UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Codifica e Decodifica di caratteri ASCII Standard con Shannon-Fano

Relatore: Prof. Antonio Giunta

Laureando: Fabio Scarparo

ANNO ACCADEMICO 2021 – 2022

Data di laurea 07/03/2022

A me stesso
Perché nessuno più di me,
ci ha creduto fino in fondo,
nonostante le difficoltà.

Contenuti

1. Codifica dell'Informazione

L'informazione digitale viene rappresentata mediante un insieme finito di simboli; In particolare, nel caso di un calcolatore, essa deve essere costituita da sequenze di {0,1} per poter essere elaborata. Con il termine "codifica" si intende la rappresentazione dell'informazione mediante un insieme ben definito di simboli, generalmente minore del numero di valori da rappresentare.

Nel caso della codifica di testi, ossia di una codifica di caratteri, la codifica associa ad un insieme di caratteri (tipicamente rappresentazioni di grafemi così come appaiono in un alfabeto utilizzato per comunicare in una lingua naturale) un insieme di numeri con lo scopo di facilitare la memorizzazione di un testo in un elaboratore o la sua trasmissione attraverso una rete di telecomunicazioni.

Un esempio comune di questa codifica è la codifica ASCII (American Standard Code for Information Interchange), essa fornisce un identificativo binario univoco per 128 caratteri diversi e permette di mappare ogni carattere nella sua rappresentazione binaria.

1.1 Codifica a lunghezza fissa (fixed)

Con la codifica a lunghezza fissa tutti i caratteri sono codificati con stringhe di bit della stessa lunghezza. La lunghezza di questa codifica dipenderà quindi unicamente dalla dimensione dell'alfabeto di caratteri da codificare. Maggiore sarà il numero di caratteri da codificare maggiore sarà la lunghezza codifica binaria associata ad ogni carattere. Dato il numero N dell'alfabeto di caratteri, la lunghezza della codifica per ogni carattere può essere calcolata nel seguente modo

$$L = \lceil \log_2 N \rceil$$

Prendendo in esame la stringa "*mamma mia!*", notiamo che l'alfabeto di caratteri, che da ora in poi chiamerò "codewords" è composto dai caratteri {*m, a, i, !, spazio*}; le codewords sono 5, di conseguenza la codifica a lunghezza fissa utilizzerà una sequenza di 3 bit per ogni codeword.

Carattere	Occorrenze	Codifica
'm'	4	000
'a'	3	001
' '	1	010
'i'	1	011
'!'	1	100

La seguente codifica è giusta e ammissibile, in quanto la codifica è priva di prefissi, ovvero nessuna codifica associata ad una codeword è prefisso di una codifica di un'altra codeword. Questa è una condizione necessaria affinché una codifica senza perdita di informazione (lossless) sia corretta.

Il peso della stringa è dato dalla somma del numero di occorrenze di una codeword per il numero di bit associati alla codeword, in questo caso: 4*3 + 3*3 + 1*3 + 1*3 + 1*3 = 30

1.2 Codifica a lunghezza variabile (variable-length)

Le codifiche a lunghezza variabile sono state introdotte per ottimizzare il numero di bit inviati in un canale di comunicazione e vengono infatti utilizzate ai fini della compressione dei dati: dato un set di codewords, si assegnano codifiche più brevi a codewords più probabili.

Nel caso specifico della codifica di testi, una codifica a lunghezza variabile assegna un numero di bit inferiore ai caratteri che si presentano con frequenza maggiore e che quindi saranno i caratteri più utilizzati nel messaggio.

Riutilizzando l'esempio fatto precedentemente, nella stringa "*mamma mia*" la codeword '*m*' ha un numero di occorrenze ben superiore rispetto alle altre; Di conseguenza la sua frequenza sarà più alta e la codifica risulterà di lunghezza inferiore:

Carattere	Occorrenze	Frequenza	Codifica
'm'	4	0.4	0
'a'	3	0.3	10
' '	1	0.1	111
'i'	1	0.1	1100
'!'	1	0.1	1101

Si può notare che, anche in questo caso, la codifica è giusta e ammissibile essendo priva di prefissi, ma viene diminuito il peso della stringa che risulta 4*1 + 3*2 + 1*3 + 1*4 + 1*4 = 21

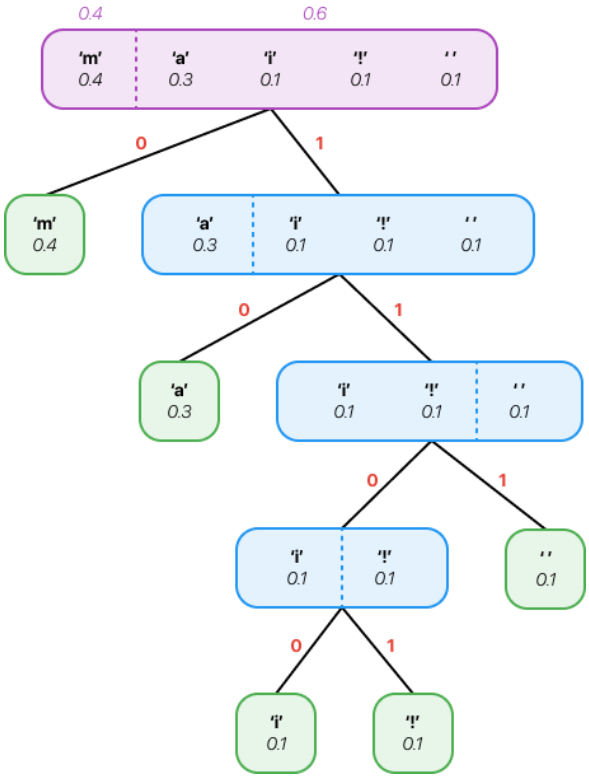
Questo risparmio sul peso è il motivo per il quale una codifica a lunghezza variabile, dal punto di vista dell'efficienza e della compressione, sia preferibile ad una codifica a lunghezza fissa.

2. Codifica di Shannon-Fano

Teorizzata e sviluppata dall'ingegnere statunitense Claudie Elwood Shannon e l'ingegnere italo-americano Roberto Mario Fano nel 1948, la codifica di Shannon-Fano è una codifica a lunghezza variabile basata sulle codewords e la loro frequenza. Si tratta di una codifica subottimale, ovvero non permette di raggiungere la lunghezza di bit minore possibile per ogni codeword come la codifica di Huffman; Tuttavia, è stato il punto di partenza da cui Huffman, 4 anni dopo, nel 1952, sviluppò la sua codifica ottimale. Entrambe le codifiche fanno uso di un albero ma, mentre l'albero della codifica di Shannon-Fano viene creato dividendo la radice in figli fino ad arrivare alle foglie, l'algoritmo di Huffman funziona nella direzione opposta, partendo dalle foglie fino ad arrivare alla radice.

Nella codifica di Shannon-Fano le codewords vengono ordinate in ordine decrescente di frequenza, quindi divise in due insiemi le cui probabilità totali sono il più possibile simili tra di loro. Al primo insieme è assegnato il valore "0" e al secondo il valore "1". Finché rimangono insiemi con più di una codeword, il processo viene ripetuto per costruire la codifica e quando l'insieme è stato ridotto ad una singola codeword, la codifica per quella codeword è stata completata.

L'algoritmo per la codifica di Shannon-Fano fa uso di un albero per implementare i passaggi descritti precedentemente, in particolare:



1. Partendo dalla radice dell'albero, tutte le codewords vengono ordinate in ordine decrescente di frequenza;
2. Le codewords presenti nella radice vengono suddivise in due parti per le quali la somma delle frequenze sia più simile possibile. Queste due parti saranno rispettivamente il figlio sinistro e destro della radice;
3. Si assegna uno "0" al cammino del figlio sinistro e un "1" al cammino del figlio destro;
4. Ripeto il punto 2 per ogni figlio fino ad arrivare a figli con un solo elemento, ovvero le foglie dell'albero, che saranno le codewords stesse;
5. La codifica per ogni codeword si otterrà percorrendo l'albero dalla radice fino alla foglia della codeword considerata e concatenando gli "0" e "1" che si incontrano durante il cammino.

Per meglio comprendere l'algoritmo che permette il calcolo della codifica di Shannon-Fano, è stata presa in esame la stringa "*mamma mia!*" utilizzata negli esempi precedenti e l'albero in figura implementa l'algoritmo descritto precedentemente.

3. Dominio applicativo del programma

Il programma in appendice è stato sviluppato in ANSI C, versione standard del linguaggio di programmazione C, ed implementa la codifica e la decodifica di file di testo utilizzando Shannon-Fano. Il programma elabora file di testo contenenti caratteri ASCII Standard (128 caratteri distinti) ad esclusione del carattere *Null*, identificato nella tabella ASCII Standard con il numero decimale 0.

4. Struttura e organizzazione dei file del programma

In appendice è dato un Makefile da collocare nella cartella principale per eseguire la compilazione automatica di tutti i file che compongono il programma. Per la corretta compilazione ed esecuzione del programma i file devono essere organizzati in una cartella nel seguente modo:

- Cartella *"Headers"* – Al suo interno saranno presenti i seguenti file con estensione *".h"* contenenti gli headers utilizzati dai file *".c"*:
    - *"utils.h":* Contiene gli headers delle funzioni secondarie ma fondamentali per la corretta esecuzione del programma. Al suo interno sono definite, le funzioni che permettono l'apertura dei file di testo in lettura e in scrittura e la funzione di ordinamento delle codewords in ordine decrescente di frequenza;
    - *"loadfunctions.h":* Contiene gli header delle funzioni adibite al caricamento delle codewords di un testo da codificare, al conteggio delle occorrenze di ogni codeword, al calcolo della frequenza di ognuna di queste ultime e al caricamento del testo da decodificare sottoforma di array di char in fase di decodifica;
    - *"printfunctions.h":* Contiene gli header delle funzioni adibite alla stampa su schermo e su file delle codeword e la loro frequenza, della codifica di Shannon-Fano e del testo codificato;
    - *"shannonfanotree.h":* Contiene la definizione della struttura dell'albero di Shannon-Fano e gli headers delle funzioni che generano i sotto-nodi partendo dalla radice, come mostrato precedentemente, per calcolare la codifica;
    - *"encode.h":* Contiene l'header della funzione che esegue la codifica dato il file contenente il testo da codificare;
    - *"decode.h":* Contiene la definizione della struttura ad albero che memorizzerà la struttura ad albero della codifica di Shannon-Fano partendo dal file contenente le informazioni della codifica per ogni codeword, l'header della funzione che esegue la decodifica e l'header di altre funzioni di supporto a quest'ultima.
- Cartella *"Support"* – Al suo interno saranno presenti tutti i file *".c"* che implementano le funzioni descritte nei file headers *".h"*.

