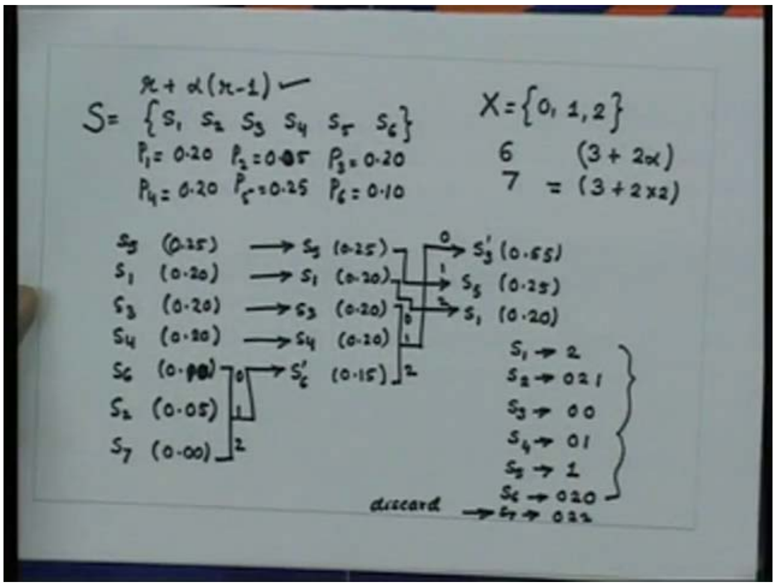
Information Theory and Coding

Prof. S. N. Merchant
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture - 14
Non-Binary Huffman Code and Other Codes

In the last class we studied the procedure to construct a non binary Huffman code. We saw that the procedure is similar to the construction of a binary Huffman code. The difference being that at each stage of source reduction we combine r symbols instead of two symbols, where r stands for the size of the code alphabet. And we have also seen that at each stage of source reduction the source symbols will reduce by r minus 1.

(Refer Slide Time: 01:40)



And if at the final stage we want that there should be r symbols then the total number of symbols which should be existing in the original alphabet should be of the form r plus alpha times r minus 1, where alpha is a integer and r is the size of the code alphabet. Now, the only hitch was that if this condition is not satisfied, then we said that we can add some dummy variables with probability 0, and then combine r symbols at each stage of source reduction.

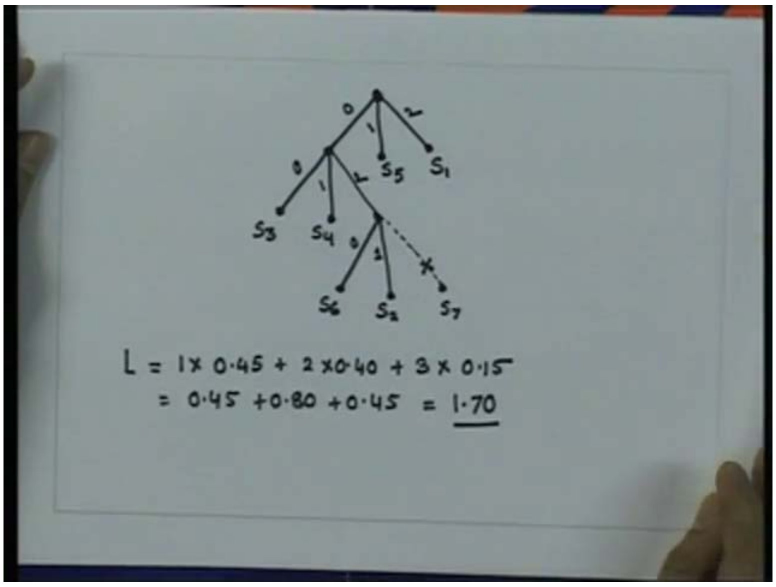
And in the final stage, when we are getting the code for the, code word for the particular symbol we ignore the code words for the dummy symbols. Now, let us look at that procedure which the aid of an illustration. Let us say that I have a source consisting of 6 symbols. So, S is given by S 1, S 2, S 3, S 4, S 5 and S 6 and I need to find out the code which is a Huffman code and the alphabet, code alphabet is a ternary. So, the code alphabet is 0, 1, 2.

So, let us assume that probabilities have been given to us as 0.2, probability of 0.25 and P 6. So, probability of S 2 is 0.05. So, the first step is to generate a non binary Huffman code for the source with this code alphabet is to arrange the symbols in the descending order of probability. So, S 5 0.25, S 1 is 0.20, S 3 0.20, S 4, S 6 0.05 sorry this is 0.01 and finally, S 2 is 0.05. So, this is 0.10 that is S 6. Now, we have six source symbols and if we look at the expression, this expression, this expression will be of form 3 plus 2 alpha.

