Data Compression

Chapter – 1
Introduction

Anokhi Shah

In the last decade we have been witnessing a transformation—some call it a revolution—in the way we communicate, and the process is still under way. This transformation includes the ever-present, ever-growing Internet; the explosive development of mobile communications; and the ever-increasing importance of video communication. Data compression is one of the enabling technologies for each of these aspects of the multimedia revolution. It would not be practical to put images, let alone audio and video, on websites if it were not for data compression algorithms. Cellular phones would not be able to provide communication with increasing clarity were it not for compression. The advent of digital TV would not be possible without compression. Data compression, which for a long time was the domain of a relatively small group of engineers and scientists, is now ubiquitous. Make a long-distance call and you are using compression. Use your modem, or your fax machine, and you will benefit from compression. Listen to music on your *mp3* player or watch a DVD and you are being entertained courtesy of compression.

Data Compression

- Data compression is art or science of representing information in a compact form
- Compact form is created by identifying and using structure that exist in the data.
  - Structure could be any Pattern, Redundancy or Model.
  - Example Data – AAABBBCCC (Every character is repeated 3 times)
- Fewer bits required to represent information

Examples of Compressed data formats

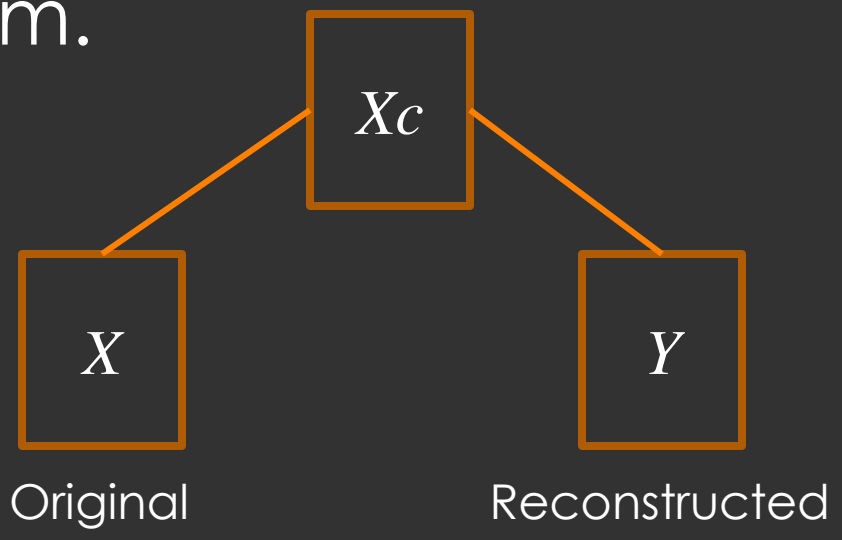
- ZIP File
- JPEG (Image)
- MP3 (Sound)
- SMS type English
  - Information : Info
  - About : Abt
  - Are: r
  - You : u
- Morse Code

Advantages & Disadvantages

- Reduce the consumption of expensive resources such as Hard disk, Bandwidth
- Faster transformation of Data
- But, compressed data can only be understood if the decoding method is known by the receiver
- Compressed data must be decompressed to be viewed or heard.

1.1 Compression Techniques

- There is a compression algorithm that takes an input X
- generates a representation Xc to generate the reconstruction Y.
- By Compression algorithm, we mean Compression as well as Reconstruction algorithm.



1.1.1 Lossless Compression

- Y is identical to X
- No loss of information
- Lossless compression is reversible so that the original data can be reconstructed exactly
- Generally used for applications that can't tolerate any difference between the original and reconstructed data
- Lossless compression algorithms usually exploit statistical redundancy

Lossless Compression Examples

- Text Compression
  - It is very important that reconstruction is identical to original
  - Very small difference can result in statements with different meanings
  - For a Bank application consider the sentences "Do not send money" and "Do now send money"
- Radiological / Satellite image
  - If data of any kind are to be processed or enhanced later to yield more information,
  - It is important that integrity be preserved.

1.1.2 Lossy Compression

- Y is different from X
- Some loss of information
- Compressed data cannot be recovered or reconstructed exactly
- Obtains much higher compression ratio than is possible with lossless compression
- Generally used for applications that can afford to have difference between the original and reconstructed data

Lossy Compression Examples

- Video / Image
  - a person viewing a picture or television video scene might not notice if some of its finest details are removed or not represented perfectly
- Sound / Speech
  - Both parties of the telephone conversation can ignore certain loss of information.

1.1.3 Measure of Performance

- A compression algorithm can be evaluated in a number of different ways
  - Measure relative complexity of the algorithm
  - Memory required to implement the algorithm
  - How fast the algorithm performs on a single machine
  - Amount of compression
  - How closely the reconstruction resembles the original

Compression Ration

- Number of bits required to represent the data before compression to the number of bits required to represent the data after compression
- Suppose an image requires 65536 bytes for storage, After compression it requires 16384 bytes
- Compression ratio is 4:1
- Ratio can be expressed as the reduction in the amount of data required as a percentage of the size of the original data
- ((65536-16384) / 65536) * 100 =75%

Compression Rate

- The average number of bits required to represent a single sample
- If we assume 8 bits per byte (pixel), the average number of bits per pixel in the compressed representation is 2
- Thus we would say that rate is 2 bits per pixel.

