# UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

## DIPARTIMENTO SCIENZE E TECNOLOGIE

## FACOLTÀ INFORMATICA

ALGORITMI E STRUTTURE DATI


RELAZIONE PROGETTO AA 2017-2018

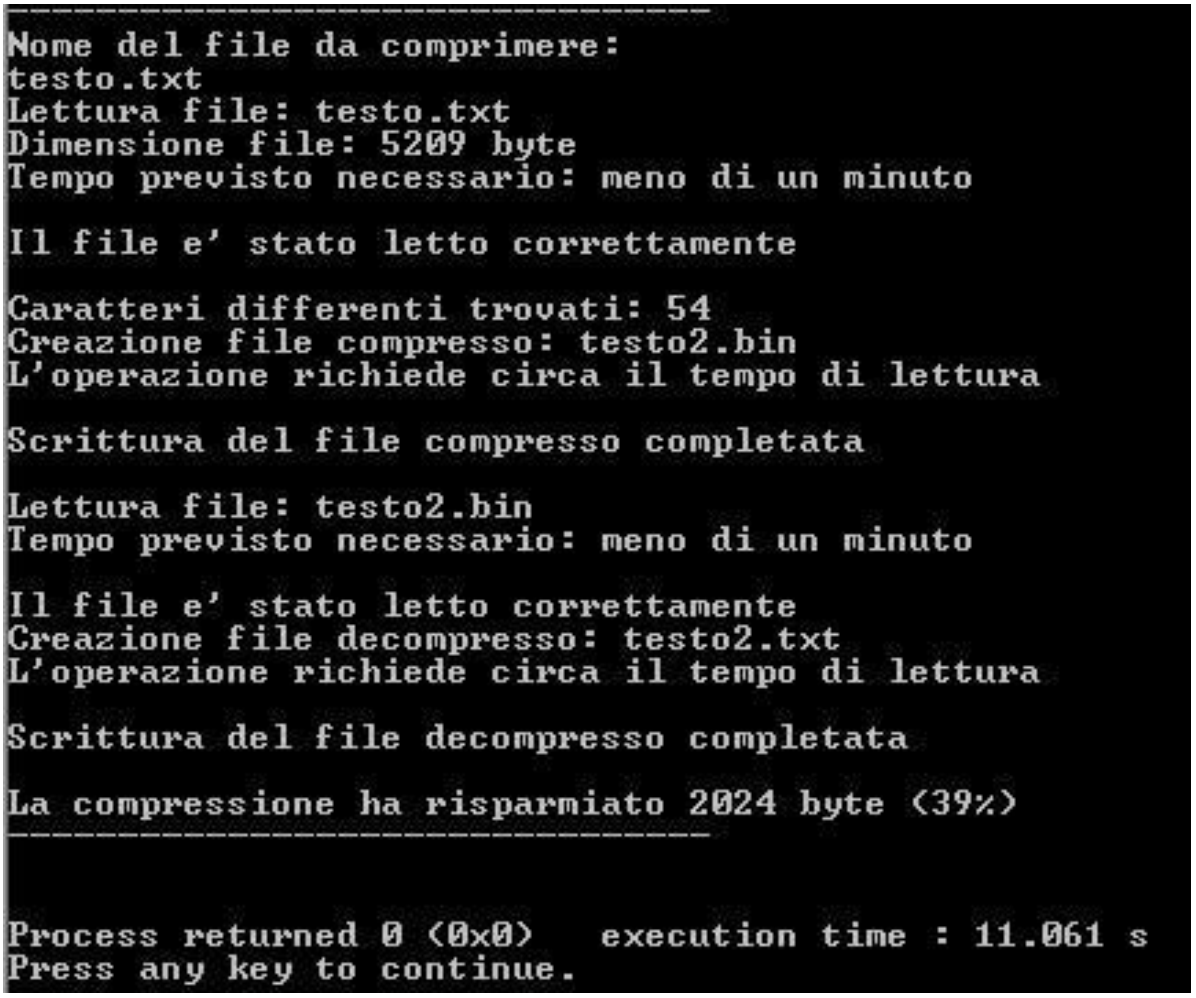
**PROF.: FRANCESCO CAMASTRA**

**STUDENTE: GIOVANNI CASTELLANO**

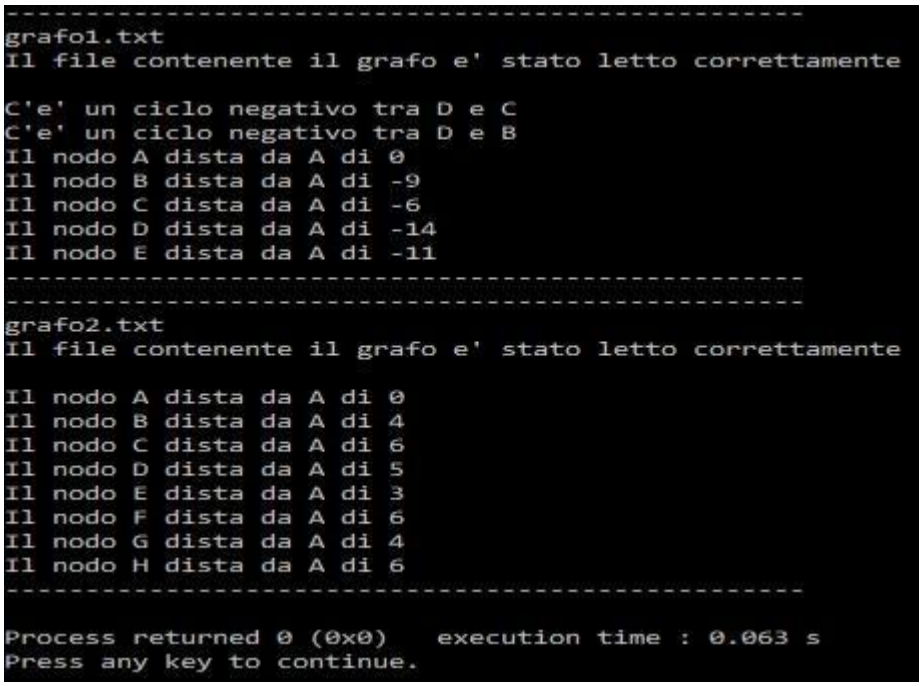
**INDICE:** **PAGINA**

## PREFAZIONE

**Esecuzione dell'algoritmo di Huffman:**



**Esecuzione algoritmo di Bellman-Ford:**




Entrambi gli algoritmi prendono dei nomi di file prestabiliti come input, quindi, laddove si voglia cambiare input, si può rinominare i nuovi file dandogli gli stessi nomi dei file già pronti e sostituendoli ad essi oppure si può cambiare il nome dei file da prendere in input nei rispettivi main.cpp. Il programma che esegue l‟algoritmo di Huffman accetta qualsiasi tipo di file, ma per aver un corretto funzionamento del programma, è consigliato rispettare i formati (file non criptati) ed è consigliato fargli elaborare file con una dimensione non superiore ai 10MB, dal momento che per file più grandi aumenta eccessivamente il tempo di esecuzione.

Nell‟algoritmo di Bellman-Ford, i file son scritti in un modo preciso dal momento che la lettura dei file avviene tramite una sequenza di controlli che riconoscono i caratteri „[„, „.“, „=“ e „\n“, associando quanto letto ad uno degli attributi della classe Graph. L‟importante, nella scrittura del file testo via editor di testo, è la scrittura corretta senza errori grammaticali e tutto in minuscolo, eccetto che per gli attributi V ed E, i quali devono essere scritti in cima al file, dato che dopo averli letti, il metodo per la lettura dei file prealloca memoria in base a quanti vertici ed archi si hanno. Gli archi possono anche essere inseriti in disordine, ma l‟importante per evitare crash di esecuzione è che vengano inseriti tutti, rispettando quanto assegnato all‟attributo E.

# 1 – ALGORITMO DI HUFFMAN

## 1.1 – Analisi della richiesta

Traccia: *Costruire un **encoder** (codificatore) e un **decoder** (decodificatore) che utilizzi la codifica di Huffman. L'implementazione deve far uso di una coda di priorità, realizzata mediante un Albero Binario di Ricerca. Definire il formato (o i formati) dei dati accettati in input e definire il formato del file codificato. Calcolare il tasso di compressione ottenuto. Il codificatore deve essere almeno in grado di codificare qualunque file di testo (.txt) e almeno un tipo di formato di immagini (bmp) e di audio ( wav, ...).*

L‟algoritmo di Huffman vuole ricodificare i caratteri di una stringa cercando di risparmiare quanto più spazio possibile, codificando i caratteri presenti con più frequenza con codifiche più piccole in termini di bit. La stringa, che essa sia un testo o una sequenza che definisce un file, viene letta, analizzata e rielaborata in modo da poter riscrivere il file con una compressione che sfrutta la **tecnica greedy**, infatti, i caratteri estratti dalla stringa vengono messi in una **coda di priorità** che, grazie alla sua struttura suggerita dal nome stesso, permette di capire quali siano i caratteri con maggiore frequenza, che nella coda avranno maggiore priorità; i caratteri e le loro occorrenze vengono quindi utilizzati per creare l‟albero binario di Huffman, le cui foglie saranno i caratteri stessi. La tecnica greedy, in questo caso, agisce scegliendo se andare a destra o a sinistra rispetto al nodo in cui si trova in base alla sequenza che sta leggendo, e appena trova una foglia, capisce che il carattere trovato è il carattere che la sequenza codificata stava a simboleggiare, in modo tale da riuscire a decodificare in modo molto veloce.