So, this 6 is not of this form where alpha is integer. So, if I add one dummy variable then I will get 7 and this can become of the form 2 into alpha is equal to 2. I will get this equal to this. So, I had one dummy variable. So, let us add a dummy variable S 7 with a probability of 0.0. Now, once we have done this then we combine this three elements at this I get as… So, I will just indicate, I get here S 5, S 1, S 3, S 4 and this is S 6 dash and this is 0.25, 0.20, 0.20, 0.20 and 0.15. Then again I combine this two three and place it on the top. I get S 3 dash 0.55, this will be S 5 that is 0.25 and this is S 1 0.20.

Once we have got the three symbols because r is equal to 3. Now, we can allot this as 0, 1, 2, allot here as 0, 1, 2, final lot 0, 1, 2 and based on this we can find out the code turns out to be S 1 is 2, S 2 is 0 2 1, S 3 is 0 0, S 4 is 0 1, S 5 is 1 and S 6 is 0 2 0, S 7 actually would be 0 2 2, but this is a dummy variable. So, we discard this. So, we get the code words corresponding to our symbols as this and this can be also depicted in form of a code tree. So, the code tree for this would like something like this.

(Refer Slide Time: 08:34)

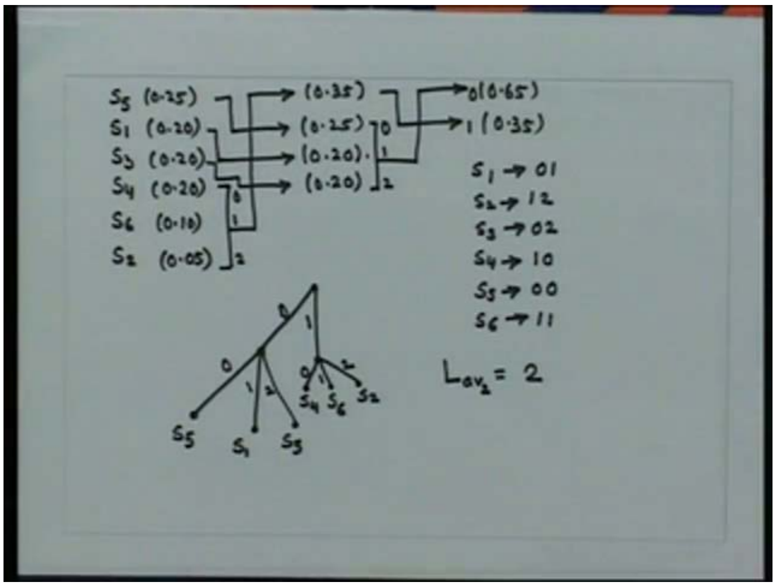


$$L = 1 \times 0.45 + 2 \times 0.40 + 3 \times 0.15$$
$$= 0.45 + 0.80 + 0.45 = \underline{1.70}$$

So, this is your root node I had here is S 1, this is my S 5, this would be S 3, this would be S 4, this would be S 6 and this would be S 2 and this is actually S 7 which is discarded and we follow the convention 0 1 2, 0 1 2, 0 1. So, once we have the tree we can also obtain the code words for this symbol by traversing from the root node to the leaves. So, S 3 would be 0 0 and S 2 would be 0 2 1 which is the same as what we have got it here. Now, we can calculate the average length for this code.

So, the average length turns out to be there, we have one code word of length 1 and two code words of length 1 and two code lengths of length 2 and code lengths of 3. So, if we calculate that this will turn out to be 1 into 0.5, 2 into 0.40 plus 3 into 0.15 and this is equal to 0.5 plus point… So, this is 1.70 ternary digits per symbol, ternary digits per symbol. Now, this is the optimum three array Huffman code for the given source. Now, if we had not followed this procedure, but an what we mean by that if we had not added this S 7 and if we had combined 3 and then again 3 and then 2 then we would have got another tree. And another way of combining would be combine 3 then combine 2 and then combine again 3. So, let us look at those tree structures.