Distortion / Fidelity / Quality

- The difference between the original and the reconstruction is called the distortion
- But when you talk about the difference between reconstruction and original its called fidelity/ quality
- i.e Y-X = Fidelity or Quality, X-Y =Distortion
- When fidelity or quality of a reconstruction is high, the difference between the reconstruction and original is small
- This difference could be mathematical or perceptual

1.3 Modeling and coding

- Compression scheme depends on various factors like characteristic of the data.
- A compression technique that works on text would not work on image because both have different kind of data
- Data compression Algorithm development is in 2 phases
  - Modeling
  - Coding

Modeling and coding

- In modeling phase we try to extract the information about any redundancy that exist in the data and describe it in terms of a model
- Coding is the description of the model and a description of how the data differs from the model are encoded generally using a binary alphabet
- The difference between data and model is referred as the residual.

Example 1.2.1

- On page no 7
- We are required to store the binary representation of the numbers.
- Max no is 21, which will require 5 bits because binary of 21 is 10101
- So rate is 5 bits per sample
- By plotting the data as seen in figure 1.2 (Pg no 7) ,the data falls on a straight line with an equation Xn'= n + 8
- Xn' is the model and En is the residual

- Residual sequence consist of 3 numbers which can be represented in binary using 2 bits

n	Xn	Xn' = n+8	En=Xn-Xn'
1	9	9	0
2	11	10	1
3	11	11	0
4	11	12	-1
5	14	13	1
6	13	14	-1
7	15	15	0
8	17	16	1
9	16	17	-1
10	17	18	-1
11	20	19	1
12	21	20	1

-1	00
0	01
1	10

We need to make use of structure that exist in data to predict the value of each element in the sequence and then encode the residual.

Note

- The more ways we have of looking at the data, the more successful we will be in developing compression schemes that take full advantage of the structures in the data.

There is a saying attributed to Bobby Knight, the basketball coach at Texas Tech University: "If the only tool you have is a hammer, you approach every problem as if it were a nail." Our intention in this book is to provide you with a large number of tools that you can use to solve the particular data compression problem. It should be remembered that data compression, if it is a science at all, is an experimental science. The approach that works best for a particular application will depend to a large extent on the redundancies inherent in the data.

- Refer text book for more examples.

Data Compression

Chapter – 2

Mathematical Preliminaries

for Lossless Compression

Anokhi Shah

Information Theory

- Mathematical foundation of Information Theory was done by Claude Elwood Shannon an electrical engineer at Bell Labs 1948.
- He defined a quantity called self−information.
- Suppose an event A, which is a set of outcomes of some random experiment.
- If P(A) is the probability that the event A will occur, then the self−information i(A) associated with A is given by,

$$i(A) = \log_b \frac{1}{P(A)}$$

Anokhi Shah

The reason for the log

- Why does the use of the logarithm makes sense in this context?
  - Recall that log(1) = 0 and −log(x) increases as x decreases from one to zero.
  - So, if the probability of an event is low, the self−information is high.
  - if the probability of an event is high, the information associated with it is low.
- This definition makes sense even from a non mathematical point of view
  - The barking of a dog during a burglary is a high probable event and therefore does not contain much information.
  - However , if the dog did not bark during a burglary, its a low probability event and contains a lot of information.

Information of independent events

- Information obtained from the occurrence of two independent events is the sum of the information obtained from the occurrence of the individual events.
- Suppose A & B are two independent events. the self information associated with the occurrence of both events is,

$$i(AB) = \log_b \frac{1}{P(AB)}$$

$$P(AB) = P(A)P(B)$$

$$i(AB) = \log_b \frac{1}{P(A)P(B)}$$

$$= \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)}$$

$$= i(A) + i(B)$$

Anokhi Shah

2.2.1 Example

- Let H and T be the outcomes of flipping coin. If the coin is fair, then P(H)=P(T)=½
- i(H)= i(T)=1bit
- If the coin is not fair, then we would expect the information associated with each event to be different. Suppose P(H)=1/8, P(T)=7/8
- Then i(H)= 3 bits i(T)= 0.193 bits

Entropy

- If the experiment is a source that puts out symbols Ai from a set A
- Entropy is a measure of the average number of binary symbols needed to code the output of the source.

$$H(S) = -\Sigma P(X) \log P(X)$$

or

$$H(S) = \Sigma P(X)\, i(X)$$

Example

- Consider the following sequence:

1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10

ai	P(ai)	1/P(ai)	i(ai)	P(ai) i(ai)
1	1/16	16	4	4/16
2	2/16	8	3	6/16
3	2/16	8	3	6/16
4	2/16	8	3	6/16
5	2/16	8	3	6/16
6	1/16	16	4	4/16
7	1/16	16	4	4/16
8	2/16	8	3	6/16
9	2/16	8	3	6/16
10	1/16	16	4	4/16
			H(S)=	**3.25 bits / sample**

Lossy Compression Examples

- However, if we assume that there was sample−to−sample correlation between the samples and we remove the correlation by taking differences of neighboring sample values,

1	2	3	2	3	4	5	4	5	6	7	8	9	8	9	10
1	1	1	-1	1	1	1	-1	1	1	1	1	1	-1	1	1

- This sequence is constructed using only two values with probabilities P(1)=13/16 and P(−1)=3/16.
- The entropy in this case is 0.70 bits per symbol.
- Of course, knowing only this sequence would not be enough for the receiver to reconstruct the original sequence.