Tecnica Greedy:

La tecnica Greedy si applica quando si cerca una soluzione ottima che sia impossibile da trovare col backtracking in tempi accettabili. Consiste, come dice il nome stesso della tecnica, a prendere decisioni “golose”, scegliendo ad ogni passo la direzione reputata migliore; in questo modo però, non è affatto assicurato che le scelte siano state le migliori nel complesso, dal momento che nel piccolo contesto la scelta può sembrare la migliore, ma nel contesto complessivo può addirittura allontanare l‟algoritmo dalla risoluzione al suo problema.

Coda di priorità:

La coda di priorità è una struttura dati che permette di dare priorità a ciò che è contenuto in testa ed eseguire prima quei compiti: può essere di max-priority o di min-priority, ma in entrambi i casi avrà sempre a disposizione delle operazioni ben precise:

*Insert*: L‟operazione inserisce un elemento nella coda in base alla sua priorità

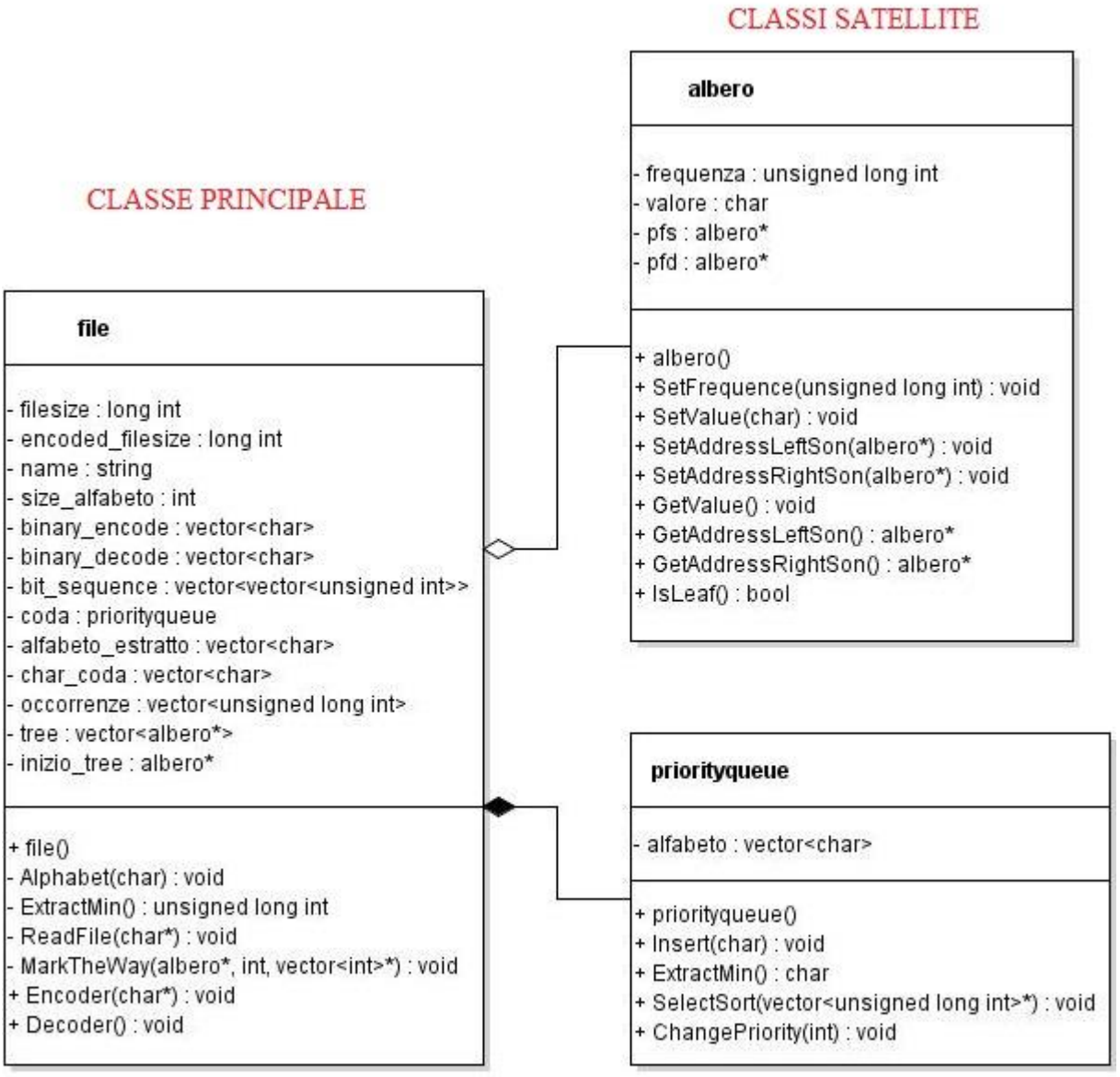
*Extract Min/Max*: estrae e cancella l‟elemento con priorità minima/massima

*Set Key*: permette di cambiare la priorità di uno degli elementi della coda o anche il valore dell‟elemento stesso.

Nel caso dell‟algoritmo di Huffman è stata utilizzata una coda di priorità di minimo, ma in ordine decrescente, in modo tale che la stl::erase() dei vector potesse essere più efficiente.

### 1.2 – Il programma

I diagrammi UML del programma aiutano a capire in modo coinciso la struttura delle classi. La classe principale è la classe file, che è composta della classe priorityqueue ed è aggregata alla classe albero. La classe file contiene tutto il necessario affinché si possa svolgere l‟algoritmo di Huffman:




Gli attributi:

- *filesize*: la dimensione del file originario letto che dovrà essere codificato

- *encoded_filesize*: la dimensione del file codificato

- *name*: stringa contenente il nome e il formato del file

- *size_alfabeto*: quanti caratteri diversi ci sono nel file, non ha un effettivo limite massimo, ma i caratteri son 256 in totale, quindi anche il size dell‟alfabeto può variare tra 0 e 256

- *binary_encode*: la sequenza di caratteri del file da codificare (la copia in memoria del file)

- *binary_decode*: la sequenza di caratteri del file da decodificare (come *binary_encode*)

- *bit_sequence*: la sequenza di bit che codifica ogni carattere trovato, è un vector di vector dato che la sequenza è scritta in memoria come singoli 0 o 1 e viene scritta come sequenza binaria solo nei metodi Encoder() e Decoder()

- *coda*: la coda di priorità

- *alfabeto_estratto*: i caratteri estratti dalla coda di priorità, l‟attributo è necessario successivamente, quando la coda è stata estratta tutta, ma bisogna associare la sequenza al carattere nella fase di encoding

- *char_coda*: caratteri inseriti nella coda, aiuta il metodo Alphabet() a creare la coda

- *occorrenze*: occorrenze dei vari caratteri estratti dal file

- *tree*: l‟albero di Huffman

- *inizio_tree*: puntatore alla radice dell‟albero di Huffman

I metodi:

file(): costruttore che inizializza a zero i size e imposta *inizio_coda* a nullptr

- Alphabet(char): metodo che crea l‟alfabeto dei caratteri, mette nella coda di priorità il carattere e la sua occorrenza e, se il carattere dovesse essere già presente nella coda, ne incrementa l‟occorrenza.

```
void file::Alphabet(char current_value)
{
    int i = 0;
    while(i < size_alfabeto && coda.GetValue(i) != current_value)
        i++;
    if(i == size_alfabeto)
    {
        coda.PushBack(current_value, 1);
        size_alfabeto++;
    }
    else
        coda.Set(coda.GetOccurrence(i)+1, i);
}
```

- ExtractMin(): metodo che chiama il metodo ExtractMin() della classe priorityqueue applicato alla variabile *coda* per poter estrarre il minimo scrivendo in *alfabeto_estratto* il valore, ritornando l‟occorrenza e cancellando dalla coda il valore.

```
unsigned long int file::ExtractMin()
{
    unsigned long int occorrenza;
    alfabeto_estratto.push_back(coda.ExtractMinValue());
    occorrenza=coda.ExtractMinOccurrence();
    coda.DeleteMin();
    return occorrenza;
}
```

- ReadFile(): metodo che apre il file col nome passatogli come parametro, lo legge e lo conserva in memoria. Chiama Alphabet per poter creare l‟alfabeto man mano che si leggono i caratteri del file.

```
for(int i=0; i<filesize; i++)
{
    fread(&current_value, sizeof(char), 1, fp);
    fseek(fp, 0, SEEK_CUR);
    binary_encode.push_back(current_value);
    Alphabet(current_value);
}
```

