

Lecture 18: Compression

CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

Overview

- When we decide how to represent something in bits, there are some competing interests:
 - easily manipulated/processed
 - short
- Common to use two representations:
 - one direct to allow for easy processing
 - one terse (compressed) to save storage and communication costs

Plan

- I'm going to try to describe one neat idea, implicit in Chapter 6: Huffman coding.
- For more information, see wikipedia:
 - http://en.wikipedia.org/wiki/Huffman_coding

Gettysburg Address

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us -- that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion -- that we here highly resolve that these dead shall not have died in vain -- that this nation, under God, shall have a new birth of freedom -- and that government of the people, by the people, for the people, shall not perish from the earth.

Character Counts

- For simplicity, let's turn the uppercase letters into lowercase letters. That leaves us with:

| | | | | | | | |
|-----|-----|-----|---|----|---|-----|---|
| 282 | <s> | 31 | c | 3 | k | 44 | s |
| 4 | | 58 | d | 42 | l | 126 | t |
| 22 | , | 165 | e | 13 | m | 21 | u |
| 15 | - | 27 | f | 77 | n | 24 | v |
| 10 | . | 28 | g | 93 | o | 28 | w |
| 0 | ? | 80 | h | 15 | p | 0 | x |
| 102 | a | 68 | i | 1 | q | 10 | y |
| 14 | b | 0 | j | 79 | r | 0 | z |

Attempt #1: ASCII

- The standard format for representing characters uses **8 bits** per character.
- The address is 1482 characters long, so a total of 11856 bits is needed using this representation.
 - 8 bits per character
 - 11856 total bits
 - 100% the size of ASCII representation.

Attempt #2: Compact

- Note that, at least in its lowercase form, there are only 32 different characters needed.
- Therefore, each can be assigned a 5 bit code (32 different 5-bits patterns).
 - 5 bits per character
 - 7410 total bits
 - 62.5% the size of ASCII representation.

5-bit Patterns

| | | | | | | | |
|-------|-----|-------|---|-------|---|-------|---|
| 00000 | <s> | 01000 | c | 10000 | k | 11000 | s |
| 00001 | | 01001 | d | 10001 | l | 11001 | t |
| 00010 | , | 01010 | e | 10010 | m | 11010 | u |
| 00011 | - | 01011 | f | 10011 | n | 11011 | v |
| 00100 | . | 01100 | g | 10100 | o | 11100 | w |
| 00101 | ? | 01101 | h | 10101 | p | 11101 | x |
| 00110 | a | 01110 | i | 10110 | q | 11110 | y |
| 00111 | b | 01111 | j | 10111 | r | 11111 | z |

Attempt #3: Variable Len

- Some characters are much more common than others.
- Give the 4 most common characters a 3-bit code, and the remaining 28 a 6-bit code.
- How many bits do we need now?

Variable Length Patterns

| | | | | | | | |
|--------|-----|--------|---|--------|---|--------|-----|
| 000 | <s> | 100100 | i | 101100 | v | 110100 | y |
| 001 | e | 100101 | d | 101101 | , | 110101 | |
| 010 | t | 100110 | s | 101110 | u | 110110 | k |
| 011 | a | 100111 | l | 101111 | - | 110111 | q |
| 100000 | o | 101000 | c | 110000 | p | 111000 | ? |
| 100001 | h | 101001 | w | 110001 | b | 111001 | j |
| 100010 | r | 101010 | g | 110010 | m | 111010 | x |
| 100011 | n | 101011 | f | 110011 | . | 111011 | z |

Decodability

- Note that the code was chosen so that the first bit of each character tells you whether the code is short (0) or long (1).
- This choice ensures that a message can actually be decoded:
- `10000110010000010100001001100010001110011`
- h i <s> t h e r e .
- 42 bits, not 45. But, harder to work with.

What Gives?

- We had assigned all 32 characters 5-bit codes.
- Now we've got 4 that have 3-bit codes and 28 that are 6-bit codes. So, more than half of the characters have actually gotten *longer*.
- How can that change help?
- Need to factor in how *many* of each characters there are.

