ICS 2303
Multimedia Systems

Chapter 3.1

Text Compression

Introduction

- The different types of texts (formatted, unformatted, and hypertext) are all represented as strings of characters selected from a defined set.

Introduction

- The strings comprise of alphanumeric characters, which are interspersed with additional control characters.
- The different types of text use and interpret the latter in different ways.

Introduction

- Any compression associated with text must be Lossless since the loss of just a single character could modify the meaning of a complete string.

Statistical Encoding

- Generally, compression is restricted to use entropy encoding and in practice, statistical encoding methods.
- Many applications use a set of codewords to transmit the source information.

Statistical Encoding

- E.g. a set of ASCII codewords are often used for the transmission of strings of characters.
- Normally all the codewords in the set comprise a fixed number of binary bits, for example 7 or 8 bits in the case of ASCII.

Statistical Encoding

- However, in many applications, the symbols and hence codewords that are present in the source information do not occur with the same frequency of occurrence i.e. with equal probability.

Statistical Encoding

- E.g. in a string of text, the character A may occur more frequently than, say, the character P which occurs more frequently than the character Z.

Statistical Encoding

- Statistical encoding exploits this property by using a set of variable-length codewords with the shortest codewords used to represent the most frequently occurring symbols.
- In practice, the use of variable-length codewords is not quite as straightforward.

Statistical Encoding

- As with run-length encoding, the destination must know the set of codewords being used by the source.

Statistical Encoding

- However, with variable-length codewords, in order for decoding operation to be carried out correctly, it is necessary to ensure that a shorter codeword in the set does not form the start of a longer codeword.

Statistical Encoding

- Otherwise the decoder will interpret the string on the wrong codeword boundaries.

Statistical Encoding

- A codeword set that avoids this happening is said to possess the prefix property and an example of an encoding scheme that generates codewords that have this property is the Huffman encoding algorithm.

Types of coding used for text.

- There are two types:
  - Static
  - Adaptive or dynamic coding.

Static Coding

- The is intended for applications in which the text to be compressed has known characteristics in terms of:
  - The characters used and
  - Their relative frequencies of occurrence.

Static Coding

- Using this information, instead of using fixed-length codewords, an optimum set of variable-length codewords is derived with the shortest codewords used to represent the most frequently occurring characters.

Static Coding

- The resulting set of codewords is then used for all subsequent transfers involving this particular type of text.

Dynamic or Adaptive coding

- This type is intended for more general applications in which the type of text being transferred may vary from one transfer to another.

Dynamic or Adaptive coding

- In this case, the optimum set of codewords is also likely to vary from one transfer to another.
- To allow for this possibility, the codeword set used to transfer a particular text string is derived as the transfer takes place.

Dynamic or Adaptive coding

- This is done by building up knowledge of both the characters that are present in the text and their relative frequency of occurrence dynamically as the characters are being transmitted.

Adaptive or Dynamic Coding

- Hence, the codewords used change as a transfer takes place, but in such a way that the receiver is able to dynamically compute the same set of codewords that are being used at each point during a transfer.

Huffman Coding

- Proposed by Dr. David A. Huffman in 1952
  - *"A Method for the Construction of Minimum Redundancy Codes"*
- Applicable to many forms of data transmission e.g text files.

Huffman Coding

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency and yet all characters are allocated the same amount of space1 char = 1 byte, be it e or x

Huffman Coding

- Any savings can be realized in tailoring codes to frequency of character.
- Codeword lengths are no longer fixed like ASCII.
- Codeword lengths vary and will be shorter for the more frequently used characters.

Static Huffman Coding

- The character string to be transmitted is first analyzed and the character types and their relative frequency determined.
- The coding operation involves creating an unbalanced tree with some branches (and hence codewords) shorter than others.

Static Huffman Coding

- The degree of imbalance is a function of the relative frequency of occurrence of the characters; the larger the spread, the more unbalanced is the tree.
- The resulting tree is called the Huffman code tree.

Static Huffman code tree

- A Huffman code tree is a binary tree with branches assigned the value 0 or 1.
- The base of the tree, normally the geometric top in practice, is called the *root node* and the point at which the branch divides is called a *branch node*.

Static Huffman code tree

- The termination point of a branch is called a *leaf node* to which the symbols being encoded are assigned.
- As each branch divides, a binary value of 0 is assigned to the left branch and a binary value of 1 for the right branch.

