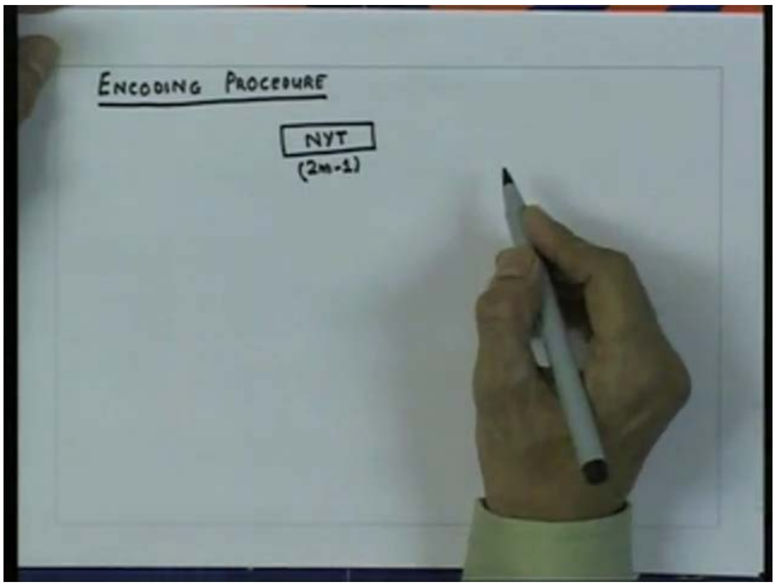
# Information Theory and Coding

**Prof. S. N. Merchant**
**Electrical Engineering**
**Indian Institute of Science, Bombay**

## Lecture - 16
## Adaptive Huffman Coding Part - II

In the last class, we studied update procedure which form an integral part of the adaptive Huffman code. In today's class, we will have a look at two more components of this adaptive Huffman code and these are encoding procedure and decoding procedure. Let us first have a look at the encoding procedure.

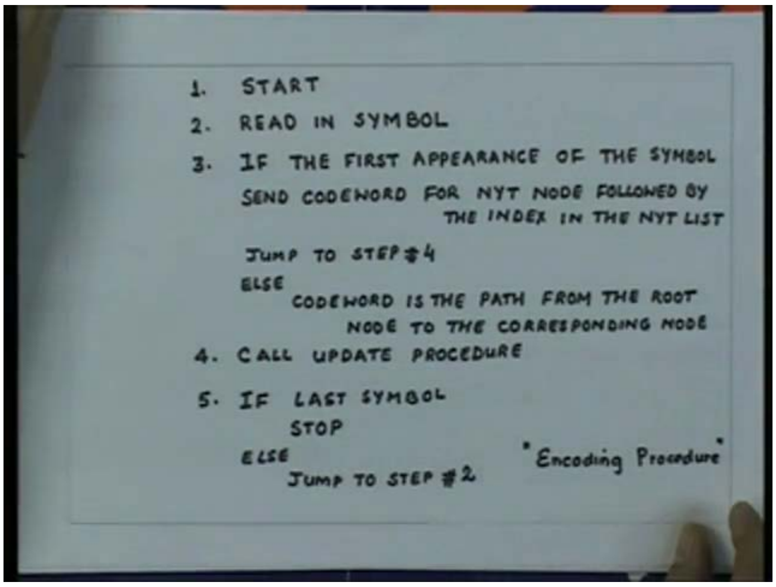
(Refer Slide Time: 01:11)



Initially, the tree at both the encoder and decoder consists of a single node which we call as the not yet transmitted node known as NYT node, and the number of this node is the maximum number that is 2 m minus one where m is the size of the alphabet. Now, the code word for the very first symbol that we encode is going to be a previously, agreed upon fixed code word.

After the very first symbol, whenever we have to encode a symbol that is being transmitted for the first time, we send the code for the NYT node followed by the previously agreed upon fixed code for that symbol. The code for the NYT node is obtain by traversing the tree from the root to the NYT node. This procedure alerts the receiver to the fact that the symbol whose code follows does not as yet have a node in the Huffman tree. If a symbol to be encoded has a corresponding node in the tree, then the code for that symbol is generated by traversing the tree from the root node to the leaf, or the external node corresponding to that symbol. So, the algorithm for an encoding procedure would be as follows.

(Refer Slide Time: 03:42)



```
1. START
2. READ IN SYMBOL
3. IF THE FIRST APPEARANCE OF THE SYMBOL
   SEND CODEWORD FOR NYT NODE FOLLOWED BY
                      THE INDEX IN THE NYT LIST
   JUMP TO STEP #4
   ELSE
       CODEWORD IS THE PATH FROM THE ROOT
            NODE TO THE CORRESPONDING NODE
4. CALL UPDATE PROCEDURE
5. IF LAST SYMBOL
       STOP
   ELSE
       JUMP TO STEP #2
```

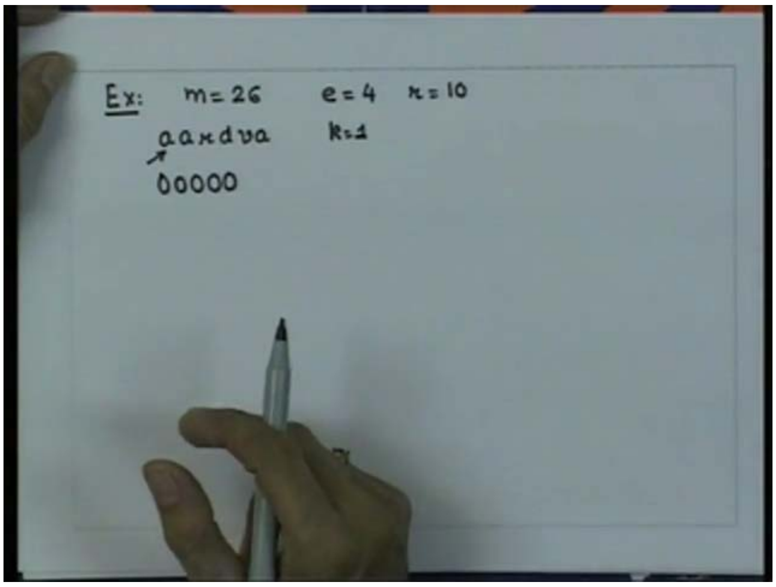
"Encoding Procedure"

First you start then after you have transmitted the first symbol. The codeword for which will be from the NYT list for the second symbol you follow the procedure as follows. Read in symbol then if this symbol is the first appearance of the symbol then you alert the receiver by sending the code for NYT node. So, send code word for NYT node followed by the index for that symbol in the NYT list.