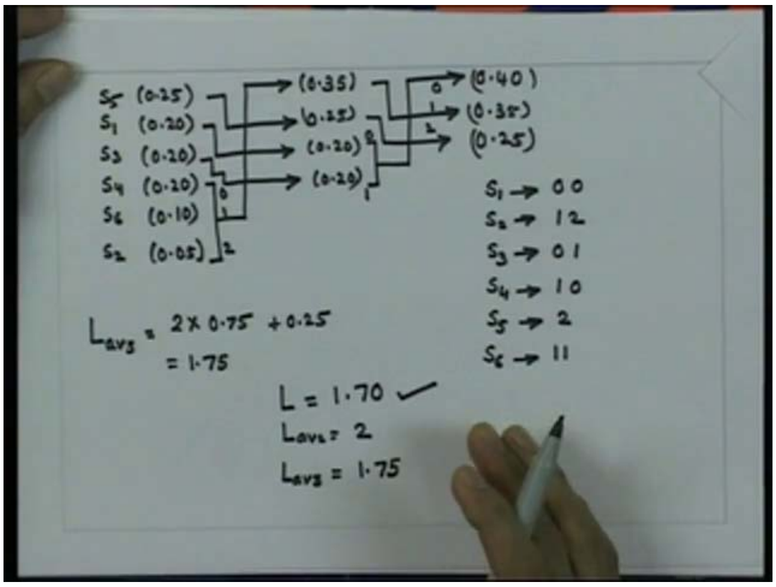
(Refer Slide Time: 11:38)



So, if I combine 3 3 and 2 then I get it as S 5 S 1 S 3 S 4 S 6 S 2 0.25 0.20 0.20 0.20 0.0 and 0.05. Now, without adding any dummy symbol we will add up, we will get a reduced source by combining the three source symbols and then place it here. I will just indicate the probabilities of the new symbols. So, this 0.35 and then this would be 0.25, this would be 0.0 and finally, this would be 0.20. And then we combine again this three and then to get finally, two symbols 0.65 and 0.35.

Now, we can allot 0 1, 0 1 2, 0 1 2 and we go from trace the path from here to here for each symbol. We will get the code word as S 1, S 2 as 1 2, S 3 as 0 2, S 4 as 1 0, S 5 as 0 0 and S 6 as 1 1 and this can be indicated in the form of a tree as, this is 0 and this is 1, this is 0 1 2, 0 1 2 and this of course this leaves or external node correspond to the symbols as shown here, S 6, S 2. So, here all the code words of length 2, so the average length will turn out to be as 2 ternary digits per source symbol. And finally, we can so let us call this as average 2 and finally, we will follow the procedure as given here.

(Refer Slide Time: 14:53)

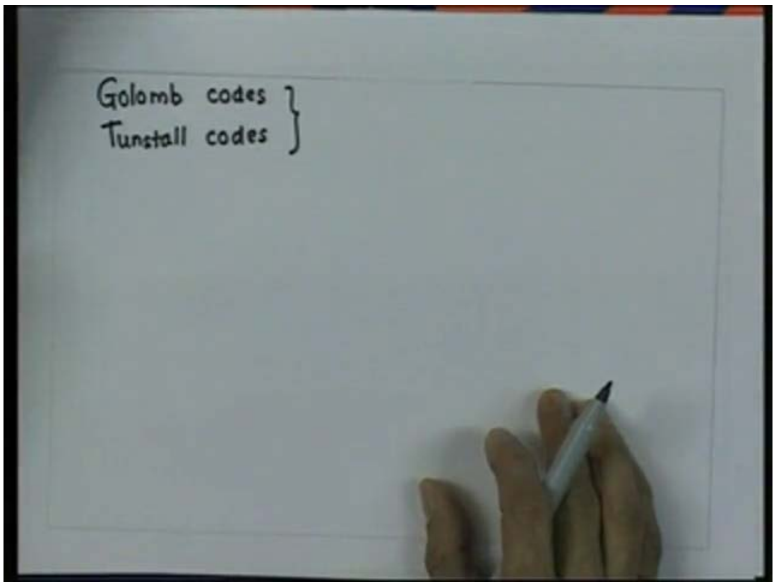


I have S 5, S 1, S 3, S 4, S 6, S 2, 0.25, 0.20, 0.20, 0.20, 0.10, 0.05. So, initially I combine three symbols I get a new source symbol. I place it here, I just indicate the new probabilities then I place it here, here and place it here. This are 0.25, 0.20 and 0.20. Now, I combine two elements to get a reduction on then source. So, this is 0.40, 0.35 and finally 0.25. So, in the final stage again I have three symbols, but in this procedure in between I have combined only two symbols.

So, for lot 0 1 2, lot 0 1, 0 1 2 and get the source code for each symbol would be given as follows, 1 1. And again I can draw the binary tree for this and the average length in this case, let us call as average length as 3. That turns out be 2 into 0.75 plus 0.25 is equal to 1.75 ternary digits per symbol. So, the first case we got L as 1.70, second case we got as 2 and the third case we got as 1.75. So, this is the lowest value which we got and this is the value which we will get if we follow the procedure discussed where we combine r symbols at a time at each stage of source reduction.

So finally, we reach with the alphabet, which consist of only r symbol. And if the initial number of symbols in the source alphabet does not follow the expression r plus alpha r minus 1 then we add some dummy symbols and with the probability 0. So, we have looked at the Huffman coding algorithms for both for the binary case and the non binary case. Although, the Huffman coding algorithm is one of the best known variable coding length algorithm there are some lesser known algorithms that can be useful in certain situations. In particular there are two algorithms.

