Sarah Depew

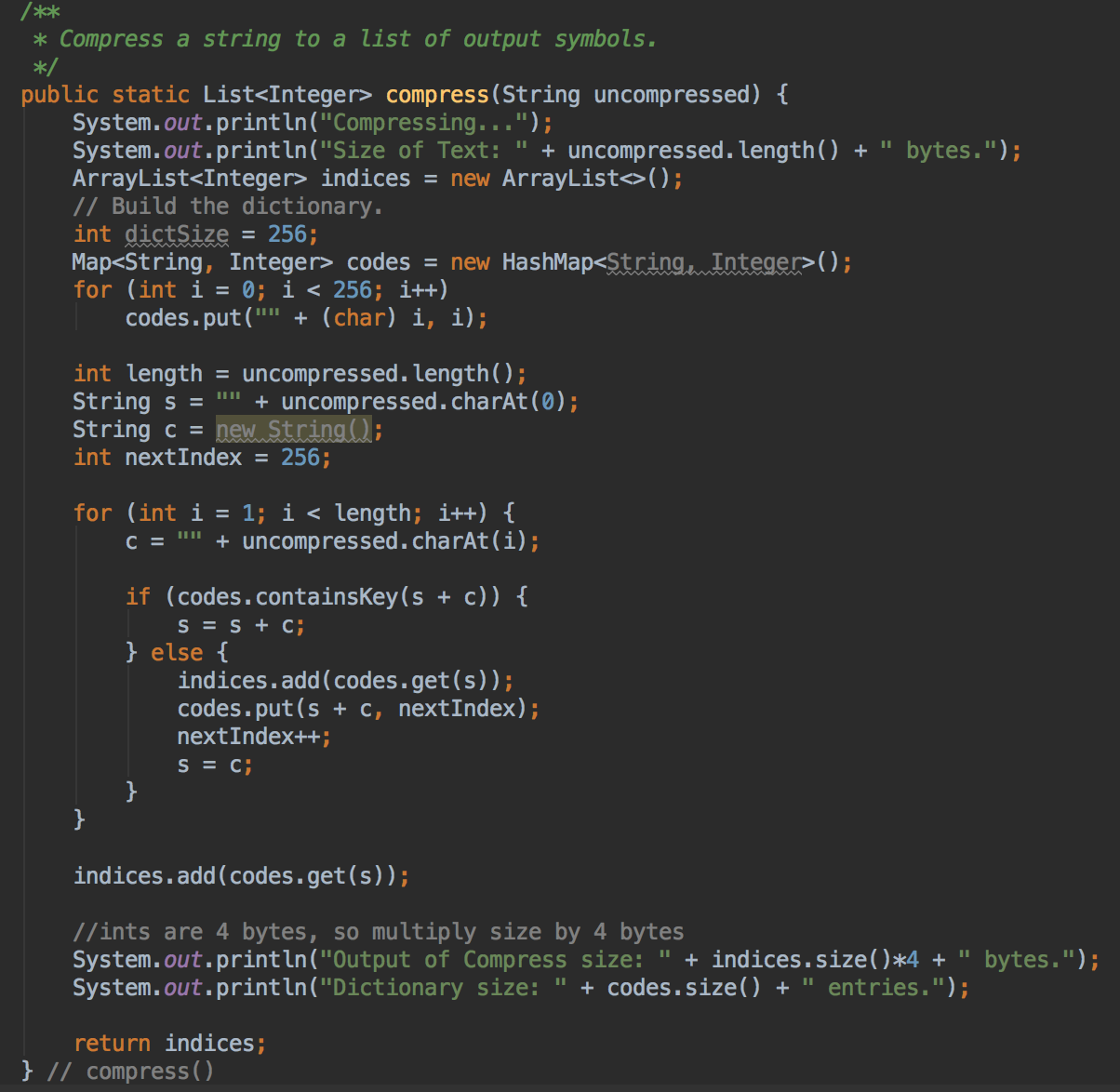
Professor Kumar

CS330 Algorithms: Design & Practice

26 April 2018

**Lab 13: Data Compression - LZW Algorithm**

**Discussion:**

**** In this lab, I implemented the Lempel-Ziv-Welch (LZW) algorithm for data compression and decompression. The decompression method was provided for us, but I wrote the compression method (Figure 1). In this method, I followed the LZW algorithm given on the first page of the lab. I used a HashMap<String, Integer> called codes to record the formed string to integer codes and used a ArrayList<Integer> called indices to return the LZW encoding to the user. I also printed this to the terminal for small enough files. Additional print methods confirmed the correctness of the encoding and decoding process.