After this step, we are supposed to update the tree at the encoder. So, we will jump to a step which involves the updating procedure. If this condition is not true, that means if the first it is not first appearance of the symbol, then else code word for that symbol is the path from the root node to the corresponding node for that symbol, and then again you call the update procedure.

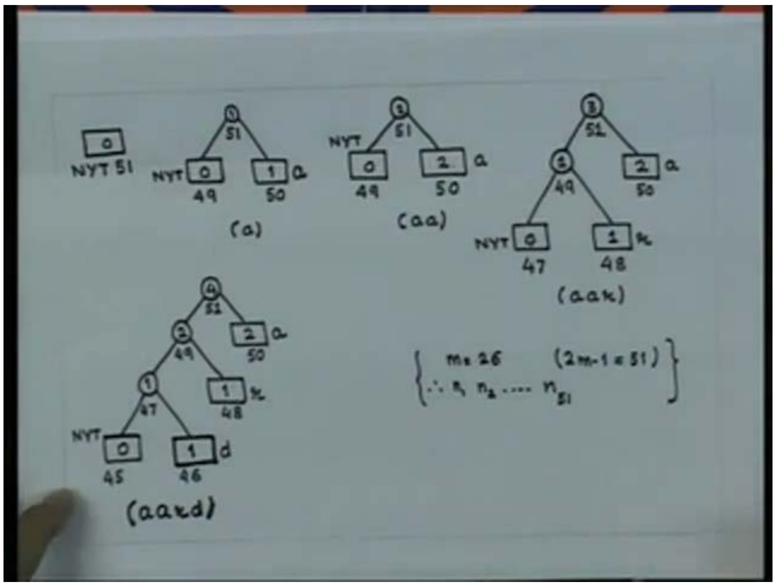
So, we have the step 4 which calls update procedure which we studied in the last class. Then find out if this is the last symbol. If it is the last symbol to be encoded then we stop else we read another symbol. So, we jumped to step 2. This would be the pseudo code for the encoding procedure. Let us try to understand this pseudo code in a much better fashion with the help of the example. Let us take the same example which we considered for the update procedure.

(Refer Slide Time: 08:19)



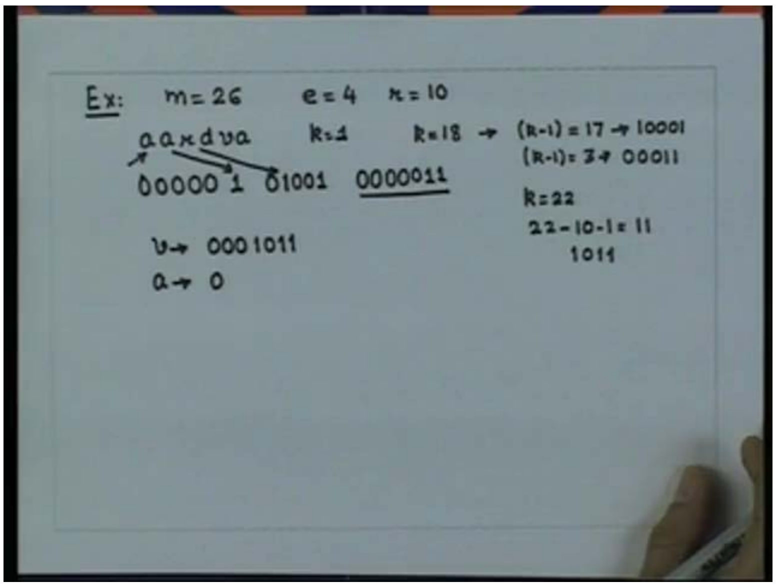
So, let us assume that we are encoding a source consisting of 26 small case letters of English alphabets. So, in our case m is equal to 26. So, to find out the fixed code, we find out this e and r values which turn out to be 4 and r is equal to 10. Let us takewe want to encode this sequence. So, the first symbol which we want to transmit is a. Now, as a is the first letter of the alphabets, k is equal to 1 and as 1 is less than 20, a is encoded as the 5 bit binary representation of k minus 1 that is 0. So, a will be encoded as 0 and the code word for that would be as given here. After you have done this the Huffman tree is updated as shown in the figure here.

(Refer Slide Time: 10:15)



So, we start with NYT equal to 51 and then after encoding a NYT 50. This node breaks up into two node, one is NYT new node and another is node corresponding to a. As a has occurred for the first time, the weight corresponding to element a is. Now, 1 and the old NYT weight is also, incremented to 1. Now, the next symbol to be transmitted or encoded is again a. Now, as a is available in this tree, therefore, the code word for encoding a of transmitting a would be obtained by simply traversing, the tree from the root node to the external node corresponding to a in order to find the code word. In this case, traversal consists of a single right branch therefore, the Huffman code for the symbol a is 1.

(Refer Slide Time: 11:48)


Ex: m = 26 e = 4 r = 10

aaxdva k = 1 k = 18 → (k-1) = 17 → 10001

(k-1) = 3 → 00011

00000 1 01001 0000011

k = 22

22-10-1 = 11

1011

v → 0001011

a → 0

So, for the next a, we transmit or encoded as 1. Now, the third symbol to be transmitted is r. As this is the first appearance of this symbol, we send the code for the NYT node followed by the previously arranged binary representation for r. Now, if it traverse the tree from the root node to the NYT node we get a code 0 for the NYT node. The letter r is the 18th in the alphabet. So, k is equal to 18. Now, since k is less than 2 r, that is 20, the binary representation of r is in terms of 5 bits. So, we represent r as binary representation of k minus 1, that is equal to 17 and that is equal to 10001.

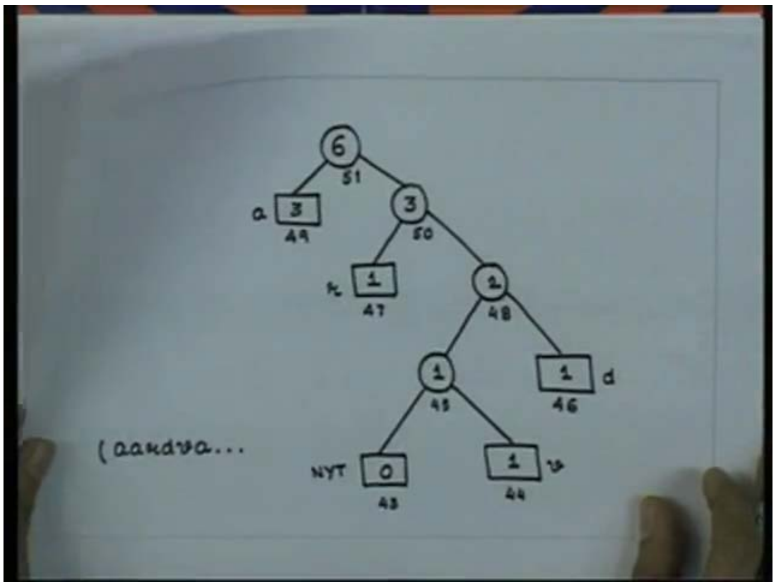
Therefore, the code word for the symbol r becomes 01001. This is the codeword for the NYT symbol. So, the tree is again updated at the end of encoding r and the tree becomes as shown here. We had looked in detail the construction of the update procedure in the last class. Now, the next symbol or letter to be transmitted is lettered and the code word for the NYT. Since, d is occurring for the first time, the code word for the NYT node is obtain again by traversing this tree from root to the NYT node.