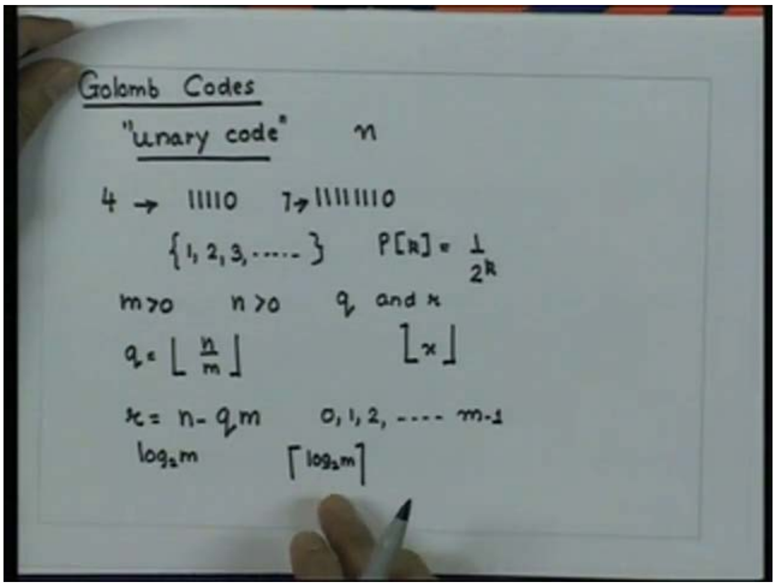
(Refer Slide Time: 18:45)



Golomb codes
Tunstall codes $\}$

One is what is known as Golomb's algorithm, that is Golomb's codes and another is Tunstall codes. Both these codes are becoming increasingly popular. So, we will discuss this codes next. So, let us take first the Golomb codes.

(Refer Slide Time: 19:19)



Golomb Codes

"unary code" n

$4 \rightarrow 11110$ $7 \rightarrow 11111110$

$\{1, 2, 3, \ldots\}$ $P[k] = \frac{1}{2^k}$

$m > 0$ $n > 0$ q and r

$q = \lfloor \frac{n}{m} \rfloor$ $\lfloor x \rfloor$

$r = n - qm$ $0, 1, 2, \ldots m-1$

$\log_2 m$ $\lceil \log_2 m \rceil$

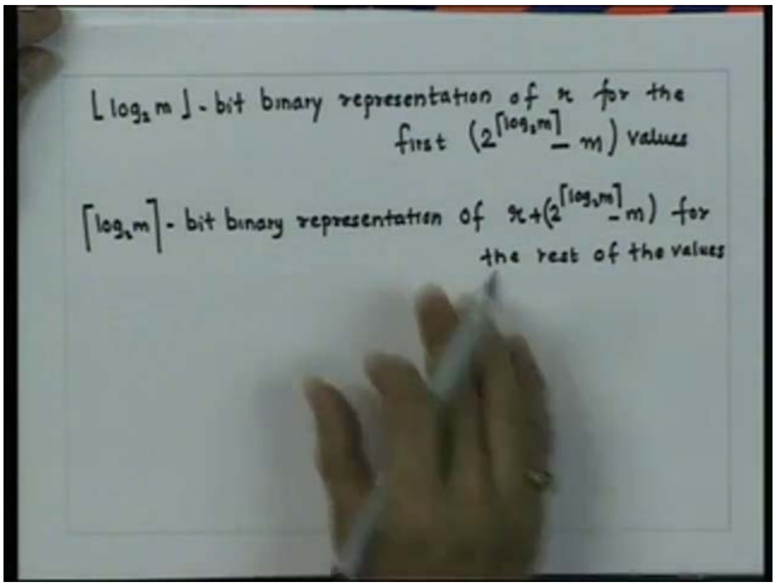
The Golomb codes belong to a family of codes designed to encode integer with the assumption that the larger an integer the lower its probability of occurrence. The simplest code for this situation is the unary code. The unary code for a positive integer n is simply n one's followed by a 0 or n zeros followed by a 1. Thus the code for 4 would be 1 1 1 1 0 and the code for 7 would be seven one's followed by a 0. The unary code is the same as the Huffman code for the semi infinite alphabet 1, 2, 3 with probability model P k equal to 1 by 2 raise to k.

Because the Huffman code is optimal the unary code is also optimal for this probability model. Although, the unary code is optimal in very restricted conditions we can see that it is certainly very simple to implement. One step higher in complexity are a number of coding schemes that split the integer into 2 parts, representing one part with a unary code and other with a different code. An example of such a code is the Golomb code. The Golomb code is actually a family of codes parameterized by integer m which is greater than 0.

