Compression and Error Checking

What is Compression?

Why is it useful?

Compression

- Resources are expensive
  - Storage space
  - Transmission bandwidth
  - Time to read/send
- Does require more computation; can be a tradeoff

Run Length Encoding

- Find "runs" (repeated sequences) in data
- Replace them with a shorter version
- Usually the sequence and a count

RLE Example

WWWWWWWWWWWWBWWWW

WWWWWWWWBBBWWWWWW

WWWWWWWWWWWWWWWWW

WBWWWWWWWWWWWWWWW

Image Source: [Wikipedia]

RLE Example

WWWWWWWWWWWWBWWWW

WWWWWWWWBBBWWWWWW

WWWWWWWWWWWWWWWWW

WBWWWWWWWWWWWWWWW

12W1B12W3B24W1B15W

Image Source: [Wikipedia]

Wait, isn't text just numbers (ASCII)?

How can we tell which numbers are text and which are not?

RLE Example

WWWWWWWWWWWWBWWWW

WWWWWWWWBBBWWWWWW

WWWWWWWWWWWWWWWWW

WBWWWWWWWWWWWWWWW

Image Source: [Wikipedia]

Escape Coding

WWWWWWWWWWWWBWWWW

WWWWWWWWBBBWWWWWW

WWWWWWWWWWWWWWWWW

WBWWWWWWWWWWWWWWW

WW12BWW12BB3WW24BWW15

Huffman Coding

- Count the occurrences of each character
- Make a binary tree with the data
- The paths of the tree give the codes

Huffman Example

THIS IS AN EXAMPLE OF A HUFFMAN TREE

288 bits (8 * 36)

Image Source: [Wikipedia]

Huffman Example

THIS IS AN EXAMPLE OF A HUFFMAN TREE

- space: 7
- a: 4
- e: 4
- f: 3
- h: 2
- i: 2
- m: 2
- n: 2

…..

