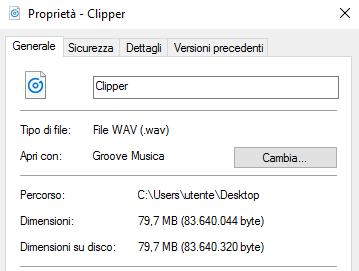
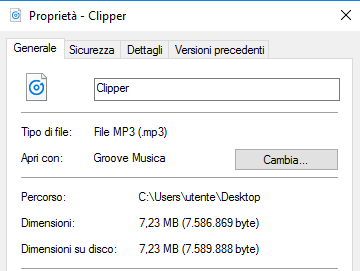
Visualizzazione di Wavelet Tree e strutture dati per la compressione

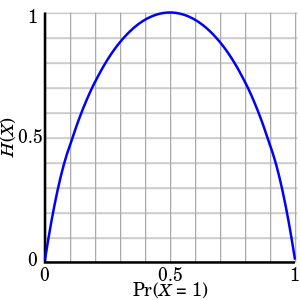
Overview Generale

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.  
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.

È possibile verificare quanto sia importante la compressione dei dati con un semplice esperimento. Solitamente la musica che ascoltiamo dai nostri computer e cellulari è in formato mp3 e lo spazio occupato per un minuto di canzone è solitamente un valore sull’ordine di 1Mb.  
Ad esempio prendiamo in considerazione un file mp3, contenente il pezzo Clipper degli Autechre, della durata di 7 min e 54 sec. Le dimensioni su disco in questo formato sono di appena 7,23 MB.  
Trasformando lo stesso file in WAV, un formato audio non compresso di codifica digitale, le dimensioni del file si decuplicano, arrivando a ben 79.7 MB.  
Anche se ci si trova in un epoca in cui lo spazio digitale non è uno dei problemi principali dell’informatica, basti pensare quanto piccole e poco costose siano diventate le schede di memoria, poter risparmiare il 90% dello spazio per memorizzare un file audio semplicemente tramite algoritmi è qualcosa di estremamente vantaggioso.



Un concetto correlato con la compressione di dati è quello di entropia. Il concetto di entropia è in comune con altri campi scientifici, come la termodinamica e la meccanica statistica e in entrambi i casi è come una sorta di “misura del disordine”. In particolare nella teoria dell’informazione è un indicatore dell’incertezza in una sorgente di informazione e può essere definita come l’informazione media contenuta in tali messaggi.  
 In un testo in italiano, alcune lettere come la ‘w’ o la ‘q’ non sono così frequenti come la ‘a’ o la ‘e’, quindi visto che diversi simboli hanno diverse probabilità di comparire, la sequenza di caratteri non è completamente casuale e l’entropia sarà dunque l’entità di misura di questa casualità.  
Sono i simboli meno frequenti a portare una maggiore quantità di informazione. Facendo un’analogia con il nostro linguaggio, in una relazione sull’agricoltura saranno parole come “pomodori”, “semina” e “raccolto” a farci capire qual è l’argomento del documento. Termini come “che”, “il” o “quindi” invece portano poca informazione, e se anche venissero tolte dal testo probabilmente non avremmo comunque problemi a capire di cosa si sta parlando.  
Per capire meglio la relazione tra entropia e probabilità di un evento consideriamo un caso elementare, ovvero una sorgente binaria X che ha probabilità p di produrre un 1, e probabilità 1-p di produrre uno 0.  
  
Come si può osservare dal grafico, man mano che la probabilità di produrre un 1 si avvicina ad uno dei due estremi, l’entropia diminuisce. Invece l’entropia raggiunge il massimo del valore consentito quando la probabilità di generare l’1 è del 50%, condizione in cui la generazione di 1 piuttosto che di 0 è completamente casuale.  
La misurazione dell’entropia può essere molto utile perché consente ai vari algoritmi di compressione di codificare con un numero minore di bit i simboli che appaiono più frequentemente.

