CSC 316 – Data Structures

Programming Project #1 – Move-to-Front Lists and Data Compression

Due Date: June 7, 2017, 11:59pm

Project Objectives

Your first programming task is to write a program to compress and decompress text files.

In normal English text – or in Java programs – some words appear much more often than others. For effective compression, we would like to represent frequent words by small numbers. The move-to-front method causes frequent words to be near the front of the list most of the time, therefore these words are likely to be represented by small numbers.

Move-to-front also exploits "locality of reference." In a long English text, the set of frequently occurring words shifts as the author moves from topic to topic, or, in a document with coauthors, as authorship passes from person to person. In a Java program, variable names are used frequently in the method in which they are declared, but not at all outside it. Furthermore, uses tend to clump together even within methods. The move-to-front technique adapts well to these changes in relative frequency. Words gradually move further back in the list as they fall into disuse.

The objectives of this project are to:

1. implement an algorithm to compress and decompress text files using linked lists with the move-to-front heuristic; and
2. collect experimental data to evaluate the performance of the compression algogorithm and compare it to well-known file compression utilities.

Description of the Compression Algorithm

For simplicity, we restrict the input files as follows:

- The file may occupy many lines.
- The lines consist of "words" made up of upper- and lower-case letters separated by blanks and special characters such "`.,/%&!@#$?-_><`". Special characters are not to be compressed, they are simply copied from input file to output file.
- The file contains **no digits** 0123456789.

The output (compressed) file is identical to the input (uncompressed) one, except as follows:

- A zero and a blank ("0␣") are prepended to the first line
- The *first* occurrence of each word remains as is, but all subsequent occurrences are replaced by positive integers, as specified further on.
- After the last line of text, a new line beginning with "0␣" is added. The rest of this line is a comment containing statistics on the compression (as shown in the example below).

Input and Output
For example, the text

```
On␣the␣future!␣␣How␣it␣tells
of␣the␣rapture␣that␣impels
to␣the␣swinging␣and␣the␣ringing␣of␣the␣bells,␣bells,␣bells,
of␣the␣bells,␣bells,␣bells,␣bells,␣bells,␣bells,␣bells,
to␣the␣rhyming␣and␣the␣chiming␣of␣the␣bells.
␣␣␣␣-␣Edgar␣Allan␣Poe
```

becomes

```
0␣On␣the␣future!␣␣How␣it␣tells
of␣6␣rapture␣that␣impels
to␣5␣swinging␣and␣3␣ringing␣9␣3␣bells,␣1,␣1,
3␣3␣3,␣1,␣1,␣1,␣1,␣1,␣1,
7␣3␣rhyming␣7␣3␣chiming␣7␣3␣7.
␣␣␣␣-␣Edgar␣Allan␣Poe
0␣Uncompressed:␣233␣bytes;␣␣Compressed:␣171␣bytes
```

after compression. Conversely, decompressing the second text should produce the original text above.

You should write a **single** program to do both compression and decompression. By examining the first character in the input file, the program can determine whether it is a plain file to be compressed, or a compressed file to be decompressed.

Where Did Those Numbers Come From?
In both modes, the program maintains a linked list of words that have appeared in the input. In compression mode, the program reads the next word from the input file and searches for it in the list. If it is not found, the program writes the word to the output file and inserts it at the front of the list. If the word is found in the i-th position in the list, the program writes out i, removes the word from the list, then reinserts it at the front (this is the move-to-front feature). For example, let us consider the text:

```
LOVE,␣LOVE␣ME␣DO!␣␣YOU␣KNOW␣I␣LOVE␣YOU,␣YOU,␣YOU.
```

which after compression becomes

```
LOVE,␣1␣ME␣DO!␣␣YOU␣KNOW␣I␣6␣4,␣1,␣1.
```

Notice that `LOVE` is replaced by both 1 and 6, and that 1 replaces both `LOVE` and `YOU`. This is due to word insertions in the list and the effect of move-to-front. After the second `LOVE` is processed, the word list is simply

```
(LOVE)
```

hence the second instance of `LOVE` is replaced by the number 1. After the `I` is processed, the word list is

```
(I, KNOW, YOU, DO, ME, LOVE)
```

Next, the third instance of `LOVE` is processed, the word is found in the 6th position (thus it is replaced by number 6), and the list becomes:

```
(LOVE, I, KNOW, YOU, DO, ME)
```

When the second `YOU` is processed, `YOU` is found in the 4th position, and the list becomes:

```
(YOU, LOVE, I, KNOW, DO, ME)
```

Finally, when the last two instances of `YOU` are processed, the word is found in the first position.

In decompression mode, a word read from input is written out again and inserted at the front of the list. When the number i is read, the word in the i-th position is written out, then the word is deleted from the list and reinserted at the front. Decompression ends when the number 0 is read. The decompressed version should be *identical* to the original in every respect! (In Unix, you can use the utility `diff` to check this.)

In compression mode, the program should also keep track of how many characters it reads and writes (between the two zeros) for the statistics line.

Deliverables: Source Code
You will implement the compression and decompression algorithms in a single Java program. You must name your file `proj1.java`, and have your program read from standard input (*System.in*) and write to standard output (*System.out*); you (and we) can use redirection to read from or write to specific files.

We will test your code on a secret file (i.e., we will not provide the test file to you in advance). In order for you to test and debug your program, I have placed three sample input files, *small*, *medium*, and *large*, in the Project 1 folder:

`http://courses.ncsu.edu/csc316/lec/001/wrap/Project1/`

Deliverables: Comparison Report
The objective of this part of the project is to compare the performance of our simple data compression algorithm to another popular file compression utility. To this end, you need to select a *reference* file compression utility available on your operating system (for instance, Unix or Mac

OS X users have a choice among `compress`, `gzip`, and `pack`; Windows users may consider `winzip`, `pkzip`, or other). Run both the reference utility and your program on the ten files in the Project 1 folder named `input1`, `input2`, $\cdots$, `input10`. Submit a report of the results that contains a table listing, for each input file: (1) the size of the uncompressed file, in bytes, (2) the size of the file compressed by the reference utility, in bytes and as a fraction of the uncompressed file size, and (3) the size of the file compressed by your program, in bytes and as a fraction of the uncompressed file size (for the comparison, do not include the statistics line generated by your program). Also, report the average over all 10 input files.

Submission

Submit all your programs, as well as your report (in PDF), using the **submit** tools, by 11:59pm on the due date indicated in the schedule of lectures.

Grading

Move-to-front linked list implementation:	30 Points
Data compression algorithm:	30 Points
Compilation and output:	30 Points
Comparison report	10 Points
	100 Points

Important reminder: Homework is an individual, not a group, project. Students may discuss approaches to problems together, but each student should write up and fully understand his/her own solution. Students may be asked to explain solutions orally if necessary.