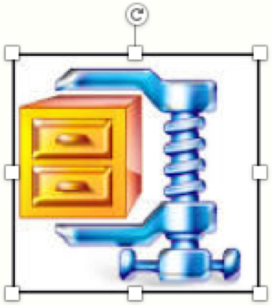
Overview

- Last Lecture
  - Data Transmission
- This Lecture
  - Data Compression
  - Source: Lecture notes
- Next Lecture
  - Data Integrity 1
  - Source : Sections 10.1, 10.3

Data Compression



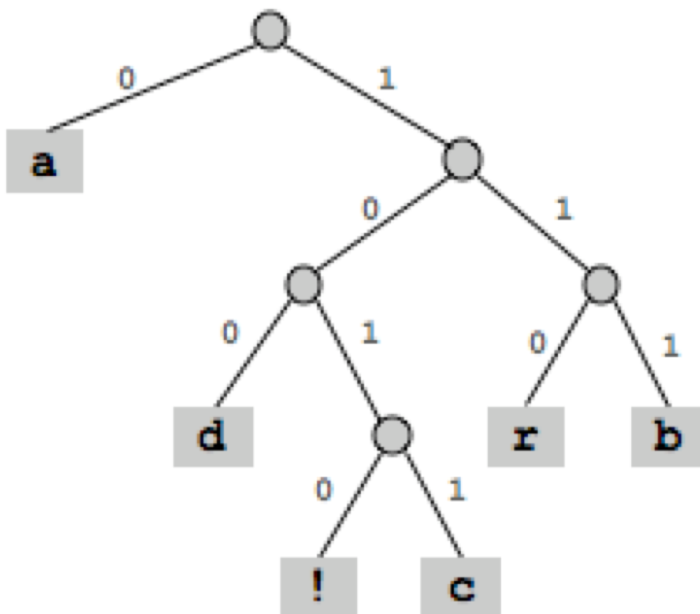
Data Compression

- Decreases space, time to transmit, and cost
- Bit rate is limited, can we send fewer bits and still deliver the data reliably? (Reduce the number of bits while retaining its meaning)
- Various approaches for data compression: Huffman, Run Length, LZW

Huffman code

Huffman coding

-- an algorithm developed by David A. Huffman while he was a Ph.D. student at MIT



char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

Huffman code

- Variable length code based on the frequency of character use.
  - Most frequently used characters -> shortest codes
  - Least frequently used characters -> longest codes
- A simple example
  - Text – EEEEAEEBFEEE (ASCII 12 * 7 = 84 bits)
  - E-0, A-100, B-101, F-110
  - Code – 000010000101110000 (18 bits)

Huffman Code (cont.)

https://en.wikipedia.org/wiki/Huffman_coding#/media/File:Huffman_huff_demo.gif

Video: https://www.youtube.com/watch?v=MleGSpPpHXs

Huffman code: Code formation

-Assign weights to each character

-Merge two lightest weights into one root node with sum of weights (why binary tree?)

