Un nuovo algoritmo per la compressione dei dati Ottimizzazione

Ho fatto Agus Dwi Suarjaya
Dipartimento di Tecnologia dell'Informazione
Università di Ulaanbaatar
Bali, Indonesia

Riassunto: **le persone tendono a conservare molti file nel loro archivio. Quando lo storage si avvicina al limite, cercano di ridurre al minimo le dimensioni di quei file utilizzando un software di compressione dati. In questo documento proponiamo un nuovo algoritmo per la compressione dati, denominato j-bit encoding (JBE). Questo algoritmo manipolerà ogni bit di dati all'interno del file per ridurre al minimo le dimensioni senza perdere dati dopo la decodifica, che è classificata come compressione lossless. Questo algoritmo di base è pensato per essere combinato con altri algoritmi di compressione dati per ottimizzare il rapporto di compressione. Le prestazioni di questo algoritmo vengono misurate confrontando la combinazione di diversi algoritmi di compressione dati.**

Parole chiave: algoritmi; compressione dei dati; codifica j-bit; JBE; senza perdite.

I. INTRODUZIONE

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 compressione dei dati può essere utilizzata anche per la tecnica di elaborazione in rete al fine di risparmiare energia poiché riduce la quantità di dati al fine di ridurre i dati trasmessi e/o diminuisce il tempo di trasferimento perché la dimensione dei dati è ridotta [1].

Esistono alcuni algoritmi di compressione dei dati ben noti. In questo documento daremo un'occhiata a vari algoritmi di compressione dati che possono essere utilizzati in combinazione con i nostri algoritmi proposti. Tali algoritmi possono essere classificati in algoritmi di trasformazione e compressione. L'algoritmo di trasformazione non comprime i dati, ma li riorganizza o li modifica per ottimizzare l'input per la successiva sequenza di algoritmo di trasformazione o compressione.

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 tramite sapendo come è stato codificato. Sono logici perché guardano solo i singoli contenuti nel flusso sorgente e sostituiscono i contenuti comuni con codici brevi. Compressione logica il metodo è utile ed efficace (raggiunge il miglior rapporto di compressione) su determinati tipi di dati [2].

II. 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* [2].

RLE viene utilizzato principalmente per comprimere sequenze dello stesso byte [3]. Questo approccio è utile quando la ripetizione avviene 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 possono quindi essere utilizzati come input per un altro algoritmo per ottenere buoni rapporti di compressione.

Poiché il 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 [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 è pensata per essere utilizzata come ottimizzazione per altri algoritmi 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 progettato in modo tale che 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 [2].

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 raggiungere Huffman (un altro algoritmo di metodo statistico) [6]. Questa tecnica è utile per la sequenza finale dell'algoritmo di combinazione della compressione dei dati e fornisce il massimo rapporto di compressione.

III. ALGORITMO PROPOSTO

La codifica J-bit (JBE) funziona manipolando bit di dati per ridurre le dimensioni e ottimizzare l'input per altri algoritmi. 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 altri algoritmi 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 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 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 dati di output combinati
   a) Scrivere la lunghezza dell'input originale.
   b) Scrivere i dati I.
   c) 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. Per quanto riguarda il processo di decompressione passo dopo passo, è possibile descriverlo di seguito:

1. Leggere la lunghezza dell'input originale.
2. Se è stato compresso separatamente, decomprimere i dati I e 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.