Anokhi Shah

- The receiver must also know the process by which this sequence was generated from the original sequence.
- The process depends on our assumptions about the structure of the sequence.
- These assumptions are called the model for the sequence.
- In this case, the model for the sequence is $x_n = x_{n-1} + r_n$
- Xn: nth element of the original sequence
  rn: nth element of the residual sequence.

Static and Adaptive Model

- static model
  - Parameters do not change with n.
- Adaptive Model
  - Parameters change or adapt with n to the changing characteristics of the data

Example

- Consider sequence: 1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2
- If we look at it one symbol at a time the probabilities are P(1) = P(2) = 0.25 and P(3) = 0.5 with entropy of 1.5 bits/symbol.
- This particular sequence consists of 20 symbols; so 30 bits required to represent sequence
- If we consider block of 2, there are only 2 symbols {1 2} and {3 3}
- The probabilities are P({1 2}) = P({3 3}) = 0.5, with entropy of 1 bit/symbol.
- 10 such symbols; so 10 bits required to represent sequence – a reduction of a factor of three.
- The theory says we can always extract the structure of the data by taking larger and larger block sizes; in practice, there are limitations to this approach.
- To avoid these limitations, we try to obtain an accurate model for the data code the source with respect to the model.

2.4 Coding

- Coding is the assignment of binary sequences to elements of an alphabet.
- The set of binary sequences is called a *code*, and the individual member of the set is called a *codeword*.
- An alphabet is a collection of symbols called *letters*.
- If a code uses the same number of bits to represent each symbol then its called *fixed-length code*. Eg ASCII code
- The average number of bits per symbol is often called the *rate* of the code.

2.4.1 Uniquely Decodable Code

- Average length of Code $\ell = \sum P(ai)n(ai)$
- n(ai) is the number of bits in the codeword for letter ai
- The average length is given in bits/symbol.

Letters	Probability	Code 1	Code 2	Code 3	Code 4
a1	0.5	0	0	0	0
a2	0.25	0	1	10	01
a3	0.125	1	00	110	011
a4	0.125	10	11	111	0111
	Average length	1.125	1.125	1.75	1.875

Unique Decodability

- We would like each symbol to be assigned a unique codeword
- Unique decodability means that any given sequence of codewords can be decoded in one, and only one way.
- Code 1 is not uniquely decodable as 0 is codeword for both a1 & a2
- In code 2 if receiver has to decode the sequence 001 then it could be decoded to a1a1a2 or a3a2; so its not unique

Code 3

- In code 3 the first three codewords all end in a 0.
- In fact, a 0 always denotes the termination of a codeword.
- The final codeword contains no 0´s and is 3 bits long.
- Because all other codewords have fewer than three 1s and terminate in a 0, the only way we can get three 1´s in a row is as a code for a4.
- The decoding rule is simple - Accumulate bits until you get a 0 or until you have three 1´s.
- There is no ambiguity in this rule, so easy to see that this code is uniquely decodable.

Code 4

- Each codeword starts with a 0, and the only time we see a 0 is in the beginning of a codeword.
- Decoding rule - Accumulate bits until you see a 0.
- The bit before the 0 is the last bit of the previous codeword.
- In the case of Code 3, the decoder knows the moment a code is complete.
- In Code 4, we have to wait till the beginning of the next codeword before we know that the current codeword is complete.
- Because of this property, Code 3 is called an *instantaneous code*.

Coding

- Suppose two binary codewords a and b, where a is k bits long, b is n bits long, and $k < n$.
- If the first k bits of b are identical to a, then a is called a *prefix* of b.
- The last $n - k$ bits of b are called the *dangling suffix*.
- For example, if a = 010 and b = 01011, then a is a prefix of b and the dangling suffix is 11.

Anokhi Shah

Test for Unique Decodability

- Construct a list of all codewords.
- Examine all pairs of codewords to see if any codeword is a prefix of another codeword.
  - Whenever you find such a pair, add the dangling suffix to the list unless you have added the same dangling suffix to the list in a previous iteration.
  - Now repeat the procedure using this larger list.
  - Continue in this fashion until one of the following two things happens:
    1. You get a dangling suffix that is a codeword.
    2. There are no more unique dangling suffixes.
- If you get the first outcome, the code is not uniquely decodable.
- However, if you get the second outcome, the code is uniquely decodable.
- For more info refer Slide 24

Example Code 5

- Code 5 {0,01,11}
- The codeword 0 is a prefix for the codeword 01 and The dangling suffix is 1.
- Let us augment the codeword list with the dangling suffix: {0,01,11,1}
- Also, 1 is a prefix of 11. This gives us a dangling suffix of 1, which is already in the list.
- There are no other pairs that would generate a dangling suffix, so we cannot augment the list any further.
- Therefore, Code 5 is uniquely decodable.

