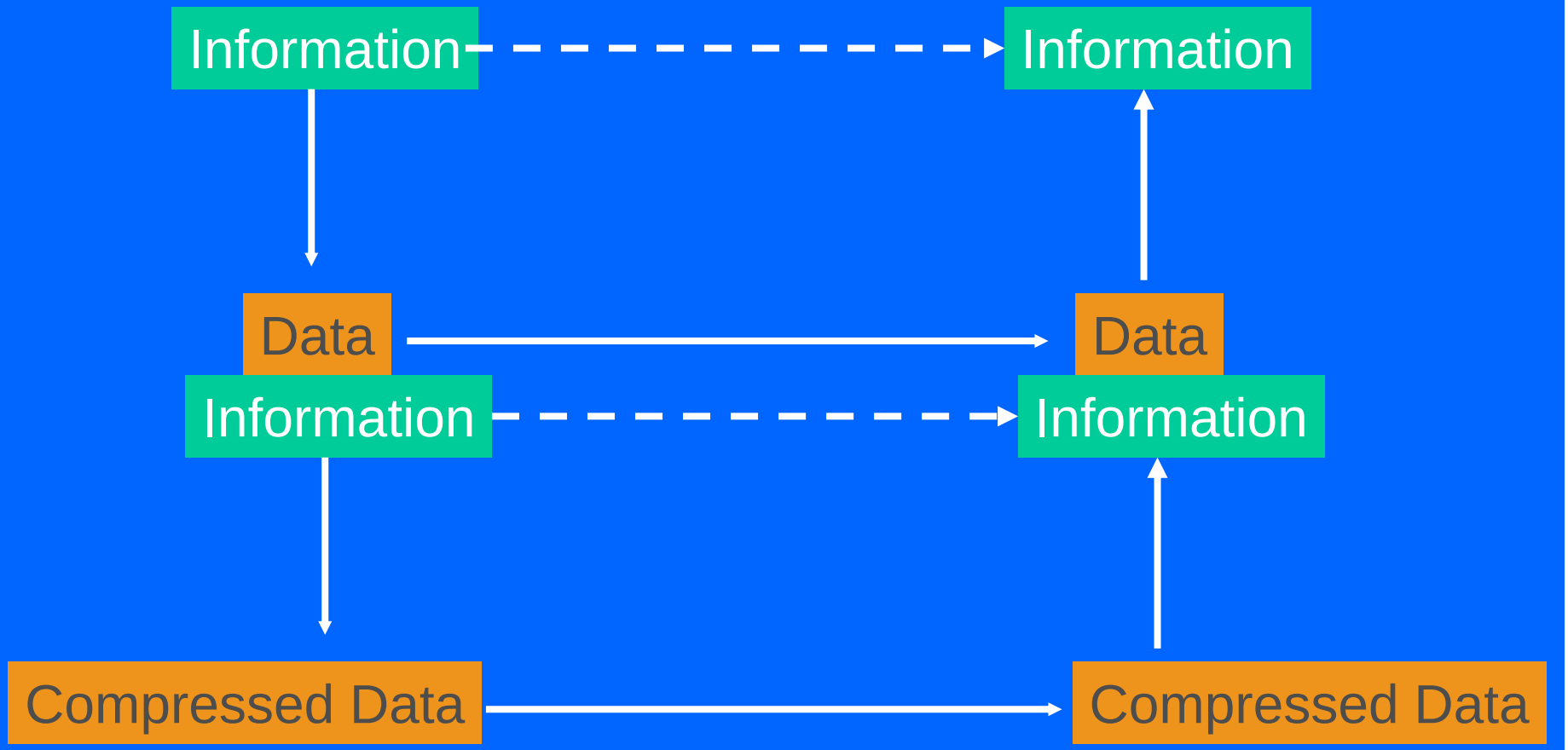
Compression

Learning Objectives

- Understand compression as an example of representation
- Understand the basic techniques of lossless compression
  - Encode frequent symbols with few bits
  - Encode frequent symbol sequences
- Understand the idea of lossy compression

Compression as Representation



Compression

- Three basic techniques
  - Encode high probability symbols with fewer bits
    - Shannon-Fano, Huffman, UNIX compact
  - Encode sequences of symbols with location of sequence in a dictionary
    - PKZIP, ARC, GIF, UNIX compress, V.42bis
  - Lossy compression
    - JPEG and MPEG
- Further Reading
  - The Data Compression Book, by Mark Nelson, 1992 M&T books

