Compression

- **Lossless**
  - Data received = data sent
  - Used for executables, text files, numeric data
- **Lossy**
  - Data received does not != data sent
  - Used for images, video, audio

Simple lossless algorithms

- **Run Length Encoding (RLE)**
  - Example: AAABBCDDDD encoding as 3A2B1C4D
  - Good for scanned text (8-to-1 compression ratio)
  - Used by fax machines
  - Can increase size for data with variation (e.g., some images)
- **Differential Pulse Code Modulation (DPCM)**
  - Example: AAABBCDDDD encoding as A0001123333
  - Change reference symbol if delta becomes too large
  - Works better than RLE for many digital images (1.5-to-1)

Huffman coding

- **Consider data as sequence of symbols**
- **Suppose you know the frequency of each symbol**
  - Assign symbols *weight* proportional to frequency
- **Use shorter code for more frequent characters**
- **Example: NYU centrex phones**
  - Consider each phone number to be a symbol
  - Most phone numbers dialed within NYU are 212-99x-xxxx
  - So compress phone number, by just dialing x-xxxx
  - But now must dial 9 for external phone numbers (longer code, but less frequently used)
- **Usually, have smaller symbol alphabet**
  - E.g., symbols are bytes, codes are in bits

Huffman trees

- **Can view Huffman codes as paths in a tree**
  - Symbols with heavier weight are higher in tree
- **Example:** $B = 16,\ A = 8,\ C = 4,\ D = 4$

```
          /\
         0  1
        /    \
       /\     B
      0  1
     /    \
    A     /\
         0  1
        /    \
       D      C
```

Symbol	Code
A	00
B	1
C	011
D	010

Building Huffman trees

- **Given weights, can build a suitable Huffman tree**
  - Take two symbols with least weight
  - Assign suffix "0" to one, suffix "1" to the other
  - Add new symbol with combined weight to figure out prefix
- **Example:** $B = 16,\ A = 8,\ C = 4,\ D = 4$
  - Let suffix of D be "0", of C be "1".
  - New weights: $B = 16,\ A = 8,\ C$-and-$D = 8$
  - Next two are A and C-and-$D = 8$; give A suffix "0"
  - Now give A-and-B-and-C suffix "0", B suffix "1"
  - Result is tree is on previous slide

Transmitting Huffman trees

- **Decoder needs Huffman tree**
- **One approach: Hard-code tree in encoder/decoder**
  - E.g., could come up with suitable weights for English text
- **Another approach: Include tree w. message**
  - Deterministically agree on one tree per set of weights:
  - All codes of a given bit length have lexicographically consecutive values, in the same order as the symbols they represent;
  - Shorter codes lexicographically precede longer codes
  - Now just need to transmit length of each symbol's code (which can be further compressed)

Arithmetic coding

- **Like Huffman, assume symbol weights known**
- **Assign each symbol a range in** $[0, 1)$
  - Example: Say weights are $B = 16,\ A = 8,\ C = 4,\ D = 4$
    $B = [0, 0.5)$, $A = [0.5, 0.75)$, $C = [0.75, 0.875)$, $D = [0.875, 1)$
  - Now for each symbol, subdivide the range
  - For String "BBA", ranges will subdivide to get:
    $[0, 0.5), [0, 0.25), [0.125, 0.1875)$
  - Transmit compressed string as rational number in range
- **Potentially better than Huffman codes**
  - Can better accommodate non-power-of-two weights
  - Less often used (in part because of patents)

Dictionary-Based Methods

- **Build dictionary of common terms**
  - Variable length strings
- **Transmit index into dictionary for each term**
- **Lempel-Ziv (LZ) is the best-known example**
- **Commonly achieve 2-to-1 ration on text**
- **Variation of LZ used to compress GIF images**
  - First reduce 24-bit color to 8-bit color
  - Treat common sequence of pixels as terms in dictionary
  - Not uncommon to achieve 10-to-1 compression (x3)

Gzip compression (LZ77)

- **Big idea: Include backreferences to past data**
- **Consider string "Blah blah blah blah blah!"**[a]
- **Notice repeat: "B<u>lah blah b</u>lah blah!"**
- **Encode "Blah blah b" as "Blah b**$[D = 5, L = 5]$**"**
  - Go back D characters, copy string of L bytes
- **But why stop there. Can say:**
  "Blah b$[D = 5, L = 18]$**!"**
  - Length L can extend into copied string!

