, 2010. Handbook of Data Compression Quinta edizione. Springer-Verlag New York, Inc.

4. Agus Dwi Suarjaya, 2012. Un nuovo algoritmo per l'ottimizzazione della compressione dei dati. Dipartimento di tecnologie dell'informazione, Università di Udayana. Bali, Indonesia.

Ricevuto: 23 maggio 2021
Accettato: 27 gennaio 2022