Lempel-Ziv Decoding

How to decode while building the dictionary?

Example: 65 66 66 256 257 259 65

Recall that dictionary contains codes up to 255.

Ideas?
- Read a code (65)
- Look it up (first code will always decode to a single character so will already be in dictionary)
- Output decoded (A)
- Build next code: concatenation of previous code + first char of next code: (A + first char of 66) = A + B = AB
- Add next code to dictionary (Add 256, AB)
ex: 65 66 66 256 257 259 65

LZ Decode(msg, dictionary)

firstcode = msg.getNextCode()
olddecoded = dictionary.translate(firstcode) // will always be // in dictionary

3. c = olddecoded
output olddecoded

while (there are still codes to read) {
nextcode = msg.getNextCode()

1. if dictionary.contains(nextcode)
   nextdecoded = dictionary.translate(nextcode)

2. else
   nextdecoded = olddecoded + c

output nextdecoded
// new string = previous decoded + first char in // new decoded

\( c = \) first char in nextdecoded
dictionary.add(olddecoded + c)

olddecoded = nextdecoded // shift window
### Dictionary

<table>
<thead>
<tr>
<th>firstcode</th>
<th>olddecoded</th>
<th>nextcode</th>
<th>nextdecoded</th>
<th>C</th>
<th>String</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>A</td>
<td>66</td>
<td>B</td>
<td>B</td>
<td>AB</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>66</td>
<td>B</td>
<td>B</td>
<td>AB</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>256</td>
<td>AB</td>
<td>A</td>
<td>BB</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>AB</td>
<td>257</td>
<td>BB</td>
<td>B</td>
<td>BA</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>259</td>
<td>ABB</td>
<td>A</td>
<td>ABB</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>65</td>
<td>A</td>
<td>A</td>
<td>BBA</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td>ABBBA</td>
<td>261</td>
</tr>
</tbody>
</table>

**ex 1**

| 65 | 66 | 66 | 256 | 257 | 259 | 65 |

**output**: A B B AB BB ABB A

(Verify that this is original string)

**ex 2**

<table>
<thead>
<tr>
<th>firstcode</th>
<th>olddecoded</th>
<th>nextcode</th>
<th>nextdecoded</th>
<th>C</th>
<th>String</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>B</td>
<td>65</td>
<td>A</td>
<td>A</td>
<td>BA</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>256</td>
<td>BA</td>
<td>B</td>
<td>AB</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>BA</td>
<td>257</td>
<td>AB</td>
<td>A</td>
<td>BAA</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td>AB</td>
<td>65</td>
<td>A</td>
<td>A</td>
<td>ABA</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>260</td>
<td>[? nextcode not in Dictionary! ]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 66 | 65 | 256 | 257 | 65 | 260 |

**output**: B A BA AB A
Why is code 260 not in table?

During encoding, there was no "lag" between encoding AA and outputting code for AA (last step of ex 2 encoding).

\[
\begin{align*}
260 \\
\text{ AAA} \\
\uparrow \uparrow \\
\text{ x+ c}
\end{align*}
\]

Encoded AA as 260, then next outputted code (in next iteration) was 260 (for last AA)

Can occur only for strings of the form \(X<\text{char}>Y\) where \(X<\text{char}> = <\text{char}>Y\)

For our example: AAA: \(X<\text{char}> = AA, <\text{char}>Y = AA\)

Note: Doesn't always occur for these types of strings, but occurs only for these types (necessary but not sufficient)

How to fix this?
What to set next-decoded to?

Notice, next-decoded should be `<char>Y` and `<char>Y = X <char>.

Look back at table: \( X <\text{char} > = \text{old-decoded} + c \),

so set next-decoded to old-decoded + c.

\[ \Rightarrow \text{Add to code 0 } + (2) \]

Almost done...

It's possible that this problem occurs at the very start. In that case, \( X \) is a single char, so \( Y \) is a single char and \( Y = X \), so the substring is a repeated char e.g. AAA. In this case, \( c = \text{old-decoded} \)

\[ \Rightarrow \text{Add to code 0} \]
Continuing example 2:

<table>
<thead>
<tr>
<th>old-decoded</th>
<th>next-code</th>
<th>next-decoded</th>
<th>c.</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>string code</td>
</tr>
<tr>
<td>A</td>
<td>260</td>
<td>AA</td>
<td>1</td>
<td>AA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>260</td>
</tr>
</tbody>
</table>

(2) output (next-decoded) .... AA
How to deal with variable-length binary strings when decoding:

ex 1: 65 66 66 256 257 259 (65, 9 bits)

↓ marker to indicate switch to 9 bits

⇒ all subsequent strings until next marker will use 9 bits (even if < 256)

No compression: 11 chars × 8 bits = 88 bits

Will z " " (3 chars × 8 bits) +
(1 × 9 bit marker) +
4 × 9 bits

= 24 + 9 + 36 = 69