La **codifica di Huffman** è un algoritmo per la compressione di dati che prende nome dall’omonimo cratore David A. Huffman.   
Per eseguire la traduzione dei simboli in codice si utilizza una tabella, in cui si assegna ai caratteri che appaiono più frequentemente una sequenza di bit più corta e viceversa.   
La codifica di Huffman usa un metodo specifico per scegliere la rappresentazione di ciascun simbolo, originando un codice prefisso, ovvero una sequenza di bit che rappresenta univocamente un simbolo e che non è un prefisso di nessun’altra sequenza di bit. Questa caratteristica è fondamentale perché consente di capire senza ambiguità in che posizione finisce la rappresentazione di un carattere all’interno della successione di bit, anche se i vari codici hanno una lunghezza differente.  
La tecnica utilizzata per ottenere questa struttura dati è l’algoritmo di Huffman, che consiste nella creazione di un albero binario a partire da una lista di nodi. Nella lista inizialmente vengono messi tutti i nodi foglia, che sono etichettati da due valori: il simbolo e il peso, che sarà proporzionale alla frequenza con cui si trova il simbolo. Ci saranno tanti nodi foglia quanta è la cardinalità dell’alfabeto in considerazione. Il peso dei nodi interni corrisponde alla somma dei pesi dei nodi figli.  
Il processo di costruzione dell’albero comincia creando un nodo interno che unisce le due foglie con peso minore. Questo nuovo nodo avrà quindi come figli le due foglie e come peso la somma del peso dei due figli. A questo punto si eliminano i figli dalla lista, e vi si inserisce il nuovo nodo creato. Questo procedimento si ripete fino a che rimane un solo nodo nella lista, che diventerà la radice dell’albero.  
In generale si può così riassumere il processo di creazione:

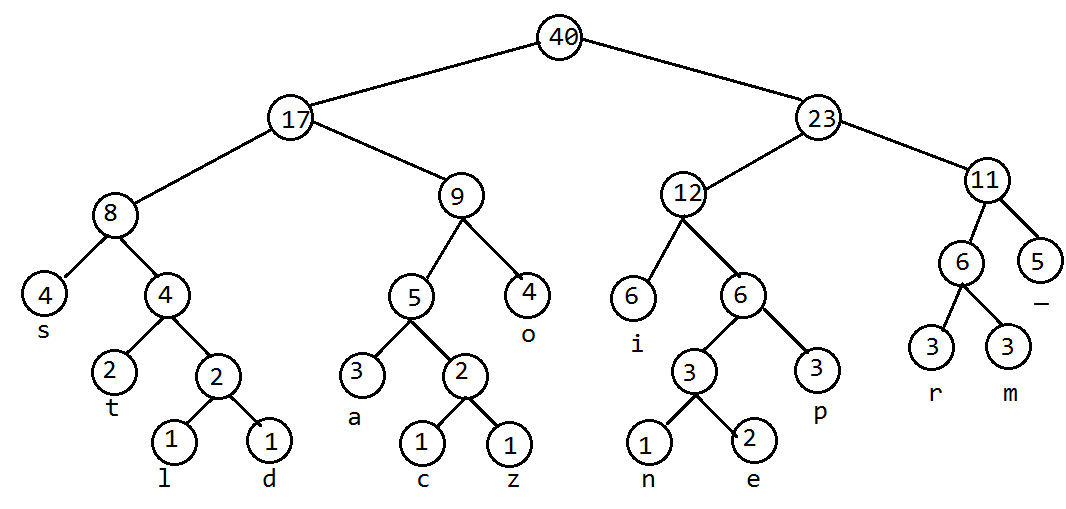
1. Creare un nodo foglia per ogni simbolo, associando il peso in base alla frequenza di apparizione e inserirlo in una lista ordinata in modo crescente.
2. Fino a che si hanno nodi nella lista:

* Eliminare i due nodi con peso minore.
* Creare un nodo interno che colleghi i nodi precedenti, assegnandogli come peso la somma dei pesi dei figli.
* Inserire il nuovo nodo nella giusta posizione della lista.

