Programming Assignment Checklist: Burrows-Wheeler Data Compression

Frequently Asked Questions

What program should I use for reading and writing the data? You must use `BinaryStdIn.java` and `BinaryStdOut.java`. These read and write sequences of bytes, whereas `StdIn.java` and `StdOut.java` read and write sequences of Unicode characters. These are in stdlib.jar.

My programs work from the command line but the autograder complains that that it can't read in the input. Any ideas? The autograder calls `main()` more than once. So, be sure that any static variables get reset appropriately (or, better yet, don't use static variables).

My programs don't work properly with binary data. Why not? Be absolutely sure that you use only `BinaryStdIn.java` and `BinaryStdOut.java` for input and output. Also, be sure that you call `BinaryStdOut.flush()` or `BinaryStdOut.close()` after you are done writing—see `Huffman.expand()` for an example.

Why does `BinaryStdIn` return the 8-bits as a (16-bit unsigned) `char` instead of as an (8-bit signed) `byte`? The primitive type `byte` is annoying to use in Java. When you operate on a `byte`, it is typically promoted to an `int` and you must be careful because the byte is signed. For example, to convert a `byte b` to a `char c`, you must write `c = (char) (b & 0xff)` instead of `c = (char) b`. By using `char`, we avoid the hassle.

For the Burrows-Wheeler encoder, in which order do I sort the suffixes? The input is a sequence of extended ASCII characters (00 to FF), which you can read in with `BinaryStdIn.readString()`. You should sort the suffixes according to extended ASCII order, which is the natural order of the `String` data type.

For the Burrows-Wheeler decoder, does `next[0]` always equal first? And wouldn't this mean that the index `first` is redundant? No, this is just a coincidence with the input string `"ABRACADABRA!"`. Consider any two input strings that are cyclic rotations of one another, e.g., `"ABRACADABRA!"` and `"CADABRA!ABRA"`. They will have the same sorted suffixes and `t[]` array—their only difference will be in the index `first`.

Can I assume that the inverseTransform() method in BurrowsWheeler receives only valid inputs (that were created by a call to the transform() method)? Yes.

How can I view the contents of a binary file and determine its size? Use HexDump.java, as in the assignment. The command-line argument specifies the number of bytes per line to print; if the argument is 0, all output except for the number of bits will be suppressed.

How much memory can my program consume? The Burrows-Wheeler encoder may use quite a bit, so you may need to use the `-Xmx` option when executing. You must use space linear in the input size n and alphabet size R. (Industrial strength Burrows-Wheeler compression algorithms typically use a fixed block size, and encode the message in these smaller chunks. This reduces the memory requirements, at the expense of some loss in compression ratio.) Therefore, depending on your operating system and configuration there may be some very large files for which your program will not have enough memory even with the `-Xmx` option.

I'm running out of memory in the Burrows-Wheeler encoder. Any ideas? Watch out for creating new strings via `substring()` or string concatenation. Remember, that, as of Java 7u6, substring extraction no longer takes constant time or space!

What is meant by "typical English text inputs"? Inputs such as `aesop.txt`, `moby.txt`, or your most recent essay. We do not mean files such as `aesop3copies.txt` or random sequences of characters.

How do I use gzip and bzip2 on Windows? It's fine to use pkzip or 7-zip instead.

I'm curious. What compression algorithm is used in PKZIP? In gzip? In bzip2? PKZIP uses LZW compression followed by Shannon-Fano (an entropy encoder similar to Huffman). The Unix utility gzip uses a variation of LZ77 (similar to LZW) followed by Huffman coding. The program bzip2 combines the Burrows-Wheeler transform, Huffman coding, and a (fancier) move-to-front style rule.

Testing

Input. Here are some sample input files. For convenience, textfiles-testing.zip contains all of these files bundled together. To fully test your program, you should also try to compress and uncompress binary files (e.g., .class or .jpg files). Be careful to download them as binary files—some browsers will corrupt them if you view the file and use File -> Save. Do not edit them in a text editor—some editors will corrupt them by inserting bogus newline characters.

Reference solutions. For reference, we have provided the output of compressing `aesop.txt` and `us.gif`. We have also provided the results of applying each of the three encoding algorithms in isolation. Note that the .gif file is a binary file and is

already compressed.

To compare the contents of two files, you can use the following commands:

- *Mac OS X or Linux:* `diff file1 file2`
- *Windows:* `fc file1 file2`

Timing your program.

- *Mac OS X or Linux.* Use the following command:

  ```
  % time java BurrowsWheeler - < mobydick.txt | java MoveToFront - | java Huffman - > mobyDickOutputFileName
  ```

  You want to record the "real" value.

- *Windows.* We recommend using a simple batch file. The process for creating and running a batch file is as follows:

  1. Create a file called `timeTest.bat` in the same directory as your Burrows wheeler assignment. This is known as a "batch file".
  2. Inside that file, put the following text (using Notepad or a similar text editor):

     ```
     echo %time%
     java BurrowsWheeler - < mobydick.txt | java MoveToFront - | java Huffman - > mobyDickOutputFileName
     echo %time%
     ```

  3. Execute `timeTest.bat` by navigating a terminal window so that it's in the same directory as `timeTest.bat` and your .java files and type `timeTest`.

  Make sure you name the batch file `timeTest.bat` instead of `timeTest.bat.txt`. Note that you can test multiple files by adding more lines to the batch file.

Timing using gzip, bzip2, 7zip, etc.

- *Mac OS X or Linux.* Use the `time` command as above, but with `gzip` or `bzip2`.
- *Windows.* There is no built-in data compression program (such as gzip or bzip2). We recommend downloading the free 7-zip program. After installing 7-zip, create a new batch file (any filename ending in `.bat`) with the following text:

  ```
  echo %time%
  7za a -tzip mobyDickOutputFileName.zip mobydick.txt
  echo %time%
  ```

  This creates a file in .zip format (the same used natively by Windows for compression). To test unzipping time, use the following:

  ```
  echo %time%
  7za e mobyDickOutputFileName.zip
  echo %time%
  ```

If you're interested in testing against other compression formats, then see this page.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps.

- Download the directory `burrows` to your system. It contains some sample input files and reference solutions. For convenience, burrows-testing.zip contains all of these files bundled together.
- Implement the `CircularSuffixArray`. Be sure not to create explicit copies of the string (e.g., via the `substring()` method in Java's `String` data type) when you sort the suffixes. That would take quadratic space. Instead for each suffix, you only need to keep an index that indicates which character is the beginning of the suffix. This way you can build the N suffixes in linear time and space. Then sort this array of indices. It's just like sorting an array of references.
- Implement the Burrows-Wheeler transform, using the `CircularSuffixArray` class.
- The Burrows-Wheeler decoding is the trickiest part, but it is very little code once you understand how it works. (Not including declarations and input, our solution is about 10 lines of code.) You may find the key-indexed counting algorithm from the string sorting lecture to be useful.

- Implement the move-to-front encoding and decoding algorithms. Not including comments and declarations, our solutions are about 10 lines of code each. If yours is significantly longer, try to simplify it.