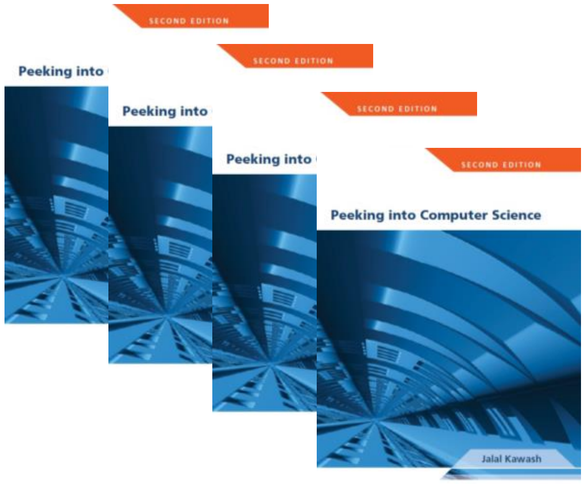
Trees & Information Coding: 3

Peeking into Computer Science

- Mandatory: Chapter 3 – Section 3.5

Reading Assignment



Information Coding

An application of trees

3

At the end of this section, the student will be able to:

1. Understand the need for variable-length coding (VLC)
2. Understand how compression works
3. Calculate the space savings with VLC
4. Understand decoding problems with VLC
5. Define non-prefix VLC
6. Use Huffman's algorithm and binary trees to assign optimized non-prefix VLC
7. Represent BL trees as nested lists

Objectives

Peeking into Computer Science

Back to Coding (JT: Review)

- Assume we have a file that contains data composed of 6 symbols only:
- A, I, C, D, E, and S (for space)

ACE DICE AIDE CAID
EAD DAICED …

Back to Coding

- Assume we have a file that contains data composed of 6 symbols only:
- A, I, C, D, E, and S (for space)

ACESDICESAIDESCAID
EADSDAICED …

- If the file has 1000 characters, how many bits (0s and 1s) are needed to code the file?

Coding

- The first question is
- How many symbols do we need to represent each character?
- The objective is to keep the size of the file as small as possible
- We have 6 characters (messages) and two symbols (0 and 1)
- 2 is not enough, since 2^2 is 4

Coding (JT: Review)

2 bits are not enough (JT: Review)

- 00 for A
- 01 for S
- 10 for I
- 11 for E

- We cannot represent the rest C and D

- 3 works, since 2^3 is 8, so we can represent up to 8 characters and we only have 6

3 bits are more than enough (JT: Review)

- Say
- 000 for A
- 001 for S
- 010 for I
- 011 for E
- 100 for C
- 101 for D
- 110 not used
- 111 not used

Coding (JT: Review)

- If the file has 1000 characters, how many bits (0s and 1s) are needed to code the file?
- Each character needs 3 bits
- Hence, we need 3x1000 = 3000 bits

Compression

- How does file compression work?
- Huffman's codes reduce the size of a file by using variable-length codes for characters

Compression

- Analyzing the file we find that
- 35% are S
- 28% of the characters are A
- 20% are E
- 7% are I
- 6% are C
- 4% are D
- Some characters are more frequent than others

Peeking into Computer Science © Jalal Kawash 2010 13

Variable-Length Codes

- To reduce the size of the file, we can use shorter codes for frequent characters
  - We can use longer codes for infrequent characters
- 35% are S (use 2 bits for S)
- 28% are A (use 2 bits for A)
- 20% are E (use 2 bits for E)
- 7% are I (use 3 bits for I)
- 6% are C (use 4 bits for C)
- 4% are D (use 4 bits for D)

Peeking into Computer Science © Jalal Kawash 2010 14

- If we use this coding, what is the size of the file?
- 350 S's (35% of 1000) require **700** bits (2 bits for each S)
- 200 E's require **400** bits (2 bits)
- 280 A's require **560** bits (2 bits)
- 70 I's require **210** bits (3 bits)
- 60 C's require **240** bits (4 bits)
- 40 D's require **160** bits (4 bits)