- Cartella *"Output"* – Al suo interno, una volta lanciato il programma ed eseguita la codifica, verranno generati i file:
    - *"nomeinputfile_A&P.txt":* Contiene per ogni codeword la sua frequenza, nel formato {*codeword probabilità*};
    - *"nomeinputfile_SF.txt":* Contiene per ogni codeword la sua codifica di Shannon-Fano. Per la codeword "Carriage Return" nel formato {*"CR" codifica*}, per tutte le altre codewords nel formato {*codeword codifica*};
    - *"nomeinputfile_ET.txt":* Testo codificato con Shannon-Fano.
- Nella fase di decodifica, invece, nella cartella principale verrà generato il file:
    - *"nomeinputfile_DT.txt":* Testo decodificato.

5. Fasi del programma

5.1 Fase di avvio ed esempio di esecuzione

Una volta linkato e compilato il progetto mediate il Makefile, sarà disponibile l'eseguibile `SFC.exe`. L'eseguibile permette sia di codificare che di decodificare mediante l'utilizzo di opportuni parametri in fase di esecuzione.

Supponiamo di voler codificare il testo *"mamma mia!"* presente nel file *"MammaMia.txt"*, si dovrà eseguire il comando

```
SFC.exe e MammaMia.txt
```

Il risultato di questa operazione saranno i file *"MammaMia_A&P.txt"*, *"MammaMia_SF.txt"* e *"MammaMia_ET.txt"*, il cui contenuto è stato descritto precedentemente.

Per decodificare il testo codificato e quindi invertire il processo di codifica, basterà eseguire il comando

```
SFC.exe d MammaMia_ET.txt MammaMia_SF.txt
```

In questo modo verrà generato un file *"MammaMia_DT.txt"* al cui interno si troverà il testo del file originario.

5.2 Codifica

5.2.1 Computo delle frequenze

Il programma legge il file di input, conta il numero di caratteri totali e il numero di occorrenze per ogni carattere per poi calcolarne la frequenza. La frequenza per ogni carattere viene poi scritta nel file *"nomeinputfile_A&P.txt"*.

5.2.2 Generazione dell'albero di Shannon-Fano

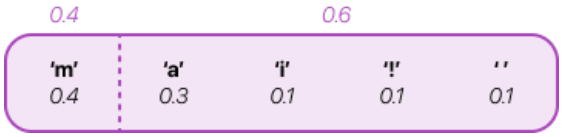
Una volta memorizzate le codeword in un array di char e le frequenze ad essere relative in un array di double e ordinate in ordine decrescente di frequenza, viene generato l'albero per la codifica di Shannon-Fano. L'albero è una struttura di nome "tree_node" contenente:

- `int nCodeword`, numero di codeword in un nodo;
- `int coding`, codifica associata al nodo che sarà 0 in un nodo sinistro, 1 in un nodo destro;
- `double codewordProb[]`, array contenente le frequenze di tutte le codeword nel nodo;
- `char codewordName[]`, array contenente il nome di tutte le codeword nel nodo;
- `int indexOfBalanceSX`, indice sinistro del bilanciamento delle frequenze in un nodo;
- `int indexOfBalancaDX`, indice destro del bilanciamento delle frequenze in un nodo;
- `double probsBalancedSX`, somma delle frequenze a sinistra dell'indice di bilanciamento;

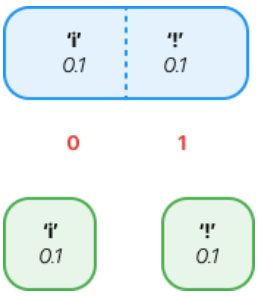
- `double probsBalancedDX`, somma delle frequenze a destra dell'indice di bilanciamento;
- `struct tree_node * left`, puntatore al nodo sinistro;
- `struct tree_node * right`, puntatore al nodo destro.

L'albero viene generato in modo top-down ricorsivamente, partendo dalla radice fino ad arrivare alle foglie. La generazione dell'albero è affidata alla funzione `computeShannonFanoTree` per l'inizializzazione del nodo radice e poi alla funzione ricorsiva `generateSubNodes` per la creazione dei nodi figli.

Prendendo in esame l'esempio del file contenente il testo da codificare *"mamma mia!"*, `computeShannonFanoTree` assegnerà le informazioni del nodo radice nella struct dell'albero



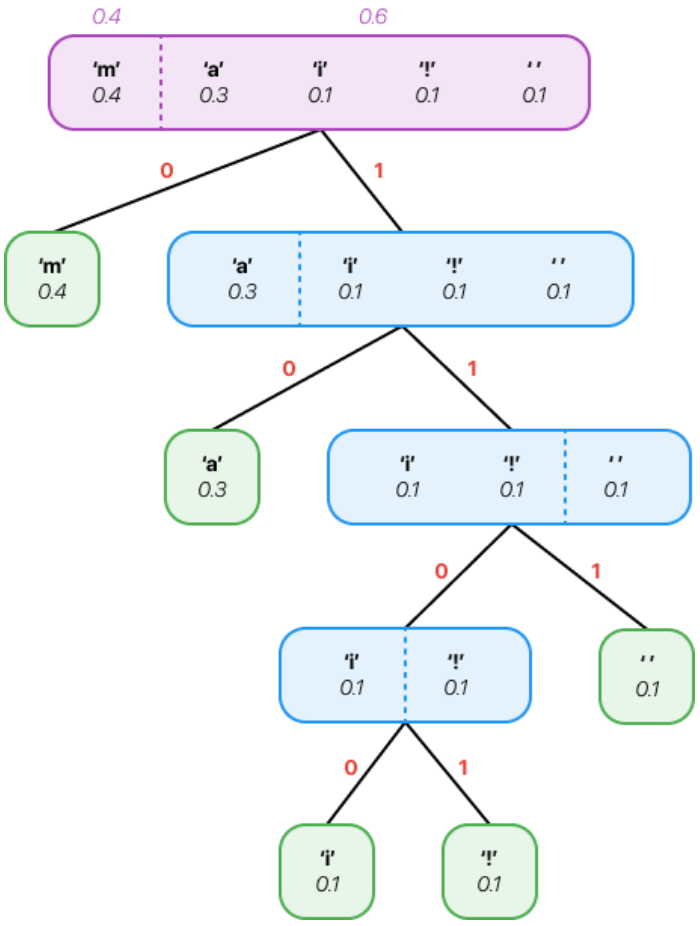
quindi, inizialmente, la struct `node_tree` sarà composta solo dalla radice. Una volta assegnati i valori della radice alla struct, viene richiamata la funzione `generateSubNodes` sull'albero in modo da generare in modo ricorsivo i figli. La generazione dei nodi figli terminerà quando si incontrerà un nodo foglia, ossia un nodo contenente solamente una codeword e che quindi avrà i figli sinistro e destro pari a `NULL`.



5.2.3 Memorizzazione della codifica per ogni codeword

La memorizzazione della codifica per ogni codeword viene fatta su un array di char bidimensionale chiamato `shannonFanoEncodingArray`, nel quale ad ogni riga è associata una codeword e ad ogni colonna uno 0 ed un 1 della codifica. La creazione dell'array bidimensionale è affidata alla funzione `computeShannonFanoEncodingArray` che, dato l'albero di Shanon-Fano e le codeword, richiama la funzione ricorsiva `recCodingBuilder`. Quest'ultima funzione non fa altro che usare un approccio umano al calcolo della codifica per ogni codeword, ossia, per ogni codeword, parte dalla radice e concatena 0 o 1 in base a ciò che trova nel cammino fino ad arrivare alla codeword interessata.

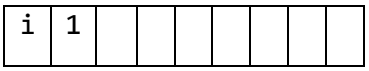
Per meglio comprendere come opera `recCodingBuilder` e la struttura dell'array `shannonFanoEncodingArray`, si prenda in considerazione l'albero che viene generato per codificare la stringa *"mamma mia!"*



Supponendo di voler calcolare la codifica di Shannon-Fano della codeword *"i"*, essa in notazione decimale ha valore 105, di conseguenza in riga 105 e in colonna 0 sarà presente la codeword *"i"* nell'array `shannonFanoEncodingArray`. La codifica associata a *"i"* viene calcolata esplorando l'albero dalla radice fino ad arrivare alla foglia relativa a quest'ultima, assegnando man mano in ogni indice di colonna gli 0 e gli 1 incontrati durante il cammino.

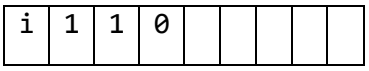
`shannonFanoEncodingArray` verrà quindi popolato, ricorsivamente, nel seguente modo e questa operazione verrà eseguita per ogni codeword presente nell'albero di Shannon-Fano

- Iterazione 1

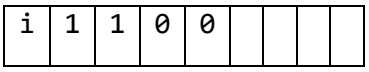


- Iterazione 2

- Iterazione 3



- Iterazione 4



Anche in questo caso, come per la generazione dell'albero di Shannon-Fano, il processo di calcolo della codifica termina quando, per ogni codeword, si è arrivati al nodo foglia, ossia un nodo contenente solamente una codeword e che quindi avrà i figli sinistro e destro pari a `NULL`.

5.2.4 Stampa della codifica per ogni codeword su file e stampa del testo codificato

La codifica di Shannon-Fano per ogni codeword viene stampata a schermo e sul file *"nomeinputfile_SF.txt"*. Viene successivamente stampato sul file *"nomeinputfile_ET.txt"* il testo codificato. Entrambi i file vengono creati grazie alle informazioni contenute nell'array popolato precedentemente.

5.3 Decodifica

5.3.1 Ricostruzione dell'albero di Shannon-Fano

La ricostruzione dell'albero è possibile grazie al file *"nomeinputfile_SF.txt"* sul quale è salvata per ogni codeword la codifica di Shannon-Fano. Ricostruire l'albero è fondamentale per poter decodificare carattere dopo carattere il testo codificato in fase di codifica.