- MarkTheWay(albero*, int, vector<int>*): il metodo crea le sequenze di bit associate ad ogni carattere letto in precedenza dalla ReadFile(char*)

```
else
{
    if(walker!=inizio_tree)
        single_bit_sequence->push_back(bit);
    if(walker->GetAddressLeftSon()!=nullptr)
    {
        MarkTheWay(walker->GetAddressLeftSon(), 0, single_bit_sequence);
        single_bit_sequence->erase(single_bit_sequence->begin()+single_bit_sequence->size()-1);
    }
    if(walker->GetAddressRightSon()!=nullptr)
    {
        MarkTheWay(walker->GetAddressRightSon(), 1, single_bit_sequence);
        single_bit_sequence->erase(single_bit_sequence->begin()+single_bit_sequence->size()-1);
    }
}
```

Nell‟immagine, si può vedere come agisce il metodo ricorsivo se il nodo su cui si trova il puntatore non è una foglia:

```
if(walker->IsLeaf())
{
    single_bit_sequence->push_back(bit);
    while(walker->GetValue()!=alfabeto_estratto.at(i))
        i++;
    for(int j=0; j<single_bit_sequence->size(); j++)
        bit_sequence.at(i).push_back(single_bit_sequence->at(j));
}
```

scrive nella variabile single_bit_sequence il bit preso come parametro, e in caso i figli siano diversi dal puntatore nullo, li visita passando come bit 0 per il sinistro e 1 per il destro, oltre che i puntatori a walker che diventa i rispettivi figli e a single_bit_sequence in modo tale che la sequenza venga ricordata. Trovata una foglia, cerca il carattere della foglia in *alfabeto_estratto* e scrive la sequenza in *bit_sequence* alla cella corrispondente a quella di *alfabeto_estratto*.

- CreateTree(): Il metodo crea l‟albero di Huffman estraendo dalla coda di priorità il minimo, quindi salva in *alfabeto_estratto* il valore e usa l‟occorrenza del valore per stabilire come costruire l‟albero. Il metodo è molto grande, ma è possibile riassumerlo concentrandosi su come lavora: prende il minimo estratto dalla coda e confronta l‟occorrenza col minimo estratto dalla variabile f_nodo, un vector che contiene tutte le occorrenze dei nodi "vuoti", ovvero quei nodi che fanno solo da ponte e hanno come occorrenza la somma delle occorrenze dei nodi figli, e sceglie se prelevare il secondo minimo dalla coda oppure se unire il primo minimo di f_nodo al secondo minimo prelevato da f_nodo. Laddove scegliesse di prelevare il secondo minimo dalla coda, lo confronta poi con il primo estratto da f_nodo, per scegliere se unire i due provenienti dalla coda o se unire i due primi provenienti uno dalla coda e l‟altro da f_nodo.

- Encoder(char*): il metodo codifica il file determinato dal nome passato come parametro e ne scrive uno binario tramite la codifica di Huffman. Se il file ha abbastanza caratteri differenti e pochi di quei caratteri son molto frequenti, la codifica avrà un risultato eccellente, che può raggiungere più dell‟80% di byte risparmiati, se invece il file è troppo grande, rendendo molti caratteri frequenti, la compressione sarà poco efficiente e in certi casi potrebbe addirittura peggiorare la dimensione del file. È infatti consigliato applicare la codifica di Huffman solo a file che contengono almeno una ventina di caratteri differenti e di dimensioni comprese tra 1Kbyte e 10Mbyte. Il metodo è molto lungo, così come per il metodo Decoder(), indi per cui non saranno messe immagini, ma si potranno trovare alla fine della spiegazione delle classi tutti i file per esteso. Il metodo in sintesi crea e apre in scrittura il file .bin, legge i caratteri e ne prende le sequenze binarie dall‟attributo *bit_sequence* per poterle scrivere sul file binario codificato.

- Decoder(): il metodo legge il file codificato tramite il metodo Encoder(char*) e prende le sequenze di 8 bit ciascuna e le ricerca nell‟albero di Huffman; se trova una foglia, scrive il carattere corrispondente sul file che ha creato e aperto in scrittura, col formato del file inizialmente letto per essere codificato.

La classe a cui è aggregata la classe file è la classe albero. Essa presenta quattro attributi, di cui due son puntatori.

Gli attributi:

- *frequenza*: la frequenza del nodo, può essere la somma delle frequenze dei nodi figli oppure, se il nodo è una foglia, è la frequenza del carattere del nodo

- *valore*: il carattere del nodo, se è un nodo ponte il valore è NULL (0x0), altrimenti ha il valore del carattere estratto dalla coda di priorità

- *pfs*: puntatore al figlio sinistro

- *pfd*: puntatore al figlio destro

I metodi:

- albero(): è il costruttore che inizializza i valori di *frequenza* e *valore* a 0

- IsLeaf(): metodo che controlla se il nodo su cui viene chiamato è una foglia. Se i puntatori ai figli sinistro e destro son entrambi nullptr, allora il metodo booleano ritorna true, altrimenti ritorna false

```
bool albero::IsLeaf()
{
    if(pfs==nullptr&&pfd==nullptr)
        return true;
    else
        return false;
}
```

- Set-: I metodi Set servono tutti ad assegnare valori agli attributi; vi è un metodo set per ogni attributo. I costruttori son inutilizzabili in questo caso per assegnare le variabili dato che nella file::CreateTree() tutti gli attributi vengono assegnati in una volta, ma è l‟unico caso in cui accade e tutte le altre variabili sparse per il programma di tipo albero non vedono i loro attributi settati in una volta, indi per cui è stato scelto di impiegare i metodi Set per l‟assegnazione.

```
void SetFrequence(unsigned long int);
void SetValue(char);
void SetAddressLeftSon(albero*);
void SetAddressRightSon(albero*);
```

- Get-: I metodi Get ritornano i valori degli attributi. Esiste un metodo Get per ogni attributo tranne che per la frequenza, dato che non è mai stato necessario.

L‟ultima classe è classe priorityqueue che va a comporre la classe file. La classe priorityqueue ha solo un attributo:

Gli attributi:

- *alfabeto*: l‟alfabeto dei caratteri letti dal file è contenuto in questo vector

I metodi:

- priorityqueue(): è il costruttore di default

- Insert(char): il metodo è un inserimento nella coda del carattere passatogli come parametro

- ExtractMin(): metodo che ritorna il minimo del vector *alfabeto*. Il sort mette tutto in ordine decrescente, quindi il minimo sarà l‟ultimo elemento del vector.

- SelectSort(): il select sort riordina array e vector in ordine crescente o decrescente, dipende da se si parla di un select sort di massimo o di minimo. Il select sort usato come metodo è di massimo, riordinando in ordine decrescente in modo tale che gli elementi minimi della coda vadano per ultimi, il che aiuta la cancellazione via erase()

- ChangePriority(): cambia la priorità dell’elemento puntato dall’indice passato come parametro, rendendolo a massima priorità

### 1.3 – I file del progetto

I file del progetto sono nove:

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```cpp
#include "main.hpp"

int main()
{
    file comprimi;
    int i=0;
    char *nomefile = new char[namesize];

    cout << "--------------------------------" << endl;
    cout << "Nome del file da comprimere: " << endl;
    fgets(nomefile, namesize, stdin);
    while(i<namesize)
    {
        if(*(nomefile+i)=='\n')
            break;
        i++;
    }
    *(nomefile+i)='\0';
    comprimi.Encoder(nomefile);
    system("cls"); cout << "Premere ENTER per avviare la decompressione"; getchar();
system("cls");
    comprimi.Decoder();
    cout << "--------------------------------" << endl;
    getchar();
    return 0;
}
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```cpp
#ifndef MAIN_HPP_INCLUDED
#define MAIN_HPP_INCLUDED

#include "librerie.hpp"
#include "priorityqueue.hpp"
#include "classalbero.hpp"
#include "classfile.hpp"

#endif
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```cpp
#include "classfile.hpp"

void file::Alphabet(char current_value)
{
    int i = 0;
    while(i < size_alfabeto && coda.GetValue(i) != current_value)
        i++;
    if(i == size_alfabeto)
    {
        coda.PushBack(current_value, 1);
        size_alfabeto++;
    }
    else
        coda.Set(coda.GetOccurrence(i)+1, i);
}

unsigned long int file::ExtractMin()
{
    unsigned long int occorrenza;
    alfabeto_estratto.push_back(coda.ExtractMinValue());
    occorrenza=coda.ExtractMinOccurrence();
    coda.DeleteMin();
    return occorrenza;
}

```