- Total is **2270** bits
- Recall that with fixed codes, the size is **3000** bits
- Compressed file size is about **76%** of original size

Compressed File Size

- Not any variable-length code works
- Assume A's code is 0
- C's code is 1
- E's code is 01

- The code 0101 could correspond to ACE, EAC, ACAC, or EE

Problems with Variable-Length Codes

Prefix Codes

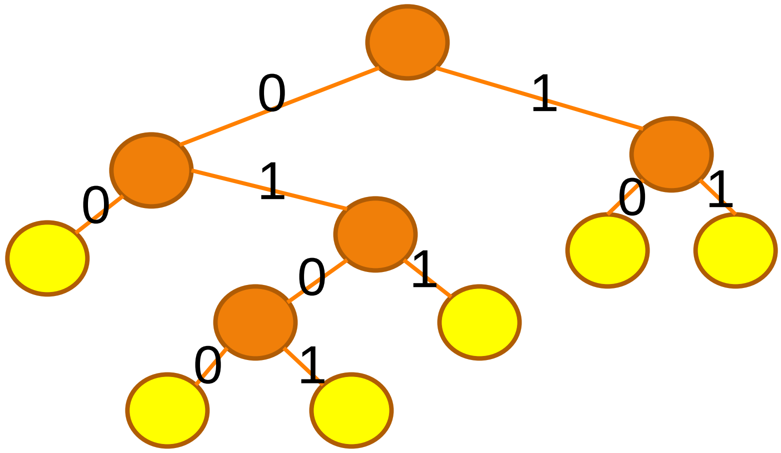
- Codes that work must have the property:
- No code can be the prefix of another code
- **Called Non-Prefix Codes**
- **0** is a prefix of **01**, this is why our coding failed
- Another example: **0101** is a prefix of **010111**

Peeking into Computer Science © Jalal Kawash 2010 17

Non-Prefix Codes

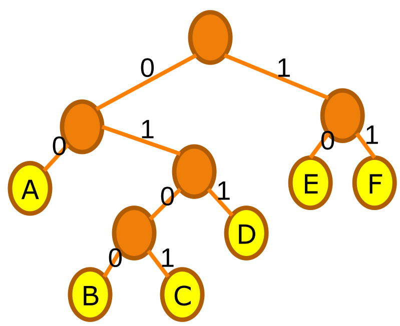
- Non-Prefix codes can be generated using a binary tree
- Start from a binary tree
- Label edges to left children with 0
- Label edges to right children with 1
- Record the labels on the path from the root to the leaves
- Each path corresponds to a non-prefix code

Peeking into Computer Science © Jalal Kawash 2010 18



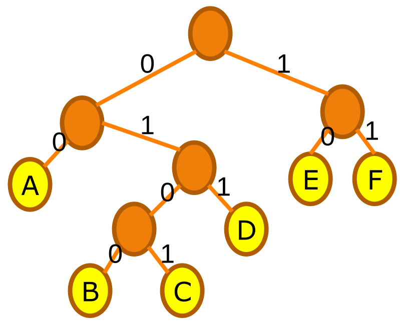
Non-Prefix Codes from a Binary Tree

- A's code: 00
- B's 0100
- C's 0101
- D's 011
- E's 10
- F's 11



Non-Prefix Codes from a Binary Tree

- 00
- 0100
- 0101
- 011
- 10
- 11



No code is a prefix of another

- A:00, B:0100, C:0101, D:011, E:10, F:11
- 0100 | 00 | 011

  B A D
- No other interpretation
  - Parsing left to right
  - (JT's extra: parsing refers to 'reading' or 'breaking into meaningful portions')

No Confusion

No Confusion

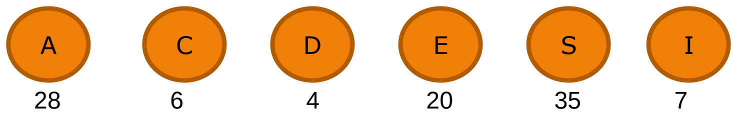
- A:00, B:0100, C:0101, D:011, E:10, F:11
- 11 10 10 011
  F E E D

- So How do we generate such codes?