Per poter ricostruire l'albero, viene utilizzata una struttura di nome "entry_node", ossia un albero denominato "Trie", in cui ogni nodo contiene le seguenti informazioni:

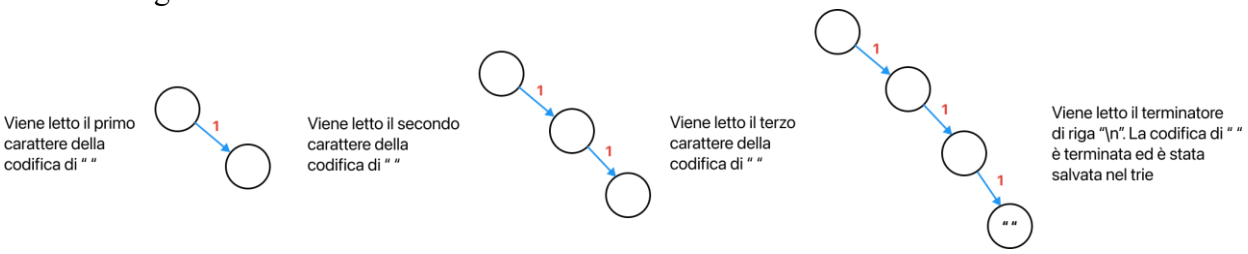
- `char c`, carattere associato alla codifica;
- `struct entry_node * zero`, puntatore al nodo sinistro, ossia uno zero;
- `struct entry_node * one`, puntatore al nodo destro, ossia un uno.

L'albero viene popolato andando a leggere riga dopo riga il contenuto del file con la codifica di Shannon-Fano per ogni carattere e, la codifica per ogni carattere viene memorizzata nella struttura mediante la funzione `addCharToTree`. Per meglio comprendere come opera quest'ultima funzione e come il trie venga popolato, si prenda in esame l'esempio del file *"MammaMia.txt"* contenente il testo da codificare *"mamma mia!"* che, una volta codificato, sarà la stringa *"0100010111011001011101"*.

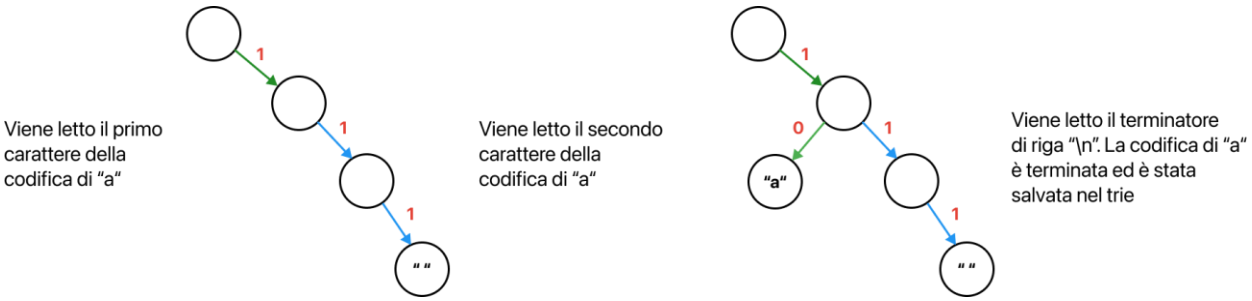
Il file *"MammaMia_SF.txt"* avrà la seguente struttura:

```
  111
a 10
! 1101
i 1100
m 0
```

Viene letta dal file la prima riga, ossia la codifica associata alla codeword " " e viene popolato il trie nel seguente modo:



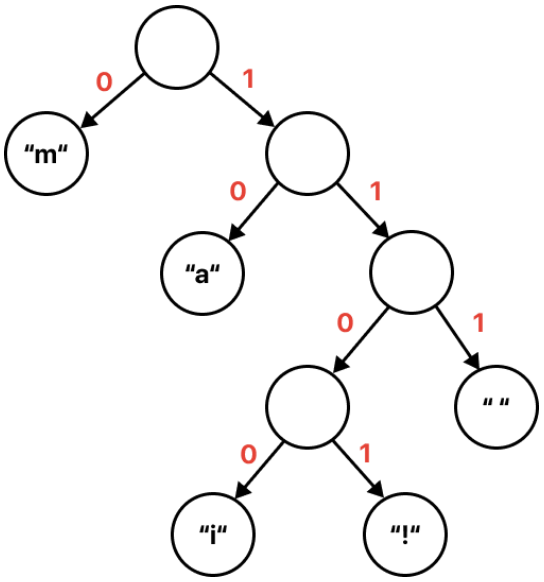
Viene successivamente letta dal file la seconda riga, ossia la codifica associata alla codeword *"a"* e viene aggiunta al trie nel seguente modo:



5.3.2 Decodifica del testo codificato e stampa su file

La decodifica vera e propria del testo codificato nel file *"nomeinputfile_ET.txt"* e il salvataggio di quest'ultima nel file di testo *"nomeinputfile_DT.txt"* è affidata alla funzione `printDecodedText`. Dato l'albero di Shannon-Fano visto precedentemente e dato il file contenente il testo codificato, la funzione legge carattere per carattere il testo codificato e, se incontra uno 0, scenderà nel nodo sinistro del trie, se incontra un 1, scenderà nel nodo destro del trie. La decodifica di una codeword sarà completata quando si arriverà ad un nodo foglia, nella quale è memorizzato, nella variabile `char c` della struct vista precedentemente, proprio il carattere a cui è associata la codifica. Arrivato al carattere, viene stampato il carattere sul file contenente il testo decodificato. Il processo si arresta quando non ci saranno più caratteri 0 e 1 da leggere.

Si prenda in esempio il testo codificato *"01000101110110010110 1"* che è associato alla stringa *"mamma mia!"*. Il trie associato alla codifica sarà il seguente



Viene letto il carattere 0, si scende nel nodo sinistro del trie che è un nodo foglia, allora si è arrivati alla fine della codifica di un carattere, ovvero del carattere "m", e viene stampato nel file. Si continua con il carattere successivo che è un 1. Si scende nel nodo destro, si legge il carattere 0, si scende nel nodo sinistro che è un nodo foglia, allora si è arrivati alla fine della codifica di un altro carattere, ovvero del carattere "a". Questo processo viene ripetuto fino alla fine della stringa del file codificato e in questo modo viene creato il file *"nomeinputfile_DT.txt"* contenente il file decodificato.

6. Osservazioni

Come discusso nell'introduzione, le codifiche a lunghezza variabile sono state introdotte per ottimizzare il numero di bit inviati in un canale di comunicazione e vengono infatti utilizzate ai fini della compressione dei dati. La codifica di Shannon-Fano, però, non è una codifica ottima, ovvero non assegna il numero di bit minore possibile ad ogni codeword come fa invece la codifica di Huffman.

Un'altra osservazione da fare è che, il programma non si occupa della vera e propria compressione del file di testo eseguendo operazioni sul bit, ma trasforma i caratteri ASCII Standard, da 1 byte ciascuno, in sequenze di caratteri 0 e 1 anch'esse da 1 byte ciascuno, appoggiandosi ad un file nel quale è salvata, per ogni codeword, la sua codifica. Ciò significa che, codificando il file, le dimensioni del file codificato saranno superiori a quelle del file originale e in fase di decodifica sarà comunque necessario avere il file contenente le codifiche per ogni codeword per poter decodificare correttamente.

7. Bibliografia

- Nevio Benvenuto, Michele Zorzi, *Principles of Communications Networks and Systems*, John Wiley & Sons Inc; 1ª edizione (26 agosto 2011)

8. Sitografia

- https://medium.com/@svm161265/why-encoding-required-in-digital-communication-480c83bbdf56
- https://www.oreilly.com/library/view/understanding-compression/9781491961520/ch04.html
- https://www.studytonight.com/advanced-data-structures/trie-data-structure-explained-with-examples

9. Appendice

File main.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Headers/encode.h"
#include "Headers/utils.h"
#include "Headers/decode.h"

/*
*Funzione principale che esegue il programma
*IP  argc  numero di argomenti passati da riga di comando
*IP  argv[]  nome dei file che utilizzerà il main in particolare:
*OR  0 se l'esecuzione va a buon fine, -1 se errore negli argomenti passati
*-2 se non è possibile aprire il file con il testo da codificare (se encode)
*o il file con il testo codificato e il file con la codifica di Shannon-Fano
(se decode)
*/
int main(int argc, char *argv[]) {

    int QCode = -1;
    if (checkInputArgs(argc, (const char **) argv)) {
        if(!strcmp(argv[1], "e")){
          QCode = encode(argv[2]);
        }else{
          QCode = decode(argv[2], argv[3]);
        }/*else*/
    }else{
      printf("Errore negli argomenti di input!\n");
      printf("Esmepio di esecuzione codifica: SFC e TextToCode.txt\n");
      printf("Esmepio di esecuzione decodifica: SFC d TextToCode_ET.txt
TextToCode_SF.txt\n");
      printf("Argomernto \"e\" per codifica, \"d\" per decodifica\n");
    }/*if*/

    printf("\n\nQCode: %d", QCode);
    return 0;
}/*main*/
```

File utils.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdbool.h>

#ifndef UTILS_H_
#define UTILS_H_

int openFile(FILE ** file, const char * name, char * type);
bool checkInputArgs(int argc, const char ** argv);
void closeFile(FILE * file);
void sort(double *probabilitiesArray, unsigned char *codewordsArray, int
dimArray);
void reverse(double *probabilitiesArray, unsigned char *codewordsArray, int
dimArray);
void convertIntArrToCharArr(int codewordCoding[], char stringcoding[], int
length);
void balanceProbabilities(double *balancingInfoArray, double
*probabilitiesArray, int dimArray);
int isCarriageReturnChar(int charASCIIcode);
int isCarriageReturnBuffer(char *line);

#endif /*UTILS_H_*/
```

File utils.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../Headers/utils.h"

/*
*Funzione che, dati i file, li chiude liberando la memoria
*IP  *file  puntatore al file da chiudere
*/
void closeFile(FILE * file){
  fclose(file);
}/*closeFiles*/

/*
*Funzione che, dati i file, li chiude liberando la memoria
*IP  **file  puntatore al puntatore al file da aprire
*IP  *name  nome del file
*OR  0 se l'apertura del file va a buon fine, 1 se no
*/
int openFile(FILE ** file, const char * name, char * type){

  /*Apro i file interessati*/
  *file = fopen(name,type);
  /*controllo che l'apertura vada a buon fine*/
  if(*file == NULL){
    return 1;
  }/*if*/

  return 0;

