T-301-REIR Assignment S5 Burrows-Wheeler: Checklist

Frequently Asked Questions

What program should I use for reading and writing the data? You must use `BinaryIn.java` and `BinaryOut.java`, which are in `algs4.jar`. These read and write sequences of bytes, whereas `In.java` and `Out.java` read and write sequences of Unicode characters.

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

Why does `BinaryIn` 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 transform, 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 `BinaryIn.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 inverse transform, 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 transform 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 `transform()` method in Burrows-Wheeler. Any ideas? Be sure not to create a new `String` object for each circular suffix created in `CircularSuffixArray`, It is OK to have multiple references to the same `String` (for example if you use a `CircularSuffix` nested class).

What is meant by "typical English text inputs"? Inputs such as Aesop's Fables, Moby Dick, or your most recent essay. We do not mean highly degenerate (such as `aesop-2copies.txt` or an input will 1 million consecutive As) or random inputs.

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.

How does this S5 assignment at RU differ from the Princeton assignment? We have modified the specification to avoid standard input/output and use I/O to files instead. This is because constructing Unix pipes is inconvenient in Eclipse. Also, we have supplied testing routines for convenience.

Testing

Input. Here are some sample text files. 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:* `cmp file1 file2`
- *Windows:* `fc file1 file2`

Timing your program. We have supplied one way to run your programs and time them, in the supplied `Tester.java`.

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 (select the 7-Zip Command Line Version). Unzip and put in your working directory (or a directory in the Path). Then, 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 like, you can test other compression formats.

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.
- Implement the `CircularSuffixArray`. Be sure not to create new `String` objects when you sort the suffixes. That would take quadratic space. A natural approach is to define a nested class `CircularSuffix` that represents a circular suffix *implicitly* (via a reference to

the input string and a pointer to the first character in the circular suffix). The constructor of `CircularSuffix` should take constant time and use constant space. You might also consider making `CircularSuffix` implement the `Comparable<CircularSuffix>` interface. Note, that while this is, perhaps, the cleanest solution, it is not the fastest.

- Implement the Burrows-Wheeler transform, using the `CircularSuffixArray` class.
- The Burrows-Wheeler decoding is the trickiest part conceptually, 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 should find the key-indexed counting algorithm from the string sorting lecture to be especially 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. Do not worry about optimizing the worst-case performance—the goal is good performance on typical English text inputs.

A video on compressing and another on expanding are provided for those wishing additional assistance. Be forewarned that video was made in early 2014 and is somewhat out of date. For example the API has changed.

T-301-REIR Reiknirit, fall 2016. School of Computer Science, Reykjavik University.
E-mail address: `mmh@ru.is`