Huffman's Coding

- Build a binary tree
- The characters in a file are the leaves
- The most frequent characters should be closer to the root, generating shorter codes
- Assign the codes, based on this tree

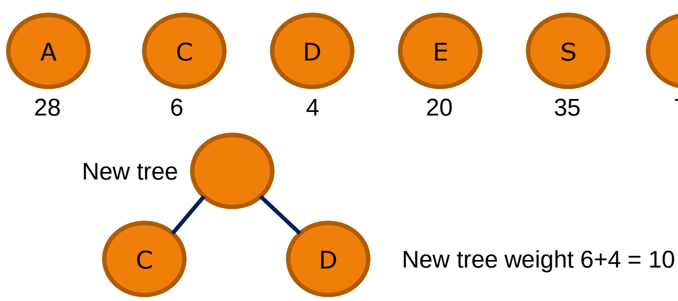
1. Assign to each symbol its weight (frequency)



Each of these is a tree of size one!

Huffman's Coding – Step 1

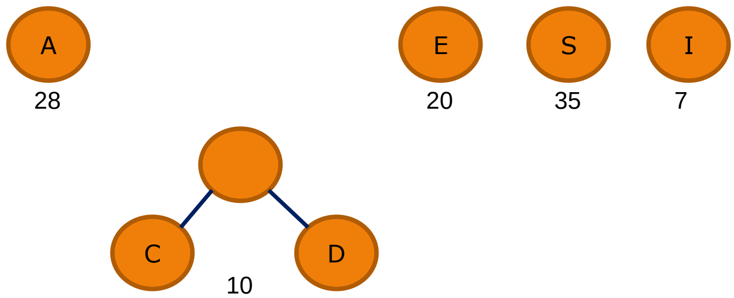
2. Choose two trees that have the minimum weights
   - Replace these two trees with a new tree with new root
   - Make the tree with the smaller weight a right child
   - The weight of the new tree is the sum of old weights (JT: next slide)



Huffman's Coding – Step 2

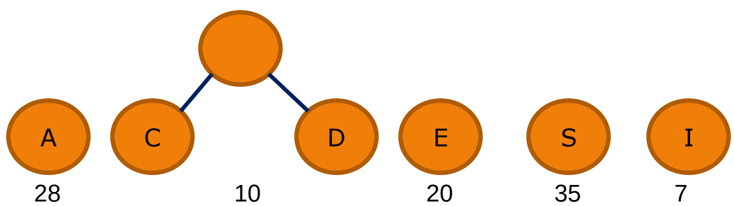
Huffman's Coding – Step 2

2. Choose two trees that have the minimum weights
   - Replace these two trees with a new tree with new root
   - Make the tree with the smaller weight a right child
   - The weight of the new tree is the sum of old weights



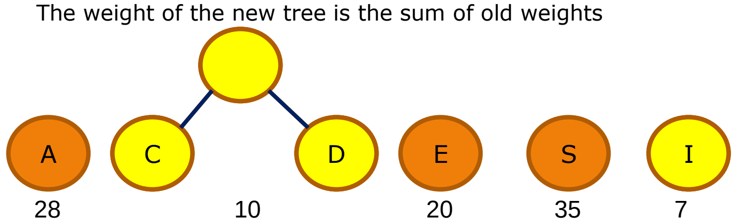
Huffman's Coding – Repeat

- Repeat Step 2 until we have a single tree



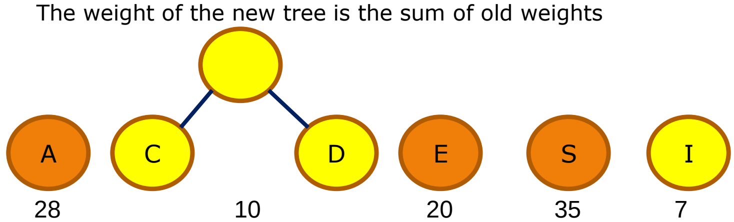
Step 2 Again

2. Choose two trees that have the minimum weights
   - Replace these two trees with a new tree with new root
   - Make the tree with the smaller weight a right child
   - The weight of the new tree is the sum of old weights