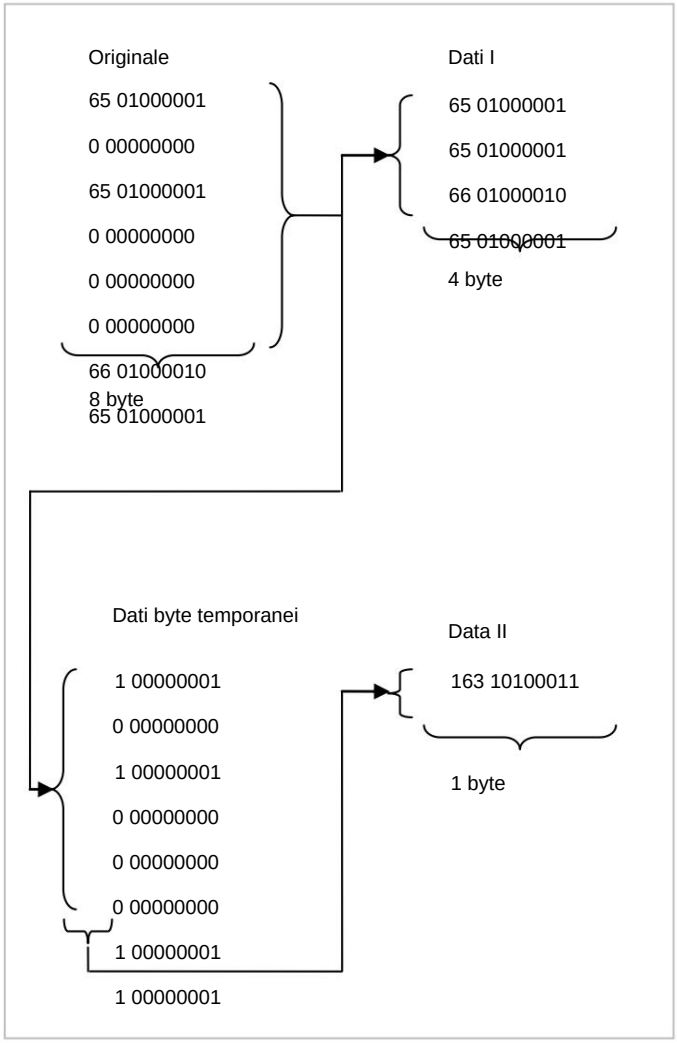

Figura 1. Processo di codifica J-bit

IV. CONFRONTO COMBINATO

Vengono utilizzate cinque combinazioni di algoritmi di compressione dei dati per scoprire quale combinazione offre il miglior rapporto di compressione. Le combinazioni sono:

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

Tali combinazioni sono testate con 5 tipi di file. Ogni tipo è composto da 50 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 dati.

compressione. Viene utilizzato il rapporto di compressione medio per ogni tipo di file. I campioni per l'esperimento sono mostrati nella tabella 1.

TABELLA I. CAMPIONI PER INPUT COMBINATO

Nessun nome		Quantità	Tipo	Specifiche
1	Immagine	50	Immagine bitmap	8 bit grezzo
2	Immagine	50	Immagine bitmap	Grezzo a 24 bit
3	Testo	50	Documento di testo	
4	Binario	50	Eseguibile, libreria	
5	Audio	50	Audio dell'onda	Crudo

V. RISULTATO

La figura 2 mostra che le immagini bitmap a 8 bit vengono compresse con un buon rapporto di compressione mediante algoritmi combinati con la codifica J-bit.

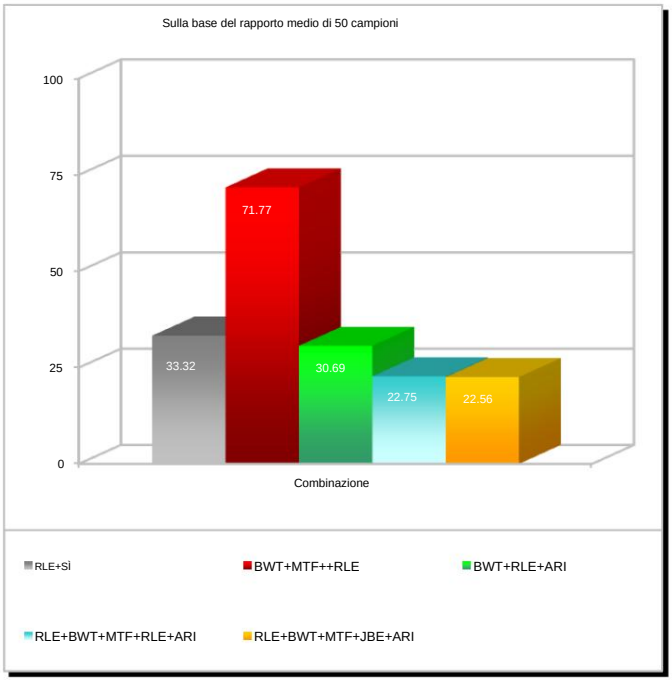

Figura 2. Confronto dei rapporti per l'immagine bitmap a 8 bit

La Figura 3 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.