}/*openFile*/
```

```
/*
*Funzione che, dati i file, li chiude liberando la memoria
*IP  argc  numero di orgomenti passati da riga di comando
*IP  **argv  testo dell'argomento passato da riga di comando
*OR  true se argomenti corretti, false se non corretti
*/
bool checkInputArgs(int argc, const char ** argv){
  if((!strcmp(argv[1], "e") && argc == 3) || (!strcmp(argv[1], "d") && argc
== 4))
    return true;
  return false;
}/*checkInputArgs*/

/*
*Funzione che, dato l'array delle probabilità delle codewords e l'array del
nome delle codewords, le ordina in modo crescente
*IP  *probabilitiesArray puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP  *codewordsArray  puntatore all'array che contiene in ogni indice il
nome di ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void sort(double *probabilitiesArray, unsigned char *codewordsArray, int
dimArray){
  int i, j;
  unsigned char tempc;
  double tempd;

  for(i=0; i<dimArray; i++){
    tempc = codewordsArray[i];
    tempd = probabilitiesArray[i];
    j = i-1;

    /*muovo gli elementi dell'array che sono maggiori del valore di temp, di
una posizione più avanti*/
    while(j>=0 && probabilitiesArray[j] > tempd){
      probabilitiesArray[j+1] = probabilitiesArray[j];
      codewordsArray[j+1] = codewordsArray[j];
      j--;
    }/*while*/

    probabilitiesArray[j+1]=tempd;
    codewordsArray[j+1]=tempc;
  }/*for*/
}/*sort*/
```

```
/*
*Funzione che, dato l'array delle probabilità delle codewords e l'array del
nome delle codewords, le ordina in modo descrescente
*effettuandone il reverse
*IP  *probabilitiesArray puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP  *codewordsArray  puntatore all'array che contiene in ogni indice il
nome di ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void reverse(double *probabilitiesArray, unsigned char *codewordsArray, int
dimArray){

  int i=0;

  for(i=0; i<dimArray/2; ++i){
    unsigned char tempc;
    double tempd;
    /*ordino l'array delle codeword*/
    tempc = codewordsArray[i];
    codewordsArray[i] = codewordsArray[dimArray - 1 - i];
    codewordsArray[dimArray - 1 - i] = tempc;

    /*ordino l'array delle probabilità*/
    tempd = probabilitiesArray[i];
    probabilitiesArray[i] = probabilitiesArray[dimArray - 1 - i];
    probabilitiesArray[dimArray - 1 - i] = tempd;
  }/*for*/
}/*reverse*/

/*
*Funzione che, dato l'array delle probabilità delle codewords e l'array del
nome delle codewords, le ordina in modo descrescente
*effettuandone il reverse
*IP  codewordCoding  array di interi che conterrà "0" e "1" man mano che
viene analizzato il cammino dalla radice alla foglia
*IP  stringcoding  array di char che conterrà la codifica sottoforma di char
*IP  length  lunghezza della codifica di una data codeword
*/
void convertIntArrToCharArr(int codewordCoding[], char stringcoding[], int
length){

  int i=0;
  /*parto da 1 e non da 0 perchè nella codifica avrei anche "-1" che è la
codifica associata alla radice, ma a me non interessa*/
  for (i = 1; i < length; i++) {
    sprintf(&stringcoding[i], "%i", codewordCoding[i]);
  }/*for*/
}/*convertIntArrToCharArr*/
```

```
/*
*Funzione che, dato un array di double, calcola gli indici in mezzo ai quali
le probabilità delle codeword sono bilanciate
*e le informazioni relatrive al bilanciamento della somma delle probabilità
delle codewords a sinistra e a destra di questi indici
*IP *balancingInfoArray  puntatore all'array di double che conterrà le
informazioni sugli indici e sulle probabilita
*IP  *probabilitiesArray  puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto

ESEMPIO
  0       1   |    2       3       4
0.39    0.19  |  0.17    0.14    0.12

La somma è bilanciata tra l'indice 1 e 2, quindi
probabilities[0] = 1
probabilities[1] = 2
probabilities[2] = Somma parziale delle probabilita' a SX
probabilities[3] = Somma parziale delle probabilita' a DX
*/
void balanceProbabilities(double *balancingInfoArray, double
*probabilitiesArray, int dimArray){
  double sumTotal = 0;
  double partialTotalSX = 0;
  double partialTotalDX = 0;
  int indexOfBalance = 0;
  int i,j,k = 0;
  int dxStartIndex = 0;;

  for(i = 0; i < dimArray; i++){
    sumTotal = sumTotal + probabilitiesArray[i];
  }/*for*/

  /*sommo tutti gli elementi uno alla volta e ogni volta controllo se la
somma è minore o uguale alla metà della somma totale di tutti gli elementi
dell'array*/
  for(j = 0; j < dimArray; j++){
    if((partialTotalSX < (sumTotal/2)) && (partialTotalSX +
probabilitiesArray[j] < (sumTotal/2))){
      partialTotalSX = partialTotalSX + probabilitiesArray[j];

      indexOfBalance++;
    }/*if*/
  }/*for*/

  if(indexOfBalance==0){
    dxStartIndex=1;
  }
```

```
  /*calcolo la somma parziale delle probabilità a destra*/
  for(k = indexOfBalance-dxStartIndex; k < dimArray; k++){
    partialTotalDX = partialTotalDX + probabilitiesArray[k];
  }/*for*/

  if(indexOfBalance==0){
    /*se l'indice di bilanciamento è 0 sigifica che il nodo sarà bilanciato
tra gli indici 0 e 1, ossia un carattere a SX e i rimanenti a DX*/
    balancingInfoArray[0] = indexOfBalance;
    balancingInfoArray[1] = indexOfBalance+1;
    balancingInfoArray[2] = partialTotalSX + probabilitiesArray[0];
    balancingInfoArray[3] = partialTotalDX;
  }else{
    balancingInfoArray[0] = indexOfBalance-1;
    balancingInfoArray[1] = indexOfBalance;
    balancingInfoArray[2] = partialTotalSX;
    balancingInfoArray[3] = partialTotalDX;
  }
}/*balanceProbabilities*/

/*
*Funzione che, dato un carattere passato con casting ad intero, ritorna se
quel carattere è il Carriage Return oppure no
*IP  charASCIIcode  intero che rappresenta il cast ad intero di un char
*OR  1 se il carattere è il Carriage Return, 0 se non lo è
*/
int isCarriageReturnChar(int charASCIIcode){
  return charASCIIcode == 10;
}/*isCarriageReturnChar*/

/*
*Funzione che, dato un buffer (riga del file Shannon-Fano), ritorna 1 se è
presente un Carriage Return, 0 altrimenti
*Nel file "ShannonFano.txt" ad ogni codeword è associata la codifica,
separata da uno spazio, se però il carattere
*è un Carriage Return, allora non sarà del tipo   a 1100    ma bensì  "CR"
1011, quindi lo spazio tra il carattere
*e la codifica non si trova nella posizione 2 ma nella posizione 5
*IP  line  riga del file Shannon-Fano
*OR  1 se il carattere è il Carriage Return, 0 se non lo è
*/
int isCarriageReturnBuffer(char *line) {
    /*nel caso che il carattere sia speciale, l'ultima posizione dello
spazio sara' oltre l'indice 2*/
    /*strrchr ritorna il puntatore all'ultima posizione del carattere
passato per parametro nella riga in parametro*/
    return strrchr(line, ' ') - line > 2;
}/*isCarriageReturnBuffer*/
```

File loadfunctions.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#ifndef LOADFUNCTIONS_H_
#define LOADFUNCTIONS_H_

int countAndLoadCodewords(FILE *fileTextToCode, int *codewordsCounter,
unsigned char *codewordArray);
void computeAndloadProbabilities(int *codewordsCounter, unsigned char
*codewordArray, double *probabilitiesArray, int dimAlphabet);
char *loadEncodedTextToCharArray(FILE *fileEncodedText);

#endif /*LOADFUNCTIONS_H_*/
```

File loadfunctions.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "../Headers/loadfunctions.h"

/*
*Funzione che dato il file contenente il testo da codificare, costruiscre
l'array contenente le occorrenze di ogni codeword e l'array delle codeword
*Ritorna la dimensione dell'alfabeto, ossia il numero di caratteri unici nel
testo
*IP *fileTextToCode  puntatore al file contenente il testo da codificare con
Shannon-Fano
*IP  *codewordsCounter  puntatore all'array sul quale verranno memorizzate
per ogni codeword le occorrenze di ognuna di esse
*IP *codewordArray  puntatore all'array che contiene in ogni indice il nome
di ciascuna codeword
*OR  count  numero di caratteri unici nel testo, ossia la dimensione
dell'alfabeto delle codeword
*/
int countAndLoadCodewords(FILE *fileTextToCode, int *codewordsCounter,
unsigned char *codewordArray){

  unsigned char temp;
  int i=0;
  int k=0;
  int count=0;

  while(fscanf(fileTextToCode, "%c", &temp) != EOF){
    codewordsCounter[temp]+=1;
  }/*while*/

  for(i=0; i<128; i++) {
    if(codewordsCounter[i] > 0) {
      codewordArray[k] = i;
      k++;
      count++;
    }/*if*/
  }/*for*/

  rewind(fileTextToCode);
  return count;
}/*countAndLoadCodewords*/
```

```
/*
*Funzione che dato il file contenente l'array contenete le occorrenze di
ogni codeword, l'array che contiene il nome di ciascuna codeword e la
dimensione dell'alfabeto
*costruiscre l'array delle probabilità associate ad ognuna delle codeword
dell'alfabeto
*IP  *codewordsCounter  puntatore all'array nel quale sono memorizzate le
occorrenze di ogni codeword
*IP  *codewordArray  puntatore all'array che contiene in ogni indice il nome
di ciascuna codeword
*IP  *probabilitiesArray  puntatore all'array che conterrà le probabilità
relative a ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void computeAndloadProbabilities(int *codewordsCounter, unsigned char
*codewordArray, double *probabilitiesArray, int dimAlphabet){
  int totalChars=0;
  int i=0;
  int k=0;
  int frequency;

  /*conto le occorrenze di tutti i caratteri per calcolae i caratteri totali
del file da codificare*/
  for(i=0; i<128; i++) {
    if(codewordsCounter[i] > 0) {
      totalChars = totalChars + codewordsCounter[i];
    }/*if*/
  }/*for*/

