

Objectives

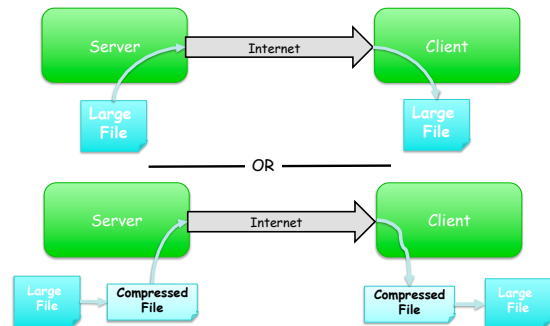
- Data Compression

Feb 19, 2010

CSCI211 - Sprenkle

1

Which Is Better?



Feb 19, 2010

CSCI211 - Sprenkle

2

Which Is Better?

- Depends on your metrics, compression time/amount
- Case 1 requires
 - More network resources
 - Less CPU time (server: compress; client: uncompress)
- Case 2 requires
 - Less network resources
 - More CPU time (client and server)
- Overall best
 - Depends on file size, network speed, compression time/amount
 - ➔ Bigger files → Case 2

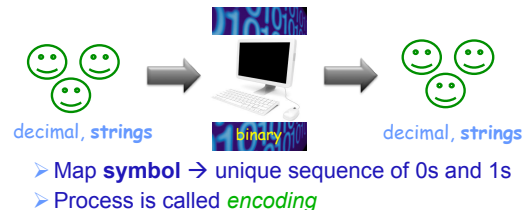
Feb 19, 2010

CSCI211 - Sprenkle

3

Problem: Encoding

- Computers use bits: 0s and 1s
- Need to represent what we (humans) know to what computers know



Feb 19, 2010

CSCI211 - Sprenkle

4

Problem: Encoding

- Let's say we want to encode characters using 0s and 1s
 - Lower case letters (26)
 - Space
 - Punctuation (, . ? ! ')

What is the **least** number of bits we would need to encode these characters?

Feb 19, 2010

CSCI211 - Sprenkle

5

Problem: Encoding Symbols

- 32 characters to encode
 - $\log_2(32) = 5$ bits
 - Can't use fewer bits
- Examples:
 - a → 00000
 - b → 00001
- Actual mapping from character to encoding doesn't matter
 - Easier if have a way to compare ...

Feb 19, 2010

CSCI211 - Sprenkle

6

For Long Strings of Characters...

- Do we need an average of 5 bits/character always?
- What if we could use *shorter encodings* for *frequently* used characters, like a, e, s, t?
- **Goal.** Optimal encoding that takes advantage of *nonuniformity* of letter frequencies
- A fundamental problem for **data compression**
 - Represent data as compactly as possible

Feb 19, 2010

CSCI211 - Sprenkle

7

Example: Morse Code

- Used for encoding messages over telegraph
- Example of *variable-length encoding*
- How are letters encoded?
- How are letters differentiated?

Feb 19, 2010

CSCI211 - Sprenkle

8

Example: Morse Code

- Used for encoding messages over telegraph
- Example of *variable-length encoding*
- How are letters encoded?
 - Dots, dashes
 - Most frequent letters use shorter sequences
 - e → dot; t → dash; a → dot-dash
- How are letters differentiated?
 - Spaces in between letters
 - Otherwise, ambiguous

Feb 19, 2010

CSCI211 - Sprenkle

9

Ambiguity in Morse Code

- Encoding:
 - e → dot; t → dash; a → dot-dash
- Example: *dot-dash-dot-dash* could correspond to

Feb 19, 2010

CSCI211 - Sprenkle

10

Ambiguity in Morse Code

- Encoding:
 - e → dot; t → dash; a → dot-dash
- Example: *dot-dash-dot-dash* could correspond to
 - etet
 - aa
 - eta
 - aet

What's the problem?

Feb 19, 2010

CSCI211 - Sprenkle

11

Problem

- *Ambiguity* caused by encoding of one character is a *prefix* of encoding of another

Feb 19, 2010

CSCI211 - Sprenkle

12

Prefix Codes

- **Problem:** Encoding of one character is a *prefix* of encoding of another
- **Solution: Prefix Codes:** map letters to bit strings such that *no encoding is a prefix of any other*
 - Won't need artificial devices like spaces to separate characters
- **Example encodings:**
 - Verify that no encoding is a prefix of another
 - What is 0010000011101?

a: 11	d: 10
b: 01	e: 000
c: 001	

Feb 19, 2010

CSCI211 - Sprenkle

13

Optimal Prefix Codes

- For typical English messages, this set of prefix codes is **not** the *optimal* set

a: 11	d: 10
b: 01	e: 000
c: 001	

- Why not?