In the Golomb code with parameter m we represent an integer n greater than 0 using two numbers q and r. So, n is represented with the help of two numbers which are q and r. Let us see what are this q and r. q is equal to the truncation value of n by m and there by definition the lower flow means the integer part of x. And r is given by n minus q m. So, q is the quotient and r is the remainder. So, q can take all values 0 1 2 all positive integers and it is represented by the unary code for q, whereas r can take all the values 0 1 2 up to m minus 1.

If m is a power of 2 we use the log 2 of m bit binary representation of r, but if m is not a power of 2 we could still use log to m the upper flow of this number to represent r. Now, when we represent using this strategy the number of bits required would be higher than the number of bits required by following the following strategy. So, if m is not a power of 2 we can reduce the number of bits required if we use the following strategy.

(Refer Slide Time: 24:18)

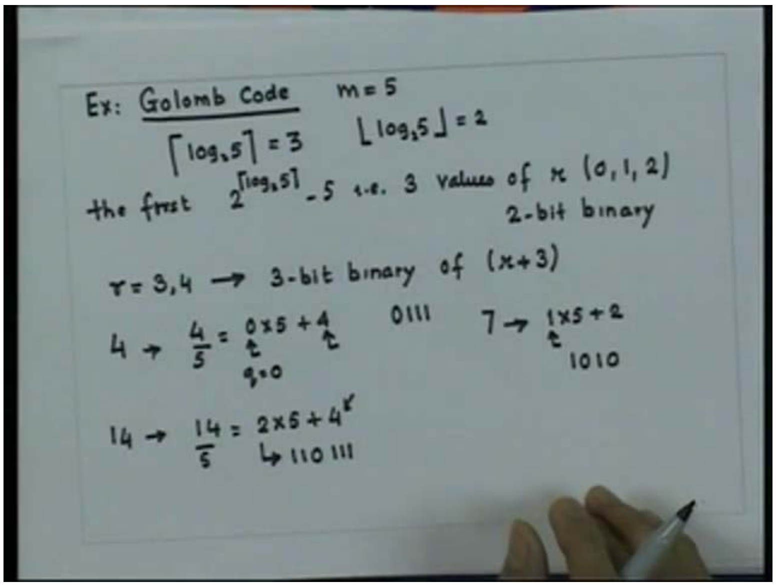


$\lfloor \log_2 m \rfloor$ - bit binary representation of r for the first $(2^{\lceil \log_2 m \rceil} - m)$ values

$\lceil \log_2 m \rceil$ - bit binary representation of $r + (2^{\lceil \log_2 m \rceil} - m)$ for the rest of the values

We represent r by the lower flow of this value with representation of r for the first 2 raise to upper flow of log 2 m minus m values. So, for all the values of r which are less then this quantity will represent it r by lower flow of log to m bit binary representation. And we will use upper flow of log 2 m bit binary representation of r plus 2 log 2 m minus m for the rest of the values. In order to understand this in a much better way let us take up a simple example.

(Refer Slide Time: 26:09)



Ex: <u>Golomb Code</u> $m = 5$

$\lceil \log_2 5 \rceil = 3$ $\lfloor \log_2 5 \rfloor = 2$

the first $2^{\lceil \log_2 5 \rceil} - 5$ i.e. 3 values of r $(0, 1, 2)$ 2-bit binary

$r = 3, 4 \rightarrow$ 3-bit binary of $(r+3)$

$4 \rightarrow \frac{4}{5} = \underset{\underset{q=0}{\uparrow}}{0} \times 5 + \underset{\uparrow}{4}$ 0111 $7 \rightarrow \underset{\uparrow}{1} \times 5 + 2$ 1010

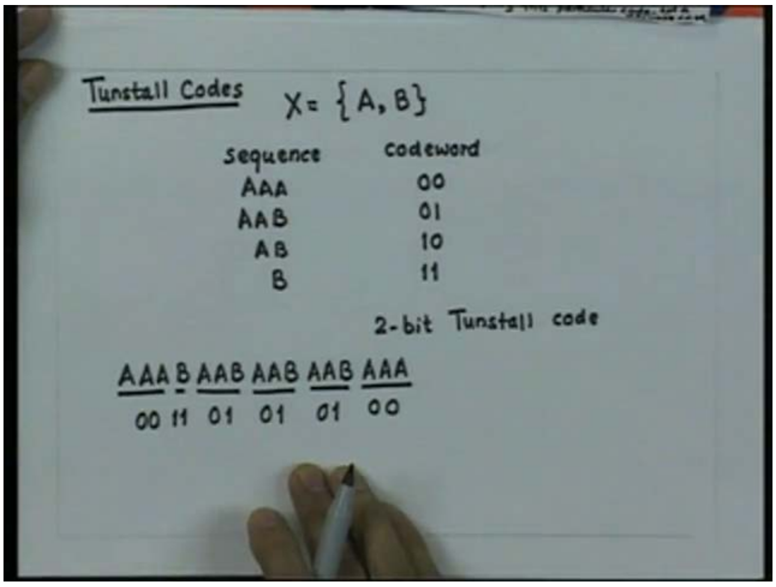
$14 \rightarrow \frac{14}{5} = 2 \times 5 + 4^{r}$ ↳ 110 111