Huffman Example

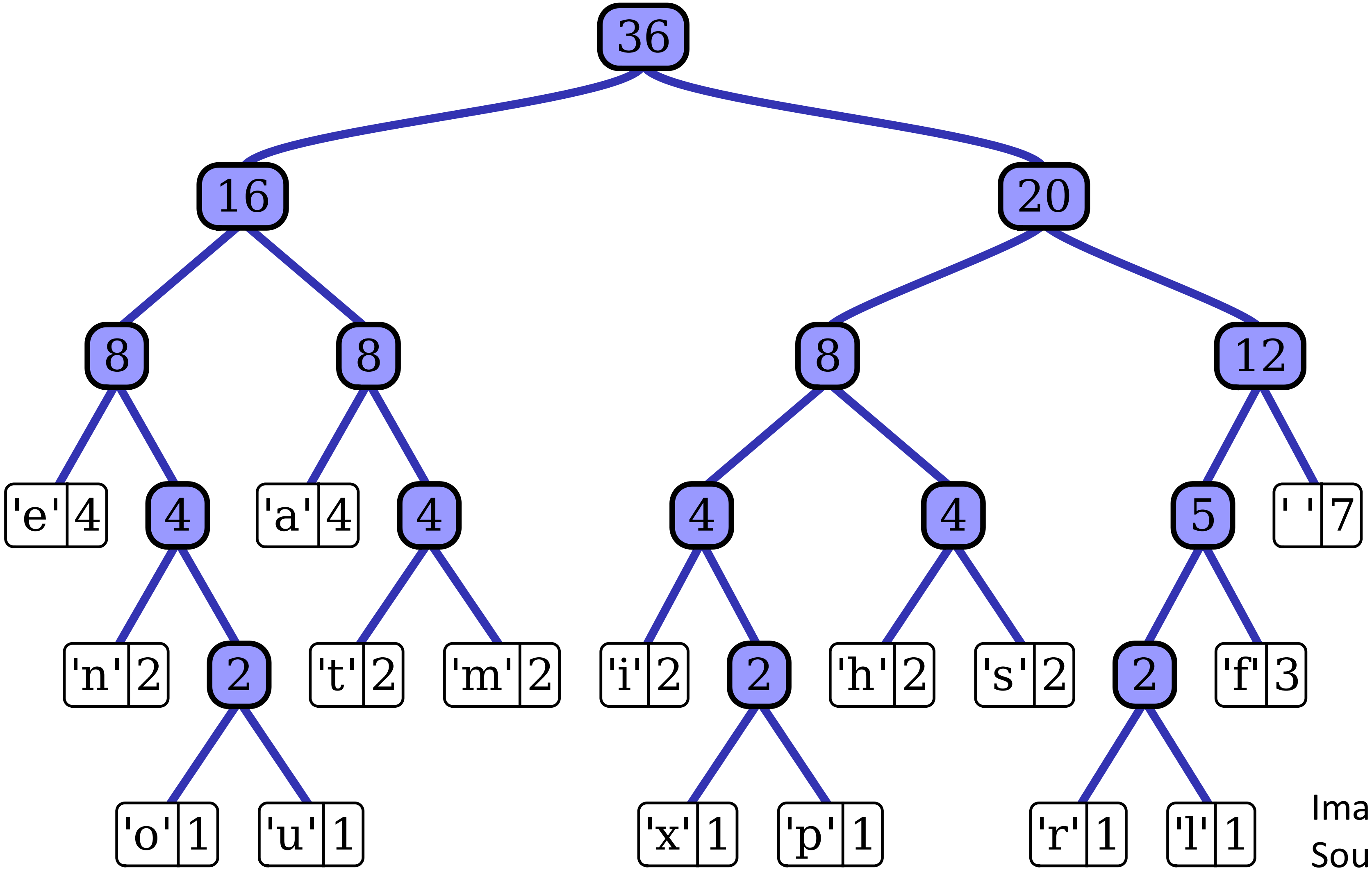


Image Source: Wikipedia

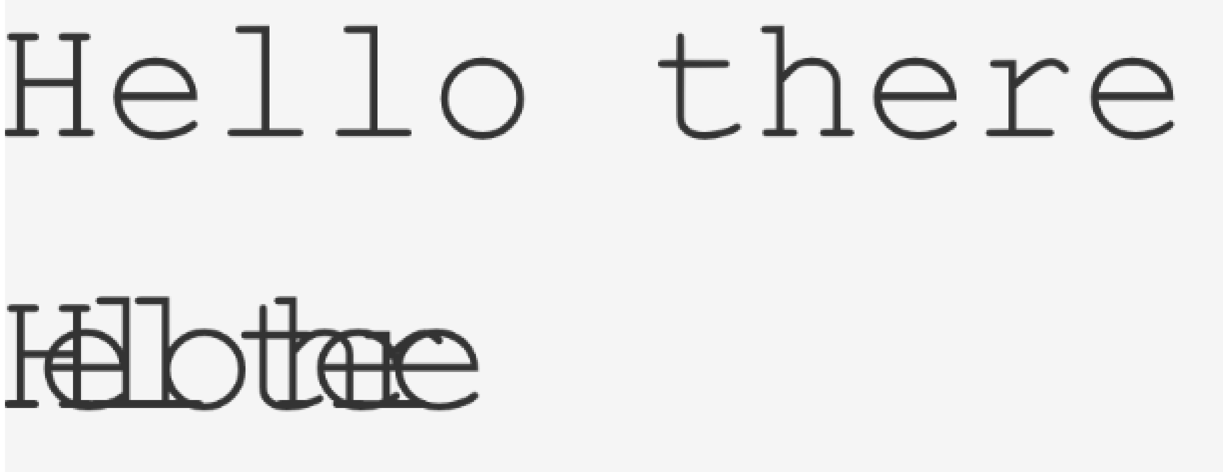
Huffman Example

```
THIS IS AN EXAMPLE OF
A HUFFMAN TREE
0110 1010 1000 1011
111 1000 1011 111
010 0010 111 000 10010
... (135 bits)
```


Image Source: Wikipedia

What is Error Checking?
Why is it useful?