Example Code 6

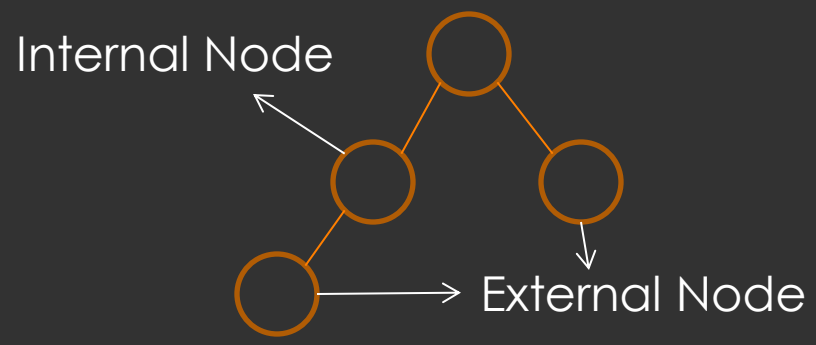
- Code 6 {0,01,10}
- The codeword 0 is a prefix for the codeword 01. The dangling suffix is 1.
- Augmenting the codeword list with 1, we obtain the list {0,01,10,1}
- In this list, 1 is a prefix for 10. The dangling suffix for this pair is 0, which is the codeword for a1.
- Therefore, Code 6 is not unique decodable.

Prefix Code

- A prefix code is a code in which no codeword is a prefix of the other.
- In this case, the set of dangling suffixes is the null set, and code in which no codeword is a prefix to another codeword is called a prefix code.
- How to check for a prefix code??? A simple way to check it is to draw the rooted binary tree of the code.
- The tree starts from the root node and has two possible branches at each node, one corresponds to a 1 and the other to a 0.
- **Convention**: we start with the root node at the top, the left branch corresponds to a 0 and the right branch corresponds to a 1.
- Using this convention, we can draw the binary tree for Code 2, 3 & 4 as given in figure 2.4 Pg 31

Nodes

- Internal nodes
  - parent nodes
- External nodes
  - Leaf nodes
  - Don't have any child nodes
  - Prefix codes are associated with ONLY external nodes



Note

- This slides are not enough material for examination
- You need to refer to your reference book for more details
- For your Practice refer 2.6 Projects and problems
- Answers of the exercise are given below

3. Entropy
   - (a) 2 bits/Symbol
   - (b) 1.75 bits/Symbol
   - (c) 1.74 bits/Symbol

7. Unique Decodability
   - (a) Not
   - (b) Not
   - (c) Yes & Prefix
   - (d) Not

Unique Decodability Test Logic

1. Construct Original List of Codewords
2. Take one by one code from the list and call it *a*. *a* is k bits long.
3. Against *a* list all possible values of code *b* where code *b* should be n bits long and n>k
4. If you find a dangling suffix from the possible values of *b* then
   - If dangling suffix is a codeword of original list then
     - Code is not unique; STOP
   - Else
     - Add dangling suffix to new augmented list and go to step2
5. If list is exhausted then code is unique.

Anokhi Shah

2.6 7. (a) Example

1. { 0 , 01, 11, 111}
2. a=0
3. a=0 b=01 df: 1 (Add it to the list)
   b=11 X
   b=111 X
4. We found a dangling suffix 1
   - Dangling suffix is not a codeword of original list then Add 1 to new augmented list { 0 , 01, 11, 111, 1} and go to step2

- So continue like this for 01,11,111 for 1 have a look at following steps

1. a=1
2. a=1 b=01 X
   b=11 df: 1 (Already in List)
   b=111 df: 11 (Codeword in original list)
3. We found a dangling suffix 11
   - Dangling suffix is a codeword of original list then
   - Code is not unique; STOP

Data Compression

Chapter – 3
Huffman Coding

Anokhi Shah

3.2 Huffman Coding Algorithm

- Lossless compression technique
- Huffman Codes are optimum prefix codes, variable length coding.
- Huffman procedure is based on two observations regarding optimum prefix codes.
  1. In an optimum code, symbols that occur more frequently (have a high probability of occurrence) will have shorter codewords than symbols that occur less frequently.
  2. In an optimum code, the two symbols that occur least frequently will have the same length.

Additional Requirement

- Huffman procedure is obtained by adding a simple requirement to two observation
- Codewords corresponding to the two least probable symbols differ only in the last bit.
- That is if γ and δ are the two least probable symbols in an alphabet, if the codeword for γ was m * 0, the codeword for δ would be m*1.
- Here m is string of 1 and 0's and * denotes concatenation

3.2.1 Design of Huffman code

- Sort letters in a descending probability order.
- Combine least two probable letters
  - Calculate the new combined probability
  - Assign the new combined probability to a new letter a'
- Continue in this manner till you find the combined probability as 1