Feb 19, 2010

CSCI211 - Sprenkle

14

Optimal Prefix Codes

- For typical English messages, this set of prefix codes is **not** the *optimal* set

a: 11	d: 10
b: 01	e: 000
c: 001	

- Why not?
 - 'e' is more commonly used than other letters and should therefore have a shorter encoding

Feb 19, 2010

CSCI211 - Sprenkle

15

Optimal Prefix Codes

- **Goal:** minimize **Average number of Bits per Letter (ABL):**

$$\sum_{x \in S} \text{frequency of } x * \text{length of encoding of } x$$

↑ For all characters in our alphabet
- f_x : frequency that letter x occurs
- $\gamma(x)$: encoding of x
 - $|\gamma(x)|$: length of encoding of x
- Minimize **ABL** = $\sum_{x \in S} f_x |\gamma(x)|$

Feb 19, 2010

CSCI211 - Sprenkle

16

Example: Calculating ABL

$f_a = .32$	a: 11
$f_b = .25$	b: 01
$f_c = .20$	c: 001
$f_d = .18$	d: 10
$f_e = .05$	e: 000

- **ABL** = $\sum_{x \in S} f_x |\gamma(x)| = ?$

Feb 19, 2010

CSCI211 - Sprenkle

handout

17

Example: Calculating ABL

$f_a = .32$	a: 11
$f_b = .25$	b: 01
$f_c = .20$	c: 001
$f_d = .18$	d: 10
$f_e = .05$	e: 000

- **ABL** = $\sum_{x \in S} f_x |\gamma(x)| = ?$
- = $.32 * 2 + .25 * 2 + .20 * 3 + .18 * 2 + .05 * 2$
- = 2.25

What about a fixed-length encoding?
Is it a prefix code? What is its ABL?

Feb 19, 2010

CSCI211 - Sprenkle

18

Fixed-Length Encodings

- Is it a prefix code?
 - Yes. Always look at fixed number of characters
- What is its ABL?
 - ABL is the length of the encoding
- For 5 characters, ABL is 3
- Variable-length prefix code's ABL (2.25) is an improvement

Feb 19, 2010

CSCI211 - Sprenkle

19

Can We Improve the ABL?

$f_a = .32$
 $f_b = .25$
 $f_c = .20$
 $f_d = .18$
 $f_e = .05$

a: 11
 b: 01
 c: 001
 d: 10
 e: 000

Feb 19, 2010

CSCI211 - Sprenkle

20

Can We Improve the ABL?

$f_a = .32$
 $f_b = .25$
 $f_c = .20$
 $f_d = .18$
 $f_e = .05$

a: 11
 b: 01
 c: 001
 d: 10
 e: 000

Swap these because
 c occurs more
 frequently than d .
 Give c the shorter
 encoding

- ABL** = $\sum_{x \in S} f_x |v(x)| = 2.23$

Feb 19, 2010

CSCI211 - Sprenkle

21

Problem Statement

- Given an alphabet and a set of frequencies for the letters, produce optimal (most efficient) prefix code
 - Minimizes average number of bits per letter

Feb 19, 2010

CSCI211 - Sprenkle

22

Approaches to Solution

- Brute force
 - Search space is complicated → all ways to map letters to bit strings that adhere to prefix code property
- Build towards greedy approach
 - Start: representing prefix codes

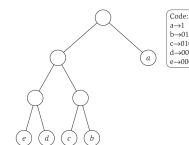
Feb 19, 2010

CSCI211 - Sprenkle

23

Binary Trees to Represent Prefix Codes

- Exposes structure better than list of mappings
 - Each leaf node is a letter
 - Follow path to the letter
 - Going left: 0
 - Going right: 1



Are these really prefix codes?
 How could we show they weren't?

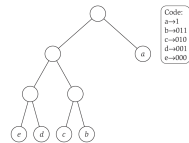
Feb 19, 2010

CSCI211 - Sprenkle

24

Binary Trees to Represent Prefix Codes

- **Proof.** If it weren't: a letter's encoding is a prefix of another letter
 - Letter is in the path of another letter
 - But, all letters are leaf nodes
 - Contradiction



Feb 19, 2010

CSCI211 - Sprenkle

25

Building the Binary Tree

- How do we build the binary tree for this mapping?
- **Tree Rules:**
 - Each leaf node is a letter
 - Follow path to the letter
 - Going left: 0
 - Going right: 1

