Travail pratique #02 - Algorithmes de compression

Objectifs

— Manipulation de structures de données en mémoire
— Développer des algorithmes de compression
— Utiliser les conteneurs standard (STL)

Règles importantes

1. Chaque classe doit avoir son propre fichier *.h* et *.cpp*
2. N'oubliez pas les *"include guard"*
3. Prenez grand soin de respecter la casse pour les noms de classe, méthode, etc
4. Veuillez commenter votre code *intelligemment*
5. Votre code doit être de qualité exemplaire
6. **Votre code doit compiler sans erreurs pour être corrigé**

Instructions

Ce travail est à réaliser en C++, avec une application de type console.

Vous devez implémenter deux algorithmes de compression très répandus : RLE et huffman. En vous inspirant des vidéos suggérées, des explications et/ou exemples de code donnés en classe, ou de toute autre source de documentation pertinente, veuillez programmer vous-même la compression et la décompression pour chacun des 2 algorithmes.

Vos classes doivent fournir les méthodes publiques suivantes, et ce sont celles-ci qui seront appelées pour la correction :

Listing 1 – Classe RLE

```
class RLE
{
    public:
        // Accepte des données non-compressées en paramètre et retourne les données compressées
        std::string Compresser(const std::string& data);

        // Accepte des données compressées en paramètre et retourne les données décompressées
        std::string Decompresser(const std::string& data);

    private:
        // ...
};
```

Listing 2 – Classe Huffman

```
class Huffman
{
    public:
        // Accepte des données non-compressées en paramètre et retourne les données compressées
        std::string Compresser(const std::string& data);

        // Accepte des données compressées en paramètre et retourne les données décompressées
        std::string Decompresser(const std::string& data);

    private:
        // ...
};
```

Les classes acceptent comme paramètres et retournent des `std::string`, un objet `std::string` pouvant contenir n'importe quel type d'octets (pas juste des caractères affichable).

Veuillez tester vos deux algorithmes de compression avec diverses données pour vous faire une idée de l'usage qui peut en être fait, et dans quels cas il est préférable de choisir l'un par rapport à l'autre.

Assurez vous que votre algorithme est fonctionnel. Lorsque vous compressez des données, vous devez valider que la décompression vous redonne les données originales sans pertes.

Votre code doit compiler sous windows et linux sans modifications.

Important : Veuillez tester vos 2 algorithmes avec 11 types de fichiers (ces fichers sont fournis sur vortex) et consigner vos résultats dans le fichier *observations.xls* (fourni lui aussi sur vortex) :

1. Un texte en langue française (01.txt)
2. Un document HTML (02.html)
3. Une image BMP (03.bmp)
4. Une image JPG (04.jpg)
5. Un fichier exécutable windows (05.exe)
6. Un fichier déjà compressé au format ZIP (06.zip)
7. Un chunk de minecraft (07.bin)
8. Un fichier de 10MB contenant juste des zéros (08.bin)
9. Un fichier de 10MB contenant des octets aléatoires (09.bin)
10. Un fichier contenant des séquences de chiffres (0..9) de longueur aléatoire, maximum 20 de long (10.bin)
11. Un fichier avec le même type de séquences que le 10.bin, mais de longueur maximale de 500 (11.bin)

Pour charger le contenu de vos fichiers en mémoire, cette fonction pourrait vous être utile :

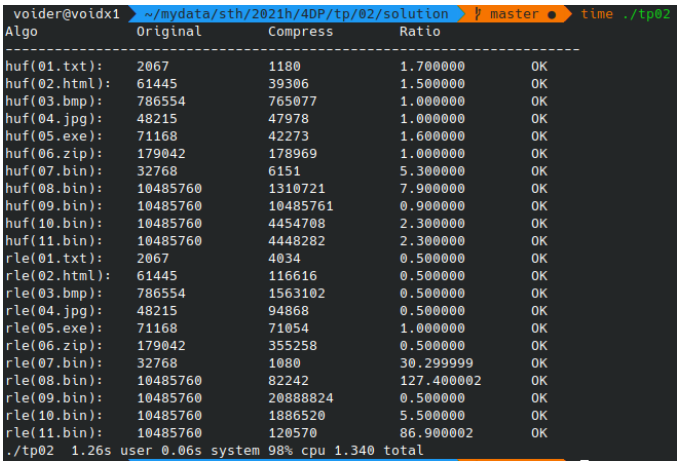
Listing 3 – Fonction LoadFile

```
std::string LoadFile(const std::string& filename)
{
    std::ifstream f(filename.c_str(), std::ios::binary);
    if(!f.is_open())
        return "";

    f.seekg(0, std::ios::end);
    unsigned int len = f.tellg();
    f.seekg(0, std::ios::beg);

    char* tmp = new char[len];

    f.read(tmp, len);
    f.close();

    std::string buffer(tmp, len);

    delete [] tmp;

    return buffer;
}
```

Exemple de résultat attendu

À titre indicatif seulement, voici le résultat attendu. Il est normal que vos chiffres divergent légèrement des miens, mais ils devraient être quand même très près. Le temps d'exécution est aussi à titre indicatif,

et correspond à l'exécution en mode release d'un test compressant l'ensemble des fichiers de tests avec RLE et huffman. Selon la puissance de votre processeur, il se peut que cela varie un peu, mais je m'attend à avoir une performance dans le même ordre de grandeur.



**Hint : Il est beaucoup plus rapide reconstruire l'arbre à partir du dictionnaire et de se servir de l'arbre pour décompresser que d'utiliser directement le dictionnaire.*

Correction

Pour la correction, vous devez vous assurer que lorsque j'utilise votre fonction *Compresser* et que j'obtiens un résultat compressé, la décompression de celui-ci avec votre méthode *Décompresser* donne **exactement** le résultat original.

`data == Decompresser(Compresser(data))`

Références

RLE : `https://www.youtube.com/watch?v=JtoziXq62XA`

Huffman : `https://www.youtube.com/watch?v=apcCVfXfcqE`

Remise

1. Votre fichier observations.xls
2. Votre code source (les fichiers *.cpp et *.h **seulement**)