So, if it traverse from the root to the node corresponding the NYT we get the codeword as 00. So, for d, the NYT node codeword would be 00 and since, d is the 4th letter in the alphabet and is less than 20, we represent it by k minus 1 that is equal to 3 and the codeword for that is 0011. Therefore, the total code word for d would be NYT code word followed by the code word for d. The next letter to be encoded is v. Since, v is appearing for the first time, we have to send the code word for the NYT node followed by the pre decided code word for v.

So, the code word for the NYT node would be given by the tree structure here because this is a tree structure which is existing at the end of encoding d. So, the NYT code word would be 000. So, we have the code word for vas NYT code word 000 followed by the pre decided code word for v. Now, since v is the 22 alphabet. So, k is equal to 22 which is larger than 20. So, we encode this as 22 minus 10 minus 1. That is equal to 11. So, we represent v by the 4 bit representation of 11 and that is 1011. So, the code word for v would be 000 followed by 1011. And at the end of encoding, we again call the update procedure and when we call update procedure, the final tree structure is as shown here.

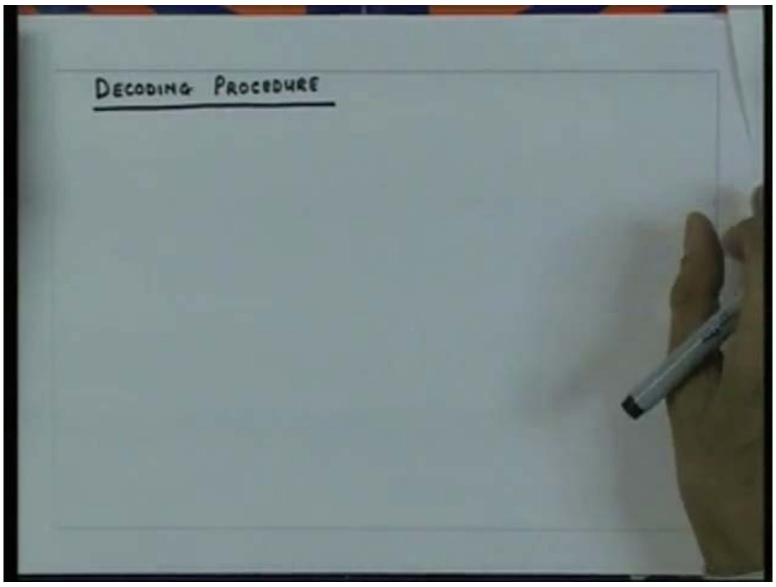
Now, the next alphabet to be the next letter to be transmitted is a. Now, if you look at this tree structure than a has already occurred. So, the code word for the letter a in this tree is given by traversing from the root to the leaf node corresponding to a, and in this case it would be 0. So, the code word for the letter a would be 0. So, if you concatenated all these binary digits, we will get the code for this sequence a a r d v a. Now, after we looked at the encoding procedure, let us look at the decoding procedure. So, at the end of encoding procedure after we have encoded a, the updated tree would look as follows.

(Refer Slide Time: 18:14)



We have just incremented the weight corresponding to the node a.

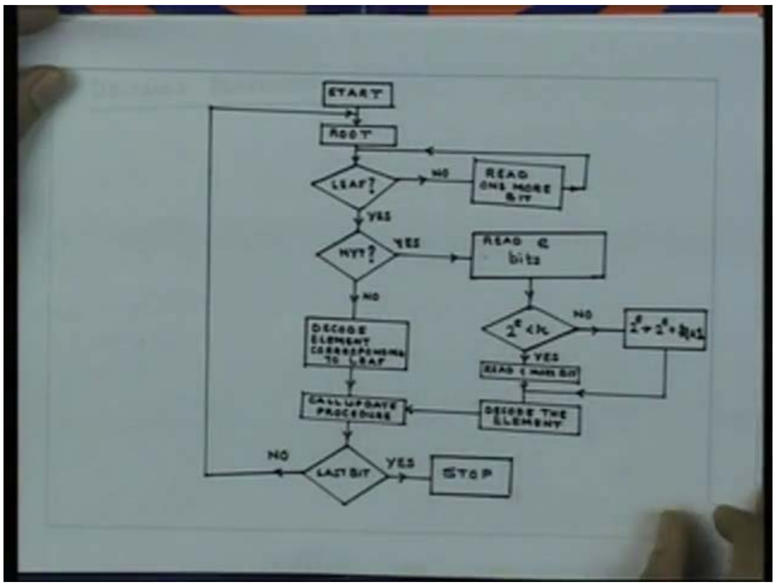
(Refer Slide Time: 18:28)



Decoding Procedure

Now, let us look at the decoding procedure. Now, as we read in the received binary string, we traverse the tree in a manner identical to that used in the encoding procedure. Let us try to understand the decoding procedure with the help of flow chart.