Static Huffman Code Tree

- The codewords used for each character (shown in the leaf nodes) are determined by tracing the path from the root node out to each leaf and forming a string of the binary values associated with each branch traced.

Static Huffman Coding Algorithm

- Initialization:
  - Put all symbols on the list sorted according to their frequency counts.
- Repeat the following until the list has only one symbol left.
  - From the list, pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node for them.
  - Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained.
  - Delete the children from the list.
- Assign a codeword for each leaf based on the path from the root.

Static Huffman Coding example

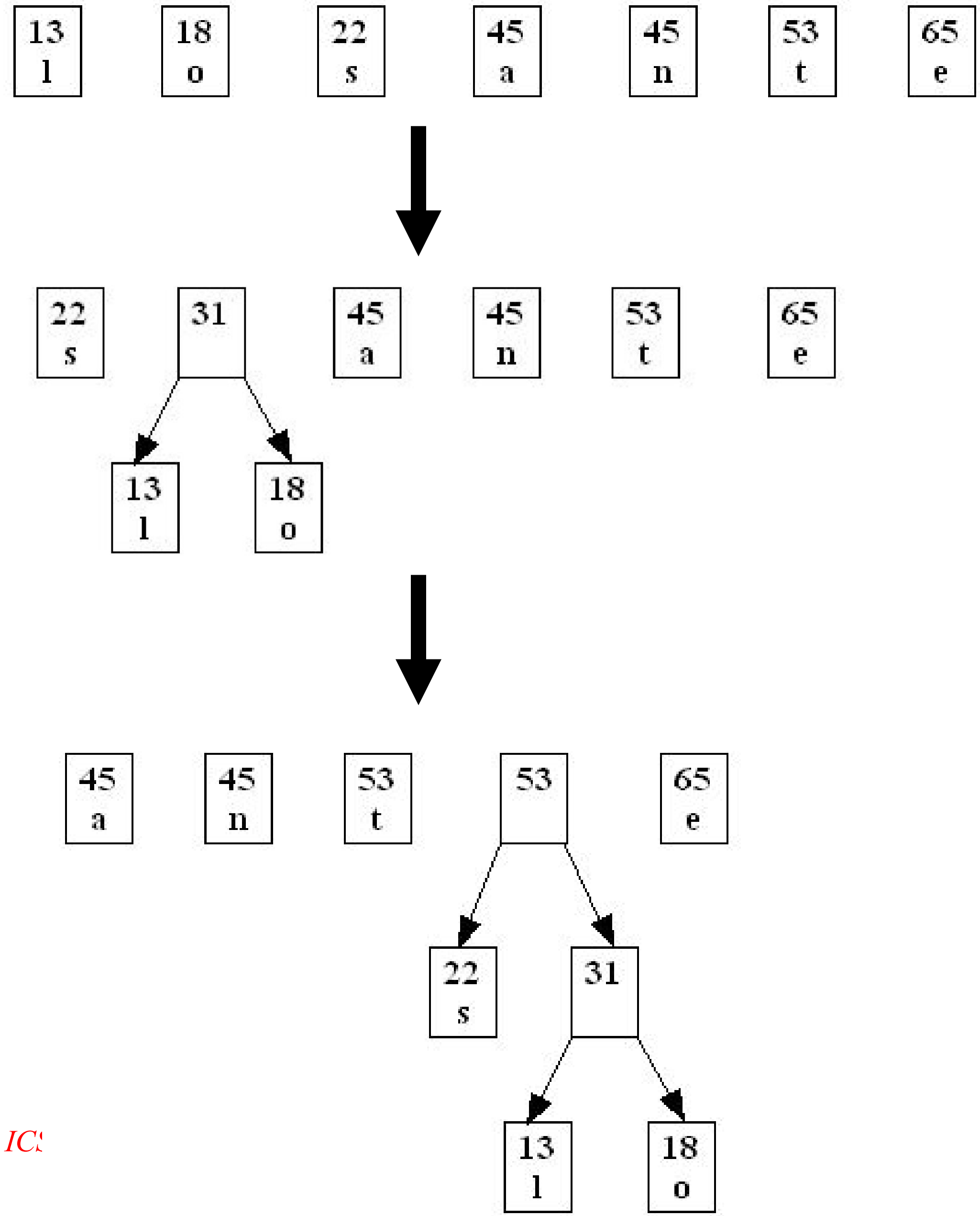
Example: Information to be transmitted over the internet contains the following characters with their associated frequencies:

