# File Compressor/Decompressor Using Lempel-Ziv-Welch Algorithm

4 Junit test cases were developed for the project 5:

* TestLZWCompression
* TestLZWCompressionWithTreeMap
* TestBitManipulation
* TestQueue

Of those, you may want to run TestLZWCompression and TestLZWCompressionWithTreeMap to test compression and decompression against the test files. Below, is a quick comparison of the 2 types of compressors’ performance:

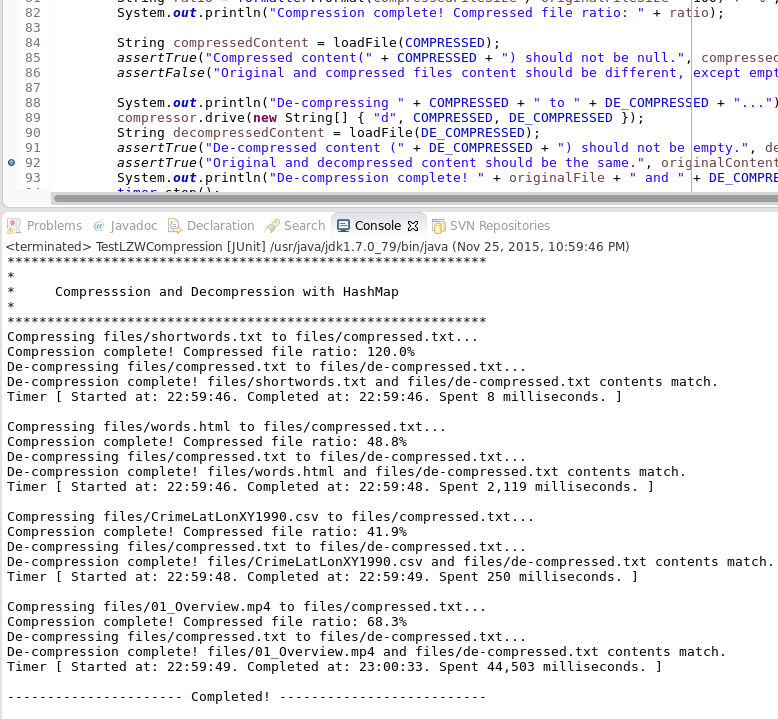
|  |  |  |
| --- | --- | --- |
| Test File | TestLZWCompression  using HashMap | TestLZWCompressionWithTreeMap  using TreeMap |
| shortwords.txt (1 KB) | 8 ms | 39 ms |
| words.html (2,436 KB) | 2,119 ms | 4,553 ms |
| CrimeLatLonXY1990.csv (270 KB) | 250 ms | 451 ms |
| 01\_Overview.mp4 (24,423 KB) | 44,503 ms | 76,783 ms |

### Host System Information

* Memory: 16 GB
* CPU: Intel® Core™ i5-4300M CPU @ 2.60GHz × 4
* OS: Fedora 23 (64 bit)

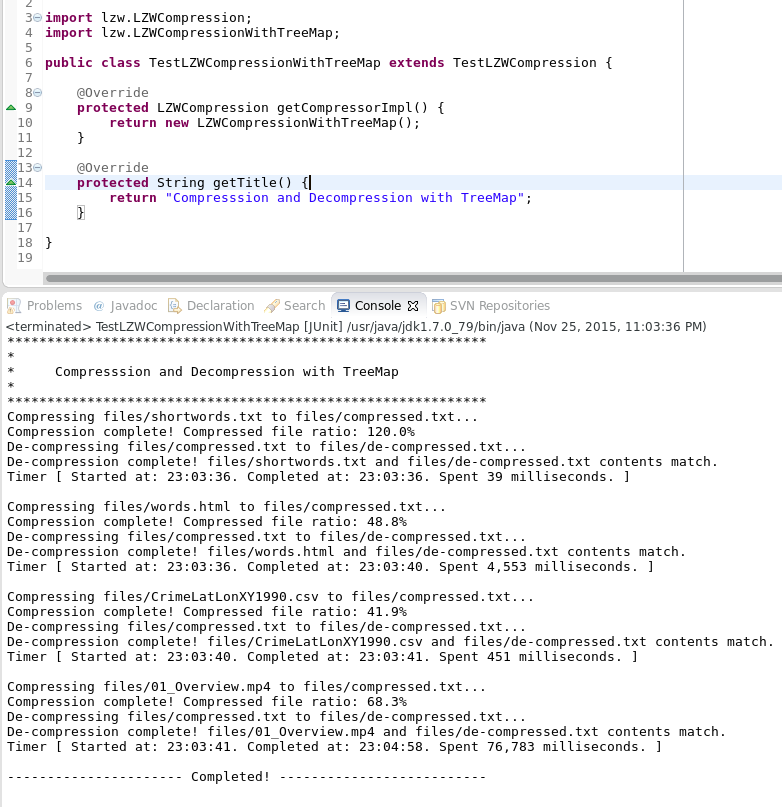
## Running TestLZWCompression

HashMap is used for the compression/decompression activities.