```
26. void file::ReadFile(char *filename)
27. {
28.     FILE *fp;
29.     int i=0;
30.     while(filename[i]!='.')
31.         name+=filename[i++];
32.     for(int j=0; j<4; j++)
33.         name+=filename[i++];
34.     char current_value;
35.     fp = fopen(filename, "rb");
36.     if(fp==nullptr)
37.         exit(-2);
38.     fseek(fp, 0L, SEEK_END);
39.     filesize = ftell(fp);
40.     fseek(fp, 0L, SEEK_SET);
41.     binary_encode.reserve(filesize+1);
42.     cout << "Lettura file: " << filename << endl;
43.     cout << "Dimensione file: " << filesize << " byte" << endl;
44.     cout << "Tempo previsto necessario: ";
45.     switch(filesize/DIVISORE_TEMPO)
46.     {
47.         case 0 : cout << "meno di un minuto" << endl; break;
48.         case 1 : cout << "circa 1 minuto" << endl; break;
49.         default: cout << "circa " << filesize/DIVISORE_TEMPO << " minuti" << endl; break;
50.     }
51.     for(int i=0; i<filesize; i++)
52.     {
53.         fread(¤t_value, sizeof(char), 1, fp);
54.         fseek(fp, 0, SEEK_CUR);
55.         binary_encode.push_back(current_value);
56.         Alphabet(current_value);
57.     }
58.     cout << "\nIl file e' stato letto correttamente\n" << endl;
59.     cout << "Caratteri differenti trovati: " << size_alfabeto << endl;
60.     coda.SelectSort();
61.     fclose(fp);
62. }
63. 
64. void file::CreateTree()
65. {
66.     vector<unsigned long int> f_nodo;
67.     vector<int> i_nodo;
68.     unsigned long int min_occ, sec_min_occ, min_coda=0, sec_min_coda=0;
69.     int min_i, sec_min_i;
70.     int i=size_alfabeto-1, j=0;
71.     do
72.     {
73.         min_occ=0xffffffff;
74.         sec_min_occ=0xffffffff;
75.         for(int k=0; k<f_nodo.size(); k++)
76.         {
77.             if(min_occ>f_nodo.at(k))
78.             {
79.                 min_occ=f_nodo.at(k);
80.                 min_i=k;
81.             }
82.             else if(sec_min_occ>f_nodo.at(k)&&f_nodo.at(k)>=min_occ)
83.             {
84.                 sec_min_occ=f_nodo.at(k);
85.                 sec_min_i=k;
86.             }
87.         }
88.         if(i>0)
89.         {
90.             if(min_coda==0)
91.             {
92.                 min_coda=ExtractMin();
```

```
93.             }
94.             if(min_coda<=min_occ||f_nodo.size()==1)
95.             {
96.                 sec_min_coda=ExtractMin();
97.                 i--;
98.                 if(sec_min_coda<=sec_min_occ)
99.                 {
100.                            i--;
101.                            tree.push_back(new albero);
102.                            tree.at(j)->SetFrequence(min_coda+sec_min_coda);
103.                            f_nodo.push_back(min_coda+sec_min_coda);
104.                            i_nodo.push_back(j);
105.
106.                            tree.push_back(new albero);
107.                            tree.at(j+1)->SetFrequence(min_coda);
108.                            tree.at(j+1)-
    >SetValue(alfabeto_estratto.at(alfabeto_estratto.size()-2));
109.                            tree.at(j)->SetAddressLeftSon(tree.at(j+1));
110.                            min_coda=0;
111.
112.                            tree.push_back(new albero);
113.                            tree.at(j+2)->SetFrequence(sec_min_coda);
114.                            tree.at(j+2)-
    >SetValue(alfabeto_estratto.at(alfabeto_estratto.size()-1));
115.                            tree.at(j)->SetAddressRightSon(tree.at(j+2));
116.                            inizio_tree=tree.at(j);
117.                            sec_min_coda=0;
118.                            j+=3;
119.                        }
120.                        else
121.                        {
122.                            tree.push_back(new albero);
123.                            tree.at(j)->SetFrequence(min_coda+min_occ);
124.                            tree.at(j)->SetAddressRightSon(tree.at(i_nodo.at(min_i)));
125.                            f_nodo.erase(f_nodo.begin()+min_i);
126.                            i_nodo.erase(i_nodo.begin()+min_i);
127.
128.                            f_nodo.push_back(min_coda+min_occ);
129.                            i_nodo.push_back(j);
130.
131.                            tree.push_back(new albero);
132.                            tree.at(j+1)->SetFrequence(min_coda);
133.                            tree.at(j+1)-
    >SetValue(alfabeto_estratto.at(alfabeto_estratto.size()-2));
134.                            tree.at(j)->SetAddressLeftSon(tree.at(j+1));
135.                            inizio_tree=tree.at(j);
136.                            if(sec_min_coda!=0)
137.                            {
138.                                min_coda=sec_min_coda;
139.                                sec_min_coda=0;
140.                            }
141.                            j+=2;
142.                        }
143.                    }
144.                    else if(f_nodo.size()>1)
145.                    {
146.                        tree.push_back(new albero);
147.                        tree.at(j)->SetFrequence(min_occ+sec_min_occ);
148.                        tree.at(j)->SetAddressLeftSon(tree.at(i_nodo.at(min_i)));
149.                        tree.at(j)->SetAddressRightSon(tree.at(i_nodo.at(sec_min_i)));
150.                        f_nodo.erase(f_nodo.begin()+min_i);
151.                        i_nodo.erase(i_nodo.begin()+min_i);
152.                        f_nodo.erase(f_nodo.begin()+sec_min_i-1);
153.                        i_nodo.erase(i_nodo.begin()+sec_min_i-1);
154.
155.                        f_nodo.push_back(min_occ+sec_min_occ);
156.                        i_nodo.push_back(j);
```

```
157.
158.                     inizio_tree=tree.at(j);
159.                     j+=1;
160.                 }
161.             }
162.             else if(f_nodo.size()>1)
163.             {
164.                 tree.push_back(new albero);
165.                 tree.at(j)->SetFrequence(min_occ+sec_min_occ);
166.                 tree.at(j)->SetAddressLeftSon(tree.at(i_nodo.at(min_i)));
167.                 tree.at(j)->SetAddressRightSon(tree.at(i_nodo.at(sec_min_i)));
168.                 f_nodo.erase(f_nodo.begin()+min_i);
169.                 i_nodo.erase(i_nodo.begin()+min_i);
170.                 f_nodo.erase(f_nodo.begin()+sec_min_i-1);
171.                 i_nodo.erase(i_nodo.begin()+sec_min_i-1);
172.
173.                 f_nodo.push_back(min_occ+sec_min_occ);
174.                 i_nodo.push_back(j);
175.
176.                 inizio_tree=tree.at(j);
177.                 j+=1;
178.             }
179.         }while(i>0||f_nodo.size()>1);
180.         if(i==0&&size_alfabeto>1)
181.         {
182.             if(min_coda==0)
183.                 min_coda=ExtractMin();
184.             tree.push_back(new albero);
185.             tree.at(j)->SetFrequence(min_coda+f_nodo.at(0));
186.             tree.at(j)->SetAddressRightSon(tree.at(i_nodo.at(0)));
187.
188.             f_nodo.erase(f_nodo.begin());
189.             i_nodo.erase(i_nodo.begin());
190.
191.             tree.push_back(new albero);
192.             tree.at(j+1)->SetFrequence(min_coda);
193.             tree.at(j+1)->SetValue(alfabeto_estratto.at(alfabeto_estratto.size()-1));
194.             tree.at(j)->SetAddressLeftSon(tree.at(j+1));
195.             inizio_tree=tree.at(j);
196.         }
197.         else if(size_alfabeto==1)
198.         {
199.             min_coda=ExtractMin();
200.             tree.push_back(new albero);
201.             tree.at(j)->SetFrequence(min_coda);
202.
203.             tree.push_back(new albero);
204.             tree.at(j+1)->SetFrequence(min_coda);
205.             tree.at(j+1)->SetValue(alfabeto_estratto.at(i));
206.             tree.at(j)->SetAddressLeftSon(tree.at(j+1));
207.             inizio_tree=tree.at(j);
208.         }
209.     }
210.
211.
212.     void file::MarkTheWay(albero *walker, int bit, vector<int> *single_bit_sequence)
213.     {
214.
215.         int i=0;
216.         if(walker->IsLeaf())
217.         {
218.             single_bit_sequence->push_back(bit);
219.             while(walker->GetValue()!=alfabeto_estratto.at(i))
220.                 i++;
221.             for(int j=0; j<single_bit_sequence->size(); j++)
222.                 bit_sequence.at(i).push_back(single_bit_sequence->at(j));
223.         }
```