3.2.1 Example

Letter	Probability
a2	0.4
a1	0.2
a3	0.2
a4	0.1
a5	0.1

0

1

a'4 0.2

Letter	Probability
a2	0.4
a1	0.2
a3	0.2
a'4	0.2

0

1

a'3 0.4

Letter	Probability
a2	0.4
a'3	0.4
a1	0.2

0

1

a''3 0.6

Letter	Probability
a''3	0.6
a2	0.4

0

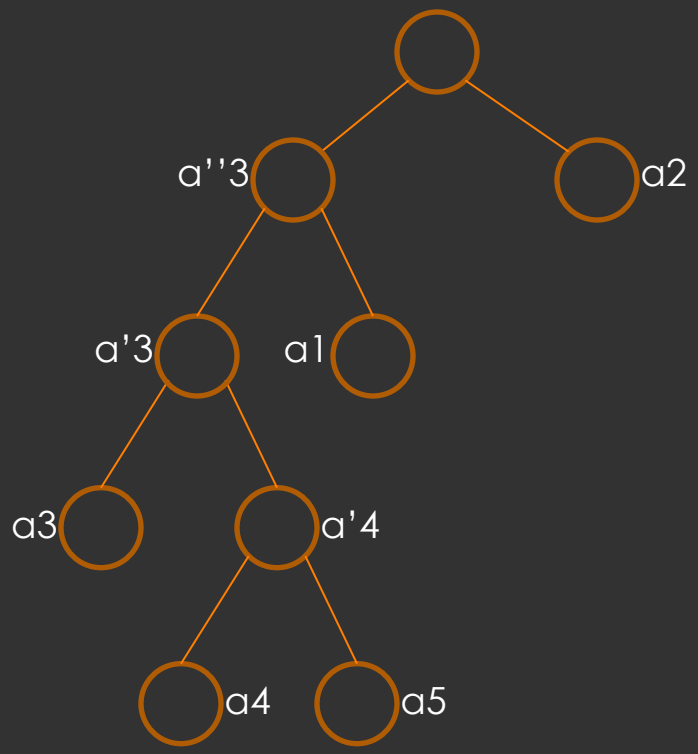
1

1

Anokhi Shah

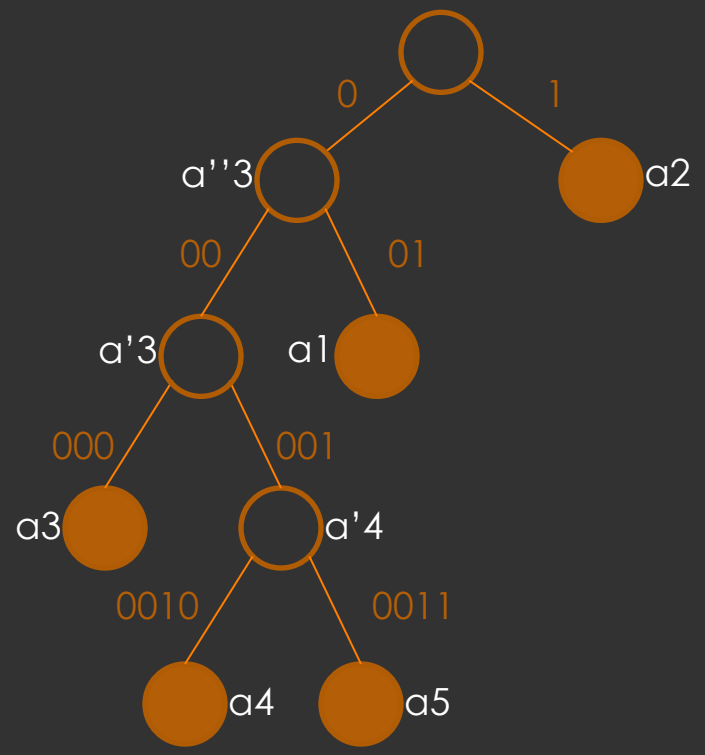
Building Huffman Code

- Huffman Code, by virtue of being a prefix code can be represented as a binary tree.
- So like prefix code Huffman code would be associated with an external node.
- Based on sorted tables created previously construct binary tree starting from the last table to first table.



3.2.1 Example's Huffman Code

- To obtain code traverse the tree from root node to each leaf node
  - Assigning a 0 to each left traversal
  - Assigning a 1 to each right traversal



Letter	Probability	Code
a2	0.4	1
a1	0.2	01
a3	0.2	000
a4	0.1	0010
a5	0.1	0011

Anokhi Shah

Redundancy

- Average length of Code ℓ is 2.2 bits/Symbol and entropy H(s) is 2.122 bits/Symbol
- A measure of the efficiency of the code is its Redundancy – the difference between the entropy and the average length
- Redundancy is 0.078 bits/symbol or 3.67% of the entropy that is (redundancy / entropy) * 100 would give %

Minimum Variance Huffman Code

- To obtain the Huffman code with minimum variance, we always put the combined letter as high in the list as possible

Letter	Probability
a2	0.4
a1	0.2
a3	0.2
a4	0.1
a5	0.1

0

a'4 0.2

1

Letter	Probability
a2	0.4
a'4	0.2
a1	0.2
a3	0.2

0

a'1 0.4

1

Letter	Probability
a'1	0.4
a2	0.4
a'4	0.2

0

a'2 0.6

1

Letter	Probability
a'2	0.6
a'1	0.4

0

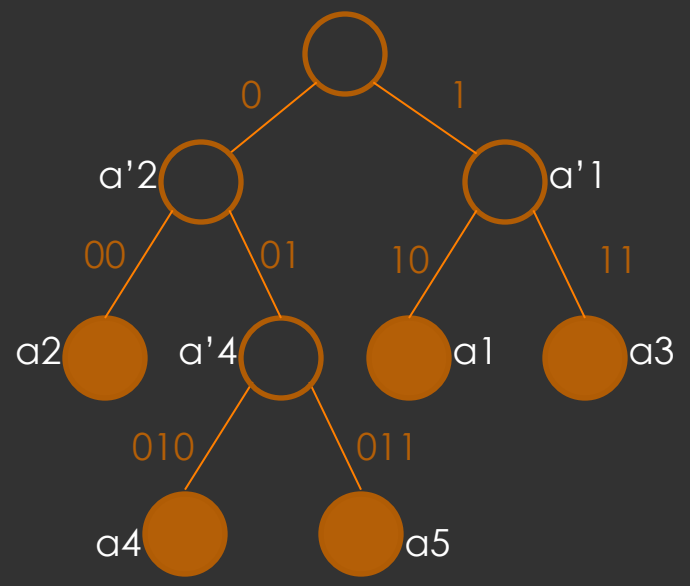
1

1

Anokhi Shah

3.2.1 Minimum Variance Huffman Code

- To obtain code follow same method as in Huffman code



Letter	Probability	Code
a1	0.2	10
a2	0.4	00
a3	0.2	11
a4	0.1	010
a5	0.1	011

it seems reasonable to elect to use the second code instead of the first. To obtain the Huffman code with minimum variance, we always put the combined letter as high in the list as possible.