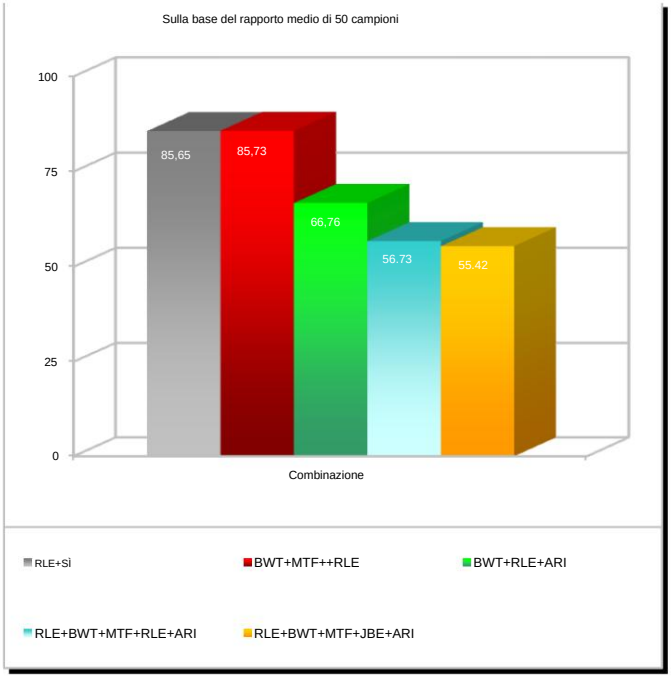

Figura 3. Confronto dei rapporti per l'immagine bitmap a 24 bit

La figura 4 mostra che i file di testo vengono compressi con un rapporto di compressione migliore mediante algoritmi combinati con la codifica J-bit.

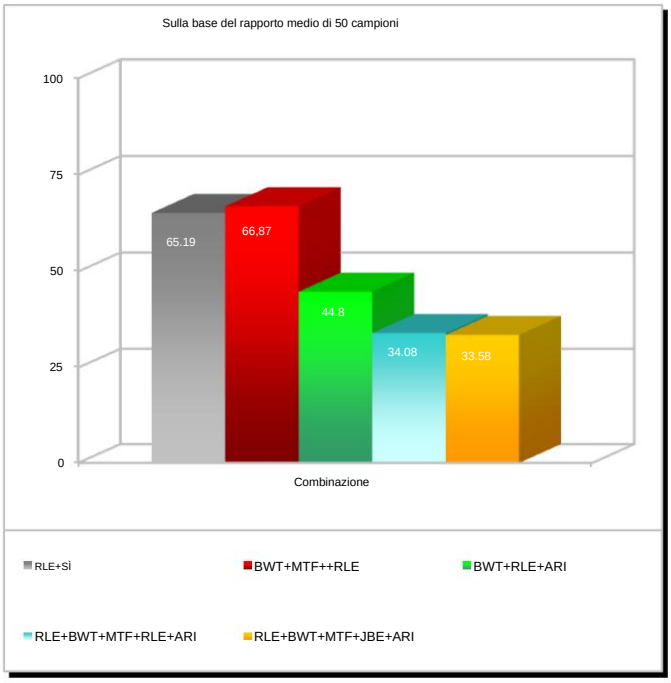

Figura 4. Confronto dei rapporti per il testo

La figura 5 mostra che i file binari vengono compressi con un rapporto di compressione migliore mediante algoritmi combinati con la codifica J-bit.

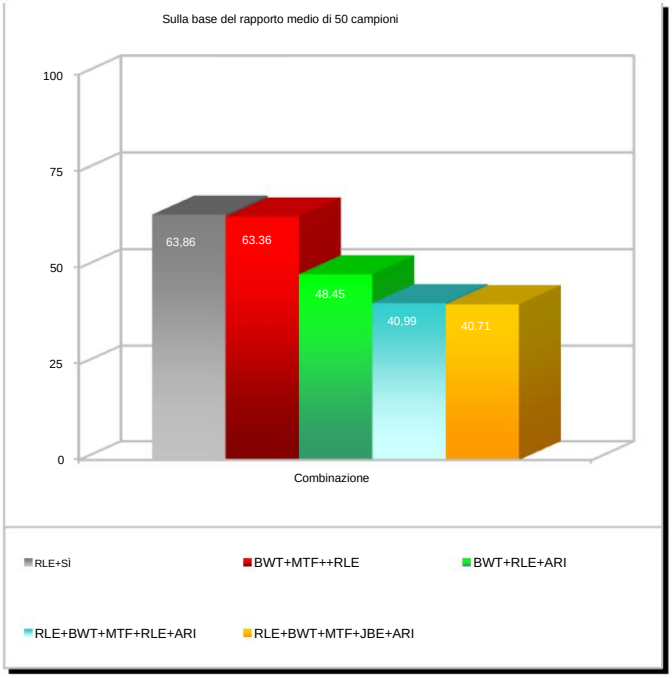

Figura 5. Confronto dei rapporti per binario

La figura 6 mostra che i file audio wave vengono compressi con un rapporto di compressione migliore mediante algoritmi combinati con la codifica J-bit.

