compressor Documentation

Release 0.2.0

Mariano Anaya

May 07, 2019

Contents

Pycompressor is a tool for compressing text files into smaller ones, as well as extracting compressed files back into the original content.

For example, in order to compress one file:

```
$ pycompress -c /usr/share/dict/words -d /tmp/compressed.zf
```

The original file, in this example has a size of ~4.8M, and the tool left the resulting file at /tmp/compressed.zf, with a size of ~2.7M.

In order to extract it:

```
$ pycompress -x /tmp/compressed.zf -d /tmp/original
```

You can specify the name of the resulting file with the -d flag. If you don't indicate a name for the resulting file, the default will be <original-file>.comp.

For the full options, run:

```
$ pycompress -h
```

Contents:

Using the Application

This section explains how the application is used from the command line interface (`cli`), detailing which parameters are accepted and how they work.

1.1 Basic usage

1.1.1 Compressing a File

You can start using the program by just running it, and telling `pycompressor` the name of the file you'd like to compress, for example:

```
$ pycompress -c /usr/share/dict/words
```

The *-c* parameter stands for "compress", and if nothing else is specified, the resulting file will be left on the current directory, with the base name of the provided file and the `.comp` suffix. In this example, the result of will be a file named `words.comp`.

You can change the name of the resulting file, by passing the *-d* (destination) flag, like in:

```
$ pycompress -c /usr/share/dict/words -d /tmp/compressed.zf
```

In this case the resulting file (after compressed) will be /tmp/compressed.zf.

1.1.2 Extracting a file

If you want to recover the original file from a binary, compressed one, use the `-x` (extract) flag:

```
$ pycompress -x /tmp/compressed.zf
```

If a name for the resulting file is not specified, it'll assume the base name provided with the `.extr` suffix, in the local path of where the command is being applied. In this case, it would be `compressed.zf.extr`.

You can also indicate the name of the destination file, again with the `-d` parameter:

```
$ pycompress -x /tmp/compressed.zf -d /tmp/original
```

The destination file in this case, indicates that after extracted the file is written in `/tmp/original`.

compressor

2.1 compressor package

2.1.1 Submodules

2.1.2 Module contents

compressor entry point

2.1.3 lib module

compressor.lib

High-level functions exposed as a library, that can be imported.

`compressor.lib.`**`compress_file`**(*filename: str*, *dest_file: str = ''*) → None
> Open the <filename> and compress its contents on a new one.

>> **Parameters**

>>> - **`filename`** (`str`) – The path to the source file to compress.
>>> - **`dest_file`** (`str`) – The name of the target file. If not provided (None), a default will be used with *<filename>.comp*

2.1.4 cli module

Compressor CLI (command-line interface) module. Exposes the entry point to the program for executing as command line.

`compressor.cli.`**`argument_parser`**() → argparse.ArgumentParser
> Create the argument parser object to be used for parsing the arguments from sys.argv

compressor.cli.**main**() → int
    Program cli

> **Returns** Status code of the program.

> **Return type** int

compressor.cli.**main_engine**(*filename: str*, *extract: bool = False*, *compress: bool = True*, *dest_file=None*) → int
    Main functionality for the program cli or call as library. *extract* & *compress* must have opposite values.

> **Return type** int

> **Parameters**

>> - **filename** (str) – Path to the source file to process.

>> - **extract** (bool) – If True, sets the program for a extraction.

>> - **compress** (bool) – If True, the program should compress a file.

>> - **dest_file** – Optional name of the target file.

> **Returns** 0 if executed without problems.

compressor.cli.**parse_arguments**(*args=None*) → dict
    Parse the command-line (cli) provided arguments, and return a mapping of the options selected by the user with their values.

> **Returns** dict with the kwargs provided in cli

2.1.5 compressor.core module

compressor.core

Low-level functionality with the core of the process that the main program makes use of.

It contains auxiliary functions.

class compressor.core.**CharNode**(*value*, *freq*, *left=None*, *right=None*)
    Bases: object

Object that wraps/encapsulates the definition of a character in the text being processed. Used for comparison, and helper with its properties & methods.

leaf
    Checks if the current node is a leaf in the tree. It is a leaf when it does not have any children (neither left nor right).

> **Returns** True if this node has no children, False otherwise.

value
    Expose the value being hold as read-only.

compressor.core.**compress_and_save_content**(*input_filename: str*, *output_file: io*, *table: dict*) → None
    Opens and processes <input_filename>. Iterates over the file and writes the contents on output_file.

> **Parameters**

>> - **input_filename** (str) – the source to be compressed

>> - **output_file** (io) – opened file where to write the outcome

>> - **table** (dict) – mapping table for the char encoding

compressor.core.**create_tree_code**(*charset: List[compressor.core.CharNode]*) → compressor.core.CharNode
  Receives a :list: of :CharNode: (characters) charset, namely leaves in the tree, and returns a tree with the corresponding prefix-free code.

  > **Return type** *[CharNode](#)*

  > **Parameters** **charset** – iterable with all the characters to process.

  > **Returns** iterable with a tree of the prefix-free code for the charset.

compressor.core.**decode_file_content**(*compfile: io*, *table: dict*, *checksum: int*) → str
  Reconstruct the remaining part of the <compfile>, starting right after the metadata, decoding each bit according to the <table>.

compressor.core.**parse_tree_code**(*tree: compressor.core.CharNode*, *table: dict = None*, *code: bytes = b''*) → dict
  Given the tree with the chars-frequency processed, return a table that maps each character with its binary representation on the new code:

  > left –> 0

  > right –> 1

  > **Return type** dict

  > **Parameters**

  > > • **tree** (*[CharNode](#)*) – iterable with the tree as returned by *create_tree_code*
  > >
  > > • **table** (dict) – Map with the translation for the characters to its code in the new system (prefix-free).
  > >
  > > • **code** (bytes) – The code prefix so far.

  > **Returns** Mapping with with the original char to its new code.

compressor.core.**process_frequencies**(*stream: Sequence[str]*)   → List[compressor.core.CharNode]
  Given a stream of text, return a list of CharNode with the frequencies for each character.

  > **Parameters** **stream** – sequence with all the characters.

compressor.core.**process_line_compression**(*buffer_line: str*, *output_file: io*, *table: dict*) → None
  Transform *buffer_line* into the new code, per-byte, based on *table* and save the new byte-stream into *output_file*.

  > **Parameters**

  > > • **buffer_line** (str) – a chunk of the text to process.
  > >
  > > • **output_file** (io) – The opened file where to write the result.
  > >
  > > • **table** (dict) – Translation table for the characters in *buffer_line*.

compressor.core.**retrieve_compressed_file**(*filename: str*, *dest_file: str = ''*) → None
  EXTRACT - Reconstruct the original file from the compressed copy. Write the output in the indicated *dest_file*.

compressor.core.**retrieve_table**(*dest_file: io*) → dict
  Read the binary file, and return the translation table as a reversed dictionary.

compressor.core.**save_compressed_file**(*filename: str*, *table: dict*, *checksum: int*, *dest_file: str = ''*) → None
  Given the original file by its *filename*, save a new one. *table* contains the new codes for each character on *filename*.

compressor.core.**save_table**(*dest_file: io*, *table: dict*) $\rightarrow$ None

> **Store the table in the destination file.** c: char L: code of c (unsigned Long)
>
> > **Parameters**
> >
> > - **dest_file** (io) – opened file where to write the *table*.
> >
> > - **table** (dict) – Mapping table with the chars and their codes.

2.1.6 functions module

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index