```
224.            else
225.            {
226.                if(walker!=inizio_tree)
227.                    single_bit_sequence->push_back(bit);
228.                if(walker->GetAddressLeftSon()!=nullptr)
229.                {
230.                    MarkTheWay(walker->GetAddressLeftSon(), 0, single_bit_sequence);
231.                    single_bit_sequence->erase(single_bit_sequence-
    >begin()+single_bit_sequence->size()-1);
232.                }
233.                if(walker->GetAddressRightSon()!=nullptr)
234.                {
235.                    MarkTheWay(walker->GetAddressRightSon(), 1, single_bit_sequence);
236.                    single_bit_sequence->erase(single_bit_sequence-
    >begin()+single_bit_sequence->size()-1);
237.                }
238.            }
239.            return;
240.        }
241.
242.        void file::Encoder()
243.        {
244.            FILE *fp;
245.            ReadFile(filename);
246.            CreateTree();
247.
248.            albero *walker=inizio_tree;
249.            vector<int>single_bit_sequence;
250.            vector<unsigned int>buffer;
251.            bit_sequence.reserve(alfabeto_estratto.size());
252.            for(int i=0; i<alfabeto_estratto.size(); i++)
253.                bit_sequence.push_back(buffer);
254.            MarkTheWay(walker, 0, &single_bit_sequence);
255.
256.            char filename[name.size()+2];
257.            char file_format[]=".bin\0";
258.            int i=0, j=0, k=0, k2=0, h=0;
259.            vector<unsigned int> sequence;
260.            char character=0x0;
261.            for(i=0; i<8; i++)
262.                sequence.push_back(0);
263.            i=0;
264.            while(name.at(i)!='.')
265.            {
266.                filename[i]=name.at(i);
267.                i++;
268.            }
269.            filename[i]='2';
270.            i++;
271.            for(j=0; j<5; j++)
272.            {
273.                filename[i]=file_format[j];
274.                i++;
275.            }
276.            fp = fopen(filename, "wb");
277.            if(fp==nullptr)
278.                exit(-3);
279.           cout << "Creazione file compresso: " << filename << endl << "L'operazione richi
    ede circa il tempo di lettura" << endl;
280.            for(i=0; i<binary_encode.size(); i++)
281.            {
282.                if(i==97)
283.                    int i=0;
284.                j=0;
285.                while(binary_encode.at(i)!=alfabeto_estratto.at(j))
286.                    j++;
287.                k2=0;
```

```
288.                while(k2<bit_sequence.at(j).size())
289.                {
290.                    sequence.at(k++)=bit_sequence.at(j).at(k2++);
291.                    if(k>=8)
292.                    {
293.                        for(h=0; h<7; h++)
294.                        {
295.                            character=character|sequence.at(h);
296.                            character=character<<1;
297.                        }
298.                        character=character|sequence.at(h);
299.                        fwrite(&character, sizeof(char), 1, fp);
300.                        character=0x0;
301.                        fseek(fp, 0, SEEK_CUR);
302.                        k=0;
303.                    }
304.                }
305.            }
306.            if(k>0||sequence.at(0)==1)
307.            {
308.                while(k<8)
309.                    sequence.at(k++)=1;
310.                for(int h=0; h<7; h++)
311.                {
312.                    character=character|sequence.at(h);
313.                    character=character<<1;
314.                }
315.                    character=character|sequence.at(h);
316.                    fwrite(&character, sizeof(char), 1, fp);
317.                    fseek(fp, 0, SEEK_CUR);
318.            }
319.            fclose(fp);
320.            cout << "\nScrittura del file compresso completata" << endl;
321.        }
322.
323.        void file::Decoder()
324.        {
325.            FILE *fp;
326.            char filename[name.size()+2], file_format[]=".bin\0", current_value;
327.            int i=0, bit, written=0;
328.            vector <int> sequence;
329.            albero *walker=inizio_tree;
330.            for(i=0; i<8; i++)
331.                sequence.push_back(0);
332.            i=0;
333.            while(name.at(i)!='.')
334.            {
335.                filename[i]=name.at(i);
336.                i++;
337.            }
338.            filename[i]='2';
339.            i++;
340.            for(int j=0; j<5; j++)
341.            {
342.                filename[i]=file_format[j];
343.                i++;
344.            }
345.            cout << "\nLettura file: " << filename << endl;
346.            fp = fopen(filename, "rb");
347.            if(fp==nullptr)
348.                exit(-2);
349.            fseek(fp, 0L, SEEK_END);
350.            encoded_filesize = ftell(fp);
351.            fseek(fp, 0L, SEEK_SET);
352.            i=0;
353.            while(name.at(i)!='.')
354.            {
```

```
355.                filename[i]=name.at(i);
356.                i++;
357.            }
358.            filename[i]='2';
359.            i++;
360.            for(int j=0; j<4; j++)
361.            {
362.                filename[i]=name.at(i-1);
363.                i++;
364.            }
365.            while(i<name.size())
366.            {
367.                filename[i]=name.at(i-1);
368.                i++;
369.            }
370.            filename[i]='\0';
371.            cout << "Tempo previsto necessario: ";
372.            switch(encoded_filesize/DIVISORE_TEMPO)
373.            {
374.                case 0 : cout << "meno di un minuto" << endl; break;
375.                case 1 : cout << "circa 1 minuto" << endl; break;
376.                default: cout << "circa " << encoded_filesize/DIVISORE_TEMPO << " minuti" <
    < endl; break;
377.            }
378.            for(i=0; i<encoded_filesize; i++)
379.            {
380.                fread(¤t_value, sizeof(char), 1, fp);
381.                fseek(fp, 0, SEEK_CUR);
382.                binary_decode.push_back(current_value);
383.            }
384.            cout << "\nIl file e' stato letto correttamente" << endl;
385.           cout << "Creazione file decompresso: " << filename << endl << "L'operazione ric
    hiede circa il tempo di lettura" << endl;
386.            fclose(fp);
387.            fp = fopen(filename, "wb");
388.            if(fp==nullptr)
389.                exit(-2);
390.
391.            for(i=0; i<binary_decode.size(); i++)
392.            {
393.                current_value=binary_decode.at(i);
394.                for(int j=7; j>=0; j--)
395.                {
396.                    bit=current_value&(int)1;
397.                    sequence.at(j)=bit;
398.                    current_value=current_value>>1;
399.                }
400.                for(int j=0; j<8; j++)
401.                {
402.                    if(sequence.at(j)==0)
403.                        walker=walker->GetAddressLeftSon();
404.                    else if(sequence.at(j)==1&&alfabeto_estratto.size()>1)
405.                        walker=walker->GetAddressRightSon();
406.                    if(walker->IsLeaf()&&written<filesize)
407.                    {
408.                        current_value=walker->GetValue();
409.                        fwrite(current_value, sizeof(char), 1, fp);
410.                        written++;
411.                        fseek(fp, 0, SEEK_CUR);
412.                    }
413.                    If(walker->IsLeaf())
414.                        walker=inizio_tree;
415.                }
416.            }
417.            fclose(fp);
418.            cout << "\nScrittura del file decompresso completata" << endl;
```

```
419.           cout << endl << "La compressione ha risparmiato " << filesize-
    encoded_filesize << " byte (" << 100-((encoded_filesize*100)/filesize) << "%)" << endl;
420.
421.       }
```

main.cpp | main.hpp | classfile.cpp | **classfile.hpp** | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```
1.  #ifndef CLASSFILE_HPP_INCLUDED
2.  #define CLASSFILE_HPP_INCLUDED
3.  #include "librerie.hpp"
4.  #include "classalbero.hpp"
5.  #include "priorityqueue.hpp"
6.  #define DIVISORE_TEMPO 10000000
7.
8.  class file{
9.  private:
10.     long int filesize, encoded_filesize;
11.     string name;
12.     int size_alfabeto;
13.     vector<char> binary_encode, binary_decode;
14.     vector<vector<unsigned int>> bit_sequence;
15.     priorityqueue coda;
16.     vector<char> alfabeto_estratto;
17.     vector<char> char_coda;
18.     vector<unsigned long int> occorrenze;
19.     vector<albero*> tree;
20.     albero *inizio_tree;
21.     void Alphabet(char);
22.     unsigned long int ExtractMin();
23.     void ReadFile(char*);
24.     void CreateTree();
25.     void MarkTheWay(albero*, int, vector<int>*);
26.
27. public:
28.     file(){filesize=0; encoded_filesize=0; size_alfabeto=0; inizio_tree=nullptr;}
29.     void Encoder(char*);
30.     void Decoder();
31. };
32.
33. #endif
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | **classalbero.cpp** | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```
1.  #include "classalbero.hpp"
2.
3.  void albero::SetFrequence(unsigned long int freq)
4.  {
5.      frequenza=freq;
6.  }
7.
8.  void albero::SetValue(char val)
9.  {
10.     valore=val;
11. }
12.
13. void albero::SetAddressLeftSon(albero* nodo)
14. {
15.     pfs=nodo;
16. }
17.
18. void albero::SetAddressRightSon(albero *nodo)
19. {
20.     pfd=nodo;
21. }
```

```
22.
23. char albero::GetValue()
24. {
25.     return valore;
26. }
27.
28. albero* albero::GetAddressLeftSon()
29. {
30.     return pfs;
31. }
32.
33. albero* albero::GetAddressRightSon()
34. {
35.     return pfd;
36. }
37.
38. bool albero::IsLeaf()
39. {
40.     if(pfs==nullptr&&pfd==nullptr)
41.         return true;
42.     else
43.         return false;
44. }
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | **classalbero.hpp** | priorityqueue.cpp | priorityqueue.hpp | librerie.hpp