  /*stampo a schermo la lunghezza del file in caratteri*/
  printf("Lunghezza testo: %d\n", totalChars);

  /*calcolo la probabilità (frequenza) di ogni codeword dell'alfabeto*/
  for(k=0; k<dimAlphabet; k++) {
    frequency = codewordsCounter[codewordArray[k]];
    probabilitiesArray[k] = (float)frequency/(float)totalChars;
  }/*for*/

}/*computeAndloadProbabilities*/
```

```
/*
*Funzione che dato il file contenente il testo codificato, crea un array
(buffer) contenente per ogni indice un carattere del testo codificato
*IP  *fileEncodedText  puntatore al file contenente il testo codificato con
Shannon-Fano
*OR  buffer  array di char contenente per ogni indice un carattere del testo
codificato
*/
char *loadEncodedTextToCharArray(FILE *fileEncodedText){

  long length; /*variabile che conterrà la lunghezza del testo*/
  char *buffer = 0; /*buffer*/

  fseek(fileEncodedText, 0, SEEK_END); /*porto il puntatore di lettura alla
fine del file*/
  length = ftell(fileEncodedText); /*salvo in length la posizione corrente
del puntatore di lettura per capire la lunghezza della stringa*/
  fseek(fileEncodedText, 0, SEEK_SET); /*riporto il puntatore di lettura
all'inizio*/
  buffer = (char *) malloc((length + 1) * sizeof(char)); /*alloco lo spazio
per il buffer*/
  assert(buffer);

  if (buffer) {
    /*sizeof(char) dimensione in byte del singolo dato essendo caratteri*/
    /*length numero di elementi da leggere*/
    /*fileEncodedText puntatore al file contenente il testo codificato*/
    fread(buffer, sizeof(char), length, fileEncodedText);
  }/*for*/

  /*alla fine aggiungo il carattere finale*/
  buffer[length] = '\0';

  /*ritorno il buffer*/
  return buffer;
}/*loadEncodedTextToCharArray*/
```

File printfunctions.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include "decode.h"

#ifndef PRINTFUNCTIONS_H_
#define PRINTFUNCTIONS_H_

void printProbabilities(double *probabilitiesArray, unsigned char
*codewordArray, int dimArray);
void printCodewords(unsigned char *codewordArray, int dimArray);
void printAlphabetInfoFile(FILE *fileAlphabet, double *probabilitiesArray,
unsigned char *codewordsArray, int dimArray);
void printShannonFanoEncodingArray(char ** shannonFanoEncodingArray);
void printShannonFanoInfoFile(FILE *fileShannonFanoCoding, char **
shannonFanoEncodingArray);
void printEncodedTextFile(FILE * fileTextToCode, FILE *fileEncodedText, char
** shannonFanoEncodingArray);
void printTextToCode(FILE * fileTextToCode);
void printDecodedText(EntryNode *tree, char *encodedText, FILE
*fileDecodedText);

#endif /*PRINTFUNCTIONS_H_*/
```

File printfunctions.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../Headers/printfunctions.h"
#include "../Headers/utils.h"
#include "../Headers/decode.h"

/*
*Funzione che, dato l'array di double contente le probabilità di tutte le
codeword, le stampa a schermo
*IP  *probabilitiesArray  puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP  *codewordsArray  puntatore all'array che contiene in ogni indice il
nome di ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void printProbabilities(double *probabilitiesArray, unsigned char
*codewordsArray, int dimArray){
  int i = 0;
  int j = 0;
  double sum = 0;
  printf("\nFREQUENZA\n");

  for(i=0; i<dimArray; i++){
    if(isCarriageReturnChar(codewordsArray[i])){
      /*stampo il nome del CARRIAGE RETURN*/
      printf("\"CR\": %.5lf\n",probabilitiesArray[i]);
    }else{
      printf("%c: %.5lf\n", codewordsArray[i],probabilitiesArray[i]);
    }/*if*/

  }/*for*/

  /*calcolo la somma delle probabilità*/
  for(j=0; j<dimArray; j++){
    sum = sum + probabilitiesArray[j];
  }/*for*/

  /*stampo la somma delle probabilità (che deve essere =1)*/
  printf("Somma frequenze: %.3lf\n", sum);
}/*printProbabilities*/
```

```
/*
*Funzione che, dato l'array di char contenente il nome di tutte le codeword,
lo stampa a schermo
*IP  *codewordsArray  puntatore all'array che contiene in ogni indice il
nome di ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void printCodewords(unsigned char *codewordsArray, int dimArray){

  int i = 0;
  printf("\nCODEWORDS\n");

  for(i=0; i<dimArray; i++){

    if(isCarriageReturnChar(codewordsArray[i])){
      /*stampo il nome del CARRIAGE RETURN*/
      printf("\"CR\" ");
    }else{
      printf("%c ",codewordsArray[i]);
    }/*if*/

  }/*for*/

  printf("\n");
}/*printCodewords*/
```

```
/*
*Funzione che, dato l'array delle probabilità e l'array delle codeword
stampa il file contenente le informazioni sull'alfabeto: CODEWORD
PROBABIILITA'
*IP  *fileAlphabet  puntatore al file sul quale verranno scritte le
informazioni sull'alfabeto, ossia ad ogni codeword la sua
probabilità/frequenza
*IP  *probabilitiesArray  puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP  *codewordsArray  puntatore all'array che contiene in ogni indice il
nome di ciascuna codeword
*IP  dimArray  dimensione dell'array delle probabilità, data dalla
dimensione dell'alfabeto
*/
void printAlphabetInfoFile(FILE *fileAlphabet, double *probabilitiesArray,
unsigned char *codewordsArray, int dimArray){

  int i = 0;
  for(i=0; i<dimArray; i++){

    if(isCarriageReturnChar(codewordsArray[i])){
      /*stampo il nome del CARRIAGE RETURN*/
      fprintf(fileAlphabet, "\"CR\" %.5lf\n", probabilitiesArray[i]);
    }else{
      fprintf(fileAlphabet, "%c %.5lf\n", codewordsArray[i],
probabilitiesArray[i]);
    }/*if*/

  }/*for*/

}/*printAlphabetInfo*/
```

```
/*
*Funzione che, dato l'array conenente le codifiche di ogni codeword
dell'alfabeto, stampa a schermo la codifica relativa ad ogni codeword
*IP  **shannonFanoEncodingArray  puntatore ad un puntatore all'arrary di
stringhe contenente la codifica di ogni codeword dell'alfabeto
*/
void printShannonFanoEncodingArray(char **shannonFanoEncodingArray){

  int i = 0;
  int j = 0;

  for (i=0; i < 128; i++) {

    /*se incontro una codeword dell'alfabeto*/
    if(shannonFanoEncodingArray[i] != '\0'){
        /*se è un CARRIAGE RETURN*/
        if(isCarriageReturnChar(i)){
          /*stampo il nome del CARRIAGE RETURN*/
          printf("CODIFICA DI \"CR\": ");
        }else{
          /*stampo il nome della codeword*/
          printf("CODIFICA DI %c: ",i);
        }/*if*/

        /*stampo la codifica associata alla codeword*/
        /*il ciclo verrò ripetuto tante volte quanti sono gli 0 e 1 della
codifica associata alla codeword*/
        /*parto da 1 e non da 0 perchè nella codifica avrei anche "-1" che è
la codifica associata alla radice, ma a me non interessa*/
        for (j=1; j < strlen(shannonFanoEncodingArray[i]); j++) {
          printf("%c",shannonFanoEncodingArray[i][j]);
        }/*for*/

      printf("\n");

    }/*if*/
  }/*for*/
}/*printShannonFanoEncodingArray*/
```

```
/*
*Funzione che, dato l'array conenente le codifiche di ogni codeword
dell'alfabeto, costruisce il file a cui ad ogni codeword è associata la sua
codifica di Shannon-Fano
*IP  *fileShannonFanoCoding  puntatore al file sul quale verranno scritte
per ogni codeword la codifica di Shannon-Fano
*IP  **shannonFanoEncodingArray  puntatore ad un puntatore all'arrary di
stringhe contenente la codifica di ogni codeword dell'alfabeto
*/
void printShannonFanoInfoFile(FILE *fileShannonFanoCoding, char
**shannonFanoEncodingArray){

  int i = 0;
  int j = 0;

  for (i=0; i < 128; i++) {

    /*se incontro una codeword dell'alfabeto*/
    if(shannonFanoEncodingArray[i] != '\0'){
        /*se è un CARRIAGE RETURN*/
        if(isCarriageReturnChar(i)){
          /*stampo il nome del CARRIAGE RETURN*/
          fprintf(fileShannonFanoCoding, "\"CR\" ");
        }else{
          /*stampo il nome della codeword*/
          fprintf(fileShannonFanoCoding,"%c ",i);
        }/*if*/

        /*stampo la codifica associata alla codeword*/
        /*il ciclo verrò ripetuto tante volte quanti sono gli 0 e 1 della
codifica associata alla codeword*/
        /*parto da 1 e non da 0 perchè nella codifica avrei anche "-1" che è
la codifica associata alla radice, ma a me non interessa*/
        for (j=1; j < strlen(shannonFanoEncodingArray[i]); j++) {

fprintf(fileShannonFanoCoding,"%c",shannonFanoEncodingArray[i][j]);
        }/*for*/

      fprintf(fileShannonFanoCoding,"\n");

    }/*if*/
  }/*for*/
  rewind(fileShannonFanoCoding);
}/*printShannonFanoInfoFile*/
```