[a] example from gzip site

Gzip details

- **Compressed file is composed of blocks of data**
- **First bit is BFINAL – set if final block**
- **Next two bits are BTYPE**
  - 00 – no compression (if compression makes things worse)
  - 01 – compressed with fixed Huffman codes
  - 10 – compressed with dynamic Huffman codes
  - 11 – reserved (error)
- **No compression blocks (up to $2^{16} - 1$ Bytes):**

```
  0   1   2   3   4...
+---+---+---+---+==============================+
|  LEN  | NLEN  |... LEN bytes of literal data...|
+---+---+---+---+==============================+
```

(NLEN is one's compliment of LEN)

Huffman coding of gzip

- **Use two Huffman trees**
  - One tree for symbols & length (L) values
  - Second tree for back distances (D)
  - Note L value always followed by D, so no ambiguity
- **Symbol/length alphabet has 285 symbols**
  - 0–255 $\rightarrow$ are data bytes
  - 256 $\rightarrow$ end of block
  - 257–285 $\rightarrow$ length of backreference
- $L \in \{3, \ldots, 258\}$**. How to encode w. 29 values?**

Encoding L

- **Add extra bits after some symbols**
  - E.g., Symbol 264 means $L = 10$
  - 265 means $L \in \{11, 12\}$, extra bit follows

Code	Extra Bits	Length(s)	Code	Extra Bits	Lengths	Code	Extra Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

Encoding D

- **Similar scheme for D, which has 30 symbols**

Code	Extra Bits	Dist	Code	Extra Bits	Dist	Code	Extra Bits	Distance
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

Gzip's fixed Huffman codes

- **D values just encoded as 5 bits**
  - Values 30 and 31 never used
- **Symbol/L values use following code:**

```
Lit Value    Bits        Codes
---------    ----        -----
  0 - 143     8          00110000 - 10111111
144 - 255     9         110010000 - 111111111
256 - 279     7           0000000 - 0010111
280 - 287     8          11000000 - 11000111
```

- **Why might these be a good choice?**

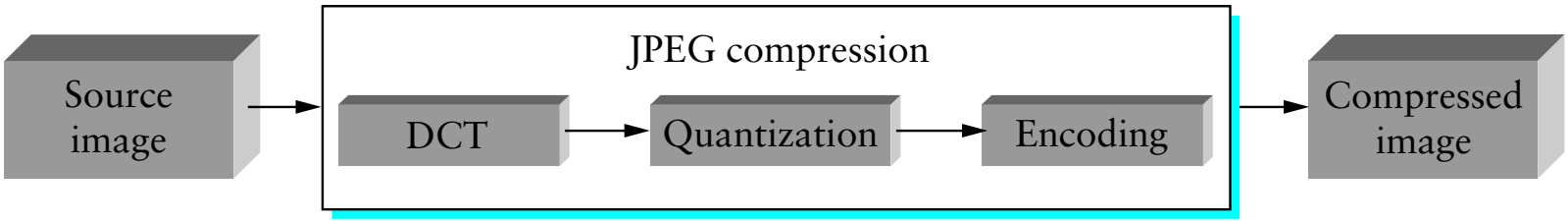
Dynamic Huffman codes

- **Must first transmit two Huffman trees (symb/L & D)**
  - Recal means xmitting code lenth for each symbol
  - Use code length 0 for unused symbols
  - Include length of table, truncate after last non-0
- **Use Huffman + RLE on the Huffman trees! Alphabet:**

```
          Extra
  Code    Bits   Meaning
------ ------ --------
0 - 15     0   Represent code lengths of 0 - 15
    16     2   Copy the previous code length 3 - 6 times
    17     3   Copy previous code length 3 - 10 times
    18     7   Copy previous code length 11 - 138 times
```

- Must transmit 3-bit code length for each of 19 symbols (0–18)
- Transmit lengths in order that often helps truncation:
  16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15

Lossy image compression

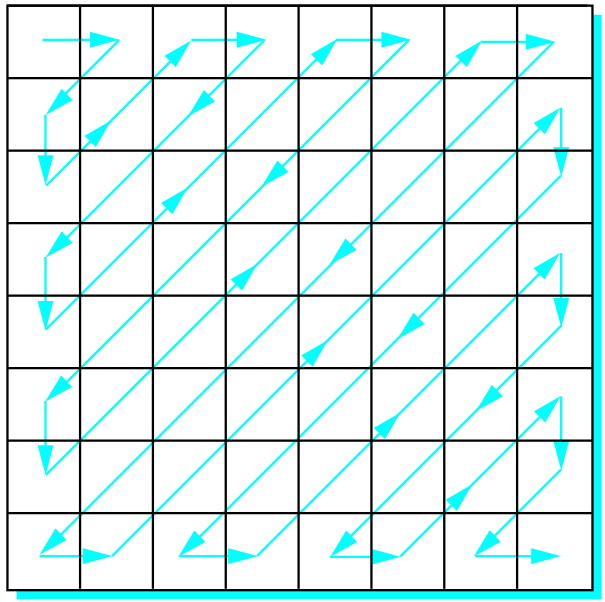


- **JPEG: Joint Photographic Expert Group (ISO/ITU)**
- **Lossy still-image compression**
- **Three phase process**
  - Process in 8x8 block chunks (macro-block)
  - DCT: transforms signal from spatial domain into and equivalent signal in the frequency domain (loss-less)
  - Apply a quantization to the results (lossy)
  - RLE (for 0s) + Huffman encoding (loss-less)
  - For color: each pixel is three values (YUV) ($Y =$ luminescence, $U = c_1(B - Y)$, $V = c_2(R - Y)$)