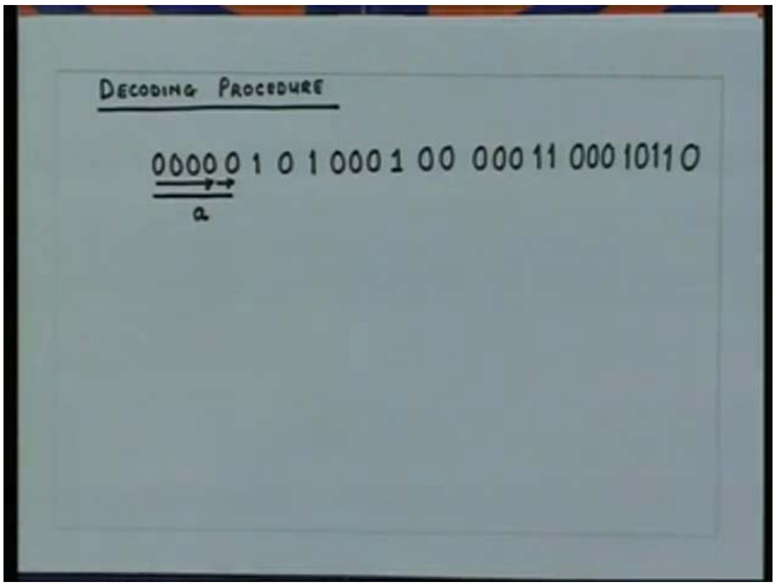
(Refer Slide Time: 19:14)



So, we start the decoding procedure by starting from the root of the tree. We keep on reading the binary bit string till we reach leaf. Now, we check for the status of this leaf. If the leaf is a NYT node then we read in e number of bits, that is four in our example and check for the value of those four bits. So, if 2 raised to e turns out to be less than r then if it is true then you read one more bit to get 5 bits and then you decode the element. But, if the value is not less than r than you decode by adding to that value of corresponding to 4 bit k plus 1 and then decode the element. After you have decoded element call the update procedure.

If the leaf is not the NYT node then decode element corresponding to that leaf and then again call the update procedure. Now, check if this is the last bit in the string of the transmitted sequence. If it is last bit you stop, otherwise, you go back to the root and start reading the bit. Now, let us try to look at this decoding procedure with the help of an example. So, let us say that we have the same binary string which we had considered earlier.

(Refer Slide Time: 21:32)



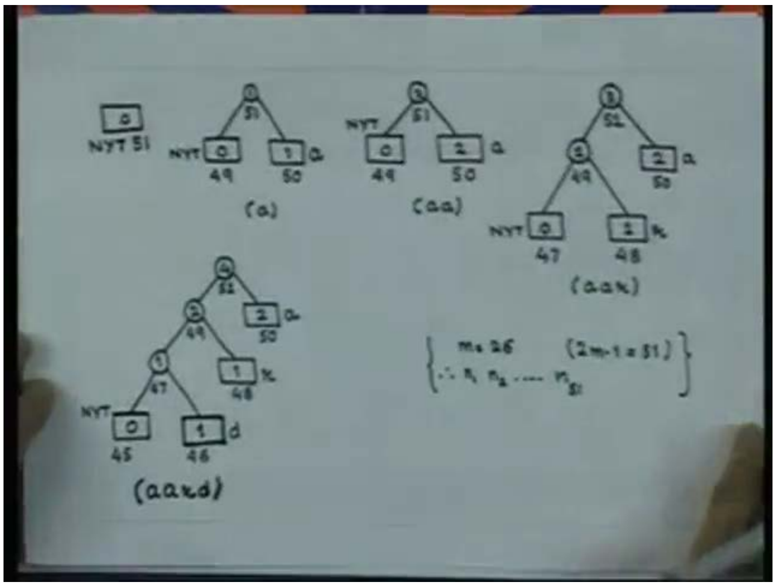
DECODING PROCEDURE

$\underline{0000\,0}\ 1\ 0\ 1\ 0001\ 00\ 00011\ 000\,1011\,0$

a

Let us assume that we received this binary string. Initially, the decoder tree consists only of the NYT node. Therefore, the first symbol to be decoded must be obtained from the NYT list. So, we read in the first four bits. This is 0000. As the value of e in our case is 4 the four bits 0000 correspond to the decimal value of 0. As this is less than the value of r, which is 10, we read in one more bit for the entire code of 00000. Now, adding one to the decimal value corresponding to this binary string, we get the index of the received symbol as 1. Now, this is the index for therefore, the first letter is decoded as a. The tree is... Now, updated as shown here.

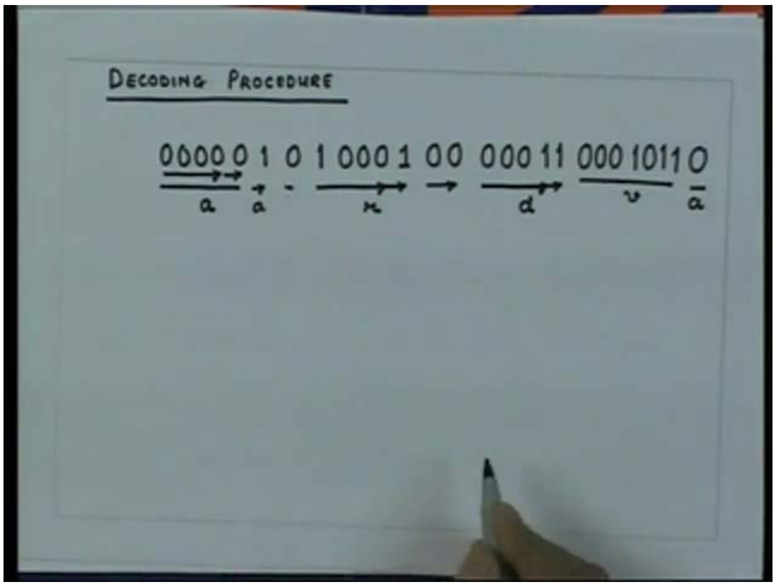
(Refer Slide Time: 23:27)



NYT 51
(a)
(aa)
(aak)
(aakd)
$\left\{ \begin{array}{ll} m = 26 & (2m-1 = 51) \\ \therefore n_1\ n_2 \cdots n_{51} & \end{array} \right\}$

So, you update that tree at the decoder as shown here. This updating procedure is exactly the same the updating procedure which we have at the encoder end.

(Refer Slide Time: 23:58)