Adding Up the Bits

- How many bits to write down just the letter “y”? Well, there are 10 “y”s and each takes 6 bits. So, 60 bits. (It was 50, before.)
- How about “t”? There are 126 and each takes 3 bits. That’s 378 (was 630).
- So, how do we total them all up?
- Let c be a character, $\text{freq}(c)$ the number of times it appears, and $\text{len}(c)$ its encoding length.
 - Total bits = $\sum_c \text{freq}(c) \times \text{len}(c)$

Summing It Up

- $282 \times 3 + 165 \times 3 + 126 \times 3 + 102 \times 3 + 93 \times 6 + 80 \times 6 + 79 \times 6 + \dots + 0 \times 6 + 0 \times 6 = 6867$

| | | | | | | | |
|-----|-----|----|---|----|---|----|-----|
| 282 | <s> | 68 | i | 24 | v | 10 | y |
| 165 | e | 58 | d | 22 | , | 4 | |
| 126 | t | 44 | s | 21 | u | 3 | k |
| 102 | a | 42 | l | 15 | - | 1 | q |
| 93 | o | 31 | c | 15 | p | 0 | ? |
| 80 | h | 28 | w | 14 | b | 0 | j |
| 79 | r | 28 | g | 13 | m | 0 | x |
| 77 | n | 27 | f | 10 | . | 0 | z |

Attempt #3: Summary

- Total for this example:
 - 4.6 bits per character
 - 6867 total bits
 - 57.9% the size of ASCII representation.

Attempt #4: Sorted

```
0      <s>
10     e
110    t
1110   a
11110  o
...
```

- Total for this example:
 - 7.1 bits per character
 - 10467 total bits
 - 88.3% the size of ASCII representation.

Attempt #5: Your Turn

- Make sure it is decodable!

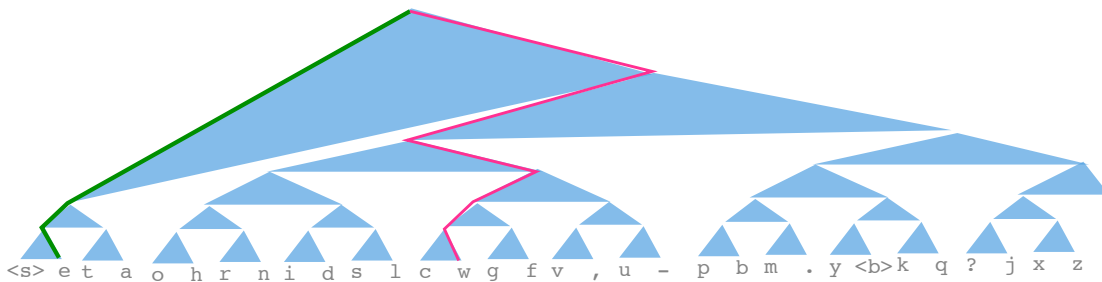
| | | | | | | | |
|-----|-----|----|---|----|---|----|-----|
| 282 | <s> | 68 | i | 24 | v | 10 | y |
| 165 | e | 58 | d | 22 | , | 4 | |
| 126 | t | 44 | s | 21 | u | 3 | k |
| 102 | a | 42 | l | 15 | - | 1 | q |
| 93 | o | 31 | c | 15 | p | 0 | ? |
| 80 | h | 28 | w | 14 | b | 0 | j |
| 79 | r | 28 | g | 13 | m | 0 | x |
| 77 | n | 27 | f | 10 | . | 0 | z |

Can We Do Better?

- Shannon invented information theory, which talks about bits and randomness and encodings.
- Fano and Shannon worked together on finding minimal size codes. They found a good heuristic.
- Fano assigned the problem to his class.
- Huffman solved it, not knowing his prof. had unsuccessfully struggled with it.

Tree (Prefix) Code

- First, notice that a code can be drawn as a tree.
- Left = "0", right = "1". So, e = "001", w = "101001".
- Tree structure ensures code is decodable: Bits tell you unambiguously which character.



Huffman Coding

- Make each character a subtree ("block") with count equal to its frequency.
- Take two blocks with smallest counts and "merge" them into left and right branches. The count for the new block is the sum of the counts of the blocks it is made out of.
- Repeat until all blocks have been merged into one big block (single tree).
- Read the code off the branches in the tree.

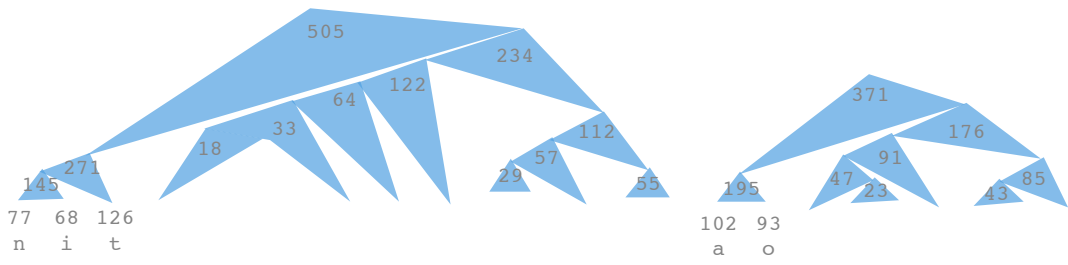
Partial Example

58 44 42 31 28 28 27 24 22 21 15 15 14 13 10 10 4 3 1
 d s l c w g f v , u - p b m . y k q