```
1.  #ifndef CLASSALBERO_HPP_INCLUDED
2.  #define CLASSALBERO_HPP_INCLUDED
3.
4.  #include "librerie.hpp"
5.
6.  class albero{
7.  private:
8.      unsigned long int frequenza;
9.      char valore;
10.     albero *pfs;
11.     albero *pfd;
12. public:
13.     albero(){frequenza=0; valore=0; pfs=nullptr; pfd=nullptr;}
14.     void SetFrequence(unsigned long int);
15.     void SetValue(char);
16.     void SetAddressLeftSon(albero*);
17.     void SetAddressRightSon(albero*);
18.     char GetValue();
19.     albero* GetAddressLeftSon();
20.     albero* GetAddressRightSon();
21.     bool IsLeaf();
22. };
23. #endif
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | **priorityqueue.cpp** | priorityqueue.hpp | librerie.hpp

```
1.  #include "priorityqueue.hpp"
2.
3.  void priorityqueue::Insert(char carattere)
4.  {
5.      alfabeto.push_back(carattere);
6.  }
7.
8.  void priorityqueue::SelectSort(vector<unsigned long int> *occorrenze)
9.  {
10.     int maxvalue, i_max, i=0, j=-1;
```

```
    char appoggio;
    do
    {
        j++;
        maxvalue=occorrenze->at(j);
        i_max=j;
        for(i=j; i<occorrenze->size(); i++)
        {
            if(maxvalue<occorrenze->at(i))
            {
                maxvalue=occorrenze->at(i);
                i_max=i;
            }
        }
        occorrenze->at(i_max)=occorrenze->at(j);
        occorrenze->at(j)=maxvalue;
        appoggio=alfabeto.at(i_max);
        alfabeto.at(i_max)=alfabeto.at(j);
        alfabeto.at(j)=appoggio;
    }while(j<alfabeto.size()-1);
}

char priorityqueue::ExtractMin()
{
    char var = alfabeto.at(alfabeto.size()-1);
    alfabeto.erase(alfabeto.begin()+alfabeto.size()-1);
    return var;
}
```

```
void priorityqueue::ChangePriority(int i)
{
    char val;
    val=alfabeto[i];
    for(int j=alfabeto.size()-1; j>=i; j--)
    {
        alfabeto[j-1]=alfabeto[j];
    }
    alfabeto[alfabeto.size()-1]=val;
}
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | **priorityqueue.hpp** | librerie.hpp

```
#ifndef PRIORITYQUEUE_HPP_INCLUDED
#define PRIORITYQUEUE_HPP_INCLUDED
#include "librerie.hpp"

class priorityqueue{
private:
    vector<char> alfabeto;
public:
    priorityqueue(){;}
    void Insert(char);
    char ExtractMin();
    void SelectSort(vector<unsigned long int>*);
    void ChangePriority(int);
};

#endif
```

main.cpp | main.hpp | classfile.cpp | classfile.hpp | classalbero.cpp | classalbero.hpp | priorityqueue.cpp | priorityqueue.hpp | **librerie.hpp**

```
#ifndef LIBRERIE_HPP_INCLUDED
```

```
2.  #define LIBRERIE_HPP_INCLUDED
3.
4.  #include <iostream>
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <string.h>
8.  #include <vector>
9.  #define namesize 30
10.
11. using namespace std;
12.
13. #endif
```

## 2 – ALGORITMO DI BELLMAN FORD

### 2.1 – Analisi della richiesta

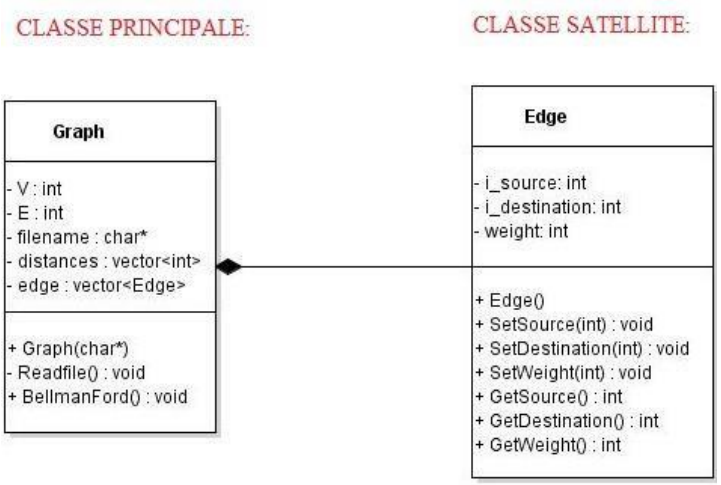
Traccia: *Si implementi l'algoritmo di **Bellman-Ford** per i cammini minimi in un grafo orientato. Si verifichi la correttezza del programma su un problema reale.*

L‟algoritmo di Bellman-Ford vuole trovare i cammini minimi per ogni nodo del grafo analizzato, a partire da uno dei nodi del grafo chiamato sorgente (source). Il grafo presenta V nodi (vertices) e E archi (edges), gli archi sono pesati e possono anche avere peso negativo. L‟algoritmo funziona grazie alla **tecnica greedy**, in modo molto simile allo Dijkstra, ma in questo caso, dato che possono esserci archi con peso negativo, invece di scegliere di analizzare l‟arco con peso minore, analizza tutti gli archi V-1 volte, dove V è il numero di vertici del grafo. Il primo passo del Bellman-Ford è rilassare tutti gli archi, successivamente individua cicli negativi che costringerebbero la visita del grafo ad incagliare in un ciclo infinito, infine rende noti all‟utente i cammini minimi di ogni vertice dalla sorgente. La complessità dell‟algoritmo è data da vertici*archi (O(|V||E|)), dal momento che analizza V volte tutti gli archi.

Tecnica Greedy: si è parlato della tecnica greedy nel primo programma

### 2.2 – Il programma

Il diagramma UML è il modo più semplice per capire la struttura del programma e come son relazionate le classi tra loro. La classe principale è la classe Graph; presenta quattro attributi, tutti privati, e tre metodi, tutti pubblici.




Gli attributi:

- *V*: numero di vertici del grafo

- *E*: numero di archi del grafo

- *distances*: vettore di interi che ricorda le distanze; è il vettore che contiene la soluzione del problema

- *edge*: vettore di Edge che contiene tutti gli archi; è del tipo della classe in composizione con Graph

I metodi:

- Graph(): è il costruttore, inizializza *V* e *E* a zero e imposta la prima cella di *distances* a zero

- ReadFile(): legge il grafo da file con una serie di accorgimenti per rendere il file del grafo leggibile anche con un semplice editor di testo, difatti, immagazzina ciò che legge in quattro variabili diverse e riesce a stabilire, in base a cosa ha letto, dove immagazzinare i dati.

Questo *switch()* controlla se il carattere letto precedentemente è uno dei caratteri riconosciuti, e in base al carattere che incontra cambia il valore della variabile *fase* per poter, con un altro *switch()* successivo, eseguire la serie di operazioni opportuna.

```
switch(carattere)//Switch sul carattere letto
{
    case '[' : fase=2; fread(&carattere, sizeof(char), 1, fp); break;
    case '.' : fase=3; fread(&carattere, sizeof(char), 1, fp); break;
    case '=' : fase=4; fread(&carattere, sizeof(char), 1, fp); break;
    case '\n': fase=5; break;
    default  : break;
}
```