-Repeat until one tree is left

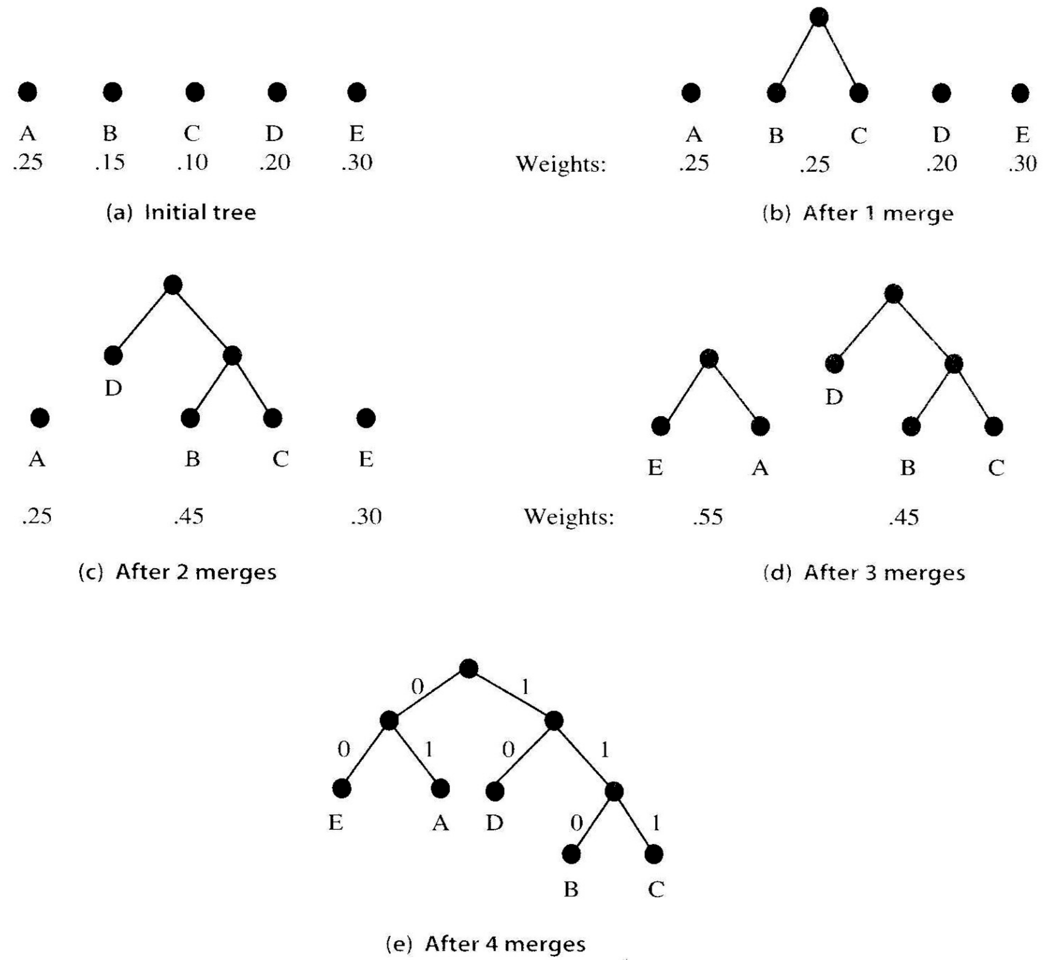
-Traverse the tree from root to the leaf (for each node, assign 0 to the left, 1 to the right)

Huffman Code (cont.)

Text: ABECADBC….

LETTER	FREQUENCY
A	25%
B	15%
C	10%
D	20%
E	30%

LETTER	CODE	ALTERNATE CODE
A	01	10
B	110	010
C	111	011
D	10	11
E	00	00



(a) Initial tree

(b) After 1 merge

(c) After 2 merges

(d) After 3 merges

(e) After 4 merges

Huffman Code (cont.)

- Huffman code: Code Interpretation
  - No prefix property (Restriction): The code for any character never appears as the prefix or start of the code for any other character. (guarantees the codes can be translated back)
  - Receiver continues to receive bits until it finds a code and forms the character
  - 01110001110110110111 (extract the string)

Huffman Code (cont.)

Huffman code steps:

- To each character, associate a binary tree consisting of just one node. To each tree, assign the character's frequency, which is called the tree's weight.
- Look for the two lightest-weight trees. If there are more than two, choose among them randomly. Merge the two into a single tree with a new root node whose left and right sub trees are the two we chose. Assign the sum of weights of the merged trees as the weight of the new tree.
- Repeat the previous step until just one tree is left.

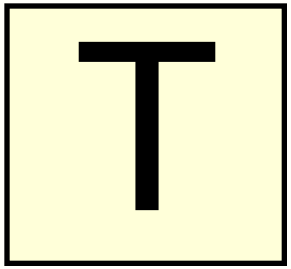
Run Length Encoding (Character-Level)

- Used for character data only
- Send an alternating set of numbers and characters.
- Example
  - HHHHHHHUFFFFFFFFFFFFFF
  - 7H1U14F

  Video: https://www.youtube.com/watch?v=ypdNscvym_E

Run Length Encoding (Bit-Level)

- Consider a picture of the letter T.



