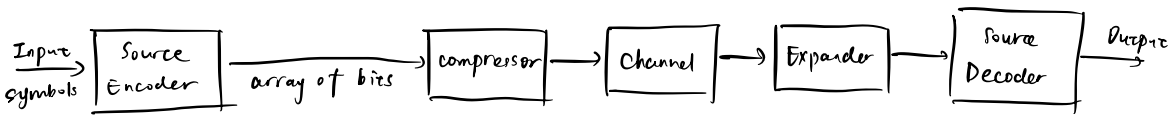
Chapter 3 Compression

data compression: a shorter string



Ideally, expander exactly reverse the action of compressor.

- Lossless (or reversible): original code ineffficient < unused bit patterns / some symbols more frequently used
- Lossy: expander produces approximation.

Methods:

① variable-length encoding: see in later chapters.

② run length encoding.

long sequences of a small number of symbols.

encoded as: a list of symbols and times it occurs. eg. BBBaaaa → 3B4a

doesn't work well for those without repeated sequences of the same symbol

③ static dictionary.

Unused codewords, assigned (as abbreviations) to frequently occurring sequences

Codebook (dictionary) static, doesn't change from one message to the next.

④ semi-adaptive dictionary.

a new dictionary defined for each message.

disadvantages: ① dictionary needs to be transmitted along

② entire message analysis beforehand cause latency

③ enough memory to store entire message to analysis

① dynamic dictionary.

using a dictionary that is calculated as the message is processed

doesn't need to accompany the message, can be used before the end of the procession

the LZW compression technique, reversible

Encoding Algorithm.

```
clear dictionary
Send start code
for each character {
    if new-entry appended with character is not in dictionary {
        Send code for new-entry.
        add new-entry appended with character as new dictionary entry
        set new-entry blank
    }
    append character to new-entry
}
Send code for new-entry
Send stop code.
```

105 i		256 (start)	
116 t	256 it	105 i	the essence is:
116 t	257 tt	116 t	① building dictionary and sending the same time
121 y	258 ty	116 t	② sequence met 2 twice then send as whole
32 space	259 y-space	121 y	⇓
98 b	260 space-b	32 space	③ new sequence must be recorded in dict
105 i	261 bi	98 b	
116 t	—	—	
116 t	262 itt	258 it	
121 y	—	—	
32 space	263 ty-space	258 ty	
98 b	—	—	
105 i	264 space-bi	260 space-b	
116 t	—	258 it	
		257 (stop)	

Decoding Algorithm

```
for each received code until stop code {
    if code is start code {
        clear dictionary
        get next code
        get string from dictionary
    }
    else {
        get string from dictionary
        update last dictionary entry by appending first character of string
    }
    add string as new dictionary entry
    output string
}
```

an exact reverse of E.A.

256 (start)
105 i
116 t
116 t
121 y
32 space
98 b
—
258 it
—
260 ty
—
262 space-b
258 it
257 (stop)

—
258 it
259 tt
260 ty
261 y-space
262 space-b
—
263 bit
—
265 ttt
265 ty-space
266 space-bi

i
t
t
y
space
b
—
it
—
ty
space-b
it

~~the~~ real difficult part is building the exact dictionary while decoding.

2-D Discrete Cosine Transformation

discrete linear transformation for image compression

takes a vector as input and returns another vector of the same size.

$\mathbb{C}$ is a discrete linear transformation. $\mathbb{O} = \mathbb{C}\mathbb{I}$

reverse: $\mathbb{I} = \mathbb{C}^{-1}\mathbb{O}$

images are 2-dimensional in nature, use matrices to represent pixels

$\mathbb{O} = \mathbb{C}\mathbb{I}\mathbb{D}$, $\mathbb{I}$ is a matrix not a vector. $\mathbb{C}$ & $\mathbb{D}$ are square.

DCT: $\mathbb{Y} = \mathbb{C}^T \mathbb{X} \mathbb{C}$ ($\mathbb{C}$ $N \times N$ size)

$\mathbb{C}_{m,n} = k_n \cos\left(\frac{(2m+1)n\pi}{2N}\right)$ where $k_n = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } n=0 \\ \sqrt{\frac{2}{N}} & \text{otherwise} \end{cases}$ $(m, n : 0 \sim N-1)$

then $\mathbb{C}^{-1} = \mathbb{C}^T$