Feb 19, 2010

CSCI211 - Sprenkle

26

Recursively Generate Tree

- All letters are in root node
- For all letters in node
 - If encoding begins with 0, letter belongs in left subtree
 - Otherwise (encoding begins with 1), letter belongs in right subtree
 - If last bit of encoding, make the letter a leaf node of that subtree
 - Shift encoding one bit
 - Process left and right children

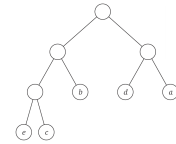
Feb 19, 2010

CSCI211 - Sprenkle

27

Tree Properties

- What is the length of a letter's encoding?
- Define our optimal goal in tree terms



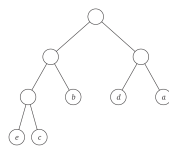
Feb 19, 2010

CSCI211 - Sprenkle

28

Tree Properties

- What is the length of a letter's encoding?
 - Length of path from root to leaf → its *depth*
- Define our optimal goal in tree terms
 - $ABL = \sum_{x \in S} f_x |y(x)| = \sum_{x \in S} f_x \text{depth}(x)$



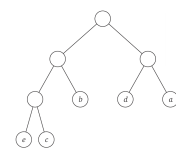
Feb 19, 2010

CSCI211 - Sprenkle

29

Tree Properties

- What do we want our tree to look like for the optimal solution?
 - How many leaves?
 - How many internal nodes?
 - Think about parent nodes vs. child nodes
 - When uniform frequencies?
 - Nonuniform frequencies?



Feb 19, 2010

CSCI211 - Sprenkle

30

Tree Properties

- **Claim.** The binary tree corresponding to the optimal prefix code is *full*, i.e., each internal node has two children.
- **Proof?**

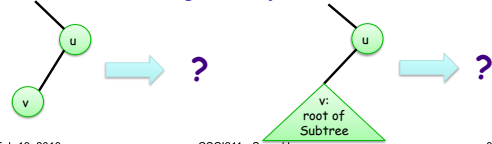
Feb 19, 2010

CSCI211 - Sprenkle

31

Tree Properties

- **Claim.** The binary tree T corresponding to the optimal prefix code is *full*, i.e., each internal node has two children.
- **Proof.** Assume that T has an internal node with only one child
 - Without loss of generality, assume left child



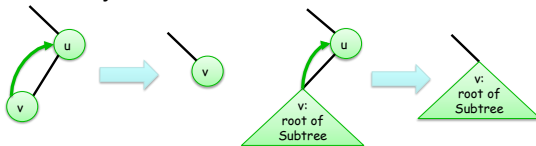
Feb 19, 2010

CSCI211 - Sprenkle

32

Tree Properties

- **Claim.** The binary tree T corresponding to the optimal prefix code is *full*, i.e., each internal node has two children.
- **Proof.** Assume that T has an internal node with only one child



Replace u with $v \rightarrow$ decrease depth \rightarrow original wasn't optimal

Toward a Solution...

- Two problems to solve:
 - Creating the prefix code tree
 - Labeling the prefix code tree with alphabet/frequencies

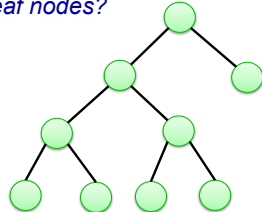
Feb 19, 2010

CSCI211 - Sprenkle

34

Simplifying: Know Optimal Prefix Code

- **Process:** assume knowledge of optimal solution to gain insight into finding solution
- Assume we knew the tree structure of the optimal prefix code, *how would you label the leaf nodes?*



Feb 19, 2010

CSCI211 - Sprenkle

35

Combining Our Conclusions

- The binary tree corresponding to the optimal prefix code is *full*, i.e., each internal node has two children
- We want to label the leaf nodes of the binary tree corresponding to the optimal prefix code such that nodes with *greatest depth* have *least frequency*

What does this mean the bottom of our tree looks like?

Feb 19, 2010

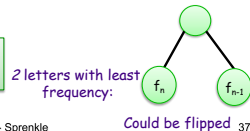
CSCI211 - Sprenkle

36

Combining Our Conclusions

- The binary tree corresponding to the optimal prefix code is *full*, i.e., each internal node has two children
- We want to label the leaf nodes of the binary tree corresponding to the optimal prefix code such that nodes with *greatest depth* have *least frequency*

What does this mean the bottom of our tree looks like?



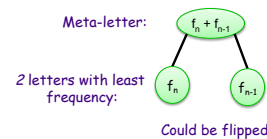
Feb 19, 2010

CSCI211 - Sprenkle

37

How Can We Use This?

