Multimedia Project Requirement

Description

The project is to find the best lossless compression technique that can be used to compress text. a set of Arabic book reviews containing over 63,000 reviews from www.goodreads.com splitted into 20 files.

You're required to make the needed data analysis (e.g. calculate probabilities …etc), compress the reviews and decode them correctly to generate the original files.

The project is a competition between teams. The winner is the one that gets the smallest number of encoded bits (highest compression ratio). There is a huge percentage of the project grade on competition (See Evaluation Criteria).

Requirements

1. Find the best possible lossless compression method for all text files.
   Note: You are free to make the modifications you want in the compression algorithm that you chose to use. You can even invent a new one.
2. Write a program that takes any input text file and constructs a lossless binary compressed file.
   Constraints: your program should write one binary file for each input text file (See Appendix)
3. Write a program that takes the compressed file only and decodes it to generate the original file.
   Constraints: this file should contain all information you need in decoding process.
   Note: Debugging is a headache in this project, so better to first take a small part of the dataset to test your system and then try it on the whole dataset.
4. Your program should be able to measure the length of the binary file and compute the average compression ratio. The compression ratio equation is as follows:

$$Compression\ Ratio = \frac{Uncompressed\ Size}{Compressed\ Size}\ , should\ be > 1$$

Deliverables

You will work in a group of **4 students**.

Delivery	Due Date
Media	Saturday 21st May, 11:59pm. **To:** m.adel0093@gmail.com **Email Subject:** [Multimedia][Semester] Team <your team no.> Example: [Multimedia][Semester] Team 12 **Attachment:** 1 zip file called: Team<your team no.>.zip Example: Team12.zip. This attachment contains the file
Discussion	In the same week. Schedule will be announced later.

Note: Late media deliveries (after 11:59pm), wrong email subject or file names will make you lose marks. No modifications are allowed after the media delivery.

The 1 attachment zip file "**Team<your team no.>.zip**" must contain the following files:

- **Readme.txt** which contains the names of the team members.
- **Statistics.txt** which contains:
  - For each text file, number of bits in the compressed binary file and compression ratio of each.
  - Total number of bits in all compressed files and total compression ratio.
- **Report.pdf** of 2-5 pages which contains:
  - **[optional]** History of experiments you made before reaching the final algorithm.
  - Clear description of your final compression and decoding algorithm.
    Note: The name of the algorithm is enough if you use a well-known algorithm as is but any modifications you made on the original algorithm or any parameters values that you use in the algorithm should be stated clearly.
  - State clearly what is the content of each of the compressed files (if it contains only code words or any extra info) and what files sent to the decoding program are.
  - Your results: total number of bits in the compressed files and total compression ratio of your final algorithm.
    [optional] the results of any previous algorithms that you tried before).
  - Work division: the tasks performed by each team member.
- **Code.zip** which contains the source codes of your project.

Evaluation Criteria

- **Compression and Decoding** [50%].
  Your code must be object oriented, at least contains encoder class and decoder class. This will be considered in evaluation too.
- **Competition** [50%].
  The winner team will take the 100% of the competition grade. The team with the highest total number of bits will take 0% of the competition grade. The other teams will take a percentage of the compression grade depending on how far they are from the winner team.

Project Files Description

[Test.zip]: this file

[DecodingCheck.exe]: that is used to compare the decode text with the original text to check the correctness of the decoding process.
The command to use it:

```
>DecodingCheck.exe “test_folder_path” “decoded_folder_path”
```

Example,

```
>DecodingCheck.exe “F:\project\test” “F:\project\decoded”
```

The “**test**” folder is the test folder above (that contains the original dataset-file).

The “**decoded**” folder is the folder that contains the decoded data.
It must contain the same number of files as the “test” folder or an error will be generated: “`ERROR#1: Numbers of Files Mismatch!!`”

Each decoded file should have the same name of its corresponding original file in test folder or an error will be generated: “`ERROR#2: Decoded File Not Found!!`”

If any decoded file is not the same as its corresponding original file, the following error will be produced: “`ERROR#3: Decoding Mismatch`”

Note that the decoding check compares the file as a whole, if it’s 100% matching, so don’t put extra new lines and make the tab as is don’t change it to spaces, …etc.

The exe will run on all files and returns the number of files that have decoding mismatch or not found.

Compression Ideas

1. Of course any lossless compression algorithm you studied in lectures. Huffman Coding, Extended Huffman Coding, Integer Arithmetic Coding and Dictionary coding are worth trying.

   Note: You may need to use Integer Arithmetic Coding instead of the original algorithm to solve the problem of underflow.

2. Try other lossless compression algorithm other than the ones studied in lectures.

3. Trying different values for the algorithms that need parameters. (e.g. block size in arithmetic coding)

4. Make modifications on any of the existing algorithms.

5. Two or more levels of compression.

6. You may try to encode group of symbols (words or group of words) instead of individual characters.

7. You may try a de-correlation method in order to try to reduce the number of bytes needed for the compression process. In particular, instead of encoding the given character values directly, encode the difference between each char and the one on its neighbors. This will not be taught in lectures, if you want to use it, search about it.

You better divide your team to try different techniques and generate numbers and take the one with the smallest number of bits.

Appendix

Binary Files

The following C++ example copies a binary file into memory and then writes its content to a new file. You have to find an equivalent way in the language you will use.

```cpp
// Copy a file
#include <fstream>      // std::ifstream, std::ofstream

int main () {
  std::ifstream infile ("test.txt",std::ifstream::binary);
  std::ofstream outfile ("new.txt",std::ofstream::binary);

  // get size of file
  infile.seekg (0,infile.end);
  long size = infile.tellg();
  infile.seekg (0);

  // allocate memory for file content
  char* buffer = new char[size];

  // read content of infile
  infile.read (buffer,size);

  // write to outfile
  outfile.write (buffer,size);

  // release dynamically-allocated memory
  delete[] buffer;

  outfile.close();
  infile.close();
  return 0;
}
```