So, let us take an example where a Golomb code parameterized by m equal to 5. So, m is equal to 5 in this case the upper flow of log 2 5 is equal to 3. This is the integer which is just larger than this value and the lower flow of log 2 5 is equal to 2. So, the first 2 raise to log 2 5 minus 5 that is 3 values of r which are 0 1 and 2 will be represented. This values of r will be represented by 2 bit binary representation and the next two values of r which is 3 and 4 will be represented by the, this will be represented by 3 bit binary representation of r plus 3.

So, the q is always represented by the unary code for q. So, if we take Golomb code of 4. So, take 4, 4 I divide by 5, I will get it as 0 multiplied by 5 plus 4. So, in this case q is equal to 0. So, the unary representation of 0 would be 0 and this is r is equal to 4. Since, r 4 has to be represented by 3 binary representations of r plus 3, so 4 plus 3 is 7. So, binary representation of 7 is 1 1 1. Similarly, the code word for 14 would be 14 by 5 would give me 2 multiplied by 5 plus 4.

So, the quotient is 2 and this by unary representation would be 1 1 0 and the remainder is 4. So, again the representation would be this. And let us take a simple case. Another finally, 7, so for 7 this can be represented as 1 into 5 plus 2. So, unary representation form 1 would be 1 0 and 2 is in this range. So, it is represented by 2 bit binary that would be 1 0. So, I get this is a Golomb representation for Golomb codes for different integer values. Now, we look at another code that is the Tunstall codes.

(Refer Slide Time: 30:01)



Tunstall Codes $X = \{A, B\}$

Sequence	Codeword
AAA	00
AAB	01
AB	10
B	11

2-bit Tunstall code

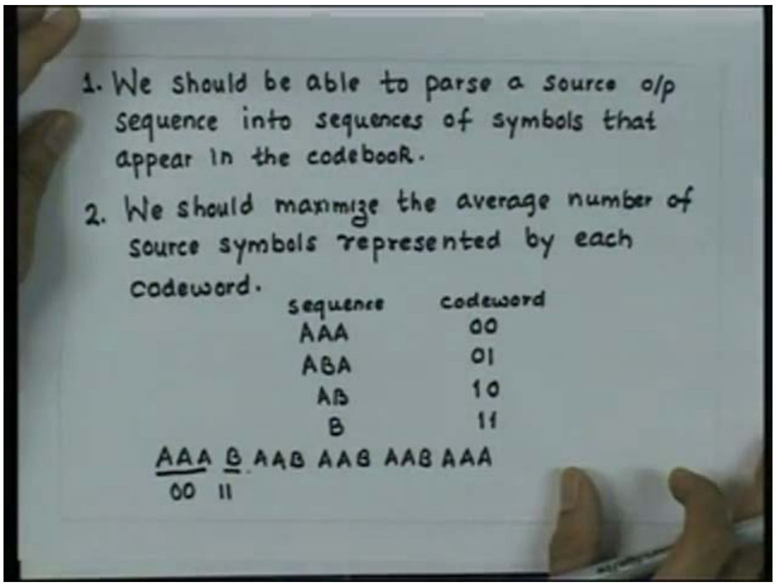
$\underbrace{AAA}_{00}\ \underbrace{B}_{11}\ \underbrace{AAB}_{01}\ \underbrace{AAB}_{01}\ \underbrace{AAB}_{01}\ \underbrace{AAA}_{00}$

So, let us look at the design of the Tunstall code. Most of the variable length coding algorithms that we have studied encode letters from the source alphabet using code words with varying numbers of bit. What it means that code words with fewer bits for letters that occur more frequently and code words with more bits for letters that occur less frequency. The Tunstall codes is an important exception. In the Tunstall code all code words are of equal length.

However, each code word represents a different number of letters. An example of a 2 bit Tunstall code would be for an alphabet X given by say A and B would be shown in this figure is the sequence and I have a code word. So, if I have a sequence A A A the code word would be 0 0, if I have a sequence A A B the code word is 0 1, if it is A B it is 1 0 and if it is B it is 1 1. So, this is an example of 2 bit Tunstall code for the alphabet A B. The main advantage of a Tunstall code is that errors in code words do not propagate unlike other variable length codes such as Huffman codes in which an error in one code word will cause a series of errors to occur.

