Cairo University
Faculty of Engineering
Computer Department

Multimedia
CMP 206
Spring 2016

Multimedia Project Requirement

Description

The project is to find the best lossless compression technique that can be used to compress text. The test data that we use in the project is 20 Shakespeare's plays (downloaded as text files from here).

You're required to make any needed data analysis (e.g. calculate probabilities …etc), compress the plays 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 bytes used in the encoded message (highest compression ratio). There is a huge percentage of the project grade on competition (See *Evaluation Criteria* Section).

Requirements

[Algorithm]
Find the best possible lossless compression method for all text files (See *Compression Ideas* Section).

[Encoder Side]
Write a program that takes any input text file and constructs a lossless binary compressed file. Your program must write one binary file for each input text file (See *Appendix* Section). This file must contain any extra information you need in the decoding process of this file (e.g. number of characters in this file …etc).

You are not allowed to save any information specific for one of the test files directly in the decoder but if it must be general for all files, you can save it once on disk and pass it to the decoder directly from a common file. For example, if there was a table (Huffman) that you want to store for each test file, you have to store it inside the corresponding compressed file of this text file. In case you have only one table for all text files, you have the right to save it in one common file (*must be binary too*) and load it directly and once to the decoder. This means that you have to write any common information in the common then load this file to the decoder; not send this information directly as parameters in your code to the decoder. That is because we'll use the size of this common file in the competition later.

Note1: concatenating the extra information specific of each text file after one another and put it in one file is considered a violation to this constraint and will be penalized. The common file mustn't have any information specific to any test file and by the way this common file is rarely used in previous projects.

Note2: the size of the common file is included in the total compressed size that we use to compare teams on in the competition but is not counted in the compressed size of each separate test file (See the equations of the *Statistics* section below).

[Decoder Side]
Write a program that takes a compressed file and decodes it to generate the original file.
Note: Debugging is a hard process in this project, so better to first take a small part of any of the attached text files to test your decoding on then try it on the whole file.

Important Note: You must write the compression and decoding algorithms by yourself. You're not allowed to use any codes or executables from any source. This will be considered as a cheating case.

[Statistics]

1. Your program should output, for each text file *i*, the number of bytes used in the compressed file (compressed size) and the compression ratio (as shown in the equation below). Compression ratio must be > 1.

$$Compression\ Ratio(i) = \frac{Uncompressed\ File\ Size(i)}{Compressed\ File\ Size(i)} \qquad (Equation\ 1)$$

2. Your program should output the total number of bytes used in all compressed files plus the size of the common file (total compressed size) and the total compression ratio.

$$Total\ Compression\ Ratio = \frac{\sum_{k=1}^{20} Uncompressed\ File\ Size(k)}{Common\ File\ Size + \sum_{k=1}^{20} Compressed\ File\ Size(k)} \qquad (Equation\ 2)$$

Note1: Any file size in the equations above (including the common file) is the size of the file on disk (not the number of bytes calculated in your code), so you have to write it as binary file (not an ordinary text file) and use your knowledge (e.g. bitset …etc) to write the minimum number of bytes.

Note2: The competition will be made by comparing the *denominator* of the total compression ratio (equation 2) that represents the total file size of the information the decoder needs to decode the files.

Deliverables

You will work in a group of **4 or 3 students**.

Delivery	**Due Date**
Media	Saturday 21st May, 11:59pm. **To:** eman.hosam.dlv@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 only which contains the files mentioned below.
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 bytes in the compressed binary file and compression ratio of each.
  - Total number of bytes 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 bytes 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.
- **EncodedFiles.zip** which contains the common binary file and the 20 compressed binary files.
- **DecodedFiles.zip** which contains the 20 decoded files.
- **Code.zip** which contains the source codes of your project.

Evaluation Criteria

- **Compression and Decoding** [60%].
  Your code must be object oriented, at least contains encoder class and decoder class. This will be considered in evaluation too.
- **Other Deliverables** [10%].
- **Competition** [30%].
  The winner team will take the 100% of the competition grade. The team with the highest total number of bytes 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 and loser teams.

Project Files Description

[Test.rar]: that contains the 20 chosen Shakespeare's plays that you'll test your algorithm on. Any modifications on the test files are NOT allowed. This will be considered as cheating.

[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 20 plays).

The "**decoded**" folder is the folder that contains the decoded plays.
It must contain the same number of files as the "test" folder (20 files) 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 a 100% match, 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 search for a modified version of Arithmetic Coding 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.

4. Make the modifications you want. It's your program; you're totally free to make whatever modifications you want in encoder and invert it to be understood and correctly-regenerated by decoder.

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 bytes.

Appendix

You should revise **bitset** and **binary files** in whatever language you decided to use. This is a quick but not sufficient example on binary files:

Binary Files

The following C++ example reads a binary file, copies its content into memory (variable buffer below) 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;
}
```