CS337 Project 1 : Compression

TA: Yan Li (yanli@cs.utexas.edu)

Due Date: Sept. 29, 2008 11:59PM

1 Part 1: Lower Bound of Lossless Compression

1.1 Description

Given a text file, which contains only ASCII characters and NO unicode characters, compute the lower bound of bits needed for lossless compression of the file. That is, compute the least number of bits you need to transmit the file without loss of information. This can be computed using Shannon's Theorem. To do this, you need to first obtain the probability of occurrence of each character in the input text file.

NOTE: include whitespaces, commas, and full-stops in your frequency calculation. Also, you can use the method giveArray() in the IO.Compress class to read characters from the file into an array; see the last section below.

1.2 Implementation

Write a program to compute the lower bound of bits you need to transmit a text file without loss of information.

1.3 Deliverables

You MUST code using UNIX/LINUX OS, NO Windows OS! (You can use any public LINUX machines in CS department.) This is because the format of a text file depends on the platform. If you code in Windows, we can not get the correct results when we grade it using LINUX.

You MUST name your program "LowerBound.java". And it MUST be able to be compiled and invoked using command line as follows:

```
 rivera.cs.utexas.edu$$ javac LowerBound.java
rivera.cs.utexas.edu$$ java LowerBound test.txt
```

You should not only output the final lower bound of bits for lossless compression, but also outputthe "Entropy" of the characters in the input file. (This is

important for partial credit.) **The expected output SHOULD be as follows:**

```
The Entropy is 1.234; The lower bound is 61.700
```

2 Part 2: Lempel-Ziv Coding

2.1 Description

The Lempel-Ziv compression algorithm accepts a string of text, which contains only ASCII characters and NO unicode characters, and produces a sequence of pairs (n, c), where n is the index in the dictionary, and c is a character. The decompression algorithm reconstructs the dictionary using the pairs that are retrieved from a compressed file . During decompression each pair in the sequence causes a new entry to be added into the dictionary. For example, when the decompression algorithm processes the pair (n, c), it is known that the index n refers to a string that is already contained in the dictionary. Therefore the new entry to be added into the dictionary is obtained by concatenating the string corresponding to n with the character represented by c. You are required to implement algorithms for compression and decompression using this method.

2.2 Implementation - Compression

Write a program that is invoked with the name of a text file. This program should create a compressed file and write it to the disk. Your implementation should use the "trie" based approach to construct the dictionary (see Implementation of the Dictionary, page 21 of handbook). The name of the compressed file is obtained by concatenating the name of the input file with the string ".myZ".

2.3 Implementation - Decompression

The decompression algorithm should be invokable with the name of a compressed file, and it should write a decompressed file to the disk. The name of the decompressed file is determined by concatenating the name of the input file with the string "."(i.e. a single period).

2.4 Deliverables

You MUST code using UNIX/LINUX OS only, NO Windows OS! (for the same reason explained in PART 1.3) You are required to submit one file called "LZcoding.java" (the file name must be exact the same as required here), which takes two command line arguments, the first argument is either the character 'c'(for compression) or 'd'(for decompression). The second argument is the name of the file to be compressed or decompressed. An example compile and run of your program should look like this:

```
rivera.cs.utexas.edu$$ javac LZcoding.java
rivera.cs.utexas.edu$$ java LZcoding c test
rivera.cs.utexas.edu$$ ls -l test*
-rw-r--r-- 1 ankur grad 5029 Mar 25 22:26 test
-rw-r--r-- 1 ankur grad 5600 Mar 26 11:06 test.myZ
rivera.cs.utexas.edu$$ java LZcoding d test.myZ
rivera.cs.utexas.edu$$ ls -l test*
-rw-r--r-- 1 ankur grad 5029 Mar 25 22:26 test
-rw-r--r-- 1 ankur grad 5600 Mar 26 11:06 test.myZ
-rw-r--r-- 1 ankur grad 5029 Mar 26 11:06 test.myZ.2
rivera.cs.utexas.edu$$ cmp test test.myZ.2
rivera.cs.utexas.edu$$
```

In some cases, you may find that the size of the ".myZ" file is actually larger than the input file. We have observed that it is so for small files. Explain why this happens in your "README" file. Along with the "LZcoding.java" and "README" files, you are also required to submit two text files: one text file, **MUST be named as "smaller"**, for which your program indeed does some compression (i.e. the size of "smaller.myZ" file is smaller than the input file "smaller"); the other text file, **MUST be named as "larger"**, for which your compression algorithm actually increases the size of the input file (i.e. the size of "larger.myZ" file is larger than the input file "larger").

3 Questions and Clarifications

Clarifications regarding this project will be emailed to you through Blackboard. Though changes are not expected, you are responsible for any modifications emailed to you (check the email account you registered with Blackboard).

4 Deadline and Sumission Instructions

All the requirements in this project description document must be strictly followed to avoid points deduction.