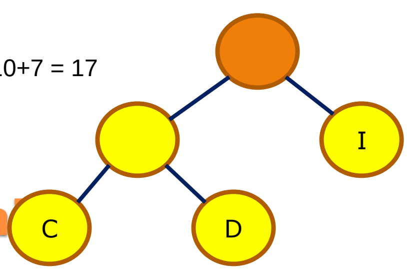


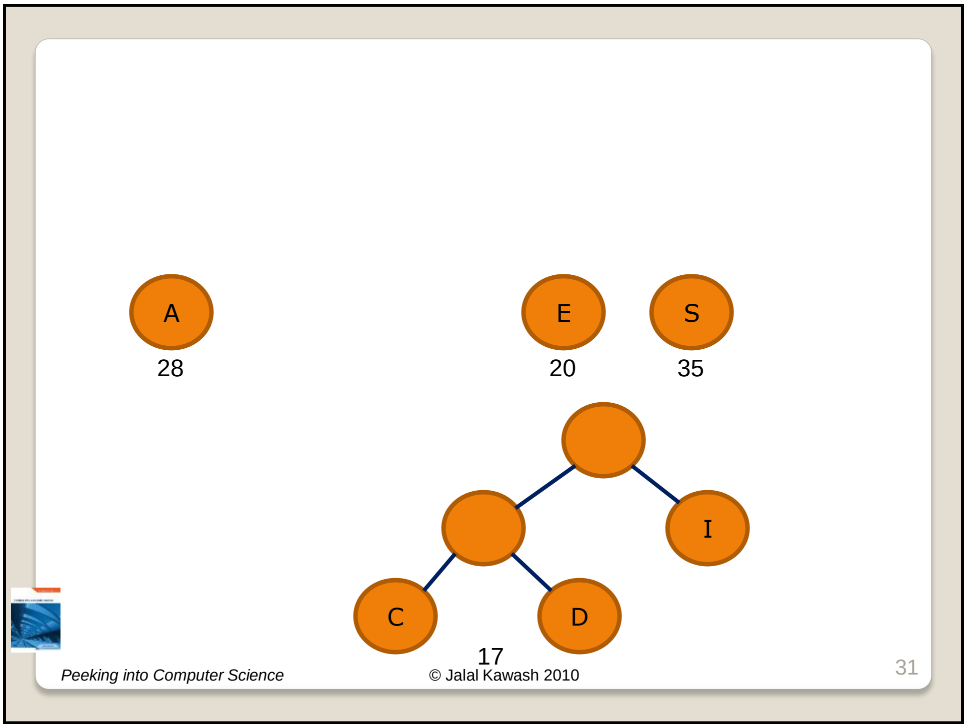
Step 2 Again

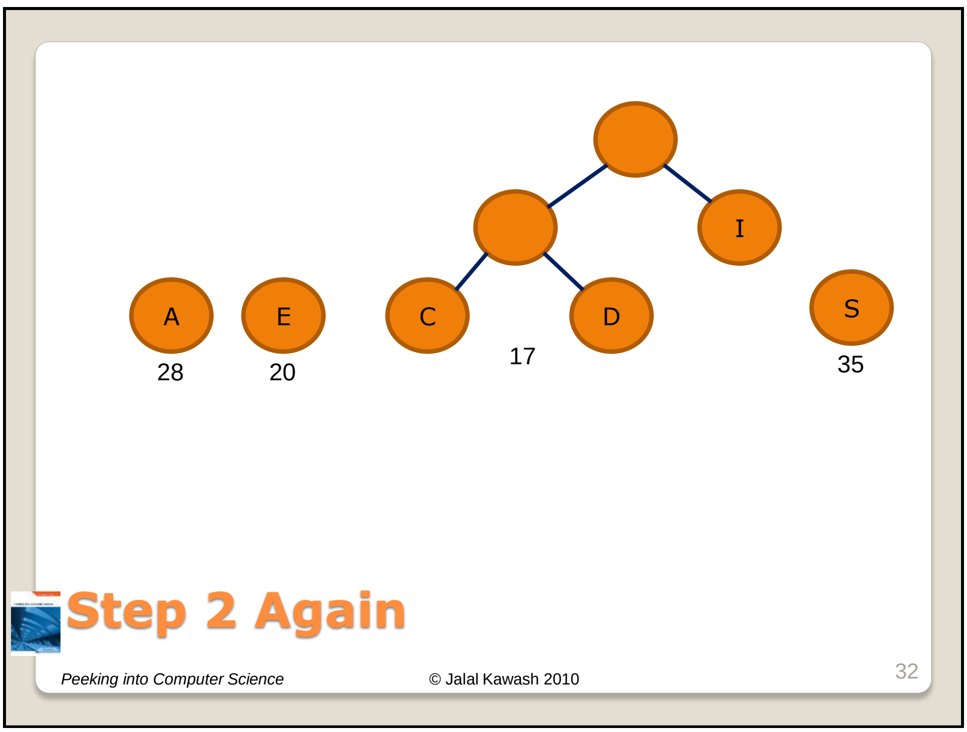
2. Choose two trees that have the minimum weights
   - Replace these two trees with a new tree with new root
   - Make the tree with the smaller weight a right child
   - The weight of the new tree is the sum of old weights