DECODING PROCEDURE

00000 1 0 10001 00 00011 0001011 0

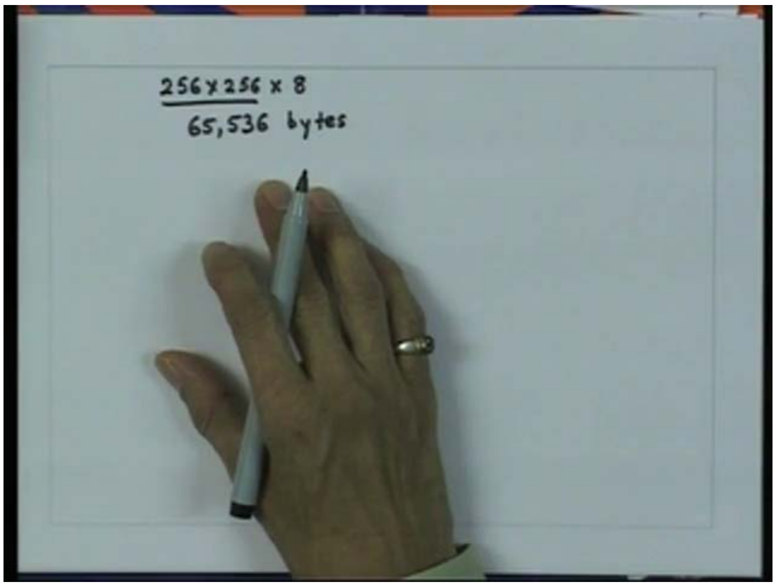
a a r d v a

The next bit in the string is one. Now, this traces the path from the root node to the external node corresponding to a. So, we decode symbol a, and then update the tree as shown here. In this case the update consists only of incrementing the weight of the external node corresponding to a. The next bit is a 0 which traces of path from the root to the NYT node. So, the next four bits are 1000 which correspond to decimal 8 which is less than 10. So, we read in one more bit to get the 5 bit word as 10001.

The decimal equivalent of this 5 bit one is 18 which is the index of r. So, we have here ‘a a r’. We decode the symbol r and then update the tree as shown here. The next two bits are 00. Again trace the path to the NYT node. So, we read the next four bits that is 0001. Since this correspond to decimal number one which is less than 10. We read another bit to get the 5 bit word as 00011. To get the index of the received signal symbol in the NYT list, we add one to the decimal value of this 5 bit word. The value of the index is four which corresponds to the symbol d. Containing in this fashion, we decode the sequence 0001011 as v, and finally this as a.

So, after having studied both adaptive and non-adaptive Huffman coding procedure, let us look at some of the examples of applications, where Huffman coding is applied. One such example is for lossless image compression. A simple application of Huffman coding to image compression would be to generate a Huffman code for the set of values that any pixel may take.

(Refer Slide Time: 28:01)

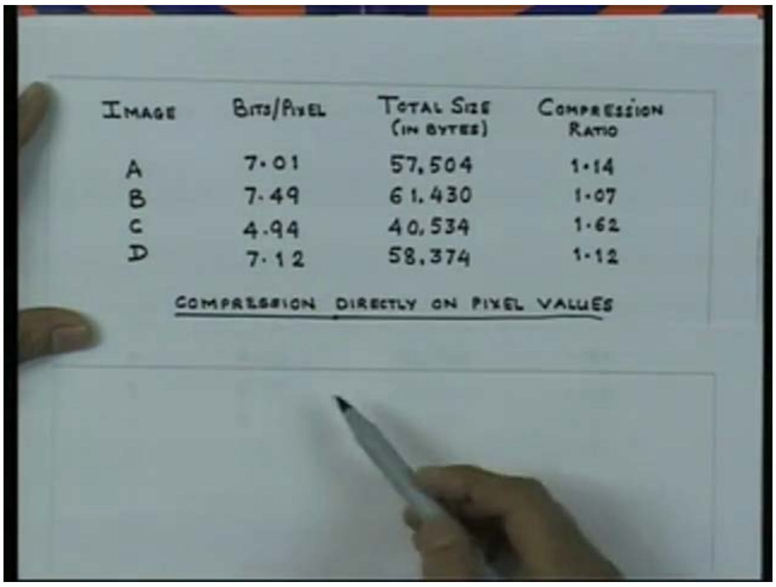


$\underline{256 \times 256} \times 8$

65,536 bytes

For monochrome images, this set usually consists of integer values from 0 to 255. In one such experiment, four images were taken and each image was of size, 256 by 256 pixels with each pixel taking values from 0 to 255. So, they are represented by 8 bit. This corresponds to 65,536 bytes. And on this four images, Huffman coding procedure was applied directly on pixel values, and the results of that experiment is as shown here.

(Refer Slide Time: 28:48)



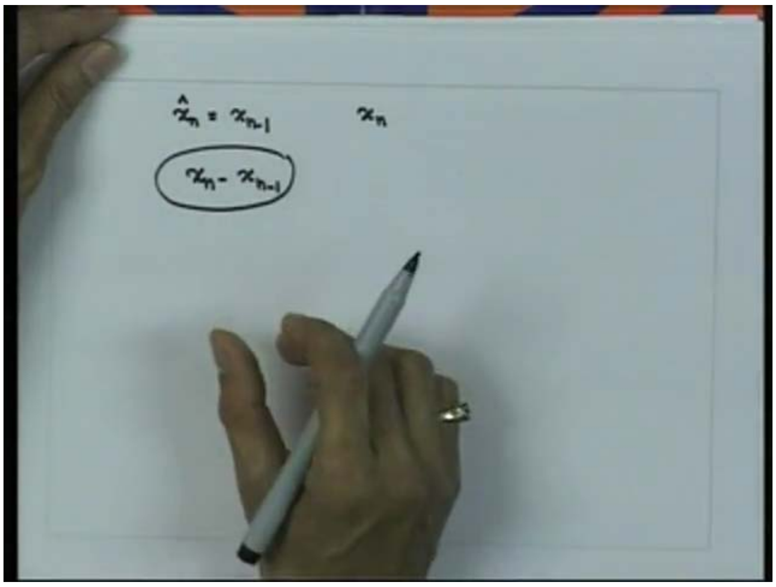
| IMAGE | BITS/PIXEL | TOTAL SIZE (IN BYTES) | COMPRESSION RATIO |
|---|---|---|---|
| A | 7.01 | 57,504 | 1.14 |
| B | 7.49 | 61,430 | 1.07 |
| C | 4.94 | 40,534 | 1.62 |
| D | 7.12 | 58,374 | 1.12 |

COMPRESSION DIRECTLY ON PIXEL VALUES

So, four images, let us call them as A, B, C and D were chosen and Hoffman code was directly applied on the pixel values and the result of that investigation is tabulated here. So, for the image A after the Huffman code was applied bits per pixel turned out to be 7.01. For b, it was 7.49 and C and D is given here. The total size in bytes are also, indicated here. Now, the compression ratio has been shown here. The compression ratio is defined as the number of bytes in the uncompressed representation divided by the number of bytes in the compress representation.

The number of bytes in the compress representation also, includes the number of bytes needed to store the Huffman code. So, from this investigation, it appears that we got the best results for the image C. Now, this is always the case the compression ratio is dependent upon the type of the images which are considered. Now, to get a better compression ratio it is necessary to utilise the statistics existing between the pixels in an image. Now, in most of the images, neighbouring pixels are highly correlated. What is implies is that the value of the current pixel can be predicted, or estimated from the values of the neighbouring pixels. So, one of the simplest school model, which can be used to estimate the current value of the pixel would be to say.