```
V=5
E=8
edge[1].i_source=0
edge[2].i_destination=2
edge[1].weight=4
edge[0].i_source=0
edge[4].i_source=1
edge[2].weight=3
edge[4].weight=2
edge[5].weight=5
edge[6].i_source=3
edge[6].i_destination=1
edge[7].i_source=4
edge[7].weight=-3
edge[3].i_source=1
edge[2].i_source=1
edge[0].weight=-1
edge[0].i_destination=1
edge[5].i_destination=2
edge[4].i_destination=4
edge[7].i_destination=3
edge[6].weight=1
edge[5].i_source=3
edge[1].i_destination=2
edge[3].weight=2
edge[3].i_destination=3
```

I file che contengono i grafi vengono letti intelligentemente, indi per cui la ReadFile() è capace di leggere anche file contenenti gli archi in ordine misto. L'importante, per il corretto funzionamento del programma, è che i primi due siano V ed E, perché nonostante la ReadFile() riesca a riconoscere l'attributo che sta leggendo, nella lettura di V ed E, il metodo alloca memoria, e se si leggessero prima gli archi, il metodo proverebbe a inserire il nuovo arco in uno spazio inesistente, andando in errore di esecuzione "segmentation fault". Infine, per non incorrere in un ciclo infinito, il file deve presentare un "a capo" (\n) dopo l'ultima riga scritta, dal momento che è stato molto più semplice inserire un semplice controllo if() per far finire la lettura, piuttosto di feof(FILE*), la quale richiede che si sia letto l'ultimo carattere prima di segnalare la fine, ed inoltre manda il puntatore del file avanti rispetto a dove si era fermato. Infine, le funzioni utilizzate per l'i/o su file son le funzioni standard del c, per via che offrono maggior controllo sulle operazioni e hanno una chiarezza di scrittura e di conseguente reinterpretazione molto semplice.

- BellmanFord():

Il metodo BellmanFord() è il metodo che viene chiamato nel main e che svolge tutto il necessario all‟algoritmo di Bellman-Ford. Per prima cosa, grazie al costruttore Graph(char*) a cui si passa come parametro il nome del file grafo, il metodo BellmanFord() sa il nome del file da cui è stato letto il grafo; successivamente, ci sono tre cicli for() che vanno a sviluppare l‟algoritmo di Bellman-Ford:

1) Il primo ciclo for rilassa gli archi, andando a riscrivere le distanze dei vertici dalla sorgente. Il ciclo interno, per ogni arco, legge qual è la sorgente e qual è la destinazione, e infine ne legge il peso: se la distanza tra la sorgente e la destinazione precedentemente

```
for(int i=0; i<V; i++)
{
    for(int j=0; j<E; j++)
    {
        s=edge.at(j).GetSource();
        d=edge.at(j).GetDestination();
        w=edge.at(j).GetWeight();
        if(distances.at(s)!=0xffffffff && distances.at(d)>distances.at(s)+w)
            distances.at(d)=distances.at(s)+w;
    }
}
```

immagazzinata in *distances* è maggiore rispetto alla distanza tra sorgente + peso dell‟arco, allora riscriverà, nella cella di *distances* corrispondente alla destinazione, la distanza come peso sorgente + peso dell‟arco. Il ciclo interno è ripetuto V volte dal momento che, man mano che svolge più volte il ciclo interno, è come se creasse la strada per raggiungere i nodi più lontani dalla sorgente, dal momento che possono risultare essere più lontani rispetto a come dovrebbe essere (facendo quindi un altro cammino)

2) Il secondo ciclo for() segnala se ci son cicli negativi nel grafo, controllando se il peso della destinazione risulta essere maggiore del peso della sorgente + il peso dell‟arco: se la

```
for(int i=0; i<E; i++)
{
    s=edge.at(i).GetSource();
    d=edge.at(i).GetDestination();
    w=edge.at(i).GetWeight();
    if(distances.at(s)!=0xffffffff && distances.at(d)>distances.at(s)+w)
        cout << "C'e' un ciclo negativo tra " << alfabeto[s] << " e " << alfabeto[d] << endl;
}
```

condizione è soddisfatta, allora manda in output un messaggio di segnalazione del ciclo negativo, specificando tra quali nodi è avvenuto.

3) Il terzo ciclo for() è un semplice output che mostra le distanze dalla sorgente, in caso di svariati cicli negativi, si potrà notare che tra i vertici che causano il ciclo, sicuramente le

```
for(int i=0; i<distances.size(); i++)
{
    cout << "Il nodo " << alfabeto[i] << " dista da " << alfabeto[0] << " di " << distances.at(i) << endl;
}
```

distanze saranno negative, altrimenti, mostra le distanze minori dalla sorgente.

Il metodo Bellman-Ford presenta un array statico denominato *alfabeto* che serve solamente per gli output, infatti presenta 26 caratteri maiuscoli, seguiti da 26 caratteri miniuscoli, seguiti infine dai numeri da 0 a 9, per un totale di 62 caratteri e di conseguenza potenzialmente 62 vertici diversi.

La classe secondaria è la classe Edge, che va a definire il tipo utilizzato per gli archi tra i vertici nella classe Graph. La classe Edge è piuttosto semplice, contiene tre attributi e 6 metodi, di cui tre Set e tre Get:

Gli attributi:

- *i_source*: variabile che contiene l‟indice della sorgente dell‟arco

- *i_destination*: variabile che contiene l‟indice della destinazione dell‟arco

- *weight*: variabile che contiene il peso dell‟arco

I metodi:

```
//Metodi Set
void SetSource(int i);
void SetDestination(int i);
void SetWeight(int w);
//Metodi Get
int GetSource();
int GetDestination();
int GetWeight();
```

I metodi sono semplici Set e Get, non ci sono operazioni da eseguire sugli archi se non leggere il file e depositare i dati in memoria tramite i Set e leggere i dati al momento opportuno con i Get. I metodi Set son stati messi, nonostante i dati siano tutti disponibili da subito, dato che la funzione ReadFile() setta i valori man mano che esegue la sua lettura, e ciò è impossibile da fare via costruttore. La funzione ReadFile() è stata strutturata in modo che leggesse da file riconoscendo cosa sta leggendo dato che, laddove fossero stati messi solo i valori, prima di tutto il file doveva contenerli in ordine, e in secondo luogo diveniva confusionaria la lettura e la conseguente modifica del file via editor di testo (son stati fatti vari test con il grafo 2 che ha più vertici ed archi, e modificarlo dopo averlo scritto diveniva complicato, specialmente per ritrovare l‟arco e l‟attributo corrispondente da modificare, spesso e volentieri modificando i valori sbagliati)

### 2.3 – I file del progetto

I file del progetto sono sette:

main.cpp | main.hpp | graph.cpp | graph.hpp | edge.cpp | edge.hpp | librerie.hpp

```
1.  #include "main.hpp"
2.
3.  int main()
4.  {
5.      Graph grafo1("grafo1.txt"), grafo2("grafo2.txt");
6.      grafo1.BellmanFord();
7.      grafo2.BellmanFord();
8.      return 0;
9.  }
```

main.cpp | main.hpp | graph.cpp | graph.hpp | edge.cpp | edge.hpp | librerie.hpp

```cpp
#ifndef MAIN_HPP_INCLUDED
#define MAIN_HPP_INCLUDED

#include "librerie.hpp"
#include "edge.hpp"
#include "graph.hpp"

#endif
```

main.cpp | main.hpp | graph.cpp | graph.hpp | edge.cpp | edge.hpp | librerie.hpp

```cpp
#include "graph.hpp"

void Graph::ReadFile()
{
    FILE *fp;
    fp=fopen(filename, "r");
    if(fp==nullptr)
        exit(-1);
    char carattere;
    string nome, posizione, variabile, valore;
    int fase=1, i=0, posizione_intero=0, valore_intero=0, end_ciclo;
    while(1)
    {
        fread(&carattere, sizeof(char), 1, fp);
        fseek(fp, 0, SEEK_CUR);
        if(fase==5&&carattere=='\n')
            break;
        else if(fase==5&&carattere!='\n')
            fase=1;
        switch(carattere)
        {
            case '[' : fase=2; fread(&carattere, sizeof(char), 1, fp); break;
            case '.' : fase=3; fread(&carattere, sizeof(char), 1, fp); break;
            case '=' : fase=4; fread(&carattere, sizeof(char), 1, fp); break;
            case '\n': fase=5; break;
            default  : break;
        }
        switch(fase)
        {
            case 1 : nome.push_back(carattere); i++; break;
            case 2 : while(carattere!=']'){posizione.push_back(carattere); i++; fread(&car
attere, sizeof(char), 1, fp);} break;
            case 3 : variabile.push_back(carattere); i++; break;
            case 4 : valore.push_back(carattere); break;
            case 5 : for(i=posizione.size()-1; i>=0; i--)
                     {
                         posizione_intero+=((posizione.at(i)&207)*pow(10,(posizione.size()
-i-1)));
                     }
                     for(i=valore.size()-1; i>=0; i--)
                     {
                         if(valore.at(i)!='-')
                             valore_intero+=((valore.at(i)&207)*pow(10,(valore.size()-i-
1)));
                         else
                             valore_intero= -valore_intero;
                     }
```

```
                        if(nome=="V")
                        {
                            V=valore_intero;
                            valore_intero=0;
                            for(i=1; i<V; i++)
                                distances.push_back(0xfffffff);
                        }
                        else if(nome=="E")
                        {
                            E=valore_intero;
                            valore_intero=0;
                            for(i=0; i<E; i++)
                                edge.push_back(Edge());
                        }
                        else if(nome=="edge")
                        {
                            if(variabile=="i_source")
                            {
                                edge.at(posizione_intero).SetSource(valore_intero);
                                valore_intero=0;
                                posizione_intero=0;
                            }
                            else if(variabile=="i_destination")
                            {
                                edge.at(posizione_intero).SetDestination(valore_intero);
                                valore_intero=0;
                                posizione_intero=0;
                            }
                            else if(variabile=="weight")
                            {
                                edge.at(posizione_intero).SetWeight(valore_intero);
                                valore_intero=0;
                                posizione_intero=0;
                            }
                            else
                            {
                                valore_intero=0;
                                posizione_intero=0;
                            }
                        }
                        else
                        {
                            cout << "Non e' stato possibile individuare a quale variabile ass
    egnare il dato" << endl;
                            cout << "L'esecuzione finisce" << endl;
                            exit(-2);
                        }
                        for(i=nome.size(); i>=0; i--)
                           nome.erase(i);
                        for(i=variabile.size(); i>=0; i--)
                           variabile.erase(i);
                        for(i=valore.size(); i>=0; i--)
                           valore.erase(i);
                        for(i=posizione.size(); i>=0; i--)
                           posizione.erase(i);
         }
            }
            fclose(fp);
           cout << "Il file contenente il grafo e' stato letto correttamente" << endl << e
    ndl;
        }

        void Graph::BellmanFord()
        {
            int s, d, w;
            char alfabeto[]="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvz0123456789";
```