3.2.4 Extended Huffman code

- Extended Huffman code is used when the alphabet size is small and probability of the occurrence of different letters is skewed.
- When we encode the output of the source in longer blocks of symbols we are guaranteed a rate closer to entropy.
- Instead of generating a codeword for every symbol, we will generate a codeword for say every two symbols
- If we look at the source sequence two at a time, the number of possible symbol pairs, or size of the extended alphabet is $3^2=9$
- To obtain the extended Huffman code , follow same method as of Huffman code.

Example 3.2.4

- A= {a1,a2,a3}, p(a1)=0.8, p(a2)=0.02,p(a3)=.18
- Probability of Letter a1a1 is obtained by P(a1).P(a1)
- The average codeword length is 1.7228 bits/symbol.
- However, each symbol in the extended code correspond to 2 symbols in original alphabet

Letter	Probability	Code
a1a1	0.64	0
a1a2	0.016	10101
a1a3	0.144	11
a2a1	0.016	101000
a2a2	0.0004	10100101
a2a3	0.0036	1010011
a3a1	0.1440	100
a3a2	0.0036	10100100
a3a3	0.0324	1011

Therefore, in terms of original alphabet, the average codeword length is 1.7228/2 = 0.8614 bits/symbol

3.3 Nonbinary Huffman Coding

- The binary Huffman coding procedure can be easily extended to nonbinary case where code elements come from m-ary alphabet, and m is not equal to 2.
- Non binary Huffman procedure is based on two observations regarding optimum prefix codes.
  1. In an optimum code, symbols that occur more frequently (have a high probability of occurrence) will have shorter codewords than symbols that occur less frequently.
  2. In an optimum code, the m symbols that occur least frequently will have the same length.

Additional Requirement

- Nonbinary Huffman procedure is also obtained by adding a simple requirement to two observation
- Codewords corresponding to the m least probable symbols differ only in the last bit.

Problem

- Consider the design of a teranry code for a source with six letter alphabet.
- Using the rules mentioned before we will first combine the three letters with lowest probability into a combined letter.
- This will give us a reduced alphabet of 4 letters
- Again performing the same step to combine 3 letters we will result in alphabet of size 2
- We have three values to assign and only 2 letters !!!!

Alternative

- Instead of combining 3 letters at the beginning, we could have combined 2 letters.
- This would result in reduced alphabet of size 5.
- If we combined 3 letters from this alphabet, we would end up with a final reduced alphabet of size 3.
- So which alternative to choose????

Correct Approach

- If at some stage we are allowed to combine less than m symbols, the logical place to do this would be very first stage.
- So in case of m-ary code and M-letter alphabet how many letters we should combine in first place? Lets call it m'
- Possible values of m' is between 2 and m
- The value to be taken should be equal to M mod (m-1)
- That is M mod (m-1) = m' mod (m-1)

Example 3.3.1

- M=6 ternary code => m=3 m-1=2
- Possible values of m' is between 2 and m
- M mod (m-1)= 6 mod (3-1) = 6 mod 2 =0

  m '=2 2 mod (3-1) = 2 mod 2 =0 √

  m'=3 3 mod (3-1) = 3 mod 2 =1
- So we will combined 2 letters in the first step and then we'll continue combining 3 letters
- That means rest of method is same as Huffman code

Letter	Probability
a5	0.25
a1	0.20
a3	0.20
a4	0.20
a6	0.10
a2	0.05

0

1

a'6 0.15

Letter	Probability
a5	0.25
a1	0.20
a3	0.20
a4	0.20
a'6	0.15

0

1

2

a'3 0.55

Letter	Probability
a'3	0.55
a5	0.25
a1	0.20

0

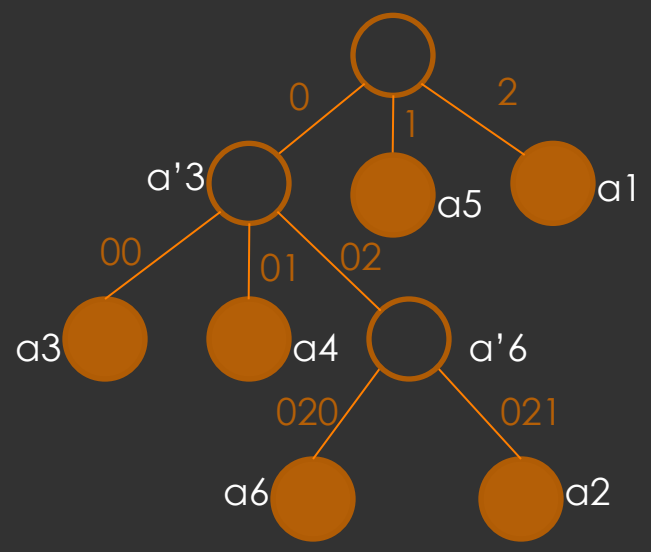
1

2

1

3.3.1 Eg's nonbinary Huffman Code

- To obtain code traverse the tree from root node to each leaf node
  - Assigning a 0 to each left traversal
  - Assigning a 1 to each middle traversal
  - Assigning a 2 to each right traversal