```
/*
*Funzione che, dato il file contenente la parola/frase da codificare con
Shannon-Fano, la stampa a schermo.
*IP  *fileTextToCode  puntatore al file contenente la parola/frase da
codificare con Shannon-Fano
*/
void printTextToCode(FILE *fileTextToCode){

  char textChar;

  while(fscanf(fileTextToCode,"%c",&textChar) != EOF){
    printf("%c",textChar);
  }/*while*/

  rewind(fileTextToCode);
}/*printTextToCode*/

/*
*Funzione che, dato il file contenente la parola/frase da codificare con
Shannon-Fano e l'arrary di stringhe contenente la codifica di ogni codeword
dell'alfabeto
*stampa il file con il testo codificato
*IP  *fileTextToCode  puntatore al file contenente la parola/frase da
codificare con Shannon-Fano
*IP  *fileEncodedText  puntatore al file che conterrà la parola/frase
codificata con Shannon-Fano
*IP  **shannonFanoEncodingArray  puntatore ad un puntatore all'arrary di
stringhe contenente la codifica di ogni codeword dell'alfabeto
*/
void printEncodedTextFile(FILE *fileTextToCode, FILE *fileEncodedText, char
**shannonFanoEncodingArray){

  unsigned char textChar;
  int i = 0;

  while(fscanf(fileTextToCode,"%c",&textChar) != EOF){
    /*stampo la codifica associata alla codeword*/
    /*il ciclo verrò ripetuto tante volte quanti sono gli 0 e 1 della
codifica associata alla codeword*/
    /*parto da 1 e non da 0 perchè nella codifica avrei anche "-1" che è la
codifica associata alla radice, ma a me non interessa*/
    for (i=1; i < strlen(shannonFanoEncodingArray[textChar]); i++) {
      fprintf(fileEncodedText,"%c",shannonFanoEncodingArray[textChar][i]);
    }/*for*/
  }/*while*/

}/*printEncodedTextFile*/
```

```
/*
*Funzione che, dato l'albero della decodifica, l'array di char contenente il
testo codificato e il file che conterrà il testo decodificato
*stampa a schermo e su file il testo decodificato
*IP  *tree  puntatore all'albero della decodifica
*IP  *encodedText  puntatore all'array di char contenente il testo
codificato
*IP  *fileDecodedText  puntatore al file che conterrà il testo decodificato
*/
void printDecodedText(EntryNode *tree, char *encodedText, FILE
*fileDecodedText) {


    int encodedTextLength = 0;
    int i = 0;
    /*istanzio il puntatore l'abero*/
    EntryNode current_node = *tree;

    /*printf("Stringa Decodificata:\n");*/

    /*inizializzo una variabile contenente la lunghezza del testo
codificato*/
    encodedTextLength = (int) strlen((const char *)encodedText);

    for (i=0; i < encodedTextLength; i++) {

        if(encodedText[i] == '0') {
            /*scendo nel ramo sinistro dell'albero*/
            current_node = *current_node.zero;
        }else{
            /*scendo nel ramo destro dell'albero*/
            current_node = *current_node.one;
        }/*else*/

        /*se il nodo è una foglia (ovvero non ha figli) allora sono arrivato
al carattere*/
        if(current_node.zero == NULL && current_node.one == NULL) {

            /*stampo il carattere decodificato a schermo*/
            /*printf("%c", current_node.c);*/
            /*stampo il carattere decodificato sul file*/
            fprintf(fileDecodedText, "%c", current_node.c);
            /*ritorno alla radice*/
            current_node = *tree;

        }/*if*/
    }/*for*/
}/*printDecodedText*/
```

File shannonfanotree.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#ifndef SHANNONFANOTREE_H_
#define SHANNONFANOTREE_H_

#include <stdlib.h>
#include <stdio.h>

typedef struct tree_node{
  int nCodeword; /*numero di codeword*/
  int coding; /*codifica associata (0 o 1)*/
  double codewordProb[128]; /*probabilità di tutte le codeword presenti nel
nodo*/
  char codewordName[128]; /*nome delle codeword presenti nel nodo*/
  int indexOfBalanceSX; /*indice sinistro del bilanciamento*/
  int indexOfBalanceDX; /*indice destro del bilanciamento*/
  double probsBalancedSX; /*somma delle probabilità a SX dell'indice*/
  double probsBalancedDX; /*somma delle probabilità a DX dell'indice*/
  struct tree_node * left; /*nodo sinistro*/
  struct tree_node * right; /*nodo destro*/
}tree_node_t;

void destroy_tree(tree_node_t * n);
void computeNodeProbabilitiesBalancingInfo(tree_node_t * n, double *
balancingInfoArray);
void computeShannonFanoTree(tree_node_t * n, unsigned char *codewordArray,
double *probabilitiesArray, double *balancingInfo, int dimAlphabet);
void generateSubNodes(tree_node_t * n);
void recCodingBuilder(tree_node_t * n, int codewordCoding[], char **
shannonFanoEncodingArray, int length);
void computeShannonFanoEncodingArray(tree_node_t * n, char **
shannonFanoEncodingArray, unsigned char *codewordArray, int dimAlphabet);

#endif /*SHANNONFANOTREE_H_*/
```

File shannonfanotree.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <string.h>
#include <assert.h>
#include "../Headers/shannonfanotree.h"
#include "../Headers/utils.h"

/*
*Funzione che, dato l'albero contenente la codifica di Shannon-Fano lo
distrugge liberando la memoria
*IP  *n  puntatore all'albero contenente la codifica di Shannon-Fano
*/
void destroy_tree(tree_node_t * n) {
  if(n){
    destroy_tree(n->left);
    destroy_tree(n->right);
    free(n);
  }/*if*/
}/*destroy_tree*/
```

```
/*
*Funzione che, dato l'array delle probabilità, l'array delle codeword,
l'array delle informazioni di bilanciamento e la dimensione dell'alfabeto
*costruisce l'albero della codifica di Shannon-Fano
*IP *n  puntatore all'albero contenente la codifica di Shannon-Fano
*IP *codewordsArray  puntatore all'array che contiene in ogni indice il nome
di ciascuna codeword
*IP *probabilitiesArray  puntatore all'array di double che contiene le
probabilità di ciascuna codeword in ogni indice
*IP *balancingInfoArray  puntatore all'array delle informazioni di
bilanciamento
*IP dimAlphabet  dimensione dell'alfabeto
*/
void computeShannonFanoTree(tree_node_t * n, unsigned char *codewordArray,
double *probabilitiesArray, double *balancingInfo, int dimAlphabet){

  int i = 0;
  n->coding = -1;
  n->nCodeword = dimAlphabet;

  for(i=0; i<dimAlphabet; i++){
    n->codewordName[i] = codewordArray[i];
  }/*for*/

  for(i=0; i<dimAlphabet; i++){
    n->codewordProb[i] = probabilitiesArray[i];
  }/*for*/

  n->indexOfBalanceSX = (int)balancingInfo[0];
  n->indexOfBalanceDX = (int)balancingInfo[1];
  n->probsBalancedSX = balancingInfo[2];
  n->probsBalancedDX = balancingInfo[3];
  n->left = NULL;
  n->right = NULL;

  generateSubNodes(n);
}/*computeShannonFanoTree*/
```

```
/*
*Funzione che, dato un nodo dell'albero (che conterrà delle codewords e le
informazioni ad esse relative)
*genera i sottonodi a quel nodo fino, rispettando le prorpietà di
bilanciamento delle probabilità a sinistra e a destra date da Shannon-Fano
*fino ad arrivare alle foglie (che sarà la codeword singola)
*IP  *n  puntatore al nodo dell'albero
*/
void generateSubNodes(tree_node_t * n){

  /*alloco la memoria per i nodi sinistro e destro del nodo passsato come
parametro,che verranno calcolati*/
  tree_node_t *sx = (tree_node_t *)malloc(sizeof(tree_node_t));
  tree_node_t *dx = (tree_node_t *)malloc(sizeof(tree_node_t));
  int nCodewordSX = 0;
  int nCodewordDX = 0;
  int maxIndexSX = 0;
  int minIndexDX = 0;

  /*array che conterrà le informazioni di bilanciamento*/
  double balancingInfo[4];
  double * balance = balancingInfo;

  int i = 0;
  int j = 0;
  int x = 0;
  int k = 0;

  /*controllo che l'allocazione dinamica della memoria vada a buon fine*/
  assert(sx);
  assert(dx);

  /*salvo per comodità di scruittura successiva gli indici di bilanciamento
del nodo a sinistra e a destra*/
  maxIndexSX = ((n)->indexOfBalanceSX);
  minIndexDX = ((n)->indexOfBalanceDX);

  /*assegno la codifica '0' ai nodi a sinistra e '1' ai nodi a destra
rispettando Shannon-Fano*/
  sx->coding = 0;
  dx->coding = 1;

  /*calcolo il numero di codeword dei due nodi sottostanti*/
  for(i=0; i<=maxIndexSX; i++){
    nCodewordSX++;
  }/*for*/

  i=0;

  for(i=minIndexDX; i<((n)->nCodeword); i++){
    nCodewordDX++;
```

```
  }/*for*/

  /*assegno ai nodi sinistro e destro il numero di codeword
rispettivamente*/
  sx->nCodeword = nCodewordSX;
  dx->nCodeword = nCodewordDX;

  i=0;

  /*assegno le codeword a sinistra*/
  for(i=0; i<=maxIndexSX; i++){
    sx->codewordName[i] = (n)->codewordName[i];
  }/*for*/

  i=0;

  /*assegno le codeword a destra*/
  for(i=minIndexDX; i<((n)->nCodeword); i++){
    dx->codewordName[k] = (n)->codewordName[i];
    k++;
  }/*for*/

  i=0;
  /*assegno le probabilità delle codeword di sinistra al nodo di sinistra*/
  for(i=0; i<=maxIndexSX; i++){
    sx->codewordProb[i] = (n)->codewordProb[i];
  }/*for*/


  /*assegno le probabilità delle codeword di destra al nodo di destra*/
  for(x=minIndexDX; x<((n)->nCodeword); x++){
    dx->codewordProb[j] = (n)->codewordProb[x];
    j++;
  }/*for*/

  /*calcolo le informazioni di bilanciamento del nodo di sinistra*/
  computeNodeProbabilitiesBalancingInfo(sx,balance);

  /*assegno al nodo di sinistra le informazioni di bilanciamento*/
  sx->indexOfBalanceSX = (int)balancingInfo[0];
  sx->indexOfBalanceDX = (int)balancingInfo[1];
  sx->probsBalancedSX = balancingInfo[2];;
  sx->probsBalancedDX = balancingInfo[3];

  /*calcolo le informazioni di bilanciamento del nodo di destra*/
  computeNodeProbabilitiesBalancingInfo(dx,balance);

  /*assegno al nodo di destra le informazioni di bilanciamento*/
  dx->indexOfBalanceSX = (int)balancingInfo[0];
  dx->indexOfBalanceDX = (int)balancingInfo[1];
  dx->probsBalancedSX = balancingInfo[2];
```

```
  dx->probsBalancedDX = balancingInfo[3];