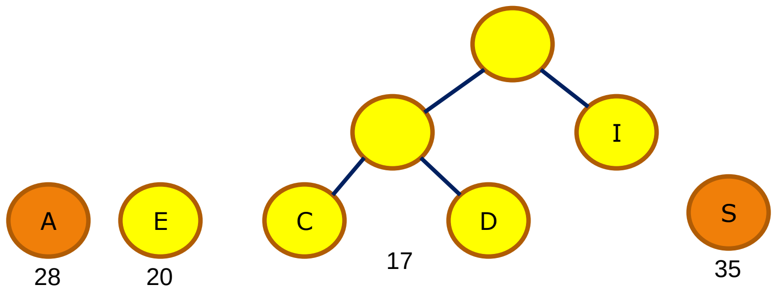


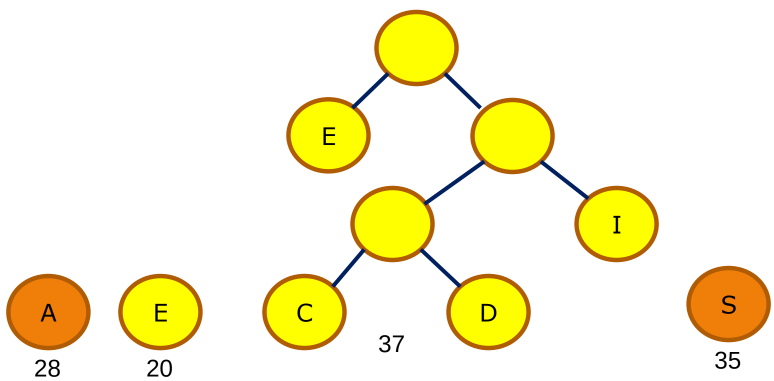
New tree weight 10+7 = 17



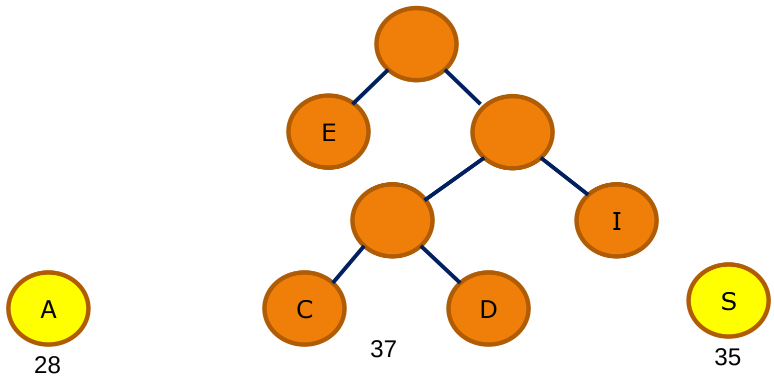

A
28
E
20
S
35
I
C
D
17
Peeking into Computer Science
© Jalal Kawash 2010
31


I
A
E
C
D
S
28
20
17
35
Step 2 Again
Peeking into Computer Science
© Jalal Kawash 2010
32