Looking at the results table, below (Figure 2), the correctness of the output can be verified by columns 1 and 4, and 3 and 5, respectively. The size of the input file should be the same as the output decompressed file, which is what is illustrated by columns 2 and 4. Too, the size of the dictionary used to encode and decode should be the same, as shown by columns 3 and 5. Since these values match, I can confidently state that my encoding was correct. Too, I confirmed the correctness by printing out the decoded files, which were all legible and correct.

Figure : Compression Method

**Results Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Size of Text (in bytes) | Output of Compress (in bytes) | #Entries in Dictionary (compress) | Output of Decompress (in bytes) | #Entries in Dictionary (Decompress) |
| Sample Text (Page 1) | **52** | **156** | **294** | **52** | **294** |
| Anassa Kata (Input from file) | **223** | **608** | **407** | **223** | **407** |
| Moby Dick (Input from file) | **1256165** | **955172** | **239048** | **1256165** | **239048** |

Figure : Results Table

**Code (Java)**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.\*;

public class Lab13 {

/\*\*

\* Compress a string to a list of output symbols.

\*/

public static List<Integer> compress(String uncompressed) {

System.out.println("Compressing...");

System.out.println("Size of Text: " + uncompressed.length() + " bytes.");

ArrayList<Integer> indices = new ArrayList<>();

// Build the dictionary.

int dictSize = 256;

Map<String, Integer> codes = new HashMap<String, Integer>();

for (int i = 0; i < 256; i++)

codes.put("" + (char) i, i);

int length = uncompressed.length();

String s = "" + uncompressed.charAt(0);

String c = new String();

int nextIndex = 256;

for (int i = 1; i < length; i++) {

c = "" + uncompressed.charAt(i);

if (codes.containsKey(s + c)) {

s = s + c;

} else {

indices.add(codes.get(s));

codes.put(s + c, nextIndex);

nextIndex++;

s = c;

}

}

indices.add(codes.get(s));

//ints are 4 bytes, so multiply size by 4 bytes

System.out.println("Output of Compress size: " + indices.size()\*4 + " bytes.");

System.out.println("Dictionary size: " + codes.size() + " entries.");

return indices;

} // compress()

/\*\*

\* Decompress a list of output indices from LZW compress() to a string.

\*/

public static String decompress(List<Integer> indices) {

System.out.println("Decompressing...");

// Build the dictionary.

int dictSize = 256;

Map<Integer, String> codes = new HashMap<Integer, String>();

for (int i = 0; i < 256; i++)

codes.put(i, "" + (char) i);

//String previous = "" + (char)(int)indices.remove(0);

StringBuffer result = new StringBuffer();

String previous = codes.get(indices.remove(0));

result.append(previous);

for (int current : indices) {

String s;

if (codes.containsKey(current))

s = codes.get(current);

else if (current == dictSize)

s = previous + previous.charAt(0);

else

throw new IllegalArgumentException("Bad compressed current: " + current);

result.append(s);

// Add w+entry[0] to the dictionary.

codes.put(dictSize++, previous + s.charAt(0));

previous = s;

}

System.out.println("Output of Decompress: " + result.toString().length() + " bytes.");

System.out.println("Dictionary size: " + codes.size() + " entries.");

return result.toString();

} // decompress()

public static void main(String[] args) {

// List<Integer> compressed = compress("It was the best of times, it was the worst of times.");

// System.out.println(compressed);

// String decompressed = decompress(compressed);

// System.out.println(decompressed);

try {

// String lines = new Scanner(new File("anassa.txt")).useDelimiter("\\Z").next();

String lines = new Scanner(new File("moby10b.txt")).useDelimiter("\\Z").next();

List<Integer> compressed = compress(lines);

// System.out.println(compressed);

String decompressed = decompress(compressed);

// System.out.println(decompressed);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

} // main()

} // class Lab13