Lempel-Ziv (msg, dictionary)
  str = msg.get_next_char()

  While (there are still chars in msg):
    c = msg.get_next_char()
    if (dictionary.contains(str+c))
      str = str + c  // build a longer string
      // (extend window length)
    else
      output str.code  // longest string with a code
      dictionary.add(str+c)
      str = c  // slide the window

  output str.code

Question: How to quickly check if a string already has a code? ⇒ Make the table a hash table (dictionary).

How good is LZ?

No compression: 8 bits x 11 chars = 88 bits
Compressed w/LZ: (8 bits) x 4 + (9 bits) x 3 = 59 bits
33% reduction.

Effectiveness: on Alice in Wonderland

Huffman’s: 15%
Lempel-Ziv: 50%

Run Time: n-length of msg, 1A1=table size
O(n) if contains is O(1)
O(1A1n) if contains searches table (unlikely)

Another example

A B R A C A D A B R A \quad (C=67, \ D=68, \ R=114)

<table>
<thead>
<tr>
<th>char</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>256</td>
</tr>
<tr>
<td>B</td>
<td>257</td>
</tr>
<tr>
<td>R</td>
<td>258</td>
</tr>
<tr>
<td>A</td>
<td>259</td>
</tr>
<tr>
<td>C</td>
<td>260</td>
</tr>
<tr>
<td>D</td>
<td>261</td>
</tr>
</tbody>
</table>

Idea: Build longest string with code

> Output string (s)
> Insert next string (s+c)
> Shift window (s=c)

Encoding: 65 66 114 65 67 65 68 256 258 256 258

No compression: 8(11) = 88 bits
Compressed: 8(7) + 2(9) = 74 bits
<table>
<thead>
<tr>
<th>old code</th>
<th>old decoded</th>
<th>c</th>
<th>nextcode</th>
<th>decoded</th>
<th>dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>A</td>
<td>B</td>
<td>66</td>
<td>B</td>
<td>string code</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>66</td>
<td>B</td>
<td>AB</td>
<td>256</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>256</td>
<td>AB</td>
<td>BB</td>
<td>257</td>
</tr>
<tr>
<td>AB</td>
<td>B</td>
<td>257</td>
<td>BB</td>
<td>BA</td>
<td>258</td>
</tr>
<tr>
<td>BB</td>
<td>A</td>
<td>259</td>
<td>ABB</td>
<td>ABB</td>
<td>259</td>
</tr>
<tr>
<td>ABB</td>
<td>A</td>
<td>259</td>
<td>65 261</td>
<td>BBA</td>
<td>260</td>
</tr>
</tbody>
</table>

next decoded (output): A B B AB BB ABB A

Verify that this is original string.

Algorithm works for all but one case when next code not in dictionary. Can this happen? When?

Only when all chars are the same!

\[
\text{Example: } A A A A A A A A \quad \text{str} \quad \text{code}
\]

\[
\begin{align*}
65, 256, 257, 65
\end{align*}
\]
Let's try decoding algorithm to see what breaks.

65 256 257 65
↑  ↑
oldcode nextcode
① ② not in dictionary

<table>
<thead>
<tr>
<th>output</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>②</td>
<td></td>
</tr>
</tbody>
</table>

What should nextdecoded be if nextcode not in dictionary?
olddecoded + first char in nextcode
= olddecoded + ch
  (A)   (A)

<Update pseudocode>
How to decode without dictionary? (Build it while reading.)

ex: 65 66 66 256 257 259 65

Safe to assume dictionary contains codes up to 255.

Ideas?

- Read a code (65)
- Look it up ..., output decoded (A)
- Build next code: concatenation of previous code + first character of next code. \( A + \text{first char of 66} = AB \)
- Add next code to dictionary.
Lenape Zev Decode (msg, dictionary)
oldcode = msg.get_next_code()
olddecoded = dictionary.translate(oldcode) // first code will always be in dictionary
output olddecoded

while (there are still codes in msg)

nextcode = msg.get_next_code()
if dictionary contains (nextcode)
nextdecoded = dictionary.translate(nextcode)
else // only occurs when all chars are the same
nextdecoded = olddecoded + c

output nextdecoded // next string = previously C = first char in nextdecoded // decoded + first char in dictionary. add (olddecoded + c) // newly decoded
olddecoded = nextdecoded // shift window
Graphs - used to model pairwise relations between entities.

**Graph** \( G = (V, E) \) (\( V \) vertices/nodes, \( E \) edges/arc/links \( (u,v) \) where \( u,v \in V \))

Nodes ~ cities

Edges ~ roads

Undirected graph - no particular ordering of vertices of an edge:

\[
V = \{a, b, c, d, e\}
\]
\[
E = \{(a,b), (a,c), (b,c), (b,d), (b,e), (c,d), (d,e)\}
\]

- or -

\[
E = \{(b,a), (c,a), (c,b), (d,b), (e,b), (d,c), (c,d)\}
\]

Directed graphs - pair of vertices of an edge are ordered:

\[
E = \{(a,b), (b,c), (b,d), (c,a), (c,d), (d,e), (e,b)\}
\]

"certain roads blocked " bc of snow"
adjacency (undirected): \( u \) adjacent to \( v \) if \((u,v)\) or \((v,u)\) \(\in E\)

adjacency (directed): \( u \) adjacent to \( v \) if \((u,v)\) \(\in E\)

path: sequence of vertices \( v_1, v_2, \ldots, v_n \) such that an edge exists for every adjacent pair in the sequence.

\[ P: c-a-b-c-d-e \rightarrow (c, a), (a, b), (b, c), (c, d), (d, e) \]

path length - number of edges in path.

\[ |P| = 5 \]

distance of vertices \( u, v \) = length of shortest path from \( u \) to \( v \).

\[ \text{dist of } c, b = 2 \]

(cycle - path of \( n \) vertices \( v_1, \ldots, v_n \) where \( v_1 = v_n \))

weighted graph - edges have weights/cost.

\( w_{u,v} \) = weight of edge \((u,v)\).

\[ \begin{align*}
&\text{amount of time it takes} \\
&\text{to traverse the edge} \\
&3 \quad 4 \quad 5 \\
&\text{C} \quad \text{D} \quad \text{E}
\end{align*} \]

Can also have weighted undirected graphs.

\( ab \) and \((b, c)\) are a pair of directly reachable from \( ab \).