The project is due **Sept. 29th 2008 11:59PM**. **No late submissions allowed!**

What you need to include in your "README" file (Failed to do this is subject to point deduction):

```
NAME of partners;
EID of partners;
OS used:  Ubuntu(Strongly prefered!!)  or Solaris, no Windows;
```

```
CODING STATUS: (very important!);
Any other stuff you want to say.
```

IMPORTANT: Summarize your code status in "README" file. If your code works perfectly, explicitly saying this in the README. If you code does not work (can not compile, or can not run, or can not output anything, etc.), you need to specifically explain this: what have been done so far and what have not been done, what is working and what is not working to avoid a very low score!! Put proper comments in all the code files.

LIST OF FILES YOU MUST SUBMIT:
(THE NAMES OF THE FILES SHOULD BE EXACTLY THE SAME AS FOLLOWS AND CASE-SENSITIVE):

```
README (explain your code status here!  and other stuff)
LowerBound.java
LZcoding.java
smaller
larger
```

You must submit your files (5 as the list above) within a SINGLE(and not recursive) directory that is the **CS username** of the partner submitting the files, followed by "-p1", like so: `username-p1`. (i.e., put all the 5 files under the directory `username-p1`.)

To submit your project, use "turn-in" program.

```
rivera.cs.utexas.edu$$ turnin --submit yanli project1 username-p1
```

where `username` is the user name of the partner submitting the files.

To confirm your submission:

```
rivera.cs.utexas.edu$$ turnin --list yanli project1
```

ONLY ONE submission per group!

Final Remark: Although you are not required to implement the Java code for file IO.java, you may develop your own interface for file IO if you wish to do so, provided it conforms with all the specifications described in this document.

5 Appendix - IO source code

We will provide a file named IO.java on the class web page. This file implements three java classes IO.pair, IO.Compress and IO.Decompress, which can be used without modifications to complete this assignment. The class IO.Compress can be used for implementing both the parts of this assignment. The class IO.Decompress can be used for implementing the decompression algorithm.

5.1 IO.pair

This class is used for storing the tuple (index,character). This is the type that will be returned by the method decode(...) in the class IO.Decompress. If p is an object of the type IO.pair, then p.index refers to the index of an entry in the dictionary, and p.extension refers to the character that must be appended to this string to create a new dictionary entry. The pair class also contains a field named 'valid'. This is helpful to determine when there are no more pairs to be decoded.

5.2 IO.Compress

The constructor of this class does two tasks. First, it initializes its internal objects so that they are ready to read from the input file. Second, it determines the name of the output file, and initializes it, so that the next call to encode(...) writes to the output file. This class provides a method giveArray() for copying all the characters from a file to a character array. It also provides a method encode(int x, char a), where the parameter x refers to an index in the dictionary, and a is the character that must be appended to the corresponding string. This method should be called during compression when a new pair is added to the sequence. The method done() should be called at the end of the compression process. Note that the method encode will be called several times during compression. The following code shows some parts of an example compression program.

```
/*infile is the name of the file to be compressed*/
public static void compress(String infile) throws Exception
  {
    IO.Compressor io ;
    /* Initialize a IO.Compressor object so that it is ready to
    * read the input file, and to write the output file to the disk */
    io = new IO.Compressor(infile);
    /* Read all characters from the input file to a character array */
    char[] carray = io.giveArray();
    ...
    /* Perform compression on the array carray, *
    * this part may call io.encode(...)  several times */
    ...
```

```
    /* Close all relevant files */
    io.done();
  }
```

5.3 IO.Decompress

The constructor of this class does two tasks. First, it initializes its internal objects so that they are ready to read from the input file (i.e. compressed file). Second, it determines the name of the output file, and initializes it, so that the next call to io.append(...) writes to the output file. This class provides a method decode() which returns the next unprocessed pair from the sequence contained in the compressed file (if no more pairs are available, then it returns a pair whose 'valid' field is set to false). This method will also be called several times during the decompression process (like the method encode in IO.Compress). It also provides a method append(String s), which appends the string s to the output generated so far. The method done should be called at the end of the decompression process. The following code shows some parts of an example decompression program.

```
/*infile is the name if the file to be decompressed*/
decompress(String infile) throws Exception
  {
    /* Initialize a IO.Decompressor object so that it is ready *
    * to read the sequence of pairs from infile, and to write *
    * the decompressed file to the disk */
    IO.Decompressor io = new IO.Decompressor(infile);
    while(...)
    {
      ....
      /* Every call to io.decode() returns the next unprocessed pair from the *
      * compressed file.  next.valid is false if no more pairs are left.  */
      IO.pair next = io.decode();
      ....
      /* output is the latest entry that was added to the *
      * dictionary, append it to the decompressed file */
      io.append(output);
      ....
    }
    /* Close all relevant files */
    io.done();
  }
```