- 70-90% of the space is white space, which means many continuous zeroes to be transmitted.
- Group the runs of zeroes and send their length instead.

Run Length Encoding (Bit-Level cont.)

- Decide the number of bits to represent a run length.
- Encoding algorithm (4 bit lengths)
  - Count the number of 0s between two 1s
  - If the number is less than 15, write it down in binary form.
  - If it is greater than or equal to 15, write down 1111, and a following binary number to indicate the rest of the 0s. If more than 30, repeat this process.
  - If the data starts with a 1, write down 0000 at the beginning.
  - If the data ends with a 1, write down 0000 at the end.
  - Send the binary string.

Run Length Encoding (Bit-Level cont.)

Decoding algorithm:

Group all the bits into 4-bit groups.

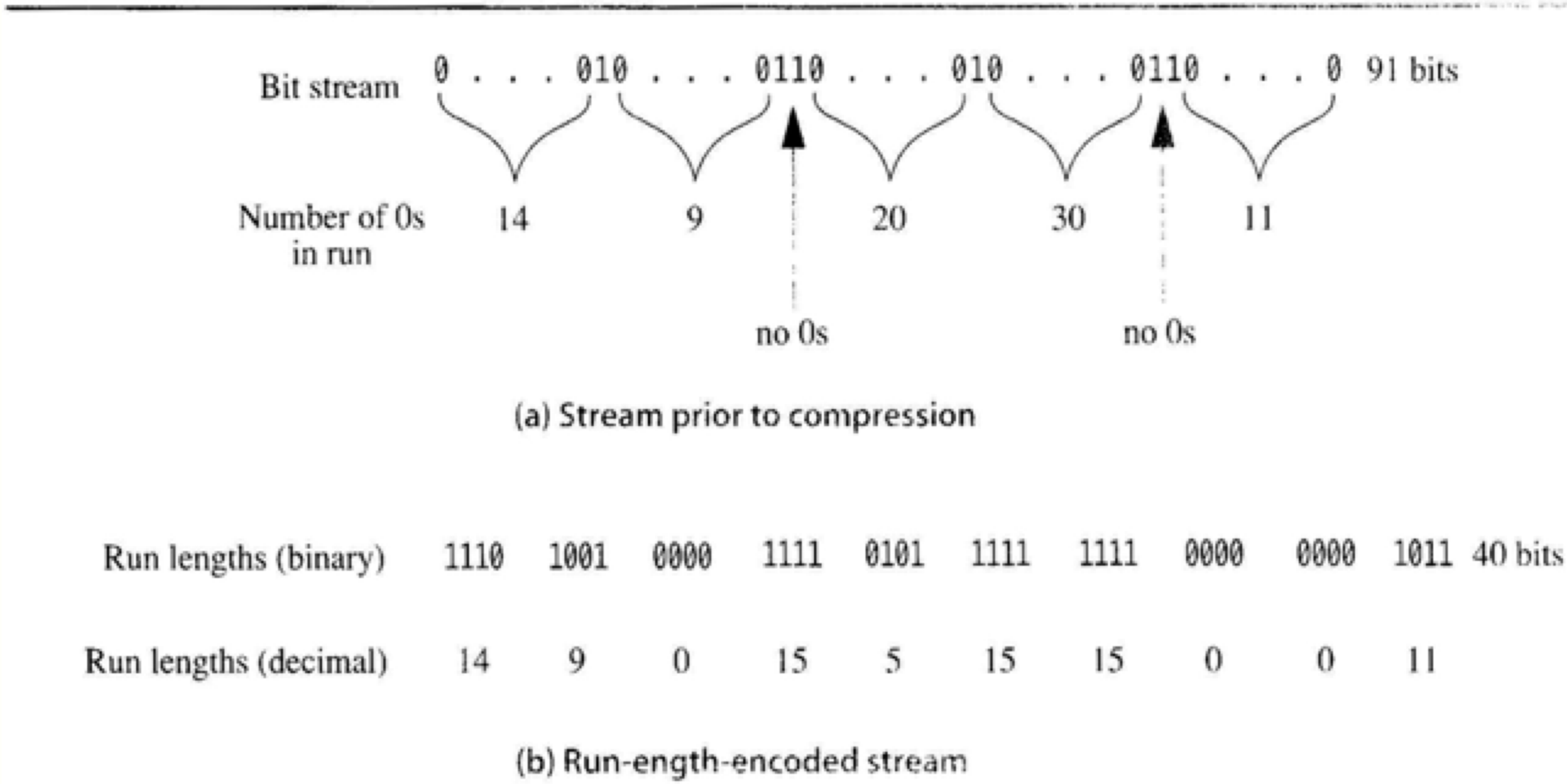
1. For each 4-bit group, write down that number of 0s.
2. If at the end of the bit string, stop.
3. If not at the end of the bit string:

