Lossless Compression Project: Issued Oct. 12, Due Nov. 5 at 5 p.m.

Overview

Natural language, such as English, is an important and fundamental data source. From an information theory standpoint, natural language is quite compressible; the entropy of English is at most 1-2 bits per character. The goal of this project is to design an encoder and decoder for compressing English.

We will focus on compressing one particular text file, a concatenation of several novels by the early 19th-century author Jane Austen. The file consists of roughly 477,000 words and about two and a half million characters. The goal is to compress it as much as possible.

Your assignment is to write a Python script that, after reading from a data file that you create, reproduces this file in its entirety. We view the data file as the compressed version of the text and the Python script as the decoder.

Rules

You are asked to submit exactly three files: a data file, a decompression script, and a report. The data file must be named `compressed` and the decompression script must be named `decompress.py`. We place no constraints whatsoever on the `compressed` file. You may place whatever data you like in this file in whatever format you like. The file `decompress.py`, on the other hand, must be a Python script that runs on the ECE School's Linux cluster (`amdpool.ece`) using the command

```
/opt/alttools/bin/python decompress.py
```

Your Python script may only import the `struct` module, which is useful for basic I/O.

This script must create a text file named `out.txt` whose contents exactly match the master file `austen.txt` that is posted on the course Blackboard site. This can be tested using the UNIX command

```
diff out.txt austen.txt
```

The contents of the two files are identical iff this command returns no output.

Your decompression script may not use more than 15 minutes of CPU time in user mode on the Linux cluster, nor may it call external executables or import any modules other than those listed above. You can determine how many CPU-seconds your script spends in user mode using the command


```
% /usr/bin/time --format %U /opt/alttools/bin/python decompress.py
```

We define the length of your compressed version of the text to be the *sum* of the file sizes of `compressed` and the size of your decompressor after it has been compiled to Python bytecode. This compilation can be accomplished in the Python interpreter via the commands

```
>>> import py_compile
>>> py_compile.compile('decompress.py')
```

This will create a file `decompress.pyc` consisting of the compiled bytecode. The length of your compressed version is defined to be the sum of the lengths of this file and the `compressed` file. This *Kolmogorov complexity* approach (cf. Chapter 14 in Cover and Thomas) is necessary because one can embed information about the text in the Python script itself. Of course, one could use the length of the uncompiled Python script instead of the length of the bytecode, but this would encourage overly concise programming that is difficult to read. In the same vein, you are not permitted to use multiple data files, or to call your data file anything other than `compressed`, because the names of the data files or the way that data is divided among them could be used to store some of the text.

Submission

The project is due on Thursday, November 5th at 5 p.m. You are asked to electronically submit your `compressed` and `decompress.py` files and your report via the Blackboard site. The report, which should be in PDF format, should describe your approach in sufficient detail to enable the reader to write a functionally equivalent version of your algorithm without looking at your code. Please also describe any novel features of your solution.

No late submissions are permitted for this project.

Collaboration

You are encouraged to discuss this project with others, including current and former students in the course, to generate ideas. You are also encouraged to consult web resources and the published literature. You can find research papers on text compression through the Engineering Library's website:

`http://engineering.library.cornell.edu/`

In your written submission, however, *you must include a separate "acknowledgment" section in which you list the sources of all ideas that you used in your*

final solution that are not your own. Also, this is an individual project so everyone must submit a distinct solution.

Grading

40% of the overall score will be determined by the compression level of the solution. Solutions that do not correctly reproduce the original text file or that require more than 15 minutes of CPU time will receive none of this credit. 35% will be determined by the quality of the technical approach. 20% will be determined by the quality of the technical writing in the written report and the commenting of the source file. 5% will be given simply for acknowledging others' ideas (if the ideas are entirely your own, simply state this to receive the 5%).

There will be a prize for the student with the whose submission achieves the most compression, assuming there is only one such submission.

Tips

1. There is a sample solution, consisting of the two required files (and the Python script used to generated the `compressed` file), on the course webpage. It does not achieve anywhere near the maximum amount of compression. It is simply intended to get you started.

2. Since you need not submit a program that creates `compressed`, your encoding does not need to be done via a Python script that you write. For this you are welcome to use UNIX tools or programs written in other languages. You should describe how this file was created in your report, however. Also note that the 15-minute limit is on the *decoding* process. Encoding can take as long as you like.

3. The text file `austen.txt` is entirely upper case and has had some of its punctuation removed. This structure can be used for compression purposes.

4. The speed of convergence of the compression performance of universal compression algorithms to the entropy limit governs how well they compress finite-length strings. As we saw in class, algorithms that are universal with respect to a small class of distributions generally achieve faster convergence than those that are universal with respect to a larger class. This suggests tailoring the compression algorithm to the particular structure of the file. One can find general-purpose universal compressors written in Python on the Internet, but these are usually rather long programs that learn the structure of our file very slowly due to their generality. While you are welcome to

encorporate these programs into your solution (as long as you properly acknowledge them and explain in detail in your report how they work), you will probably obtain better results by creating your own scheme that is tailored to this specific text. In past years, many students have obtained solutions that beat general purpose compressors such as `gzip` by a significant margin, even if one does not include the length of the `gzip` executable in the calculation.

5. If you choose to develop your solution on a machine other than the ECE School's Linux cluster or using a different Python interpreter, you should verify periodically that your solution runs correctly on the Linux cluster using the command above.

6. You might find it helpful to debug your code by first compressing a shortened version of `austen.txt`.

Good luck!