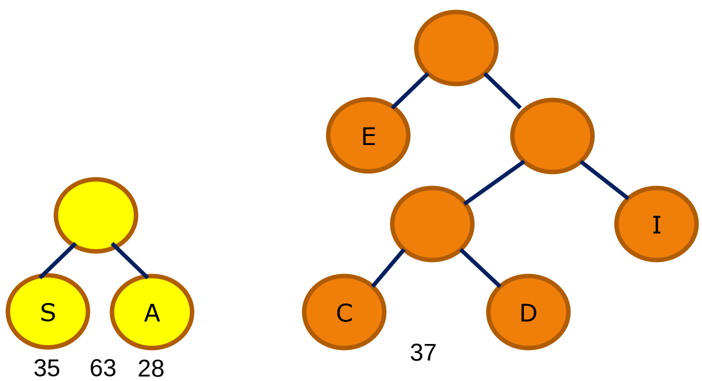




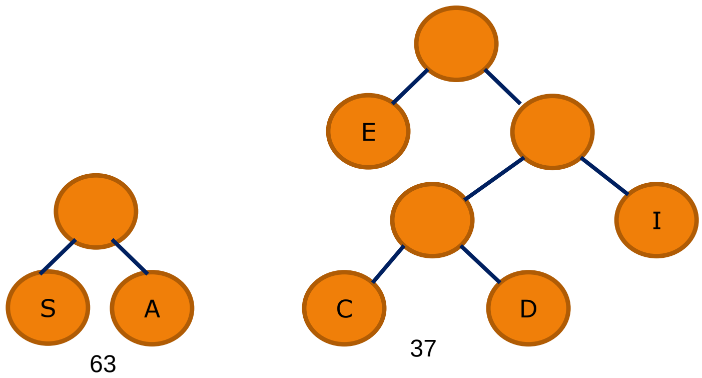
Weight is 20+17 = 37

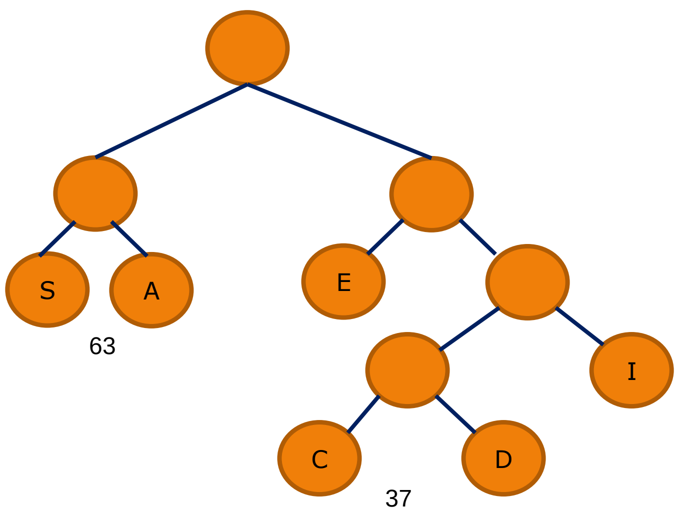


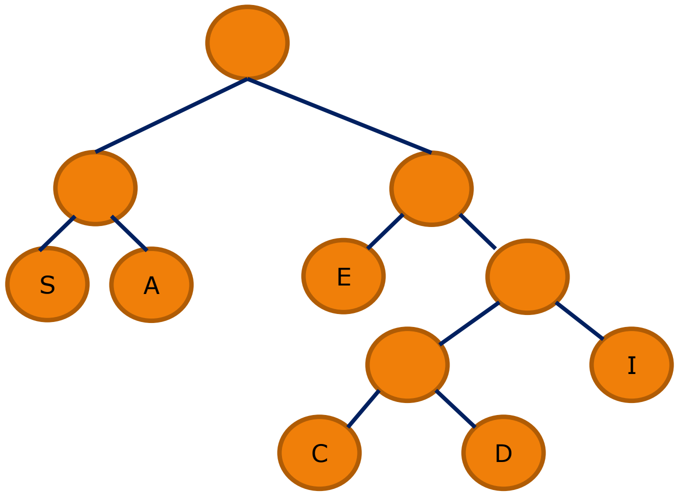
Weight is 20+17 = 37