(Refer Slide Time: 31:11)

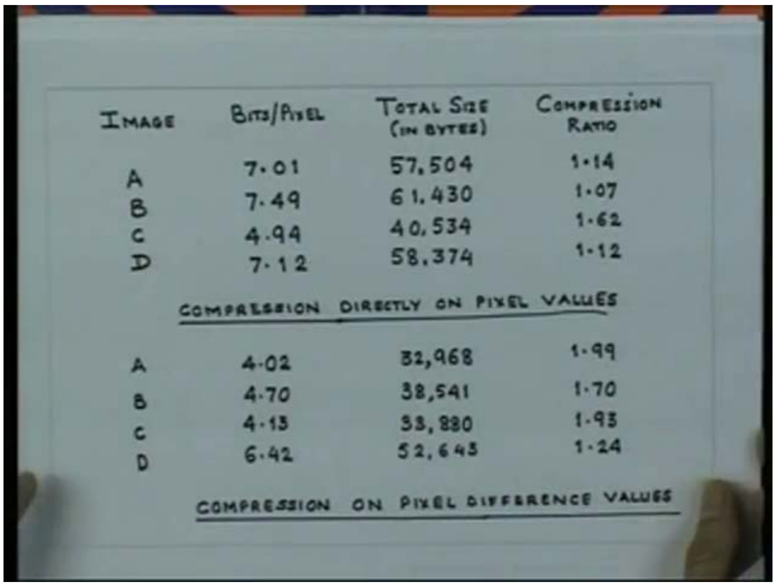


$$\hat{x}_n = x_{n-1} \qquad x_n$$

$$x_n - x_{n-1}$$

That estimation of the current value is equal to the past value. So, instead of encoding the pixel value x n, if you encode the residual between the pixel value and its estimate using this model, we would get as x n minus x n minus 1. So, this would be the residual value which will obtain after taking the different between the neighbouring pixels. Now, if you take this difference values and use the Huffman code on the residuals then we get the results as shown here.

(Refer Slide Time: 32:01)



| IMAGE | BITS/PIXEL | TOTAL SIZE (IN BYTES) | COMPRESSION RATIO |
|---|---|---|---|
| A | 7.01 | 57,504 | 1.14 |
| B | 7.49 | 61,430 | 1.07 |
| C | 4.94 | 40,534 | 1.62 |
| D | 7.12 | 58,374 | 1.12 |

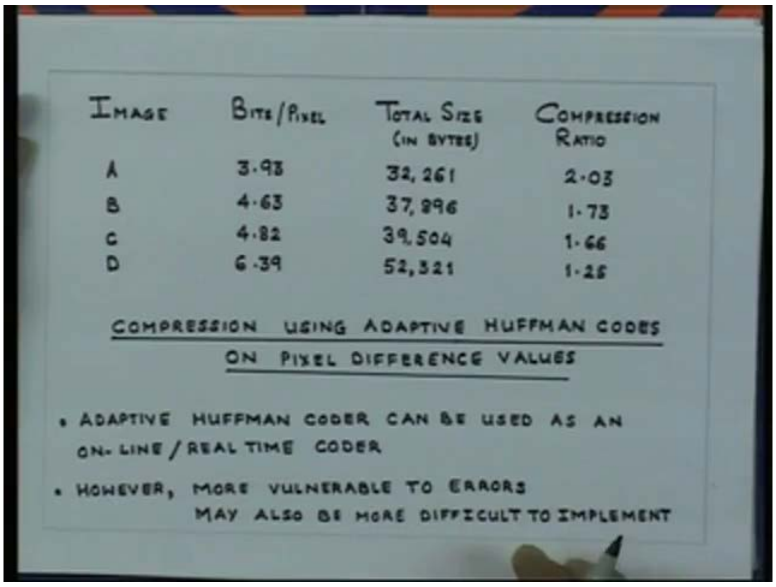
COMPRESSION DIRECTLY ON PIXEL VALUES

| IMAGE | BITS/PIXEL | TOTAL SIZE (IN BYTES) | COMPRESSION RATIO |
|---|---|---|---|
| A | 4.02 | 32,968 | 1.99 |
| B | 4.70 | 38,541 | 1.70 |
| C | 4.13 | 33,880 | 1.93 |
| D | 6.42 | 52,643 | 1.24 |

COMPRESSION ON PIXEL DIFFERENCE VALUES

The same images were considered, but now, the bits per pixel representation of this is given here and the compression ratios are again indicated out here. So, if you follow this procedure then it shows that for the image A we get the best result. Now, both these experiment were conducted by constructing the Huffman code based on 2 pass procedure. In the first pass, the statistics of the symbols in the source that is 0 to 255 integer values was connected and in the next pass it was used to code the same values. Now, if we had to use the adaptive Huffman coding procedure than the results are as shown here.

(Refer Slide Time: 33:20)



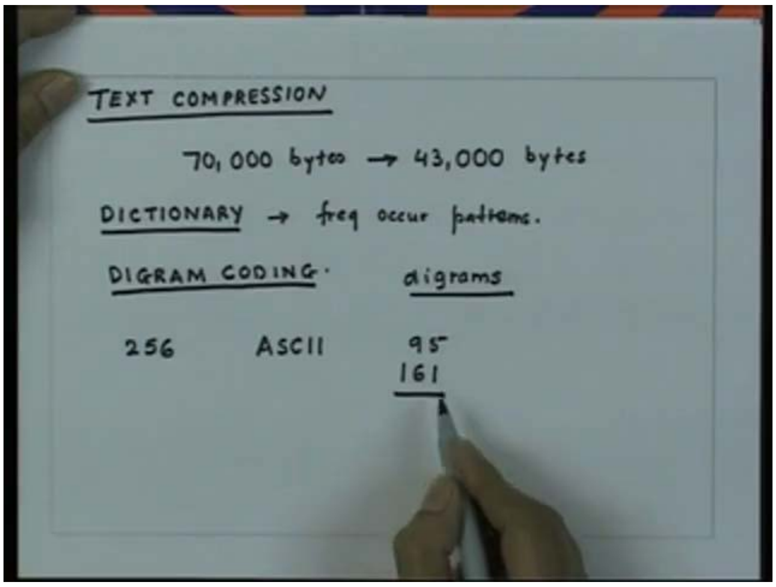
| IMAGE | BITS/PIXEL | TOTAL SIZE (IN BYTES) | COMPRESSION RATIO |
|---|---|---|---|
| A | 3.93 | 32,261 | 2.03 |
| B | 4.63 | 37,896 | 1.73 |
| C | 4.82 | 39,504 | 1.66 |
| D | 6.39 | 52,321 | 1.25 |