Character	a	e	l	n	o	s	t
Frequency	45	65	13	45	18	22	53

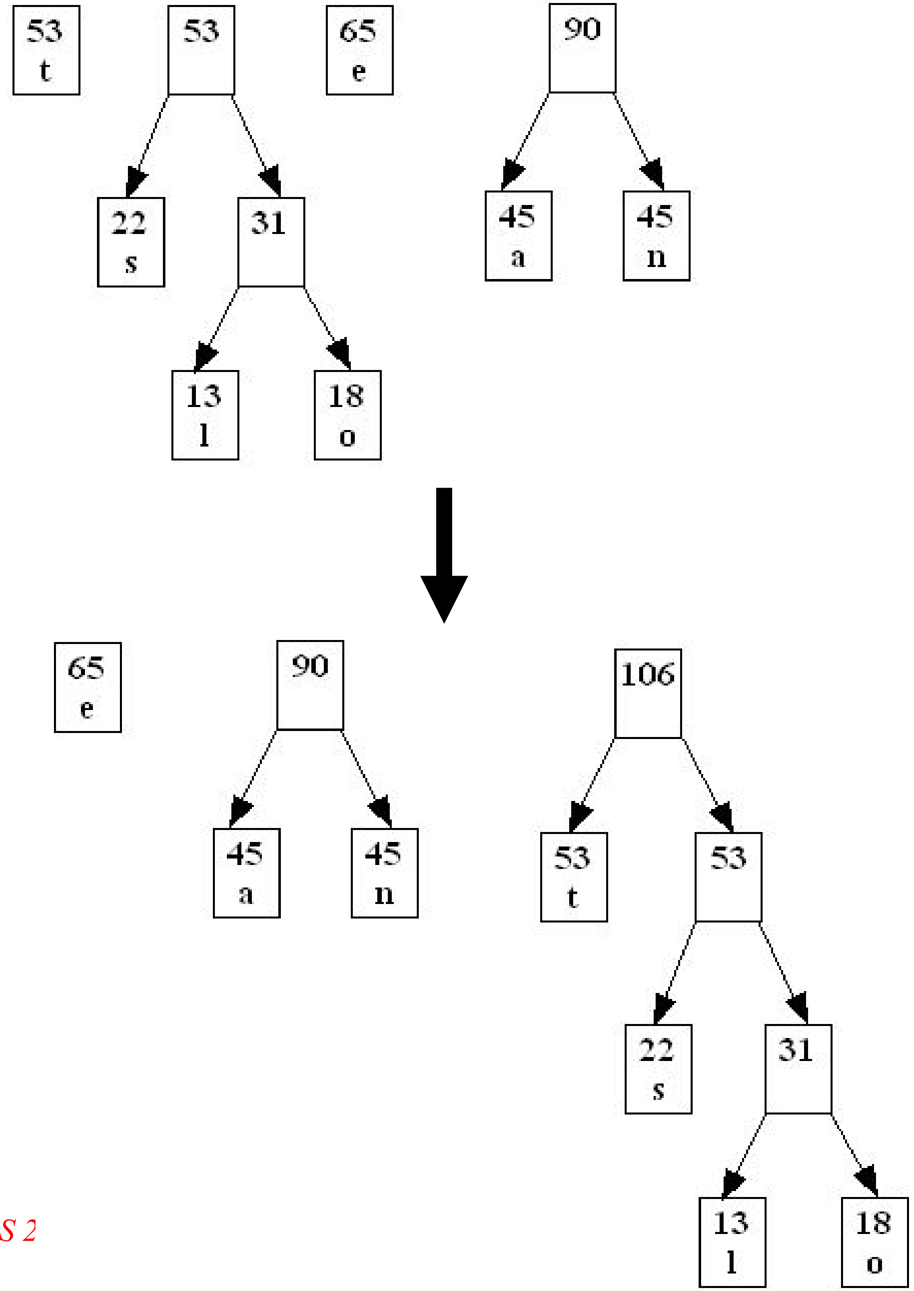
Use Huffman technique to answer the following questions:

- Build the Huffman code tree for the message.
- Use the Huffman tree to find the codeword for each character.
- If the data consists of only these characters, what is the total number of bits to be transmitted? What is the compression ratio?
- Verify that your computed Huffman codewords satisfy the Prefix property.