   If the 4-bit group was less than 15, write down a 1. Go to step 1.

   If the 4-bit group is 15, go to step 1.

Run Length Encoding (Bit-Level cont.)

Figure 3.33 Stream Prior to Compression and Run-Length-Encoded Stream



Lempel-Ziv Compression

- In text, phrases or entire words are repeated very often.
- Look for repeated strings. Store them and a code in a dictionary.
- In the output, replace these repeated strings with the code.
- zip, unzip, *compress* command in Unix.

Lempel-Ziv Compression (cont.)

- From Crichton, M. *Jurassic Park.*

The tropical rain fell in drenching sheets, hammering the corrugated roof of the clinic building, roaring down the metal gutters, splashing on the ground in a torrent.

- Some repetitions:
  - the - #
  - ro - $
  - ing - %
  - en - &
  - rr - *

Lempel-Ziv Compression (cont.)

The tropical rain fell in drenching sheets, hammering the corrugated roof of the clinic building, roaring down the metal gutters, splashing on the ground in a torrent.

#t$pical rain fell in dr&ch% sheets, hammer% #co*ugated $of of #clinic build%, $ar% down # metal gutters, splash% on #g$und in a to*&t.

Lempel-Ziv-Welch (LZW) algorithm (cont.)

- Add all possible character codes to the dictionary
- w = "";
- for (every character c in the incoming data) {
-    if ((w + c) exists in the dictionary) {
-       w = w + c;
-    } else {
-       add (w + c) to the dictionary;
-       add the dictionary code for w to output;
-       w = c;
-    }
- }
- add the dictionary code for w to output;
- display output;

Lempel-Ziv-Welch (LZW) algorithm

- A dictionary is initialized to contain all the single characters.
- Scan through the input string for successively longer substrings (w+c) that is not in the dictionary.
- When such a string (w+c) is found, the index for the string less the last character (i.e., the longest substring that *is* in the dictionary, w) is sent to output.
- The new string (including the last character, w+c) is added to the dictionary with the next available code.
- The last input character (c) is then used as the next starting point to scan for substrings.

Lempel-Ziv-Welch (LZW) algorithm (cont.)

Data: a b b a a b b a a b a b b a a a a b a a b b a

0 1 1 0 2 4 2 6 5 5 7 3 0

Dictionary			
Index	Entry	Index	Entry
0	a	7	b a a
1	b	8	a b a
2	a b	9	a b b a
3	b b	10	a a a
4	b a	11	a a b
5	a a	12	b a a b
6	a b b	13	b b a

Lempel-Ziv-Welch (LZW) algorithm (cont.)

- No need to send the dictionary except the initial encoding for the alphabet letters.
- Need to agree on the initial coding between the sender and the receiver.
- The dictionary can be reconstructed as decompression is done.

Summary

- Huffman encoding
- Run-length encoding
- Lempel-Ziv Compression