COMPRESSION USING ADAPTIVE HUFFMAN CODES ON PIXEL DIFFERENCE VALUES

- ADAPTIVE HUFFMAN CODER CAN BE USED AS AN ON-LINE / REAL TIME CODER
- HOWEVER, MORE VULNERABLE TO ERRORS
  MAY ALSO BE MORE DIFFICULT TO IMPLEMENT

This result is basically obtained after applying adaptive Huffman codes on pixel difference values and we see that there is a further improvement for the image A. Adaptive Huffman coder can be used as an online or real time coder. However, it is venerable to errors and also it is more difficult to implement. We will have look at better strategies for exploiting the statistical relationships between the pixels in the images. Just now, we have used the simplest model of utilising this statistical relationship in the form of estimation of the current value by the passed neighbouring pixel. Another application of Huffman code is to text compression.

(Refer Slide Time: 34:32)



TEXT COMPRESSION

70,000 bytes → 43,000 bytes

DICTIONARY → freq occur patterns.

DIGRAM CODING. digrams

256 ASCII 95
161

So, for text compression, application it seems that Huffman coding is a natural form of coding procedure. In text, we have a discrete alphabet that in a given class has relatively stationary probabilities. For example, the probability model for a particular English novel will not differ significantly from the probability model for another English novel. Similarly, probability model for a set of a C programs is not going to be much different than the probability model for a different set of C programs.

Now, simplest way to obtain the probability models is by counting the frequency of occurrence of letters in a document pertaining to a certain application. For example, English novel, in an experiment a document consisting of 70,000 bytes was Huffman coded and the result was represented by 43,000 bytes. There is a substantial drop in file size. While this reduction in file size is useful, a better compression can be obtained if we first remove the structure existing in the form of correlation between the symbols in the file.

Unfortunately, this correlation is not amenable to simple numerical models as it is in the case of image files. However, there are other somewhat more complex techniques that can be used to remove the correlation in text file. These techniques are based on what is known as dictionary concept. There are two types of dictionary. One is a stationary dictionary and other is adaptive dictionary. Let us consider only one example of a static dictionary. So, we will use dictionary concept for the encoding of the texts material.

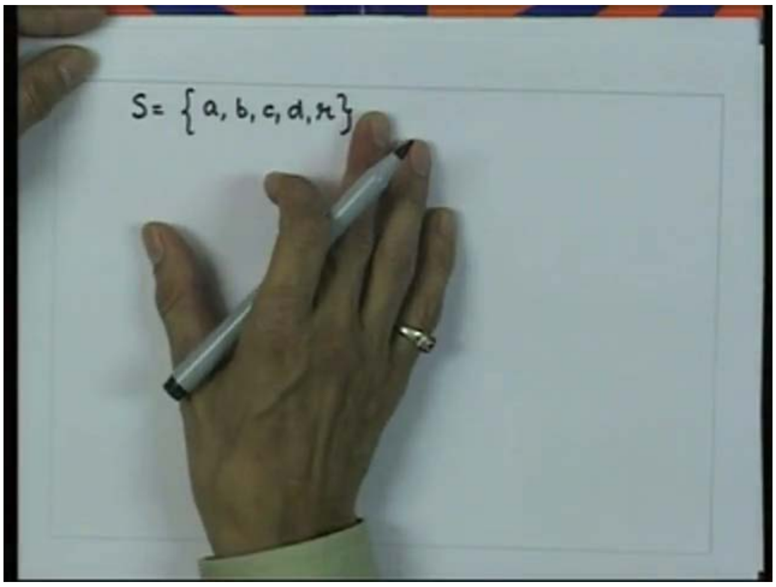
So, very reasonable approach to encode such source is to keep a list or dictionary of frequently occurring patterns. When these patterns appear in the source output, they are encoded with a reference to the dictionary. If the pattern does not appear in the dictionary then it can be encoded using some other less efficient method. In effect we are splitting the input into two classes. One class consist of frequently occurring patterns and other class consist of infrequently occurring patterns.

For this technique to be effective, the class of frequently occurring patterns and hence the size of the dictionary must be much smaller than the number of all possible patterns. As I said dictionary can be of two types static and adaptive. Choosing a static dictionary is most appropriate when considerable prior knowledge about the source is available. A static dictionary technique that is less specific to a single application is known as diagram coding.

In this form of coding, the dictionary consists of all letters of the source alphabet followed by as many pairs of letters called diagrams as can be accommodated by the dictionary. For example, suppose we were to construct a dictionary of size 256. For diagram coding of all printable ASCII characters. Then the first 95 entries of the dictionary would be the 95 printable ASCII characters.

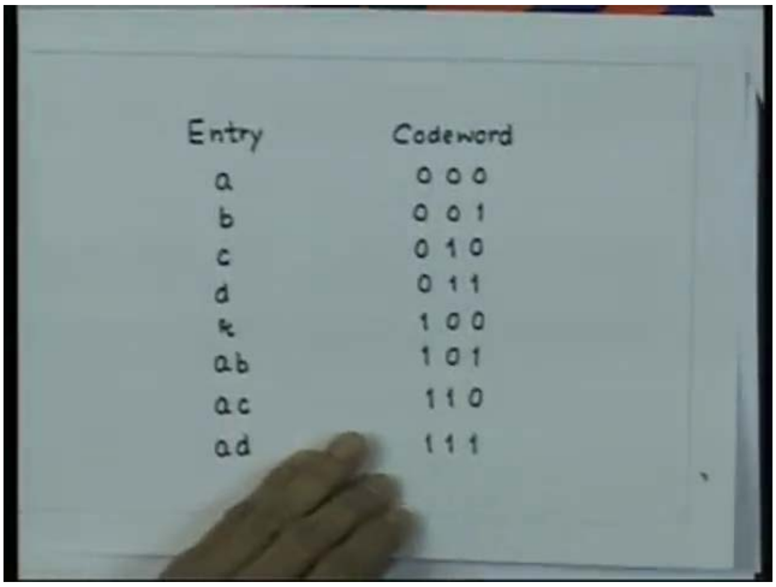
The remaining 161 entries would be the most frequently used pairs of characters. The diagram encoder works as follows. It first reads a two character input and searches the dictionary to see if this input exist in the dictionary. If it does, corresponding index is encoded and transmitted. If it does not, the first character of the pair is encoded. The second character in the pair then becomes the first character of the next diagram. The encoder reads another character to complete that diagram and search procedure is repeated. So, let us take an example.