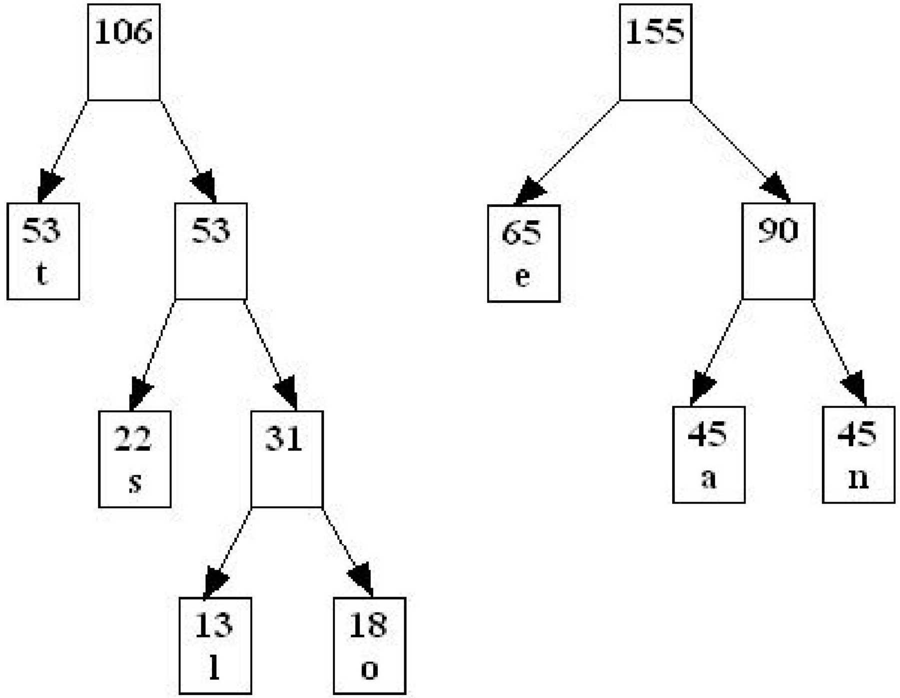
Static Huffman Coding example (cont'd)



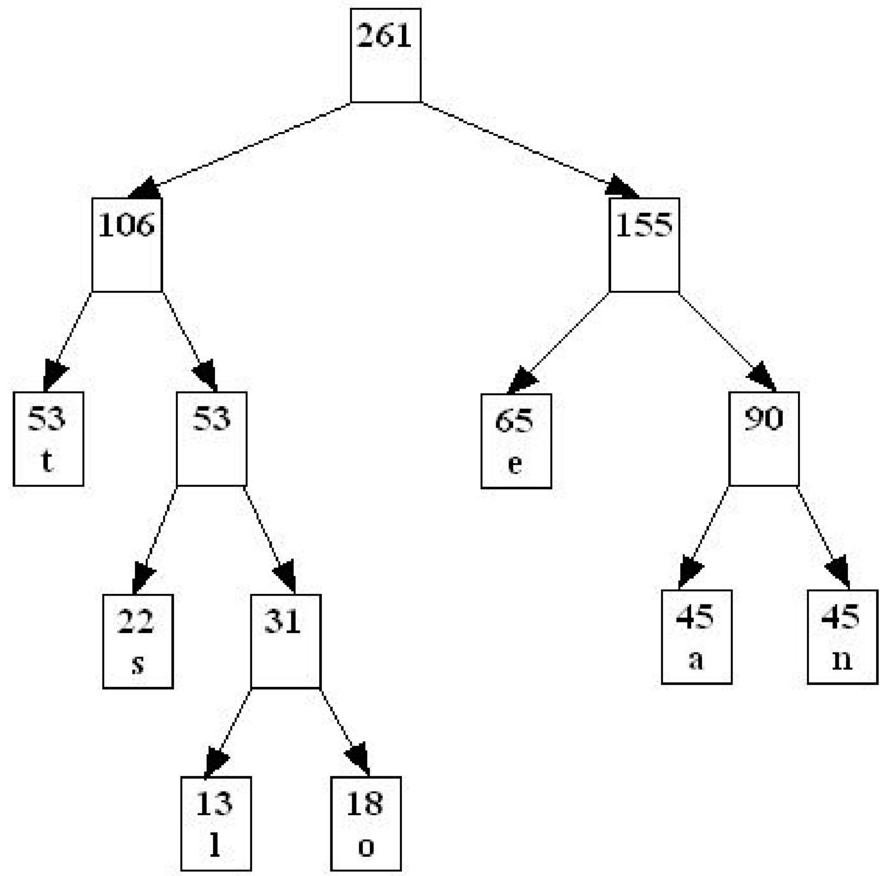
Static Huffman Coding example (cont'd)