## Running TestLZWCompressionWithTreeMap

TreeMap is used for the compression/decompression activities. This is considerably slower than the compressor that uses HashMap.



## TestLZWCompression Impementation Details

**package** test;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** junit.framework.TestCase;

**import** lzw.LZWCompression;

**import** lzw.Timer;

/\*\*

\* Test compressor/decompressor tool that uses Lempel-Ziv-Welch algorithm.<br>

\* Test compression/decompression methods against ASCII and Binary files. <br>

\* **@author** bsanchin

\*/

**public** **class** TestLZWCompression **extends** TestCase {

**private** **final** String COMPRESSED = "files/compressed.txt";

**private** **final** String DE\_COMPRESSED = "files/de-compressed.txt";

**private** **final** Timer timer = **new** Timer();

**private** **final** NumberFormat formatter = **new** DecimalFormat("#0.0");

@Override

**protected** **void** setUp() **throws** Exception {

// Remove compressed and decompressed files from previous run.

(**new** File(COMPRESSED)).delete();

(**new** File(DE\_COMPRESSED)).delete();

}

**protected** **final** String files[] = (

"shortwords.txt" // short ascii file

+ ",words.html" // long ascii file

+ ",CrimeLatLonXY1990.csv" // long ascii file

+ ",01\_Overview.mp4" // long binary file

).split(",");

/\*\*

\* Test suite may vary by map usage.

\*/

**protected** LZWCompression getCompressorImpl() {

**return** **new** LZWCompression();

}

/\*\*

\* Returns banner for the test suite.

\*/

**protected** String getTitle() {

**return** "Compresssion and Decompression with HashMap";

}

/\*\*

\* Runs Compression/De-compression test for each specified file.

\*/

**public** **void** testCompressionAndDecompression() **throws** IOException {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("\*");

System.***out***.println("\* " + getTitle());

System.***out***.println("\*");

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

**for** (String file : files) {

testCompression("files/" + file);

}

System.***out***.println("---------------------- Completed! --------------------------");

}

/\*\*

\* Compression/De-compression test for a single file.

\*/

**private** **void** testCompression(String originalFile) **throws** IOException {

timer.start();

String originalContent = loadFile(originalFile);

*assertTrue*(originalFile + " should not be empty.", originalContent != **null**);

LZWCompression compressor = getCompressorImpl();

System.***out***.println("Compressing " + originalFile + " to " + COMPRESSED + "...");

compressor.drive(**new** String[] { "c", originalFile, COMPRESSED });

**double** originalFileSize = **new** File(originalFile).length();

**double** compressedFileSize = **new** File(COMPRESSED).length();

String ratio = formatter.format(compressedFileSize / originalFileSize \* 100) + "%";

System.***out***.println("Compression complete! Compressed file ratio: " + ratio);

String compressedContent = loadFile(COMPRESSED);

*assertTrue*("Compressed content(" + COMPRESSED + ") should not be null.", compressedContent != **null**);

*assertFalse*("Original and compressed files content should be different, except empty files.", originalContent.equals(compressedContent));

System.***out***.println("De-compressing " + COMPRESSED + " to " + DE\_COMPRESSED + "...");

compressor.drive(**new** String[] { "d", COMPRESSED, DE\_COMPRESSED });

String decompressedContent = loadFile(DE\_COMPRESSED);

*assertTrue*("De-compressed content (" + DE\_COMPRESSED + ") should not be empty.", decompressedContent != **null**);

*assertTrue*("Original and decompressed content should be the same.", originalContent.equals(decompressedContent));

System.***out***.println("De-compression complete! " + originalFile + " and " + DE\_COMPRESSED + " contents match.");

timer.stop();

System.***out***.println(timer.getStatus());

System.***out***.println();

}

/\*\*

\* Loads given file content as one giant String object.

\*/

**private** String loadFile(String string) **throws** IOException {

String line;

StringBuilder result = **new** StringBuilder();

BufferedReader reader = **new** BufferedReader(**new** FileReader(string));

**while** ((line = reader.readLine()) != **null**) {

result.append(line.trim()).append("\n");

}

reader.close();

**return** result.toString().trim();

}

}

## TestLZWCompressionWithTreeMap Impementation Details

**package** test;

**import** lzw.LZWCompression;

**import** lzw.LZWCompressionWithTreeMap;

**public** **class** TestLZWCompressionWithTreeMap **extends** TestLZWCompression {

@Override

**protected** LZWCompression getCompressorImpl() {

**return** **new** LZWCompressionWithTreeMap();

}

@Override

**protected** String getTitle() {

**return** "Compresssion and Decompression with TreeMap";

}

}