(Refer Slide Time: 41:06)



Suppose I have a source consisting of 5letters a, b, c, d and r. Now, based on the knowledge about the source, we build the dictionary shown in the table here.

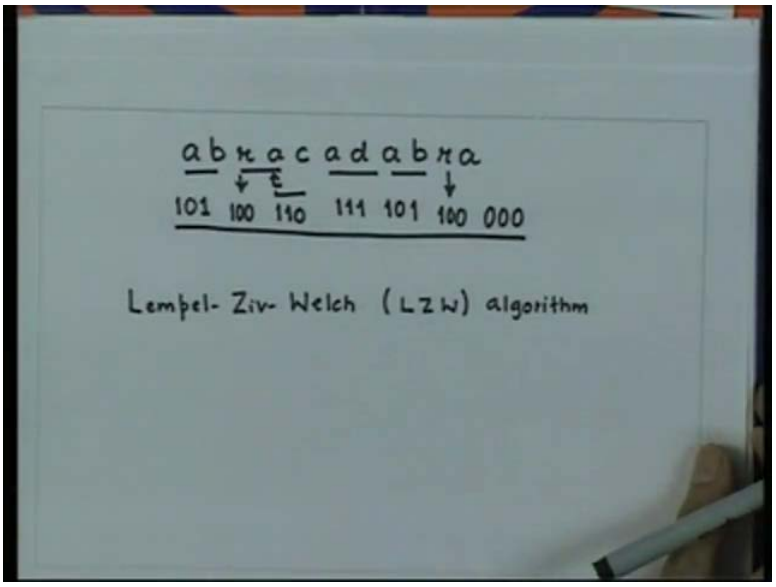
(Refer Slide Time: 41:26)



| Entry | Codeword |
|---|---|
| a | 0 0 0 |
| b | 0 0 1 |
| c | 0 1 0 |
| d | 0 1 1 |
| r | 1 0 0 |
| ab | 1 0 1 |
| ac | 1 1 0 |
| ad | 1 1 1 |

Let us assume, so the dictionary is built as follows. All letters of the source alphabet are included in the dictionary. So, all a, b, c, d and r are included and then we have provision for including three more letters in this dictionary, if you want a size of a dictionary to be represented by three bits. So, if you want the size to be of 8, then we have provision to take care of three most frequently occurring pairs. So, let us assume based on some knowledge about the source we find that a b, a c and a d occurred very frequently, so if you form a dictionary based on this than if we had to encode a sequence.

(Refer Slide Time: 42:25)



a b r a c a d a b r a

101 100 110 111 101 100 000

Lempel- Ziv- Welch (LZW) algorithm

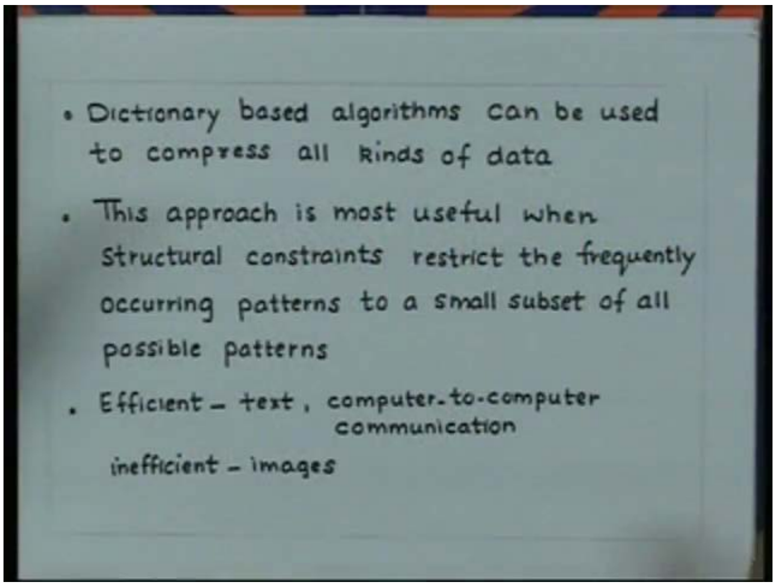
Let us say a, b, r, a, c, a, d, a, b, r, a then the procedure is as follows. First we read the two characters they formed a diagram. Now, check for this in the code whether the code word exists for this diagram a b. Now, if you look in the code which we have designed here based on some knowledge, we find that the code word for the diagram a b exists and that is 101. So, we code a b as 101.

The next diagram is r a. Now, if you look for r a we find that r a is not existing in the code. So, we take the first character of this diagram r and coded individually and the code word for r based on this code is 100. So, we code this as 100. The second character or the second letter in this diagram forms the first character of the diagram to follow. So, we read in the second letter and form another diagram that is a c. The code word for a c exists in our code that is 110. So, we code this as 110.

The next diagram is a d for which the code exists as 111. For a b again it exists as 101. For this it does not exist. So, we code this as 100. Now, only a is left out. So, we coded as 000. So, this would be the sequence of the code word being generated based on the concepts discussed. Now, most adaptive dictionary based techniques are based on a very popular algorithm that is known as Lempel-Ziv-Welch - LZW algorithm.

This algorithm is well explained in the literature. We will not go into the details as for this class is concerned. The idea behind this algorithm is that it dynamically constructs a dictionary from pattern observed in the source output. Dictionary based algorithms can be used to compress all kinds of data. However, care should be taken with their use. Otherwise we may end up with data expansion instead of data compression. So, the advantages of the dictionary based techniques can be summarised as shown here.

(Refer Slide Time: 46:20)



- Dictionary based algorithms can be used to compress all kinds of data
- This approach is most useful when Structural constraints restrict the frequently occurring patterns to a small subset of all possible patterns
- Efficient – text, computer-to-computer communication

  inefficient – images

This approach is most useful when structure constraint restrict the frequency occurring patterns to a small subset of all possible patterns. These techniques are very efficient for text and computer to computer type of communication, but it is not very efficient for the images. Now, we have looked into quite of few number of techniques for coding. We have seen that the Huffman codes is the optimal or compact code. Huffman code is a variable length coding approach.

Now, there is another popular variable length coding approach which is known as arithmetic coding. Arithmetic coding is very useful whenever you have source alphabet of small sizes, and when the probability of occurrence of the letters in the alphabet here are squealed. That means they are not uniform. In such cases Huffman coding procedures do not give very good results. So, in the next class we will have a look at arithmetic coding, but before we study arithmetic coding, we will study a simplified version arithmetic coding and that is based on what is known as Shannon Fano Elias coding.