

Data Compression

- reduce size of file to save $\begin{cases} \text{space} \\ \text{time} \end{cases}$

Lossless compression and expansion

- message = binary data B we want to compress
- compress = generates a compressed representation $C(B)$
- expand = reconstruct original bitstream B
- compression ratio = Bits in $C(B)$ / bits in B

Proposition : No algorithm can compress every bitstream

Run-length encoding

- * simple type of redundancy in a bitstream
- \Rightarrow long runs of repeated bits

=> 4-bit counts to represent alternating runs of 0s and 1s

$\underbrace{1111}_{15} \quad \underbrace{0111}_7 \quad \underbrace{0111}_7 \quad \underbrace{1011}_{11}$

Q: How many bits to store the counts?

A: 8 in practice => if longer than 255

=> intersperse runs of length 0

! Check out RunLength.java algorithm

Huffman Compression

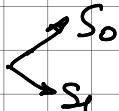
- variable length - codes
- use different number of bits to encode different chars
- how to avoid ambiguity -> no codeword is a prefix of another
- how to represent the prefix-free code?
 - => a binary tree $\left\{ \begin{array}{l} \text{chars in leaves} \\ \text{codeword's path} \end{array} \right.$
from root to leaf

Compression

M1) start at leaf: follow path up to the root
print bits in reverse

M2) create ST of key-value pairs
symbol table

Expansion

- start at root
- go left if bit is 0, go right if 1
- if leaf node, print char and return to root
- How to find best prefix-free code?
 - * Shannon-Fano algorithm
 - => divide symbols in two subsets 
 - roughly equal frequency
 - => codewords for symbols in S_0 start with 0, S_1 start with 1

! not optimal

Huffman

- count frequency of each character in input
- start with 1 node for each character with weight equal to the frequency (in sorted order)
- select 2 ties with minimum weight
- merge into single tie with cumulative weight

! Running time: Using a binary heap

$$\begin{array}{ccc} N + R \log R \\ \uparrow & & \uparrow \\ \text{input size} & & \text{alphabet size} \end{array}$$

LZW Compression

Static Model

- same model for all texts \Rightarrow static model

- not optimal

Dynamic model \Rightarrow Huffman

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

\rightarrow more accurate modeling = better compression

\Rightarrow decoding must start from the beginning

LZW Compression

- create ST associating w -bit codewords with string keys
- init ST with codewords for single-char keys
- find longest string s in ST that is a prefix of unscanned part of input
- write the w -bit codeword associated with s

- add $s+c$ to $ST \Rightarrow$ where c is next char in the input

Q: How to represent LZW compression code tables?

A: A trie to support longest prefix match

LZW Expansion

- create ST associating string values with w -bit keys
- initializes ST to contain single-char values
- read a w -bit key
- find associated string value in ST and write it out
- update ST