Technique 1:
Variable Length Bit Coding

- Suppose ‘A’ appears 50 times in text, but ‘B’ appears only 10 times
- ASCII coding assigns 8 bits per character, so total bits for ‘A’ and ‘B’ is 60 * 8 = 480
- If ‘A’ gets a 4-bit code and ‘B’ gets a 12-bit code, total is 50 * 4 + 10 * 12 = 320
  - Use same scheme be used for decoding as coding!
- Why not assign 4-bit codes to all letters?
- No code is a prefix of another
  - **For example, can’t have ‘A’ map to 10 and ‘B’ map to 100, because 10 is a prefix (the start of) 100.**
  - **Enables left-to-right, unambiguous decoding**
  - **That is, if you see 10, you know it’s ‘A’, not the start of another character.**

Huffman Coding Example

- Character (or symbol) frequencies
  - A: 20% (.20)
    - **e.g., ‘A’ occurs 20 times in a 100 character document, 1000 times in a 5000 character document, etc.**
  - B: 9% (.09)
  - C: 15%
  - D: 11%
  - E: 40%
  - F: 5%
- Also works if you use **character counts**
- Must know frequency of every characters in the document

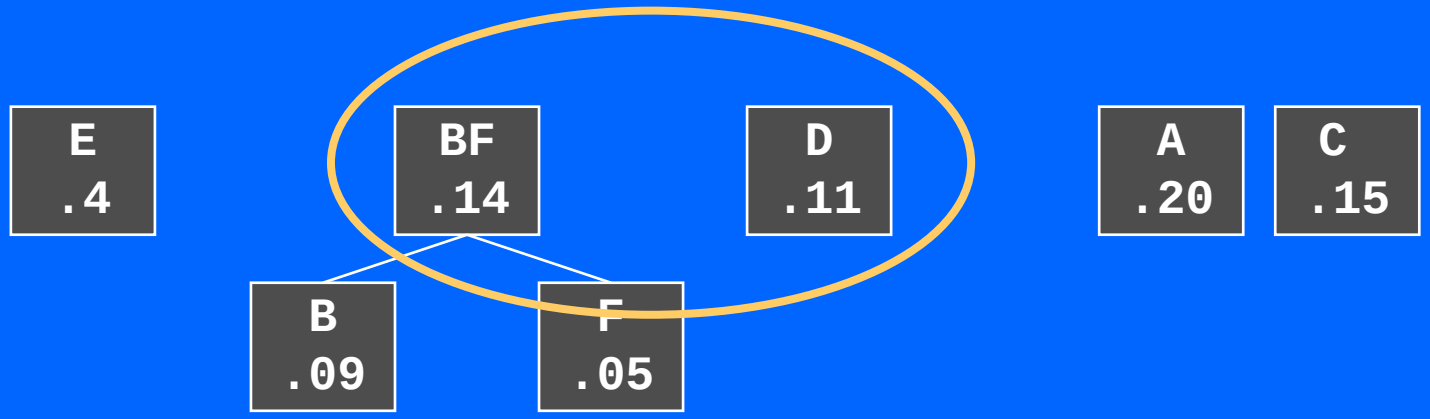
Huffman Coding Example

- Here are the symbols and their associated frequencies.
- Now we combine the two least common symbols (those with the smallest frequencies) to make a new symbol string and corresponding frequency.