1. Il nodo che rimane diventa la radice dell’albero.

Ecco ora un esempio con la stringa “comprimendo\_il\_testo\_si\_risparmia\_spazio”

|  |  |  |
| --- | --- | --- |
| Carattere | Frequenza | Codice |
| i | 6 | 00 |
| \_ | 5 | 100 |
| o | 4 | 1100 |
| s | 4 | 1110 |
| m | 3 | 0100 |
| a | 3 | 0101 |
| r | 3 | 1010 |
| p | 3 | 1011 |
| t | 2 | 0110 |
| e | 2 | 0111 |
| z | 1 | 11010 |
| n | 1 | 110110 |
| c | 1 | 110111 |
| d | 1 | 111100 |
| l | 1 | 111101 |

Una volta ultimata la tabella si può procedere alla creazione dell’albero.

Si noti che numero delle foglie, 15, corrisponde alla cardinalità dell’alfabeto e che il peso della radice ha lo stesso valore della lunghezza complessiva della stringa.

Assumiamo di avere come input un alfabeto A = {a1, a2, …, an} di grandezza n e l’insieme W = {w1, w2, …, wn}  composto dai pesi dei vari simboli in cui vale wi = peso ai con 1<= i <= n.  
La quantità di informazione h presente in ogni simbolo ai con probabilità non nulla è tale che h(ai) = log2 1/wi.L’entropia, espressa in bit, è la somma ponderata della quantità di informazione di tutti i simboli ai con probabilità non nulla.entropia.png  
Per la decodifica di ogni simbolo bisogna attraversare l’albero da radice a foglia, operazione che richiede mediamente la visita di O(log n) nodi. La complessità temporale complessiva sarà dunque di O (k log n), attribuendo a k il valore della lunghezza totale del testo in considerazione.  
La complessità spaziale invece è O(k) per il testo codificato ed O(n) per l’albero, avendo questo una dimensione che è direttamente proporzionale al numero di caratteri di cui è costituito l’alfabeto da rappresentare.

La **codifica aritmetica** è una forma di codifica entropica utilizzata nella compressione di dati loseless. Se normalmente una stringa si rappresenta con un numero fisso di bit per carattere, con la codifica aritmetica i caratteri che appaiono più frequentemente sono rappresentati da un numero minore di bit rispetto a quelli che appaiono in maniera più sporadica.La codifica aritmetica si differenzia dalle altre forme di codifica entropica perché invece di suddividere l’input in simboli e sostituire ognuno di essi con un codice univoco, codifica l’intero messaggio in un solo numero decimale S compreso tra 0 e 1.  
L’uso di queste tecniche in campo commerciale è però poco diffuso per il fatto che molte delle implementazioni della codifica a partire dagli anni 80 sono protette da brevetti molto costosi e restrittivi. Le immagini di tipo JPEG, ad esempio, possono usare internamente sia la codifica di Huffman che la codifica aritmetica ma quest’ultima è quasi utilizzata, perché anche se molti dei brevetti sono ormai scaduti ci sono ancora pochi programmi e sistemi operativi che supportano automaticamente questa tecnica, che pure consentirebbe di risparmiare quasi il 15% di spazio senza alcuna perdita della qualità.  
 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.