Now, let us take this 2 bit Tunstall code and encode the following sequence. So, if I have a sequence given by triple A B and let us use this 2 bit Tunstall code. So, the first thing is basically you pass the source output sequence into sequences of symbols that appear in the code book. So, the first step is when I get the sequence I can pass it as sequences like this. And now for this sub sequences in the source sequence you code it using this 2 bit Tunstall code which we have given here. So, A A would be 0 0, B would be 1 1, A A B would be 0 1 0 0. Now, the design of a code that has a fixed code word lengths, but a variable number of symbols for code word should satisfy the following conditions.

(Refer Slide Time: 34:07)



1. We should be able to parse a source o/p sequence into sequences of symbols that appear in the codebook.
2. We should maximize the average number of source symbols represented by each codeword.

sequence	codeword
AAA	00
ABA	01
AB	10
B	11

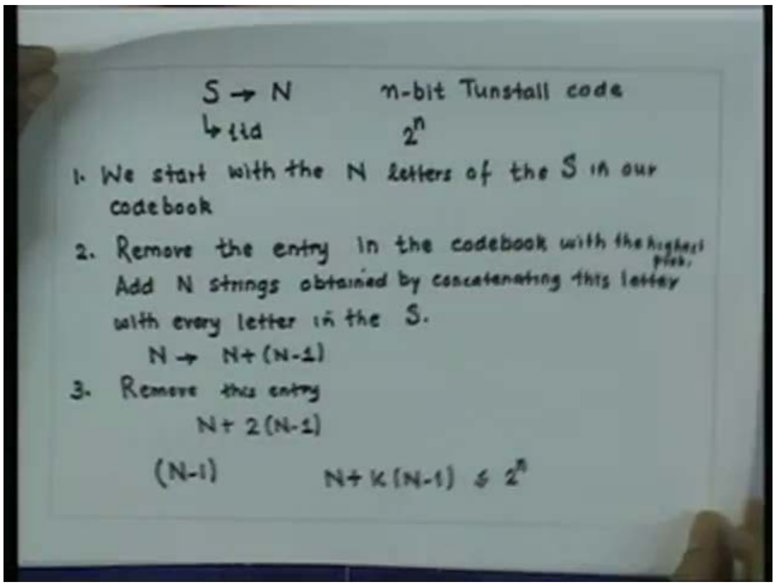
$\underset{00}{\underline{AAA}}\ \underset{11}{\underline{B}}$ AAB AAB AAB AAA

So, the first condition which it should satisfy is we should be able to pass a source output sequence into sub sequences of symbols that appear in the code book. Second, we should maximize the average number of source symbols represented by each code word. Now, in order to understand what we mean, what we mean by the first condition consider codes shown in the table below. I have a sequence here and the code word as follows for A A I have 0 0, for A B A I have 0 1, A B I have 1 0 and B I have 1 1. Now, actually this an example of 2 bit non Tunstall code.

Let us see if I have a code of this form. Then can I represent it my the original sequence which I considered earlier of the form A A, B, A A B. If we take this source output sequence and then the first thing we should do is we should able to pass a source output sequence into sub sequences of symbols that appear in the code book. So, if I get this I can pass it as it like this. So, for 0 0 because available in the code I can give a 0 0, for B i can put as 1 1, but now, from this point onwards I cannot pass this sequence into sequences of symbols that appear in the code book.

So, this sequence is uncodable using this particular code and this is not a desirable condition. So, Tunstall he gave a simple algorithm that fulfills this condition of able to pass a source output sequences into a sequences of symbol that appear in the code book. The algorithm is as follows. So, the first things is suppose we want an n bit Tunstall code for the source that generates identically independent distributed letters from an alphabet of size capital N.

(Refer Slide Time: 38:34)



$S \rightarrow N$ n-bit Tunstall code

$\hookrightarrow$ iid 2^n

1. We start with the N letters of the S in our codebook
2. Remove the entry in the codebook with the highest prob. Add N strings obtained by concatenating this letter with every letter in the S.