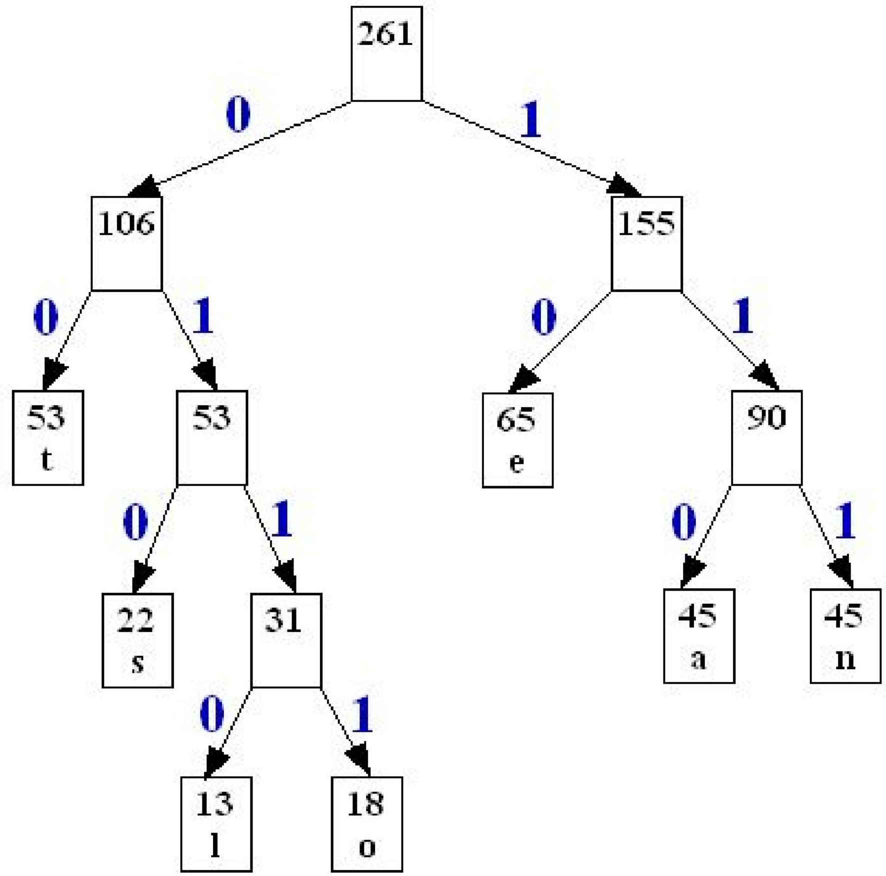
Static Huffman Coding example (cont'd)



Static Huffman Coding example (cont'd)



Static Huffman Coding example (cont'd)



The sequence of zeros and ones that are the arcs in the path from the root to each leaf node are the desired codes:

character	a	e	l	n	o	s	t
Huffman codeword	110	10	0110	111	0111	010	00

Static Huffman Coding example (cont'd)

If we assume the message consists of only the characters a,e,l,n,o,s,t then the number of bits for the compressed message will be 696:

character	a	e	l	n	o	s	t	
codeword	110	10	0110	111	0111	010	00	
codeword bits	3	2	4	3	4	3	2	
character frequency	45	65	13	45	18	22	53	
codeword bits * frequency	135	130	52	135	72	66	106	sum 696

If the message is sent uncompressed with 8-bit ASCII representation for the characters, we have 261*8 = 2088 bits.