Si prenda ad esempio la decodifica del numero binario 0.0011 (0.1875 in decimale ) con i seguenti intervalli di probabilità A = 0.3, B = 0.5, C = 0.2. Supponiamo che la stringa abbia la lunghezza pari a 4 caratteri.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Intervallo corrente | A | B | C | output |
| [0 , 1) | [0 , 0.3) | [0.3 , 0.8) | [0.8 , 1) | A |
| [0 , 0.3) | [0 , 0.09) | [0.09 , 0.24) | [0.24 , 0.3) | B |
| [0.09 , 0.24) | [0.09 , 0.135) | [0.135, 0.21) | [0.21 , 0.24) | B |
| [0.135, 0.21) | [0.135 , 0.1575) | [0.1575 , 0.195) | [0.195 , 0.21 | B |

Sarà quindi sufficiente scrivere il numero binario 0.0011 per codificare la stringa ABBB  
In generale in ogni passo del processo di decodifica il cifrario deve considerare solamente tre elementi:

* Il seguente simbolo da cifrare
* L’intervallo attuale, sempre impostato a [0 , 1) ma aggiornato ad ogni passo
* Le probabilità che il modello dà ad ognuno dei diversi simboli che è possibile incontrare

Il cifrario divide l’intervallo attuale in sottointervalli, ognuno dei quali rappresentato da una frazione dell’intervallo corrente proporzionale alla probabilità del simbolo nel contesto.  
Nell’esempio precedentemente riportato 0.1875 è racchiuso al primo passaggio nel primo intervallo (quello con il 30% di probabilità) e nei restanti tre passaggi dal secondo intervallo, quello con probabilità maggiore. In questo caso ovviamente si suppone di conoscere all’inizio la lunghezza del testo.

**Prediction by Partial Matching** (PPM) è un algoritmo adattivo di compressione dei dati basato su un modello di previsione statistica. Inventato da John G. Cleary e Ian H. Witten nel 1984, l’algoritmo iniziale fu oggetto di numerose modifiche e miglioramenti e oggi fa parte degli standard della compressione loseless. 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. L’algoritmo prende in considerazione, più modelli, ognuno caratterizzato da un ordine k, in cui k rappresenta il numero di caratteri precedenti consecutivi presi in considerazione.   
Se k = 0 il valore abbinato ai simboli sarà dato esclusivamente dalla sua frequenza del testo.  
Il numero massimo di ordini a cui si può arrivare è stabilito empiricamente prima di iniziare l’algoritmo, e di solito non si supera il limite di 5, che come si evince dal grafico sottostante è di solito il valore massimo più efficiente.

Ai vari simboli è assegnato un valore, che sarà direttamente proporzionale alla frequenza con cui tale simbolo appare in un determinato contesto.   
Quando si analizza il simbolo successivo next ci sono due possibilità:

* next è conosciuto, in questo caso si aggiorna solamente la probabilità di incontrare next nel dato contesto.
* next non è mai apparso in precedenza. Occorre quindi scrivere nella tabella delle probabilità il cosiddetto simbolo di fuga, che permette di regredire al sistema di ordine minore. Questa operazione, se necessario, viene ripetuta fino ad arrivare al modello di ordine 0, dove il simbolo viene codificato senza alcuna rielaborazione.

Ecco ora un esempio di come funziona il Prediction by Partial Matching dopo aver elaborato la stringa BANANA\_BANALE.

|  |  |  |
| --- | --- | --- |
| Ordine k = 2 | Ordine k = 1 | Ordine k =0 |
| BA 🡪 esc = 1/3  🡪 N = 2/3  AN 🡪 esc = 1/3  🡪 A = 2/3  NA 🡪 esc = 3/6  🡪 N = 1/6  🡪 \_ = 1/6  🡪 L = 1/6  A\_ 🡪 esc = 1/2  🡪 B = 1/2  \_B 🡪 esc = 1/2  🡪 A = 1/2  BA 🡪 esc = 1/3  🡪 N = 2/3  AL 🡪 esc = ½  🡪 E = ½ | B 🡪 esc = 1/3  🡪 A = 2/3  A 🡪 esc = 3/8  🡪 N = 3/8  🡪 \_ = 1/8  🡪 L = 1/8  N 🡪 esc = 1/4  🡪 A = 3/4  \_ 🡪 esc = 1/2  🡪 B = 1/2  L 🡪 esc = 1/2  🡪 E = 1/2 | 🡪 esc = 6/19  🡪 B = 2/19  🡪 N = 3/19  🡪 A = 5/19  🡪 \_ = 1/19  🡪 L = 1/19  🡪 E = 1/19 |

L’ordine massimo è stato impostato a 2 e si è indicato con “esc” il simbolo di fuga (escape character).