```
109.            cout << "-------------------------------------------------" << endl;
110.            cout << filename << endl;
111.            for(int i=0; i<V; i++)
112.            {
113.                for(int j=0; j<E; j++)
114.                {
115.                    s=edge.at(j).GetSource();
116.                    d=edge.at(j).GetDestination();
117.                    w=edge.at(j).GetWeight();
118.                    if(distances.at(s)!=0xfffffff && distances.at(d)>distances.at(s)+w)
119.                        distances.at(d)=distances.at(s)+w;
120.                }
121.            }
122.            for(int i=0; i<E; i++)
123.            {
124.                s=edge.at(i).GetSource();
125.                d=edge.at(i).GetDestination();
126.                w=edge.at(i).GetWeight();
127.                if(distances.at(s)!=0xfffffff && distances.at(d)>distances.at(s)+w)
128.                    cout << "C'e' un ciclo negativo tra " << alfabeto[s] << " e " << alfabe
    to[d] << endl;
129.            }
130.            for(int i=0; i<distances.size(); i++)
131.                cout << "Il nodo " << alfabeto[i] << " dista da " << alfabeto[0] << " di "
    << distances.at(i) << endl;
132.            }
133.            cout << "-------------------------------------------------" << endl;
134.        }
```

main.cpp | main.hpp | graph.cpp | **graph.hpp** | edge.cpp | edge.hpp | librerie.hpp

```
1.  #ifndef GRAPH_HPP_INCLUDED
2.  #define GRAPH_HPP_INCLUDED
3.  #include "librerie.hpp"
4.  #include "edge.hpp"
5.
6.  class Graph
7.  {
8.  private:
9.      int V;
10.     int E;
11.     char *filename;
12.     vector<int> distances;
13.     vector<Edge> edge;
14.     void ReadFile();
15. public:
16.     Graph(char *nome){V=0; E=0; distances.push_back(0); filename=nome; ReadFile();}
17.     void BellmanFord();
18. };
19.
20. #endif
```

main.cpp | main.hpp | graph.cpp | graph.hpp | **edge.cpp** | edge.hpp | librerie.hpp

```
1.  #include "edge.hpp"
2.
3.  void Edge::SetSource(int i)
4.  {
5.      i_source=i;
6.  }
```

```
7.
8.  void Edge::SetDestination(int i)
9.  {
10.     i_destination=i;
11. }
12.
13. void Edge::SetWeight(int w)
14. {
15.     weight=w;
16. }
17.
18. int Edge::GetSource()
19. {
20.     return i_source;
21. }
22.
23. int Edge::GetDestination()
24. {
25.     return i_destination;
26. }
27.
28. int Edge::GetWeight()
29. {
30.     return weight;
31. }
```

main.cpp | main.hpp | graph.cpp | graph.hpp | edge.cpp | **edge.hpp** | librerie.hpp

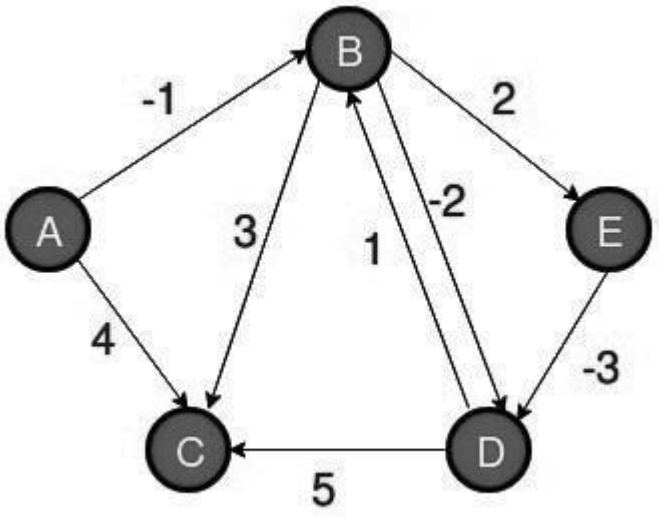
```
1.  #ifndef EDGE_HPP_INCLUDED
2.  #define EDGE_HPP_INCLUDED
3.  #include "librerie.hpp"
4.
5.  class Edge
6.  {
7.  private:
8.      int i_source;
9.      int i_destination;
10.     int weight;
11. public:
12.     Edge(){i_source=0xffffffff; i_destination=0xffffffff; weight=0xffffffff;}
13.     void SetSource(int i);
14.     void SetDestination(int i);
15.     void SetWeight(int w);
16.     int GetSource();
17.     int GetDestination();
18.     int GetWeight();
19. };
20.
21. #endif
```

main.cpp | main.hpp | graph.cpp | graph.hpp | edge.cpp | edge.hpp | **librerie.hpp**

```
1.  #ifndef LIBRERIE_HPP_INCLUDED
2.  #define LIBRERIE_HPP_INCLUDED
3.
4.  #include <iostream>
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <math.h>
8.  #include <vector>
9.  #include <string>
```

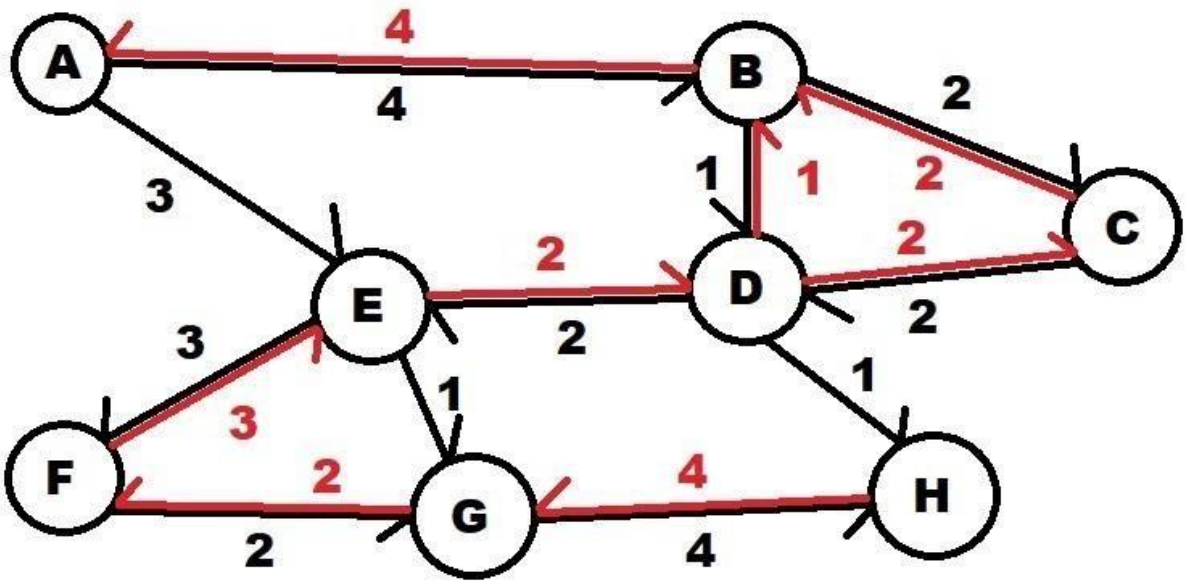
```
10. using namespace std;
11.
12. #endif
```

### 2.4 – I grafi




Il grafo del primo file è un grafo esempio con un ciclo negativo:

Il grafo del secondo file rappresenta un caso reale, con un"ipotetica mappa di una città che presenta la strada principale e dei vicoli scorciatoia. I valori sulla mappa indicano i tempi di percorrenza, e nonostante l"algoritmo di Bellman-Ford è pensato per ovviare i cicli negativi, non esistono tempi negativi in casi reali, quindi i valori saranno tutti positivi. In questo caso quindi, si potrà verificare il funzionamento dell"algoritmo per quanto riguarda i cammini minimi:



**BIBLIOGRAFIA**:

- Trascrizione da Code::Blocks a Word: http://www.planetb.ca/syntax-highlight-word

- C++ Linguaggio, libreria standard, principi di programmazione – Bjarne Stroustrup

- Slide del corso 18-19 laboratorio, 14.1-14.2 teoria