  (n)->left = sx; /*il nodo di sinistra sarà, ovviamente, il nodo di
sinistra del nodo passato per paramentro*/
  (n)->right = dx; /*il nodo di destra sarà, ovviamente, il nodo di destra
del nodo passato per paramentro*/


  /*vado avanti finchè non arrivo alle foglie (ovvero finchè non ho una
singola codeword)*/
  if((sx)->nCodeword > 1){
    generateSubNodes(sx);
  }/*if*/

  if((dx)->nCodeword > 1){
    generateSubNodes(dx);
  }/*if*/
}/*generateSubNodes*/
```

```
/*
*Funzione che, dato un nodo dell'albero (che conterrà delle codewords e le
informazioni ad esse relative)
*e l'array di 4 elementi che conterrà le informazioni relative al
bilanciamento delle codeword di quel nodo, calcola le informazioni di
bilanciamento di quel nodo
*IP  *n  puntatore al nodo dell'albero
*IP  *balancingInfoArray  puntatore all'array delle informazioni di
bilanciamento
*/
void computeNodeProbabilitiesBalancingInfo(tree_node_t * n, double *
balancingInfoArray){
  double sumTotal = 0.0;
  double partialTotalSX = 0.0;
  double partialTotalDX = 0.0;
  int indexOfBalance = 0;
  int i,j,k = 0;
  int dxStartIndex = 0;

  /*calcolo la probabilità totale in quel nodo*/
  for(i=0; i<=((n)->nCodeword); i++){
    sumTotal = sumTotal + (n)->codewordProb[i];

  }/*for*/

  /*sommo tutti gli elementi uno alla volta e ogni volta controllo se la
somma è minore o uguale alla metà della somma totale di tutti gli elementi
dell'array*/
  /*calcolo la somma parziale delle probabilità a sinistra*/
  for(j = 0; j<=((n)->nCodeword); j++){

    if((partialTotalSX < (sumTotal/2)) && (partialTotalSX + (n)-
>codewordProb[j+1] < (sumTotal/2))){
      partialTotalSX = partialTotalSX + (n)->codewordProb[j];
      indexOfBalance++;
    }/*if*/
  }/*for*/

  if(indexOfBalance==0){
    dxStartIndex=1;
  }

  /*calcolo la somma parziale delle probabilità a destra*/
  for(k = indexOfBalance-dxStartIndex; k <=((n)->nCodeword); k++){
    partialTotalDX = partialTotalDX + (n)->codewordProb[k];
  }/*for*/

  /*popolo l'array delle informazioni di bilanciamento*/
  if(indexOfBalance==0){
    /*se l'indice di bilanciamento è 0 sigifica che il nodo sarà bilanciato
tra gli indici 0 e 1, ossia un carattere a SX e i rimanenti a DX*/
```

```
    balancingInfoArray[0] = indexOfBalance;
    balancingInfoArray[1] = indexOfBalance+1;
    balancingInfoArray[2] = partialTotalSX + (n)->codewordProb[0];
    balancingInfoArray[3] = partialTotalDX;
  }else{
    balancingInfoArray[0] = indexOfBalance-1;
    balancingInfoArray[1] = indexOfBalance;
    balancingInfoArray[2] = partialTotalSX;
    balancingInfoArray[3] = partialTotalDX;
  }
}/*computeNodeProbabilitiesBalancingInfo*/

/**
*Funzione che, dato l'albero della codifica di Shannon-Fano, l'array di
stringhe che conterrò la codifica per ogni codeword dell'alfabeto,
*l'array delle codeword e la dimensione dell'alfabeto, costruisce l'array
contenente la codifica di Shannon-Fano di ogni codeword
*IP  *n  puntatore all'albero contenente la codifica di Shannon-Fano
*IP  **shannonFanoEncodingArray  puntatore ad un puntatore all'arrary di
stringhe contenente la codifica di ogni codeword dell'alfabeto
*IP  dimAlphabet  dimensione dell'alfabeto
*/
void computeShannonFanoEncodingArray(tree_node_t * n, char **
shannonFanoEncodingArray, unsigned char *codewordArray, int dimAlphabet){
  int codewordCoding[8];
  recCodingBuilder(n, codewordCoding, shannonFanoEncodingArray, 0);
}/*computeShannonFanoEncodingArray*/
```

```
/*
*Funzione ricorsiva che, dato l'albero della codifica di Shannon-Fano e
l'array di stringhe che conterrò la codifica per ogni codeword
dell'alfabeto,
*calcola la codifica di ogni codeword analizzando ogni cammino dalla radice
dell'albero fino alla foglia (codeword)
*IP  *n  puntatore all'albero contenente la codifica di Shannon-Fano
*IP  codewordCoding  array di interi che conterrà "0" e "1" man mano che
viene analizzato il cammino dalla radice alla foglia
*IP  **shannonFanoEncodingArray  puntatore ad un puntatore all'arrary di
stringhe contenente la codifica di ogni codeword dell'alfabeto
*IP  length  lunghezza della codifica di una data codeword
*/
void recCodingBuilder(tree_node_t * n, int codewordCoding[], char **
shannonFanoEncodingArray, int length){

  /*array di char che conterrà la codifica sottoforma di char*/
  /*una codifica di 0 e 1 può essere al massimo lunga log(2,128), ossia 7!,
ciò capita se uso tutti caratteri diversi nel testo.*/
  /*avrà allora dimensione 8*/
  char stringcoding[8];
  int i=0;
  /*aggiungo la codifica relativa a quel nodo alla codifica complessiva*/
  codewordCoding[length] = n->coding;
  length++;

  /*se il nodo è un nodo foglia (ovvero una singola codeword), stampo il
cammino dalla radice a quella foglia*/
  if (n->left==NULL && n->right==NULL){
    /*converto l'array di int contenente la codifica della foglia in array
di char per poterlo copiare in shannonFanoEncodingArray*/
    convertIntArrToCharArr(codewordCoding,stringcoding,length);

    /*alloco lo spazio in shannonFanoEncodingArray per contenere la stringa
contenente la codifica*/
    shannonFanoEncodingArray[(int)n-
>codewordName[0]]=calloc(strlen(stringcoding),sizeof(stringcoding[i]));

    /*copio in shannonFanoEncodingArray in indice della codeword, la sua
codifica*/
    strcpy(shannonFanoEncodingArray[(int)n->codewordName[0]],stringcoding);

  }else{

    /*se non è foglia scansiono i sottoalberi destro e sinistro*/
    recCodingBuilder(n->left, codewordCoding, shannonFanoEncodingArray,
length);
    recCodingBuilder(n->right, codewordCoding, shannonFanoEncodingArray,
length);
  }/*if*/
}/*recPrintCoding*/
```

File encode.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#ifndef ENCODE_H_
#define ENCODE_H_

int encode(char * fTTC);

#endif /*ENCODE_H_*/
```

File encode.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "../Headers/loadfunctions.h"
#include "../Headers/printfunctions.h"
#include "../Headers/shannonfanotree.h"
#include "../Headers/encode.h"
#include "../Headers/utils.h"

/**
*Funzione che esegue l'encoding del testo
*IP  fTTC  nome del file contenente il testo da codificare
*OR  0 se l'esecuzione va a buon fine, -2 se non è possibile aprire il file
di testo da codificare
*/
int encode(char * fTTC){

  /*creo i puntatori ai file*/
  FILE *fileTextToCode = NULL;
  FILE *fileAlphabet = NULL;
  FILE *fileShannonFanoCoding = NULL;
  FILE *fileEncodedText = NULL;

  char alphabetNameFileBuffer[100];
  char shannonFanoNameFileBuffer[100];
  char fileEncodedTextNameFileBuffer[100];

  /*conterrà la dimensione dell'alfabeto*/
  int dimAlphabet = 0;

  /*definisco l'array dove memorizzerò il nome delle codeword prese dal
file*/
  unsigned char codewords[128];
  unsigned char * charcodeword = codewords;

  /*definisco l'array dove memorizzerò tutte le probabilità delle codeword
prese dal file*/
  double probabilities[128] = { 0 };
  double * probs = probabilities;
```

```
  /*buffer che tiene conto del numero di caratteri da 0,...,256; ogni
carattere avrò il suo indice "i" dato che viene usata la codifica intera del
carattere*/
  /*Inizializzato a 0 per ogni elemento*/
  int codewordsCounter[128] = { 0 };

  /*
    definisco l'array dove memorizzerò le informazioni relatrive al
bilanciamento della somma delle probabilità delle codewords
    balancingInfo[0] e balancingInfo[1] =  indici in mezzo ai quali l'array
è bilanciato

    ESEMPIO
      0       1   |    2       3       4
    0.39    0.19  |  0.17    0.14    0.12

    La somma è bilanciata tra l'indice 1 e 2

    balancingInfo[3] = Somma parziale delle probabilita' a SX
    balancingInfo[4] = Somma parziale delle probabilita' a DX
  */

  double balancingInfo[4] = { 0.0 };
  double * balance = balancingInfo;

  /*definisco l'array dove memorizzerò per ogni codeword trasformata in
intero, un array di 0 e 1 ossia la sua codifica*/
  /*ad ogni riga è assicata una codeword, che possono essere al massimo 128
dato che utilizzo l'ASCII STANDARD*/
  /*inizializzo l'array con il carattere '\0'*/
  char * shannonFanoEncodingArray[128] = {'\0'};

  /*definisco l'albero che conterrà i nodi con le informazioni relative alla
codifica e creo il nodo radice*/
  /*il nodo radice contiene tutte le codeword e tutte le informazioni ad
esse relative*/
  tree_node_t *root = (tree_node_t *)malloc(sizeof(tree_node_t));
  assert(root);

  /*Costruisco i percorsi di output per i file*/
  strcpy(alphabetNameFileBuffer,"Output/");
  strcpy(shannonFanoNameFileBuffer,"Output/");
  strcpy(fileEncodedTextNameFileBuffer,"Output/");

  /*copio il nome del file di input senza l'estensione .txt*/
  strncat(alphabetNameFileBuffer,fTTC,strlen(fTTC)-4);
  strncat(shannonFanoNameFileBuffer,fTTC,strlen(fTTC)-4);
  strncat(fileEncodedTextNameFileBuffer,fTTC,strlen(fTTC)-4);

  /*aggiungo il nome dei file di output e l'estensione*/
  strcat(alphabetNameFileBuffer,"_A&P.txt");
```