La **trasformata di Burrows-Wheeler (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. Si noti che questa trasformata non riduce l’entropia del messaggio ma riordina solo i bit per rendere altri algoritmi di compressione più efficienti.  
L’output della BWT non è altro che una permutazione della stringa originaria e potrà contenere lunghe sequenze di caratteri uguali posizionati adiacentemente. 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.  
Per ottenere la permutazione desiderata di una stringa è sufficiente eseguire tutte le possibili permutazioni della stringa e ordinarle in una colonna in ordine alfabetico. L’unione dell’ultimo carattere di tutte le stringhe presenti nella colonna costituisce l’output.  
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.  
Per compiere questa operazione su una stringa s di lunghezza n è innanzitutto necessario creare una tabella vuota. Poi si inserisce s verticalmente nella tabella (nel senso che si aggiunge un carattere a ognuna delle n righe della colonna) e si ordina la colonna in senso alfabetico crescente. Questa operazione, si ripete n volte, ovvero fino a quando si ottengono n stringhe di lunghezza n. A questo punto quale delle stringhe ha come ultimo carattere il simbolo di end of file e tale stringa sarà il valore di ritorno della funzione.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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  $MACACO | Output:  MACACO$ |

Si possono fare però delle ottimizzazioni a questo algoritmo in modo da farlo funzionare più efficientemente. Nella BWT non c’è necessità di rappresentare la tabella né nel processo di codifica né in quello di decodifica. Nella codifica ogni linea della tabella può essere rappresentata attraverso un unico puntatore tra stringhe e realizzare l’ordinamento mediante gli indici.

**Move to front (MTF)** è una trasformata che si usa per diminuire l’entropia di un input prima che venga codificato.   
In questo algoritmo, pubblicato per la prima volta da Ryabko Boris Yakovlevich nel 1980, i caratteri dei simboli in input sono sostituiti da degli indici. Inizialmente si crea una lista contenente i simboli dell’alfabeto del testo. Ad ogni passaggio si sostituisce il carattere in input con il relativo indice della lista e successivamente si modifica l’ordine dell’alfabeto spostando il simbolo in considerazione in posizione 0. In questo modo i caratteri che sono apparsi di recente saranno nelle prime posizioni dell’alfabeto, quindi anche i simboli che appaiono più frequentemente si dovrebbero trovare all’inizio della lista. È chiaro che se il testo è costituito da caratteri alfanumerici casuali non vi è nessuna riduzione dell’entropia e l’unico effetto della trasformata è un aumento dell’overhead.  
Anche in testi di senso compiuto un uso non adeguato di questo algoritmo può essere inutile o addirittura dannoso. Ad esempio nel monologo di Amleto sono stati calcolati 7033 bit di entropia. Applicando direttamente la trasformata l’entropia aumenta a 7807, più del 10% rispetto a prima. Invece applicando prima la trasformata di Burrows-Wheeler e poi MTF l’entropia si riduce a 6187 bit.  
Ecco ora un esempio con la stringa “macaco\_cattivo”.