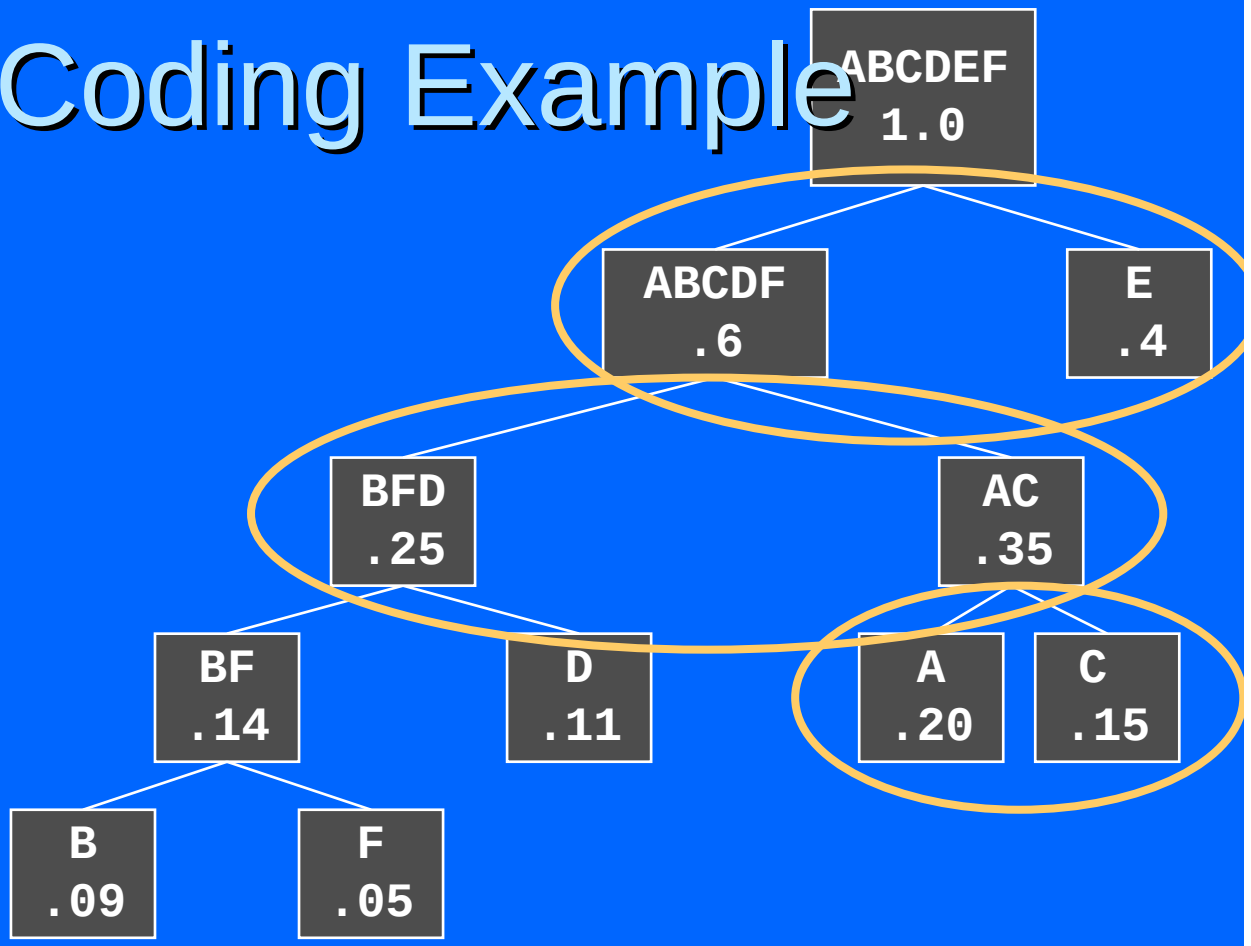
  - What are the two least common symbols?
  - What do we get when we combine the symbols? The frequencies?

Huffman Coding Example

- Here's the result of combining symbols once.
- Now repeat until you've combined all the symbols into a single string.
- Take 2 minutes to do this on your own, then pair up and discuss.

