LZW (Lempel-Ziv-Welch) Compression

Static model: same model for all texts

* Fast
* Not optimal: different texts have different statistical properties
* Ex: ASCII, Morse code

Dynamic model: generate model based on text

* Preliminary pass needed to generate model
* Must transmit the model
* Ex. Huffman code

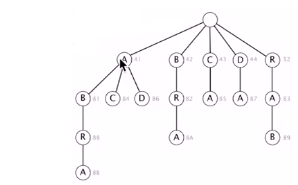
Adaptive model: progressively learn and update model as you read text

* More accurate modeling produces better compression
* Decoding must start from beginning
* Ex: LZW

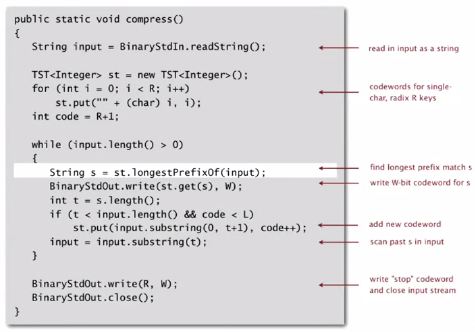
LZW Compression

1. Create ST associating W-bit codewords with string keys
2. Initialize ST with codewords for single-char keys
3. Find the longest string s in ST that is a prefix of unscanned part of input
4. Write the W-bit codewords associated with s
5. Add s + c to ST, where c is next char in the input

How to represent LZW compression code table: Use a trie for longest prefix



LZT compression Java implementation



LZW expansion: very similar to compression

* Create ST associating string values with W-bit keys
* Initialize ST to contain single-char values
* Read a W-bit key
* Find associated string value in ST and write it out
* Update ST

How to represent LZW expansion code table: an array of size 2W

How big to make ST?

* How long is the message?
* Whole message similar model?
* [many variations have been developed]

What to do when ST fills up?

* Throw away and start over [GIF]
* Throw away when not effective [Unix compress]
* [many other variations]

Why not put longer substrings in ST?

Lossless data-compression benchmarks