•

Static Huffman Coding example (cont'd)

Assuming that the number of character-codeword pairs and the pairs are included at the beginning of the binary file containing the compressed message in the following format:

7 ← in binary (significant bits)
a110
e10
l0110 ← Characters are in 8-bit ASCII codes
n111
o0111
s010
t00
sequence of zeroes and ones for the compressed message

Number of bits for the transmitted file = bits(7) + bits(characters) + bits(codewords) + bits(compressed message)

= 3 + (7*8) + 21 + 696 = 776

Compression ratio = bits for ASCII representation / number of bits transmitted

= 2088 / 776 = 2.69

Thus, the size of the transmitted file is 100 / 2.69 = 37% of the original ASCII file

The Prefix Property

- Data encoded using Huffman coding is uniquely decodable. This is because Huffman codes satisfy an important property called the prefix property:

  In a given set of Huffman codewords, no codeword is a prefix of another Huffman codeword

- For example, in a given set of Huffman codewords, 10 and 101 cannot simultaneously be valid Huffman codewords because the first is a prefix of the second.

- We can see by inspection that the codewords we generated in the previous example are valid Huffman codewords.

The Prefix Property (cont'd)

character	a	b	c	d	e	f
codeword	0	101	100	111	110	1100

The decoding of 11000100110 is ambiguous:

11000100110 => face

To see why the prefix property is essential, consider the codewords given below in which "e" is encoded with **110** which is a prefix of "f"

11000100110 => eaace

Encoding and decoding examples

- Encode (compress) the message **tenseas** using the following codewords:

character	a	e	l	n	o	s	t
Huffman codeword	110	10	0110	111	0111	010	00

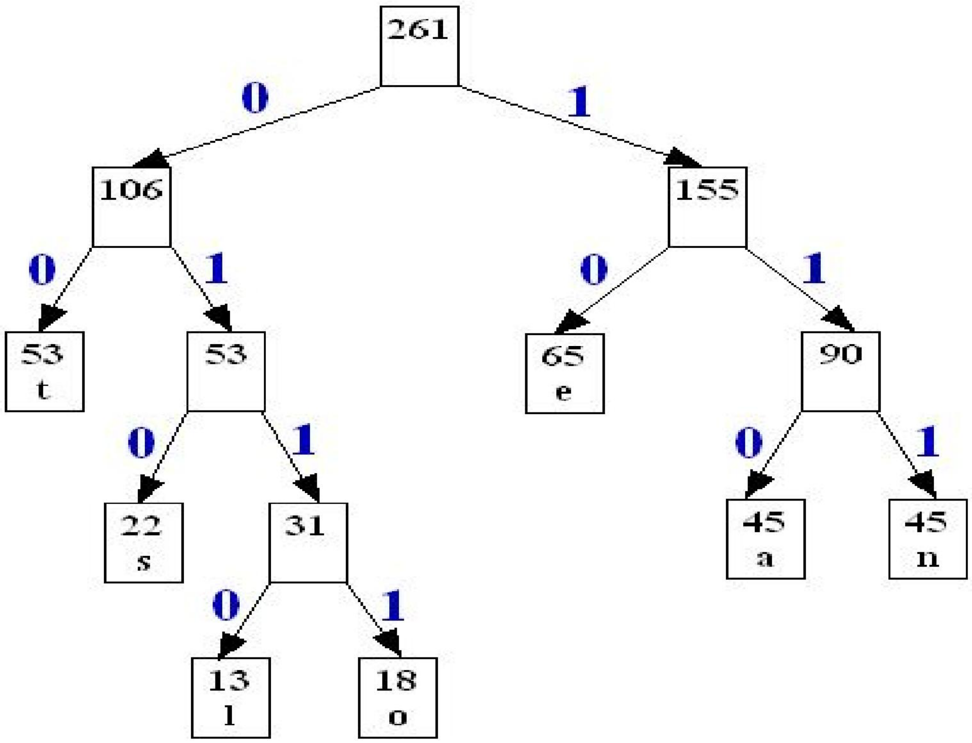
Answer: Replace each character with its codeword:

0010111010101100 10

Encoding and decoding examples

- Decode (decompress) each of the following encoded messages, if possible, using the Huffman codeword tree given on the next slide **0110011101000** and **11101110101011**:

Encoding and decoding examples



Answer

- Decode a bit-stream by starting at the root and proceeding down the tree according to the bits in the message (0 = left, 1 = right). When a leaf is encountered, output the character at that leaf and restart at the root .If a leaf cannot be reached, the bit-stream cannot be decoded.

End of chapter 3.1