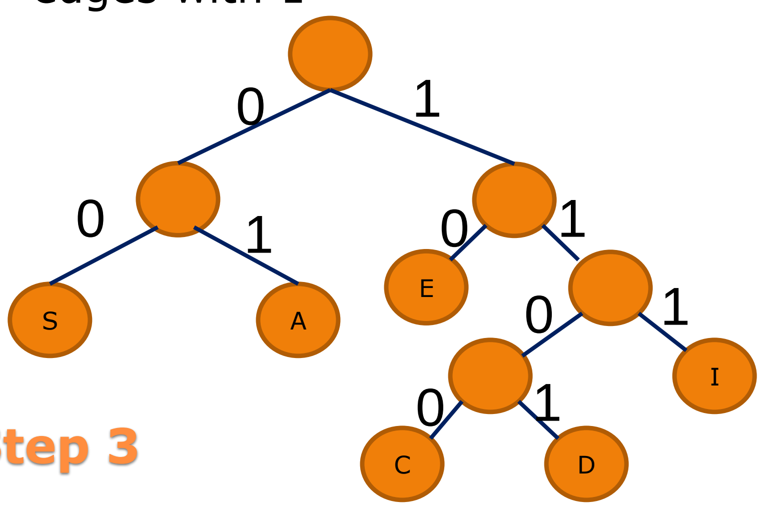
Weight is 35+28 = 63


E
I
S
A
C
D
63
37

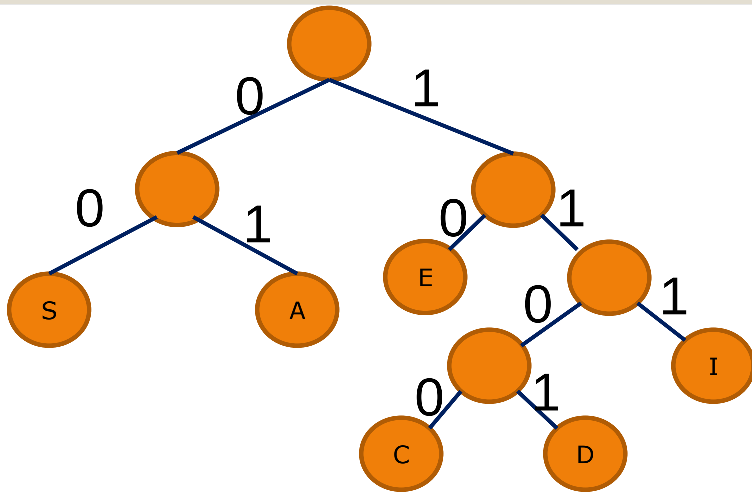

S
A
E
I
63
C
D
37



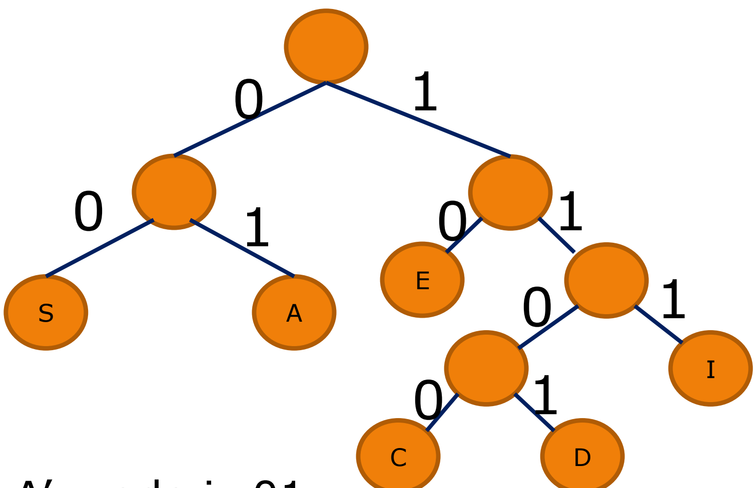
3. Label left edges with 0 and right edges with 1



