FREQUENTLY ASKED QUESTIONS (GENERAL)

How should I read and write the data? You must use BinaryStdIn and BinaryStdOut, which read and write sequences of bytes. Do *not* use either StdIn or StdOut, which 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 `BinaryStdIn` and `BinaryStdOut` for reading and writing data. Also, be sure to call either `BinaryStdOut.flush()` or `BinaryStdOut.close()` after you are done writing; for an example, see RunLength.java.

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 type `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.

I'm curious. Which 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 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.

FREQUENTLY ASKED QUESTIONS (CIRCULARSUFFIXARRAY)

Must I use `BinaryStdIn` and `BinaryStdOut` with `CircularSuffixArray`? The constructor and methods neither write to standard output nor read from standard input, so there is no need to use either `BinaryStdIn` or `BinaryStdOut`. You are free to use `StdOut.println()` when testing in `main()`.

Can I form the *n* circular suffixes using the `substring()` method from the `String` data type? No. Beginning with Java 7, Update 6, the `substring()` method takes time and space proportional to the length of the substring&mdash. So, explicitly forming the *n* circular suffixes in this way would take both quadratic time and space.

FREQUENTLY ASKED QUESTIONS (MOVETOFRONT)

How should i read the binary input in `encode()`? The input is a sequence of extended ASCII characters (`0x00` to `0xFF`). You should read them in one character at a time using `BinaryStdIn.readChar()` until `BinaryStdIn.isEmpty()`.

How should i read the binary input in `decode()`? Same as `encode()`.

FREQUENTLY ASKED QUESTIONS (BURROWSWHEELER)

How should i read the binary input in `transform()`? The input is a sequence of extended ASCII characters (`0x00` to `0xFF`). You can read it using `BinaryStdIn.readString()`.

How should i read the binary input in `inverseTransform()`? The input is an integer, followed by a sequence of extended ASCII characters (`0x00` to `0xFF`). You can read it using `BinaryStdIn.readInt()`, followed by `BinaryStdIn.readString()`.

Can I assume that `inverseTransform()` receives only valid inputs (e.g., that correspond to the output of `transform()`)? Yes.

For the Burrows–Wheeler transform, in which order do I use to sort the suffixes? Use lexicographic order to sort the suffixes, which is the natural order of the `String` data type.

For the Burrows–Wheeler inverse transform, does `next[0]` always equal `first`? 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`.

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 inputs with very long repeated substrings (such as `aesop-2copies.txt` or an input will 1 million consecutive As) or random inputs.

Testing

Input. To fully test your programs, you should use not only text files but also binary files (such as `.class` or `.jpg` files).

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 binary file `us.gif` is already compressed.

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

```
~/Desktop/burrows> cmp aesop.txt us.gif
aesop.txt us.gif differ: byte 1, line 1

~/Desktop/burrows> cmp us.gif us.copy.gif
```

Compression ratio. You can use the `ls` command to determine the size of a file (in bytes).

```
~/Desktop/burrows> ls -l
total 60536
-rw-rw-r--@  1 wayne  staff       723 Jul 14 10:24 COS 226.iml
-rw-rw-r--@  1 wayne  staff     24567 May  4  2012 CS_bricks.jpg
-rw-rw-r--@  1 wayne  staff         1 May  4  2012 a.txt
-rw-rw-r--@  1 wayne  staff        12 Mar 22  2009 abra.txt
-rw-rw-r--@  1 wayne  staff        19 Mar 22  2009 abra.txt.bwt.mtf.huf
-rw-rw-r--@  1 wayne  staff    191943 Nov 10  2005 aesop.txt
-rw-rw-r--@  1 wayne  staff     66026 Mar 22  2009 aesop.txt.bwt.mtf.huf
...
```

For example, `aesop.txt` uses 191,943 bytes; after compression, it (`aesop.txt.bwt.mtf.huf`) uses only 66,026 bytes; the compression ratio is 66026/191943 = 0.344.

Timing your program. Use the following bash commands for compression and expansion, respectively:

```
~/Desktop/burrows> time java-algs4 BurrowsWheeler - < mobydick.txt | java-algs4 MoveToFront - | java-algs4 edu.princeton.cs.algs4.Huffman - > mobyDick
real    0m1.341s
user    0m1.447s
sys 0m0.477s

~/Desktop/burrows> time java-algs4 edu.princeton.cs.algs4.Huffman + < mobyDickOutputFileName | java-algs4 MoveToFront + | java-algs4 BurrowsWheeler +
real    0m0.419s
user    0m0.750s
sys 0m0.372s
```

The "real" value is the wall clock time; the "user" value is the amount of CPU time spent in user-mode; the "system" value is the amount of CPU time spent in kernel mode. The CPU time may exceed the real time if your computer is using multiple CPUs.

Timing using gzip or bzip2. If you are using bash, you should have access to the the following data compression and expansion commands: `gzip`, `gunzip`, `bzip2`, or `bunzip2`. You can time them using the following bash commands:

```
~/Desktop/burrows> time gzip mobydick.txt
real    0m0.218s
user    0m0.133s
sys 0m0.007s

~/Desktop/burrows> time gunzip mobydick.txt.gz
real    0m0.063s
user    0m0.026s
sys 0m0.008s

~/Desktop/burrows> time bzip2 mobydick.txt
real    0m0.176s
user    0m0.145s
sys 0m0.007s

~/Desktop/burrows> time bunzip2 mobydick.txt.bz2
real    0m0.099s
user    0m0.057s
sys 0m0.008s
```

Possible Progress Steps

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

- Download burrows.zip . It contains 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.