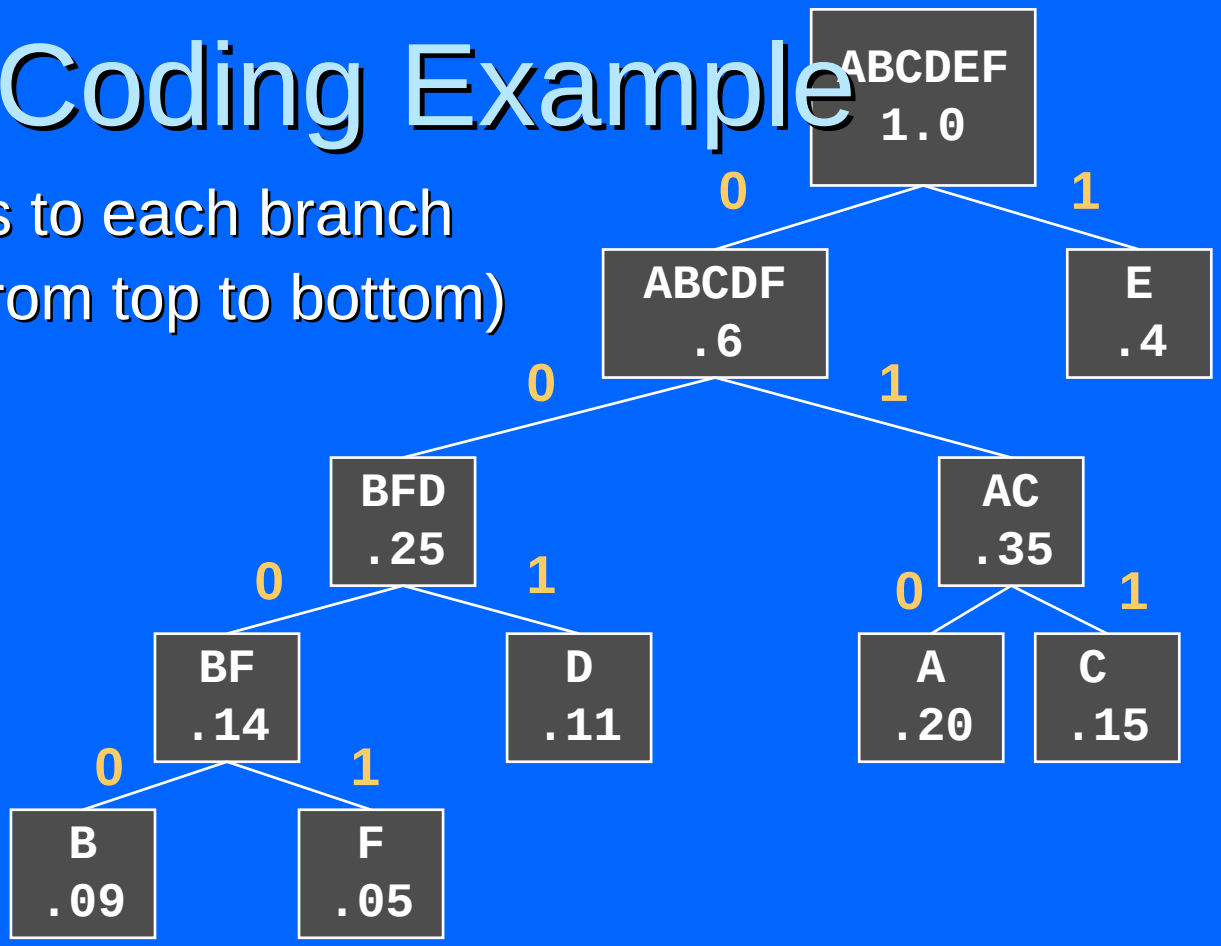


Huffman Coding Example



Huffman Coding Example

- Now assign **0**s/**1**s to each branch
- Codes (reading from top to bottom)
  - A: 010
  - B: 0000
  - C: 011
  - D: 001
  - E: 1
  - F: 0001



- Note
  - None are prefixes of another
- Decode this.
  - 0100111100010000
  - What's the first character? The second?

Try decoding right to left

Context-Sensitive Coding

- ‘u’ is unlikely
  - normally it’s code should have many bits
- After a ‘q’ it’s very likely
  - its code should have few bits
- Decompression has to take account of context in same way!
- See for yourself: encode ‘aqua’ and ‘auqa’
  - How many bits long is each?

Note that Huffman is not context sensitive

aqua is 22 bits long; auqa is 29 bits

State 1

Symbol	Code	Next state
‘a’	010	1
‘b’	0110	2
…		
‘q’	0111000111000	7
‘u’	1000100001	1

State 7

Symbol	Code	Next state
‘a’	1010	1
‘b’	01101	1
…		
‘q’	0111000111000	7
‘u’	010	1

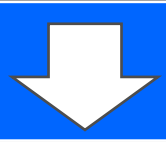
Technique 2:
Dictionary Based Coding

- Keep a dictionary of common words and phrases
- Translate symbol string in input to a location in the dictionary
- Example: “fantastic” appears on p. 327, line 13
  - How many bits in ASCII representation?
    - Remember, ASCII uses 8 bits/character
  - How many in (327, 13)?
    - How many bits do you need to represent 327 as a binary number? 13?

Sliding Window Dictionary

- Even more clever: use previous characters in document as the dictionary!
  - Called a sliding window.
- 1 bit indicates ASCII or dictionary index follows
  - Like the parenthesis is used in the example below
- 4 bits: starting how many characters back (0..15)
- 4 bits: length of dictionary string (0..15)

```
Even though seven bought enough socks, some others...
```



```
Even though se(13,4)b(12,4)t (10,2)(8,4) socks,
(7,2)me others...
```

Technique 3:
Lossy Compression

- Can’t exactly regenerate original bit string
- But close enough for intended use
- Commonly used for images
  - How many different shades of blue are there in a photograph of the blue sky?
  - How many different shades of blue do you notice in the photograph?

Applying The Techniques to Image Compression

- Variable Length Coding
  - E.g, blue with fewer bits than yellow (assuming there's more purple than yellow – like on this slide)
- Dictionary Techniques
  - E.g., repeating image fragments represented with reference to location of first occurrence
- Lossy compression
  - E.g., group sets of four pixels and represent each by code for the "average" color

Image fragment on left could be repeated several times on this screen

Applying The Techniques to Video Compression

- Variable Length Coding
  - As for still images
- Dictionary techniques

  E.g., short bit strings encode parts of the frame that haven't changed since previous frame
- Lossy compression

  E.g., Omit frames

Application to Sound Compression

- Variable Length Coding
- Dictionary techniques
- Lossy compression