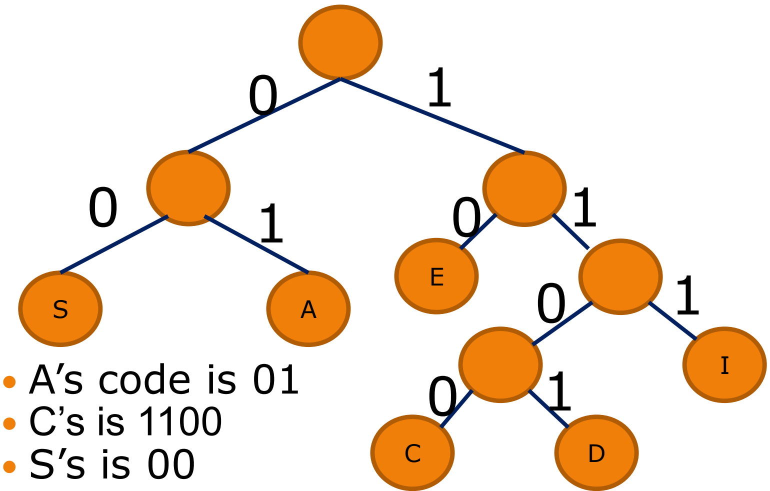
Step 3



- Read the labels on the path from the root to each leaf, this is its code



- A's code is 01



- A's code is 01
- C's is 1100
- S's is 00
- E's is 10
- D's is1101
- I's is 111

- 35% are S
- 28% are A
- 20% are E
- 7% are I
- 6% are C
- 4% are D

More Frequent Characters have shorter codes

Recall This Analysis?

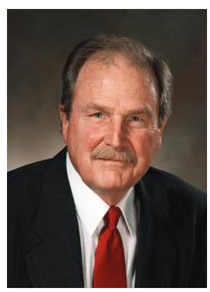
- If we use this coding, what is the size of the file?
- 350 S's (35% of 1000) require **700** bits (2 bits for each S)
- 280 A's require **560** bits
- 200 E's require **400** bits
- 70 I's require **210** bits
- 60 C's require **240** bits
- 40 D's require **160** bits

- Total is **2270** bits
- Recall that with fixed codes, the size is **3000** bits
- Compressed file size is about **76%** of original size

Peeking into Computer Science © Jalal Kawash 2010 45

David Huffman

- 1925-1999
- US Electrical Engineer
- Contributions in coding theory, signal design for radar and communications, and logic circuits
- He wrote his coding algorithm as a graduate students at MIT



Peeking into Computer Science © Jalal Kawash 2010 46

Concluding Notes

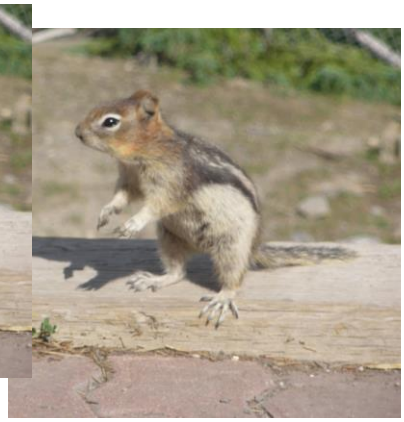
- Even though coding gave C and D codes of length 4 (compared to 3 in fixed coding), it was beneficial
- No code generated by Huffman’s method can be a prefix of another code
- Many compression tools use a combination of different coding methods, Huffman’s is among them

Peeking into Computer Science © Jalal Kawash 2010 47

JT’s Extra: Uses Of Compression

- Image and video files

Hi-res: JPEG 322 KB



Hi-res: TIF 14.1 MB

Images: Courtesy of James Tam

Peeking into Computer Science © Jalal Kawash 2010