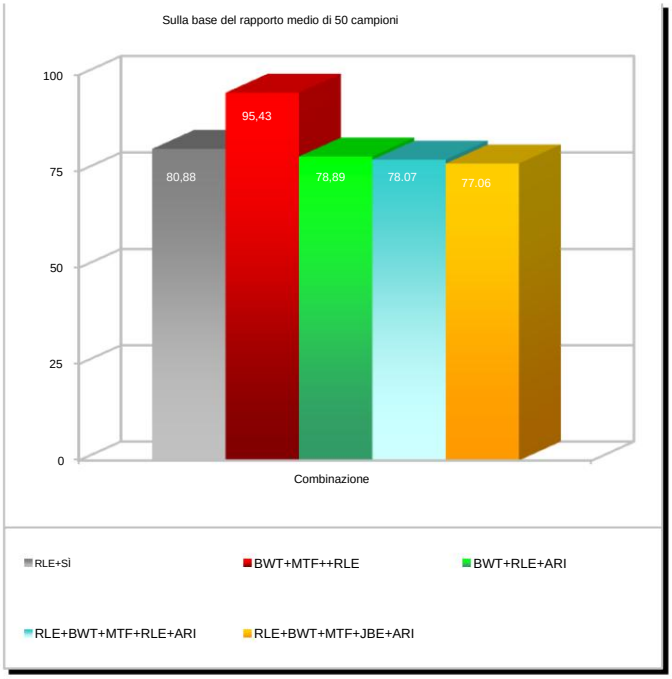

Figura 6. Confronto dei rapporti per onda

VI. CONCLUSIONE

Questo documento propone e conferma un algoritmo di compressione dati che può essere utilizzato per ottimizzare altri algoritmi. È stato condotto un esperimento utilizzando 5 tipi di file con 50 dimensioni diverse per ogni tipo, sono stati testati e confrontati 5 algoritmi di combinazione. Questo algoritmo fornisce un rapporto di compressione migliore quando inserito tra la trasformazione di spostamento in avanti (MTF) e la codifica aritmetica (ARI).

Poiché alcuni file sono costituiti da contenuti ibridi (testo, audio, video, binari in un unico file, proprio come i file di documento), la capacità di riconoscere i contenuti indipendentemente dal tipo di file, di dividerli e quindi di comprimerli separatamente con un algoritmo appropriato al contenuto è potenziale oggetto di ulteriori ricerche in futuro per ottenere un rapporto di compressione migliore.

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. Compressione dei dati i riferimenti completi Terzo Edizione. Springer-Verlag New York, Inc.

[3] Nelson, M. 1996. Compressione dei dati con la trasformata di Burrows-Wheeler. Dott. Il diario di Dobb.

[4] Campos, AS http:// E. Codifica della lunghezza di esecuzione. Disponibile: www.arturocampos.com/ac_rle.html (ultimo accesso luglio 2012).

[5] Campos, Fronte. UN. S. E. Spostarsi a Disponibile: http://www.arturocampos.com/ac_mtf.html (ultimo accesso luglio 2012).

[6] Campos, ASE Basic arithmetic coding. Disponibile: http://www.arturocampos.com/ac_arithmetic.html (ultimo accesso luglio 2012).

PROFILO AUTORI

I Made Agus Dwi Suarjaya ha conseguito la laurea triennale in Computer System and Information Science nel 2007 presso l'Università di Udayana e la laurea magistrale in Information Technology nel 2009 presso l'Università di Gadjah Mada. Ha lavorato come docente a tempo pieno presso la Facoltà di Ingegneria, Dipartimento di Information Technology presso l'Università di Udayana. I suoi interessi di ricerca includono l'ingegneria del software, il networking, la sicurezza, l'informatica, l'intelligenza artificiale, i sistemi operativi e i multimedia.