$N \rightarrow N + (N-1)$

3. Remove this entry

$N + 2(N-1)$

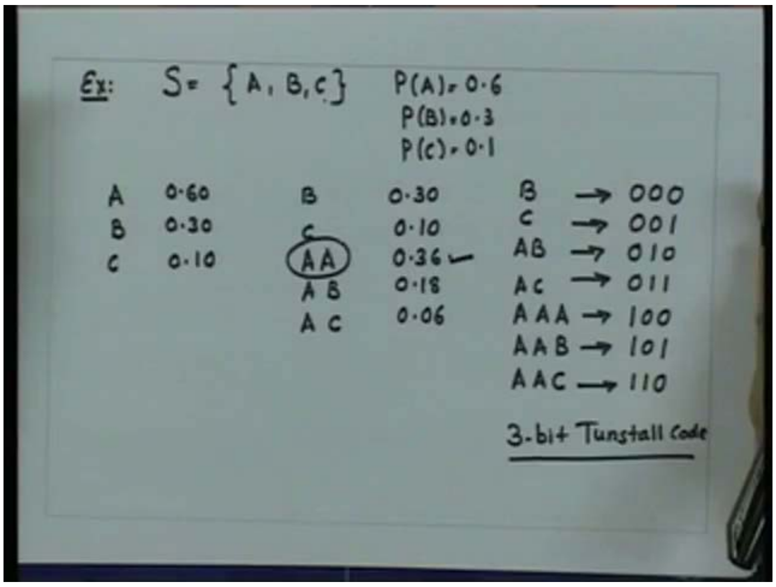
$(N-1)$ $N + k(N-1) \leq 2^n$

So, the alphabet size that is the source alphabet is N and we require to generate n bit Tunstall code for this source. And this source generates identically independent distributed letters. So, the number of code words which we can have would be equal to 2 raise to n. So, the first thing is to, we start with the N letters of the source alphabet in our code book. So, we place in our code book the N letters of the source alphabet. Then the next step is remove the entry in the code book that has the highest probability. And then you add N strings obtained by concatenating this letter which you have removed with every letter in the alphabet.

So, add N strings obtained concatenating this letter which we have removed with every letter in the source alphabet including that letter itself. So, this will increase the size of the code book from N to N plus N minus 1. And the probabilities of the new entries will be the product of the probabilities of the letters concatenated to form the new entry. And now, look through the N plus N minus 1 entries in the whole book and find the entry that has the highest probability keeping in mind that the entry with the highest probability may be a concatenation of symbols.

Then, once we have found out the, that symbol with the highest probability, remove this entry and add the N strings obtained by concatenating this entry which we have removed with every letter in the alphabet. So, again you repeat the same procedure of the step 2 of obtaining N strings by concatenating the letter which we are removing from the code book. Now, the size of code book will become N plus 2 N minus 1. So, each time we perform this operation we increase the size of the code book by N minus 1. Therefore, this operation can be performed K times where N plus K N minus 1 should be less than equal to 2 raise to n which is the number of code books, code words which we can form out of n bit. Let us take a simple example to understand this.

(Refer Slide Time: 43:33)



Ex: $S = \{A, B, C\}$ $P(A) = 0.6$ $P(B) = 0.3$ $P(C) = 0.1$

A	0.60	B	0.30	B	$\rightarrow$ 000
B	0.30	C	0.10	C	$\rightarrow$ 001
C	0.10	$\boxed{AA}$	0.36 ✓	AB	$\rightarrow$ 010
		AB	0.18	AC	$\rightarrow$ 011
		AC	0.06	AAA	$\rightarrow$ 100
				AAB	$\rightarrow$ 101
				AAC	$\rightarrow$ 110

3-bit Tunstall Code

So, if I have a source consisting of a 3 alphabets with probability of A equal to 0.6, probability of B equal to 0.3 and probability of C equal to 0.1. Then the first thing you do is you start with the three entries A B C, this have probability 0.60, 0.30, 0.10. In this the first step, letter A has the highest probability. So, remove it. So, what is left out is B and C and then you get 3 more strings because the size of the alphabet is 3, get 3 more strings which have been obtained by concatenating the letters from this source to this A.

So, I will get A A, A B and A C. So, the probabilities would be 0.30, 0.10, 0.36, 0.18 and 0.06. Now, again from here I find this is of the highest probability. So, remove it. So, what is left out is B C, A B, A C and now after I remove this I get 3 more strings by concatenating the letters from this alphabet to this entry. So, I get A A B and A A C and now if I carry out one more iteration I will get 9 sequences. And since 9 exceeds 2 raise to 3 I have to stop at this point. And now, once I have reached this I can give the code word as 0 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0.

So, this I have my code words which I have formed for this sequence and this would be 3 bit Tunstall code. So, using this 3 bit Tunstall code you can again code the earlier sequence. So, we have looked at some of the coding strategies and we have seen that Huffman coding requires I mean it is the optimal and it also requires the knowledge of the probabilities of the source sequence. Now, for Huffman coding which is an optimal coding scheme if this knowledge is not available then Huffman coding becomes a 2 pass procedure.

In the first pass the statistics are collected and in the second pass the source is encoded. Now, in order to convert this algorithm into a 1 pass procedure adaptive Huffman coding algorithms have been developed. The algorithm constructs the Huffman code based on the statistics of the symbols already encountered. In the next few class we will have a look at the design of adaptive Huffman code.