- **Quantization Table**

$$\begin{pmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{pmatrix}$$

- **Encoding Pattern**

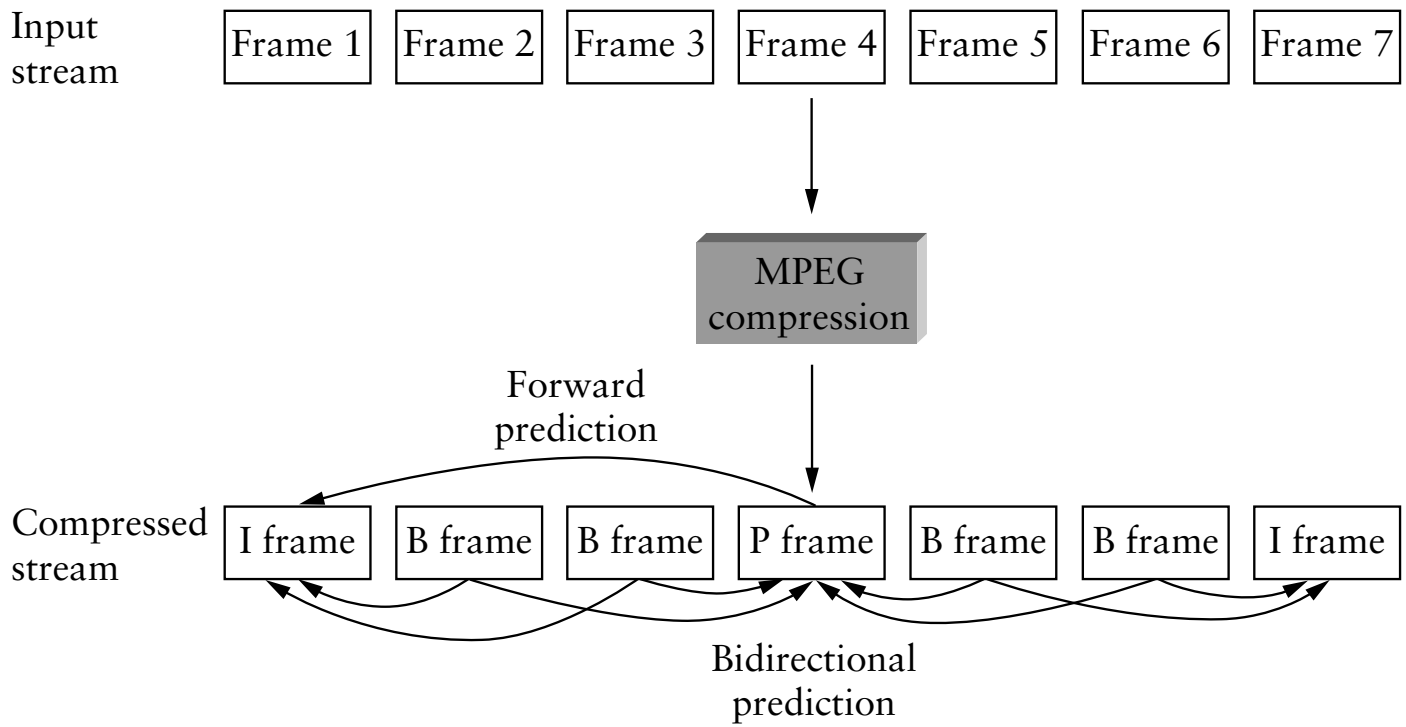


MPEG

- **Motion Picture Expert Group**
- **Lossy compression of video**
- **First approximation: JPEG on each frame**
  - Y value encoded w. 16×16 "macroblock"
  - U, V channels downsampled into 8×8 macroblocks
- **Also remove inter-frame redundancy**
  - E.g., when large objects move across image

MPEG (cont)

- **Frame types**
  - I frames: intrapicture
  - P frames: predicted picture
  - B frames: bidirectional predicted picture



- **Example sequence transmitted as I P B B I B B**

MPEG (cont)

- **B and P frames**
  - Coordinate for the macroblock in the frame
  - Motion vector relative to previous reference frame (B, P)
  - Motion vector relative to subsequent reference frame (B)
  - Delta for each pixel in the macro block
- **Effectiveness**
  - Typically 90-to-1
  - As high as 150-to-1
  - 30-to-1 for I frames
  - P and B frames get another 3 to 5$\times$

Transmitting MPEG

- **Adapt the encoding**
  - Resolution
  - Frame rate
  - Quantization table
  - Specified in GOP (group of pictures) header
- **Packetization (best to lose whole frames)**
- **Dealing with loss**
- **GOP-induced latency**

MP3

- **CD Quality**
  - 44.1 kHz sampling rate
  - 2 x 44.1 x 1000 x 16 = 1.41 Mbps
  - Actually requires more for synchronization & error correction
- **Strategy**
  - Split into some number of frequency bands
  - Divide each subband into a sequence of blocks
  - Encode each block using DCT + Quantization + Huffman
  - Trick: how many bits assigned to each subband