Intuition - Squeezing



Simple - Addition

```
Hello there
72 101 108 108 111 32
116 104 101 114 101 33
= 1101
```

Simple - Addition

- If we only have the sum, can we recover the original?
- How well does this detect errors?
- Are there errors it cannot detect?
- 1101 is larger than 8 bits. How should we handle that?
- Can we do better?

Better - Pinpoint

4837543622563997

4	8	3	7	?
5	4	3	6	?
2	2	5	6	?
3	9	9	7	?
?	?	?	?	

Better - Pinpoint

4	8	3	7	2
5	4	3	6	8
2	2	5	6	5
3	9	9	7	8
4	3	0	6	

4837254368225653997843O6

Better - Pinpoint

48372543682756539978 4306

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		

Better – Fletcher's Checksum

INPUT: a data word (e.g., a sequence of ASCII-numbers)
OUTPUT: two checksums for the word, each sized to fit in one byte
ALGORITHM:

1. divide the Word into a sequence of equally-sized blocks, b_1 b_2 ... b_n
2. define two checksums, starting at C1 = 0 and C2 = 0
3. for each block, b_i,
   add b_i to C1
   add the new value of C1 to C2
4. compute Checksum1 = C1 mod 255 and Checksum2 = C2 mod 255
5. return Checksum1 and Checksum2

Better – Fletcher's Checksum

```
72 101 108 108 111 32 116 104 101 114 101 33
```

Block	C1	C2
72	72	72
101	173	245
108	281	526
108	389	915
Total:	1101 (81)	7336 (196)

Testing Fletcher's

- 72 101 108
- 72 108 101
- 74 99 108
- 72 101 0 108

- Which ones does it catch?

Can we do better?

Cyclic Redundancy Check (CRC)

```
Input: 010100001001
Check: 1011

    010100001001
XOR  1011
    ------------
        10001001
    XOR 1011
       ---------
        00111001
      XOR 1011
         -------
           10101
       XOR 1011
          ------
   Checksum: 011
```

Hash Codes

Choose a "hash base", b (e.g., b= 2 or b= 10 or b= 37)

For a word of integers of length n+1:

$w = x_0 \; x_1 \; x_2 \; ... \; x_{n-1} \; x_n$,

Compute this hash number:

$hash(w) = (x_0 * b^n) + (x_1 * b^{n-1}) + (x_2 * b^{n-2}) + ...$
$(x_{n-1} * b^1) + x_n$

Hash Codes

```
For  word = 4 5 6,
when  b = 10,
     hash(word) = (4 * 10²) + (5 * 10¹) + 6  =
     400 + 50 + 6  =  456
when b = 100,
     hash(word) = (4 * 100²) + (5 * 100¹) + 6  =
     40000 + 500 + 6  =  40506
when b = 5,
     hash(word) = (4 * 5²) + (5 * 5¹) + 6  =
     100 + 20 + 6  =  126
when b = 2,
     hash(word) = 16 + 10 + 6 =  32
when b = 1,
     hash(word) = 4 + 5 + 6 = 15
```

Hash Codes

```
For word = 12 33 08,
when b = 10,
      hash(word) = (12 * 10^2) + (33 * 10^1) + 8  =
      1200 + 330 + 8  =  1538
when b = 100,
      hash(word) = (12 * 100^2) + (33 * 100^1) + 8  =
      120000 + 3300 + 8  =  123308
when b = 5,
      hash(word) = (12 * 5^2) + (33 * 5^1) + 8  =
      300 + 165 + 8  =  473
when b = 2,
      hash(word) = 48 + 66 + 8 =  122
```

Hamming Codes

```
BIT#:     1        2         3  4        5  6  7
        -------------------------------------------
PURPOSE: P3,5,7  P3,6,7    D  P5,6,7   D  D  D

where  D  is a data bit, and
Pa,b,c,...  is the parity bit for data bits at  a,b,c,...

Examples:
DATA 3 5 6 7  P3,5,7  P3,6,7  P5,6,7   HAMMING CODE
------------------------------------------------------
     0 1 0 0     1       0       1      1 0 0 1 1 0 0

     1 0 1 1     0       1       0      0 1 1 0 0 1 1
```