|  |  |  |  |
| --- | --- | --- | --- |
| Simbolo in input | Indice | Alfabeto | Sequenza di indici |
| m | 3 | {a, c, i, m, o, t, v, \_} | 3 |
| a | 1 | {m, a, c, i, o, t, v, \_} | 31 |
| c | 2 | {a, m, c, i, o, t, v, \_} | 312 |
| a | 2 | {c, a, m, i, o, t, v, \_} | 3122 |
| c | 1 | {a, c, m, i, o, t, v, \_} | 31221 |
| o | 4 | {c, a, m, i, o, t, v, \_} | 312214 |
| \_ | 7 | {o, c, a, m, i, t, v, \_} | 3122147 |
| c | 2 | {\_, o, c, a, m, i, t, v} | 31221472 |
| a | 3 | {c, \_, o, a, m, i, t, v} | 312214723 |
| t | 6 | {a, c, \_, o, m, i, t, v} | 3122147236 |
| t | 0 | {t, a, c, \_, o, m, i, v} | 31221472360 |
| i | 6 | {t, a, c, \_, o, m, i, v} | 312214723606 |
| v | 7 | {i, t, a, c, \_, o, m, v} | 3122147236067 |
| o | 5 | {v, i, a, t, c, \_, o, m} | 31221472360675 |
| \ | \ | {o, v, i, a, t, c, \_, m} | **31221472360675** |

L’output della trasformata è 31221472360675.   
MTF è facilmente reversibile. Per ottenere la stringa originare dalla sequenza di interi è sufficiente ripetere in modo inverso le operazioni a partire dall’output e dall’alfabeto finale: la stringa verrà riprodotta recuperando carattere per carattere a partire dall’ultimo fino ad arrivare al primo.   
Ad ogni passo il carattere recuperato sarà il primo nell’alfabeto. Dopodiché si toglie l’ultimo indice dalla sequenza di interi e si modifica l’ordine dell’alfabeto facendo sì che il simbolo recuperato abbia tale indice all’interno della lista. Ripetendo questa operazione fino che ci sono interi nella lista si ottiene la stringa originale.  
La complessità temporale di questo algoritmo è nel peggior caso O(nk), dove si intende con n la lunghezza complessiva del testo e con k la cardinalità dell’alfabeto.

**Run-length encoding (RLE)** è un algoritmo loseless di compressione in cui sequenze di dati con lo stesso valore sono codificate con un valore unico unito ad un intero che rappresenta quante volte questo si ripete consecutivamente. È usato principalmente per contenuti audiovisivi e immagini ed è particolarmente efficiente nelle icone, dove capita sovente di trovare vari pixel adiacenti con lo stesso colore. L’uso di questo algoritmo non è affatto recente, già a partire dal 1967 infatti veniva utilizzato per la trasmissione di segnali televisivi.  
Prendiamo come esempio la stringa AAAAAAABBBACCCCBBBDDAAAAAAACCCCCCEEE in cui le diverse lettere maiuscole rappresentano i colori dei pixel di un’immagine 6x6.  
La stringa sarà codificata come 7A3B1A4C3B2D7A6C3E

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$

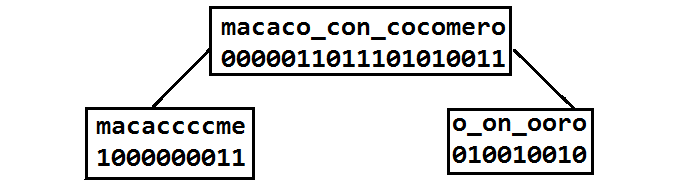
|  |  |  |  |
| --- | --- | --- | --- |
| Suffissi | indice | Suffissi ordinati | Indice |
| MACACO$  ACACO$  CACO$  ACO$  CO$  O$  $ | 1  2  3  4  5  6  7 | ACACO$  ACO$  CACO$  CO$  MACACO$  O$  $ | 2  4  3  5  1  6  7 |