```
  strcat(shannonFanoNameFileBuffer,"_SF.txt");
  strcat(fileEncodedTextNameFileBuffer,"_ET.txt");

  /*apro i file che dovrò scrivere*/
  openFile(&fileAlphabet,alphabetNameFileBuffer,"w");
  openFile(&fileShannonFanoCoding,shannonFanoNameFileBuffer,"w");
  openFile(&fileEncodedText,fileEncodedTextNameFileBuffer,"w");

  /*controllo l'input*/
  if(openFile(&fileTextToCode, fTTC, "r")){
    printf("Impossibile Aprire il file da codificare!\n");
    return -2;
  }

  /*determino la dimensione dell'alfabeto e carico le codewords*/
  dimAlphabet = countAndLoadCodewords(fileTextToCode, codewordsCounter,
charcodeword);

  /*stampo la quantità del mio alfabeto*/
  printf("Dimensione alfabeto: %d\n",dimAlphabet);

  /*determino la probabilità associata ad ogni codeword*/
  computeAndloadProbabilities(codewordsCounter,charcodeword, probs,
dimAlphabet);

  /*stampo il nome delle codewords*/
  printCodewords(charcodeword,dimAlphabet);

  /*stampo la probabilità di ciascuna codeword*/
  printProbabilities(probs,charcodeword,dimAlphabet);

  /*ordino le codewords, le probabilità e le lunghezze di Shannon associate
alle codewords in ordine di probabilità descrescente*/
  printf("\nOrdino le codewords in ordine decrescente di frequenza e le
informazioni ad essere relative...\n");
  sort(probs,charcodeword,dimAlphabet);
  reverse(probs,charcodeword,dimAlphabet);

  /*creo il file contenente le informazioni sull'alfabeto*/
  printAlphabetInfoFile(fileAlphabet, probs, charcodeword, dimAlphabet);

  /*calcolo gli indici e le probabilità bilanciate a SX e ad DX*/
  balanceProbabilities(balance,probs,dimAlphabet);

  /*genero l'albero della codifica di Shannon-Fano*/
  computeShannonFanoTree(root, charcodeword, probs, balance, dimAlphabet);

  printf("\nCalcolo la codifica di Shannon-Fano...\n\n");

  /*calcolo la codifica di Shannon-Fano associata ad ogni codeword
salvandola nell'array shannonFanoEncodingArray*/
```

```
  computeShannonFanoEncodingArray(root, shannonFanoEncodingArray,
charcodeword, dimAlphabet);

  /*stampo a schermo la codifica per ogni codeword*/
  printShannonFanoEncodingArray(shannonFanoEncodingArray);

  /*creo il file contenente le informazioni sulla codifica di Shannon-Fano*/
  printShannonFanoInfoFile(fileShannonFanoCoding, shannonFanoEncodingArray);

  /*elimino l'albero associato alla codifica*/
  destroy_tree(root);

  /*stampo la stringa da codificare con Shannon-Fano*/
  /*printf("\nStringa da codificare:\n");*/
  /*printTextToCode(fileTextToCode);*/

  /*faccio il rewind sul file contenente la codifica di Shannon Fano per
ogni codeword e sul file da codifcare*/
  rewind(fileShannonFanoCoding);
  rewind(fileTextToCode);

  /*eseguo la codifica della parola da codificare*/
  printEncodedTextFile(fileTextToCode, fileEncodedText,
shannonFanoEncodingArray);

  /*chiudo i file*/
  closeFile(fileTextToCode);
  closeFile(fileAlphabet);
  closeFile(fileShannonFanoCoding);
  closeFile(fileEncodedText);

  return 0;
}/*encode*/
```

File decode.h

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con SHannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#ifndef SHANNONFANO_DECODE_H
#define SHANNONFANO_DECODE_H

typedef struct entry_node {
    char c; /*carattere associato alla codifica*/
    struct entry_node *zero; /*nodo sinistro*/
    struct entry_node *one; /*nodo destro*/
}EntryNode;

int decode(char *text, char *dict);
EntryNode *initNode();
void addCharToTree(EntryNode *tree, char *line);
void destroy_decode_tree(EntryNode *tree);

#endif /*SHANNONFANO_DECODE_H*/
```

File decode.c

```
/*
* Progetto: Codifica e Decodifica di caratteri ASCII Standard con Shannon-
Fano
* Autore: Fabio Scarparo
* Matricola: 1163162
*/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "../Headers/decode.h"
#include "../Headers/loadfunctions.h"
#include "../Headers/printfunctions.h"
#include "../Headers/utils.h"

/**
*Funzione che esegue la decodifica del testo
*IP  encodedtxt  nome del file contenente il testo da decodificare
*IP  shannonfanotxt  nome del file contenente la codifica di Shannon-Fano
per ogni codeword
*OR  0 se l'esecuzione va a buon fine, -2 se non è possibile aprire il file
di testo da codificare
*/
int decode(char *encodedtxt, char *shannonfanotxt) {

    /*creo il pntatore al file che conterrà il testo decodificato e apro il
file*/
    FILE *fileDecodedText = NULL;
    /*creo il pnutatore al file che contiene la codifica di Shannon-Fano per
ogni codeword e apro il file*/
    FILE *fileShannonFano = NULL;
    /*creo il pnutatore al file che contiene il testo codificato con
Shannon-Fano e apro il file*/
    FILE *fileEncodedText = NULL;
    /*Albero che conterrà la codifica*/
    EntryNode tree;
    /*Array che conterrà il path del file del testo con la codifica di
Shannon-Fano*/
    char fileShannonFanoPath[100];
    /*Array che conterrà il path del file del testo codificato*/
    char fileEncodedTextPath[100];
    /*Array che conterrà il path del file del testo decodificato*/
    char fileTextDecodedPath[100];
    /*inizializzo il buffer che mi conterrà ogni riga del file contente pe
ogni carattere la sua codifica di Shanon-Fano*/
    char line[1024];
    /*creo il pntatore all'array che conterrà il testo codificato*/
    char * encodedText;
```

```
    strcpy(fileTextDecodedPath,"");
    strncat(fileTextDecodedPath,encodedtxt,strlen(encodedtxt)-6);
    strcat(fileTextDecodedPath,"DT.txt");

    openFile(&fileDecodedText,fileTextDecodedPath,"w");

    strcpy(fileShannonFanoPath, "./Output/");
    strcat(fileShannonFanoPath, shannonfanotxt);

    if(openFile(&fileShannonFano, fileShannonFanoPath, "r")){
      printf("Impossibile aprire il file contenente la codifica di Shannon-
Fano!\n");
      return -2;
    }

    strcpy(fileEncodedTextPath, "./Output/");
    strcat(fileEncodedTextPath, encodedtxt);

    if(openFile(&fileEncodedText, fileEncodedTextPath, "r")){
      printf("Impossibile aprire il file contenente il testo
codificato!\n");
      return -2;
    }

    /*dal file contenete il testo codificato, creo un array di char
contenente tutti i caratteri*/
    encodedText = loadEncodedTextToCharArray(fileEncodedText);

    /*inizializzo l'albero contenente la codifica*/
    tree.zero = NULL;
    tree.one = NULL;

    /*costruisco l'albero*/
    while (fgets(line, 1024, fileShannonFano)) {
        addCharToTree(&tree, line);
    }
    /*stampo la stringa da decodificare*/
    /*printf("\nStringa Codificata:\n%s\n\n", encodedText);*/
    printf("\nDecodifico...\n");
    /*stampo la stringa decodificata*/
    printDecodedText(&tree, encodedText, fileDecodedText);
    /*elimino l'albero per la decodifica*/
    destroy_decode_tree(&tree);
    /*chiudo i file*/
    closeFile(fileDecodedText);
    closeFile(fileShannonFano);
    closeFile(fileEncodedText);

    return 0;
}/*decode*/
```

```
/*funzione di supporto per la funzione addCharToTree(..)*/
EntryNode *initNode() {
    EntryNode *node = (EntryNode *) malloc(sizeof(EntryNode));
    assert(node);
    node->zero = NULL;
    node->one = NULL;
    return node;
}/*initNode*/

/**
*Funzione che, dato l'albero della decodifica, l'array di char contenente la
riga di testo del file ShannonFano
*popola l'albero della decodifica
*IP  *tree  puntatore all'albero della decodifica
*IP  *line  puntatore all'array di char contenente la riga di testo del file
ShannonFano
*/
void addCharToTree(EntryNode *tree, char *line) {
    char c; /*carattere*/
    char value[8]; /*codifica di Shannon-Fano del carattere*/
    int value_length = 0; /*lunghezza della codifica del singolo carattere*/
    EntryNode *current_tree = tree; /*albero per la decodifica*/
    int i = 0;

    /*se il carattere è carriage return, assegno a "c" il carriage return e
leggo la codifica associata dopo lo spazio altrimenti legge il carattere e
la codifica associata*/
    if (isCarriageReturnBuffer(line)) {
        c = '\n';
        sscanf(line, "\"CR\" %s", value);
    } else {
        sscanf(line, "%c %s", &c, value);
    }/*if*/

    value_length = strlen(value); /*lunghezza della codifica del singolo
carattere*/

    for (i = 0; i < value_length; i++) {
        if (value[i] == '0') {
            /*scendo nel ramo sinistro dell'albero*/
            if (current_tree->zero == NULL) {
                /*se il nodo non ha figli, li inizializzo a NULL con la
funzione initNode()*/
                current_tree->zero = initNode();
            }
            /*passo al nodo successivo*/
            current_tree = current_tree->zero;
        }else{
            /*scendo nel ramo destro dell'albero*/
            if (current_tree->one == NULL) {
```

```
                /*se il nodo non ha figli, li inizializzo a NULL con la
funzione initNode()*/
                current_tree->one = initNode();
            }/*if*/
            /*passo al nodo successivo*/
            current_tree = current_tree->one;
        }/*else*/
    }/*for*/

    /*arrivato alla fine della codifica memorizzo il carattere nell'ultimo
nodo*/
    current_tree->c = c;
}/*addCharToTree*/

/**
*Funzione che, dato l'albero per la decodifica, lo distrugge liberando la
memoria
*IP  *n  puntatore all'albero della decodifica
*/
void destroy_decode_tree(EntryNode * n) {
  if(n){
    destroy_decode_tree(n->zero);
    destroy_decode_tree(n->one);
    free(n);
  }/*if*/
}/*destroy_decode_tree*/
```