58 44 42 31 28 28 27 24 22 21 15 15 14 13 10 10 3 1 4
 d s l c w g f v , u - p b m . y k q



58 44 42 31 28 28 27 24 22 21 15 15 14 13 10 10 3 1 4
 d s l c w g f v , u - p b m . y k q





Partial Example


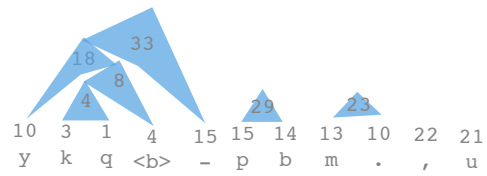
22 21 15 15 14 13 10 10 4 3 1
 , u - p b m . y k q




22 21 15 15 14 13 10 10 3 1 4
 , u - p b m . y k q




22 21 15 15 14 13 10 10 3 1 4
 , u - p b m . y k q

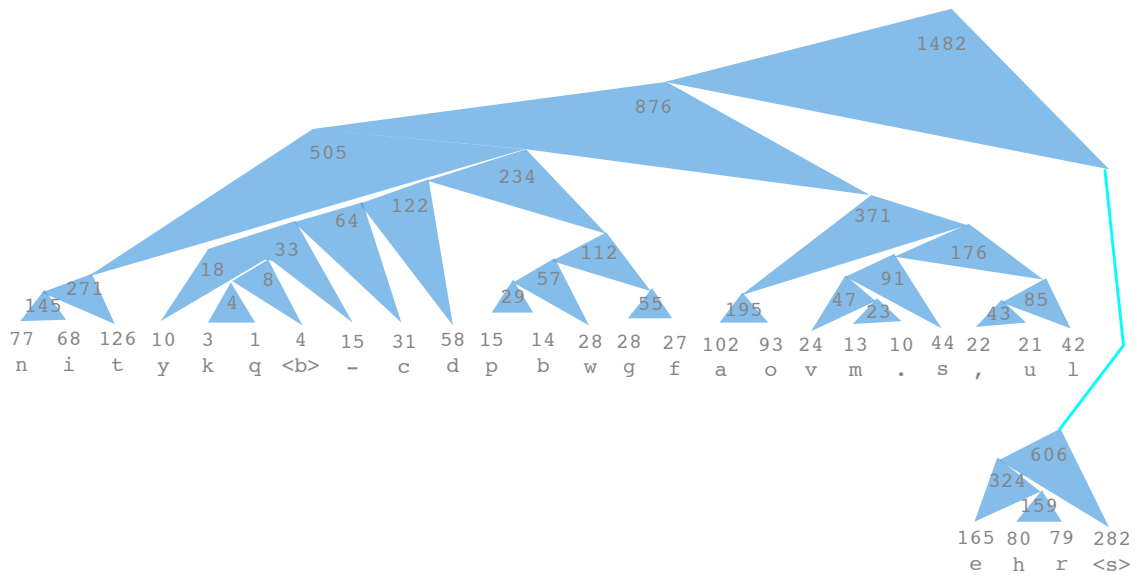
22 21 10 3 1 4 15 15 14 13 10
 , u y k q - p b m .



13 10 22 21 10 3 1 4 15 15 14
 m . , u y k q - p b



Completed Code Tree



Created Code

| | | | | | | | |
|-------|-----|--------|---|---------|---|------------|-----|
| 11 | <s> | 00001 | i | 011000 | v | 00100000 | y |
| 100 | e | 00101 | d | 011100 | , | 001000011 | |
| 0001 | t | 01101 | s | 011101 | u | | |
| 0100 | a | 01111 | l | 0010001 | - | 0010000100 | |
| 0101 | o | 001001 | c | 0011000 | p | | k |
| 1010 | h | 001101 | w | 0011001 | b | 0010000101 | |
| 1011 | r | 001110 | g | 0110010 | m | | q |
| 00000 | n | 001111 | f | 0110011 | . | | |

Huffman: Summary

- Total for this example:
 - 4.1 bits per character
 - 6135 total bits
 - 51.7% the size of ASCII representation.
- Minimal for this type of code.

Other Codes

- error detecting: Know if something has been modified (bit flip).
- error correcting: Know *which* bit has been modified.
- multicharacter: Encode *sequences* (like “the”) with their own codes. Can get much closer to minimum possible code length: “Shannon’s entropy”.

What To Know

- construct a Huffman code from frequencies
- decode a message using a Huffman code
- encode a message using a Huffman code
- (Let's try some examples, as time permits.)

Next Time

- Hillis Chapter 8.