**Implementazione della Codifica di Huffman – Cattaneo Kevin – S4944382**

Per l’implementazione della codifica si è scelto di utilizzare il linguaggio C++.

Di seguito il codice in C++

#include <iostream>

#include <fstream>

#include <unordered\_map>

#include <queue>

#include <string>

#include <tuple>

#include <math.h> // funzione log2

using Map         = std::unordered\_map<char,int>; // Mappa <carattere,frequenza>

using CodecMap    = std::unordered\_map<char, std::string>; // Mappa <carattere,codifica>

using Queue       = std::queue<std::string>; // Queue per inserimento stringhe da input

using std::string;

using std::vector;

using std::greater;

using std::tuple, std::get; // per gestire tuple

struct node{

  char c;   // carattere che il nodo rappresenta

  int freq; // frequenza carattere

  node\* left; // nodo figlio sinistro

  node\* right;  // nodo figlio destro

  // Costruttore

  node(char \_char, int \_freq = 0) :

  c(\_char), freq(std::forward<int>(\_freq)),

  left(nullptr),

  right(nullptr){}

};

// Albero binario

using Tree = node\*;

// Heap binario di tuple <frequenza emp, carattere, nodo>

// La dichiarazione segue: contenuto<>, contenitore<>, operatore di confronto<>

using Heap = std::priority\_queue<tuple<int, char, Tree>, vector<tuple<int, char, Tree>>, greater<tuple<int, char, Tree>>>;

// Queue per salvare la codifica associata ad ogni nodo (contenente un char) tramite BFS

using Bfsq = std::queue<tuple<Tree, string>>;

void destroy\_tree(Tree tree){

  if (!tree) return;

  destroy\_tree(tree -> left);

  destroy\_tree(tree -> right);

  delete tree;

}

CodecMap huffmanEncoding(const Map& map){

  Heap heap;

  CodecMap cmap;

  Bfsq q;

  for (const auto [c,f] : map) heap.emplace(f,c, new node(c,f));

  // Albero con la posizione di ogni simbolo

  Tree aux;

  for (int i=0; i < map.size()-1; ++i){

    auto tupla1 = heap.top();

    heap.pop();

    auto tupla2 = heap.top();

    heap.pop();

    // Creazione nodo fittizio (la cui frequenza è somma dei figli...

    // ...per ottenere in seguito la somma di probabilità)

    aux = new node('|',get<0>(tupla1)+get<0>(tupla2));

    aux->left = get<2>(tupla1);

    aux->right = get<2>(tupla2);

    heap.emplace(aux->freq, aux->c, aux);

  }

  // BFS per visitare l'albero e determinare la codifica di ogni nodo...

  // ...inserendola all'interno di una coda

  q.emplace(aux, ""); // nodo radice, sapendo che aux rappresenta tale

  while(!q.empty()){

    auto [n, cod] = q.front();

    q.pop();

    // Se giungo a una foglia (no figli), inserisco la sua codifica...

    // ...nella mappa alla posizione corrispondente al char

    if(!(n->left) && !(n->right)) cmap[n->c] = cod;

    else{

      // Costruisco la codifica, sinistra 0, destra 1

      if(n->left) q.emplace(n->left, cod+"0");

      if(n->right) q.emplace(n->right, cod+"1");

    }

  }

  destroy\_tree(aux);

  return cmap;

}

// Funzione di calcolo dell'entropia di Shannon

double Shannon(Map& map){

  double somma = 0.0;

  double H = 0.0;

  for(const auto& [c,freq] : map)

    somma += freq;

  for(const auto& [c,freq] : map)

    H += (freq/somma \* log2(somma/freq));

  return H;

}

// Funzione di calcolo della lunghezza media della codifica

double Lmedia(CodecMap comp, Map map){

  double somma = 0.0;

  double L = 0.0;

  for(const auto& [c,freq] : map)

    somma += freq;

  for(const auto& [c,cod] : comp){

    double prob = map[c]/somma;

    L += (prob \* cod.size());

  }

  return L;

}

// Creazione mappa di coppie <Carattere, Frequenza>

Map create\_map(int& dim)

{

  Map map;

  std::ifstream f("Genova.txt");

        if (f.is\_open())

        {

                std::string aux;

    while (std::getline(f, aux)){ // getline memorizza gli spazi

      if(f.peek() != EOF) aux.append("\n"); // std::getline() salta '\n' dunque inserisco a mano, finchè non EOF

      for(auto c : aux){

        dim += 8; // ogni carattere ascii = 8 bit

        map[c]++; // allocazione automatica della coppia (c,frequenza);

      }

    }

        }

        else std::cout << "Errore nell'apertura del file";

  return map;

}

void compressionToFile(CodecMap& cmap, int& dim){

  std::ifstream f1("Genova.txt");

  std::ofstream f2("GenovaZIP.txt");

        if (f1.is\_open() && f2.is\_open())

        {

                std::string aux;

    while (std::getline(f1, aux)){ // getline memorizza gli spazi

      if(f1.peek() != EOF) aux.append("\n"); // std::getline() salta '\n' dunque inserisco a mano, finchè non EOF

      for(auto k : aux){

        f2 << cmap[k];

        dim += cmap[k].size();

      }

      if(f1.peek() != EOF) f2 << "\n"; // vado manualmente a capo, in linea con il testo originale

    }

        }

        else std::cout << "Errore nell'apertura di un file";

        f1.close();

        f2.close();

}

int main(){

  int dimNonCod = 0; // dimensione in bit del testo non compresso

  int dimCod = 0; // dimensione in bit del testo compresso

  Map map = create\_map(dimNonCod);

  CodecMap cmap = huffmanEncoding(map);

  // Visualizzazione su terminale

  for(const auto& [c,freq] : map)

    std::cout << c << " " << freq << std::endl;

  std::cout << "\n\n";

  for(const auto& [c,cod] : cmap)

    std::cout << c << " " << cod << std::endl;

  std::cout << "\nEntropia di Shannon: " << Shannon(map) << std::endl;

  std::cout << "Lunghezza media della codifica: " << Lmedia(cmap, map) << std::endl;

  std::cout << "Dimensione (bit) testo non compresso: " << dimNonCod << " bit" << std::endl;

  compressionToFile(cmap, dimCod);

  std::cout << "Dimensione (bit) testo compresso: " << dimCod << " bit" << std::endl;

  return 0;

}

Screenshot dei risultati ottenuti:

Immagine che contiene testo

Descrizione generata automaticamente

*Commento*

Nella cartella è possibile visualizzare sia i sorgenti in formato cpp che i risultati ottenuti in formato txt.

Dalle statistiche ottenute è possibile desumere che, come conferma alla teoria, la codifica di Huffman è ottimale, in quanto si avvicina notevolmente a quello che è il limite inferiore dato dall’entropia di Shannon: si osserva infatti una differenza nell’ordine dei centesimi fra la lunghezza media della codifica utilizzata e l’entropia di Shannon.

Infine, si osserva come la compressione riduca più di 10.000 bit di dimensione dal file originale in ASCII nella versione compressa sottoforma di stringhe binarie di 0 e 1. Si vuole specificare però che la compressione non viene attuata su stringhe binarie (ovvero il binario del file) ma su caratteri la cui dimensione naturalmente è maggiore rispetto al singolo bit; ovviamente una codifica non ottimale poteva portare a una compressione non così efficiente. Nonostante tale operazione non avvenga da stringhe binarie in stringhe binarie è comunque possibile visualizzare il funzionamento e l’efficacia della compressione data dalla codifica di Huffman.