Wavelet Tree

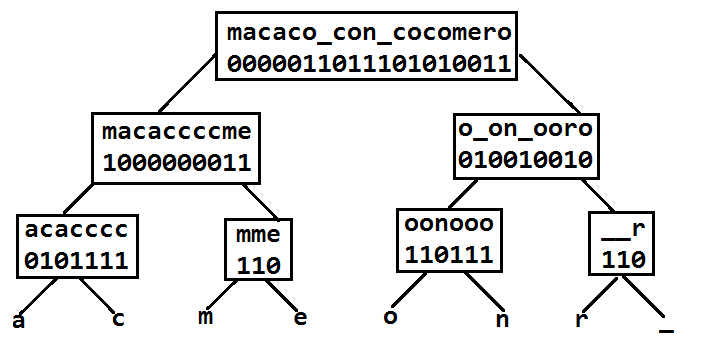
I wavelet treesono una struttura dati inventata nel 2003 da Roberto Grossi, Ankur Gupta e Jeffrey Scott Vitter. Il nome deriva dalla trasformata wavelet, uno strumento matematico per l’analisi e l’elaborazione dei segnali.  
Con i wavelet tree la stringa in input viene convertita in un albero binario bilanciato, dove ogni nodo è costituito da un vettore di bit. L’ennesimo bit di un nodo indica se il carattere rappresentato dal bit in posizione n appartiene alla prima o alla seconda metà dell’alfabeto costituito dall’insieme dei simboli rappresentati nel nodo. Si supponga per convenzione di rappresentare con il simbolo “0” i bit che appartengono alla prima metà del nodo e con il simbolo “1” i rimanenti. L’insieme di tutti i bit etichettati come 0 costituirà il contenuto del nodo figlio sinistro, analogamente tutti gli “1” comporranno il nodo figlio destro. Se il nodo è costituito da soli 1 e quindi vi è un solo carattere rappresentato in esso, allora le ricorsioni si fermeranno ed il nodo in questione sarà un cosiddetta foglia dell’albero.   
Assumiamo che la cardinalità dell’alfabeto in questione sia |∑|=n: la divisione per due dell’alfabeto ad ogni passaggio garantisce che l’altezza dell’albero corrisponda al logaritmo in base due di n, sempre arrotondato per eccesso.

Ecco ora un esempio dove “macaco\_con\_cocomero” è la stringa in input da rappresentare.

L’alfabeto della stringa, costituito da 8 caratteri {a, c, m, e, o, n, r, \_ }, deve essere ordinato, generalmente usando come criterio per questa operazione il valore numerico ASCII o Unicode associato ai simboli in questione.  
La stringa “macaco\_con\_cocomero” viene quindi inizialmente mappata come “0000011011101010011” in quanto i caratteri {a, c, m, e} verranno sostituiti con lo 0, mentre {o, n, r, \_ } saranno sostituiti con 1.



Si continua ricorsivamente con questo modus operandi fino ad arrivare alle foglie. L’altezza dell’albero quindi sarà log2 (8) = 3. Ecco il risultato finale:

  
I bit saranno l’unica informazione contenuta nel nodo, oltre ai vari puntatori per raggiungere i nodi figli ed il nodo padre all’interno dell’albero.

I motivi fondamentali però per cui si usano i wavelet tree è la possibilità di poter interrogare l’albero con le query rankq e selectq in modo efficiente.

Per Rank(Node root, char ch, int index) si intende la funzione che ha come valore di ritorno il numero di occorrenze del carattere ch fino all’indice index della stringa in input.  
 Se si vuole sapere il numero di occorrenze in un intervallo, ad esempio [3 ; 11], sarà sufficiente sottrarre al valore di Rank (root, ch, 11) il valore di Rank (root, ch, 3).

Bibliografia

* Wikipedia
* <http://alexbowe.com/>
* <https://scholarworks.iupui.edu/bitstream/handle/1805/2266/sekharthesis.pdf?sequence=1>
* <http://www.cs.waikato.ac.nz/ml/publications/1995/Cleary-Teahan-Witten-ppm.pdf>
* http://dictionnaire.sensagent.leparisien.fr/Move-to-front%20transform/en-en/