- Two letters with least frequency are definitely going to be siblings
 - Tie them together
 - Their parent is a "meta-letter"
 - Frequency is sum of $f_n + f_{n-1}$



Feb 19, 2010

CSCI211 - Sprenkle

38

Constructing an Optimal Prefix Code

Huffman's Algorithm:

To construct a prefix code for an alphabet S with given frequencies:

```

if S has two letters:
    Encode one letter as 0 and the other letter as 1
else:
    Let y* and z* be the two lowest-frequency letters
    Form a new alphabet S' by deleting y* and z* and replacing
    them with a new letter w of freq  $f_{y^*} + f_{z^*}$ 
    Recursively construct a prefix code  $y'$  for S' with tree T'
    Define a prefix code for S as follows:
    Start with T'
    Take the leaf labeled w and add two children below it
    labeled y* and z*
  
```

Feb 19, 2010

CSCI211 - Sprenkle

39

Alternative Description

- Create a leaf node for each symbol, labeled by its frequency, and add to a queue
- While there is more than one node in the queue
 - Remove the two nodes of lowest frequency
 - Create a new internal node with these two nodes as children and with frequency equal to the sum of the two nodes' probabilities
 - Add the new node to the queue
- The remaining node is the tree's root node

Feb 19, 2010

CSCI211 - Sprenkle

40

Creating the Optimal Prefix Code

$f_a = .32$
 $f_b = .25$
 $f_c = .20$
 $f_d = .18$
 $f_e = .05$

Feb 19, 2010

CSCI211 - Sprenkle

41

Creating the Optimal Prefix Code

$f_a = .32$
 $f_b = .25$
 $f_c = .20$
 $f_d = .18$
 $f_e = .05$

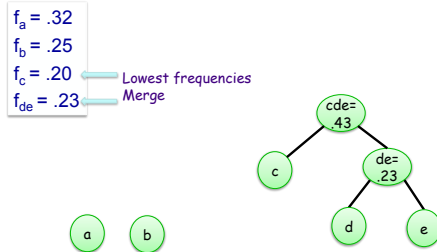


Feb 19, 2010

CSCI211 - Sprenkle

42

Creating the Optimal Prefix Code

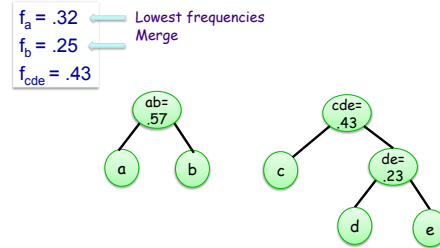


Feb 19, 2010

CSCI211 - Sprenkle

43

Creating the Optimal Prefix Code

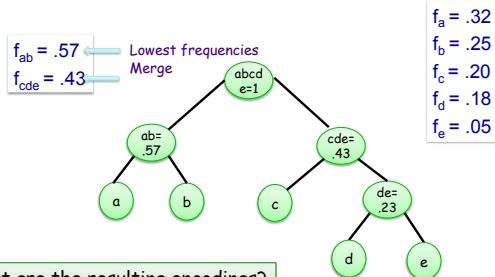


Feb 19, 2010

CSCI211 - Sprenkle

44

Creating the Optimal Prefix Code



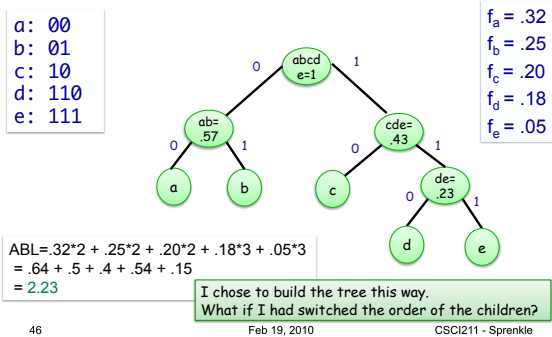
What are the resulting encodings?
What is the ABL?

Feb 19, 2010

CSCI211 - Sprenkle

45

Creating the Optimal Prefix Code



46

Feb 19, 2010

CSCI211 - Sprenkle

Exam 1 Feedback

- Can't say much because only 3 exams
- Common issues
 - Incorrect algorithms (a test case that doesn't work)
 - Not being specific about how something is represented
 - E.g., list, adjacency matrix
 - Missing the complexity of some step
 - Could be optimized

Feb 19, 2010

CSCI211 - Sprenkle

47

Assignments

- Continue reading Chapter 4
- Start reading Chapter 5
- PS5 due Friday after break

Feb 19, 2010

CSCI211 - Sprenkle

48