Letter	Probability	Code
a1	0.20	2
a2	0.05	021
a3	0.20	00
a4	0.20	01
a5	0.25	1
a6	0.10	020

Anokhi Shah

3 .5 Golomb Codes

- The Golomb codes belong to a family of codes designed to encode integers 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

1. Using '1' is : n 1s followed by a 0
   - e.g. code for 4 is 11110, code for 7 is 11111110
2. Using '0' is : n 0s followed by a 1
   - e.g. code for 4 is 00001, code for 7 is 00000001

3 .5 Golomb Codes

- One step higher in complexity are a number of coding schemes that split the integer into two parts, representing one part with a unary code and the other part with a different code.
- An example of such a code is the Golomb code.
- The Golomb code is parameterized by an integer $m > 0$.
- We represent an integer $n > 0$ using two numbers q and r, where $q = floor(n/m)$ and

  $r = n - qm$

3 .5 Golomb Codes

- In other words, q is the quotient and r is the remainder when n is divided by m.
- The quotient q can take on values 0, 1, 2, ... and is represented by the unary code of q
- The remainder r can take on the values 0, 1, 2, ... $m - 1$
- Golomb code for an integer n is
  - q's unary code(using '1') appended by r's code

Finding Code for Remainder r

- $C = ceiling\ (\log_2 m)$
- $F = floor\ (\log_2 m)$
- $X = 2^C - m$
- Possible values for r is 0 to $m\text{-}1$
- To generate the code for r
  - For the first C possible values of r : f bit binary of r
  - For the rest of the values of r : C bit binary of $r + X$

Anokhi Shah

Example : 3.5.1

- If m = 5 then r can be 0,1, 2, 3 and 4
  - C = 3, $\mathcal{F}$ = 2, $\mathcal{X}$ = 8-5 = 3
- To generate the code for r
  - For the first 3 possible values of r (viz. 0, 1 ,2)
    - Find 2 bit binary of r
  - For the rest of the values of r (viz. 3, 4)
    - Find 3 bit binary of r+3

r	r+3	2 bit binary of r	3 bit binary of r+3
0		00	
1		01	
2		10	
3	6		110
4	7		111

Example : 3.5.1

- Golomb code for n = 13
  - 13/5 : q = 2, r = 3 code for q is 110, code for r is 011
  - Golomb code for n = 13 will be 110011
- Golomb code for n = 0
  - 0/5 : q = 0, r = 0 code for q is 0, code for r is 00
  - Golomb code for n = 0 will be 000
- Golomb code for n = 1
  - 1/5 : q = 0, r = 1 code for q is 0, code for r is 01
  - Golomb code for n = 1 will be 001

3.6 Rice Code

- Rice code is adaptive Golomb code
- In the Rice code, a sequence of nonnegative integers (which might have been obtained from the pre-processing of other data) is divided into blocks of J integers each.
- Each block is then coded using one of several options, most of which are a form of Golomb codes.

Pre-Processing for Rice Code

- Given a sequence $\{y_i\}$, for each y_i we generate a prediction y'_i
- A simple way to generate a prediction would be to take the previous value of the sequence to be a prediction of the current value of the sequence: $y'_i = y'_{i-1}$
- We then generate a sequence whose elements are the difference between y'_i and its predicted value y'_i such as $d_i = y_i - y'_i$
- The d_i value will have a small magnitude when our prediction is good else a large value.

Pre-Processing for Rice Code

- Let y_{max} and y_{min} be the largest and smallest values that the sequence $\{y_i\}$ takes on.
- It is reasonable to assume that the value of y will be confined to the range [y_{max} , y_{min}]
- Define $T_i = \min(y_{max} - y', y' - y_{min})$
- The sequence $\{d_i\}$ can be converted into a sequence of nonnegative integers $\{x_i\}$ using the following mapping

$$x_i = \begin{cases} 2d_i & 0 \le d_i \le T_i \\ 2|d_i| - 1 & -T_i \le d_i < 0 \\ T_i + |d_i| & otherwise \end{cases}$$

Rice Code

- {xi} is divided into segments with each segment being further divided into blocks of size J.
- Each block is then coded using one of the following options.

1. Fundamental sequence
   - This is a unary code.
   - Unary code using '1' or or using '0'

Rice Code

2. Split sample options
   - These options consist of a set of codes indexed by a parameter m.
   - The code for a k-bit number n using the mth split sample option consists of, The m least significant bits of k followed by a unary code representing the k-m most significant bits
   - For example, suppose we wanted to encode the 8-bit number 23 using the 3rd split sample option. The 8-bit representation of 23 is 00010111.
   - The three least significant bits are 111.
   - The remaining bits (00010) correspond to the number 2, which has a unary code 001.
   - Therefore, the code for 23 using the third split sample option is 111001.

Sample Values	4-bit Binary Representation	FS Code, $k = 0$	$k = 1$ 1 LSB + FS Code	$k = 2$ 2 LSB + FS Code
8	1000	000000001	0 00001	00 001
7	0111	00000001	1 0001	11 01
1	0001	01	1 1	01 1
4	0100	00001	0 001	00 01
2	0010	001	0 01	10 1
5	0101	000001	1 001	01 01
0	0000	1	0 1	00 1
3	0011	0001	1 01	11 1
Total Bits	32	38	29	29

Table 1-2: Examples of split-sample options using fundamental sequence codes

Rice Code

3. Second Extension option
   - Useful for sequences with low entropy-when, in general, many of the values of Xi will be zero.
   - The sequence is divided into consecutive pairs of samples.
   - Each pair is used to obtain an index γ using the following transformation:

$$\gamma = \frac{1}{2}(x_i + x_{i+1})(x_i + x_{i+1} + 1) + x_{i+1}$$

   - the value of γ is encoded using a unary code. The value of γ is an index to a lookup table with each value of γ corresponding to a pair of values x_i, x_{i+1}

Rice Code

4. Zero block option
   - The zero block option is used when one or more of the blocks of Xi are zero—generally when we have long sequences of yi that have the same value.
   - In this case the number of zero blocks are transmitted using the code shown in Table 3.17.
   - The ROS code (Remainder-of-Segment) is used when the last five or more blocks in a segment are all zero.

3.7 Tunstall Code

- Tunstall Code is a fix length code - all code words are of equal length.
- However, each codeword represents a different number of letters
- The main advantage of a Tunstall code is that errors in codewords do not propagate, unlike other variable-length codes, such as Huffman codes, in which an error in one codeword will cause a series of errors to occur.

Tunstall Code Algorithm

- Suppose we want an n-bit Tunstall code for an alphabet of size N.
- The number of codewords will be 2^n.

1. Remove the entry in the codebook that has the highest probability
2. Combine this removed letter with every letter in the alphabet and these combination to codebook (This will increase the size of the codebook by N-1)
3. Repeat above steps till $N + K(N-1) <= 2^n$ where K is the index of iteration E.g for 1st Iteration K =0, 2nd Iteration K=1 , 3rd Iteration K=2 …

That is – Loop till new size of the codebook is <= 2^n

Example 3.7.1

- Design a 3-bit Tunstall code for a source with the following alphabet: A = {A, B, C} where P(A)= 0.6, P(B)=0.3, F(C) = 0.1
- Size of the alphabet N =3, n = 3 bit
- Possible codewords = $2^n = 2^3 = 8$
- Each iteration will increase the size of codebook by 2 so loop till new size is <=8 or For each Iteration check $N + K(N-1) <= 2^n$

Iteration 1, k=0 **Current : 3<=8 Next : 5 <=8**	
Letter	**Probability**
A	0.6
B	0.3
C	0.1

Removing A and combining with all letter in alphabet that is with A,B & C for next iteration

New Size will be 5 and 5<=8
So, Perform next iteration

Example 3.7.1 (Iteration 2)

Iteration 2, k=1 Current : 5<=8 Next : 7 <=8	
Letter	**Probability**
B	0.30
C	0.10
AA	0.36
AB	0.18
AC	0.06

Removing A and combining with all letter in alphabet that is with A,B & C for next iteration

New Size will be 7 and 7<=8
So, Perform next iteration

Example 3.7.1 (Iteration 3)

Iteration 3, k=2 Current : 7<=8 Next : 9<=8	
Letter	**Probability**
B	0.300
C	0.100
AB	0.180
AC	0.060
AAA	0.216
AAB	0.108
AAC	0.036

New Size will be 9 and 9<=8 is not satisfied so STOP

Example 3.7.1 (Generate Code)

Letter	Code
B	000
C	001
AB	010
AC	011
AAA	100
AAB	101
AAC	110

Assign 3 bit Code to all letters in codebook

Note

- This slides are not enough material for examination
- You need to refer to your reference book for more details
- For your Practice refer 3.10 Projects and problems 4, 5 and 7
- Always keep calculator with you!

Data Compression

Chapter – 4
Arithmetic Coding

Anokhi Shah

4.1 Overview

- Arithmetic coding is another method of generating variable-length codes.
- Arithmetic coding is especially useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly skewed probabilities.
- It is also a very useful approach when, for various reasons, the modeling and coding aspects of lossless compression are to be kept separate

4.2 Introduction

- It is more efficient to generate codewords for groups / sequences of symbols rather than generating a separate codeword for each symbol in a sequence.
- In order to find the Huffman codeword for a particular sequence of length m, we need codewords for all possible sequences of length m.
- This fact causes an exponential growth in the size of the codebook.

4.2 Introduction

- We need a way of assigning codewords to particular sequences without having to generate codes for all sequences of that length.
- In arithmetic coding a unique identifier or Tag is generated for the sequence to be encoded.
- This tag corresponds to a binary fraction, which becomes the binary code for the sequence.

4.3 Coding a Sequence

- In order to distinguish a sequence of symbols from another sequence of symbols we need to tag it with a unique identifier.
- One possible set of tags for representing sequences of symbols are the numbers in the unit interval [0, 1)
- Because the numbers in the unit interval is infinite, it should be possible to assign a unique tag to each distinct sequence of symbols.

4.3.1 Generating a Tag

- The procedure for generating the tag works by reducing the size of the interval in which the tag resides as more and more elements of the sequence are received.
- We start out by first dividing the unit interval into subintervals of the form

  [Fx(i-1), Fx(i)) for i = 1,…, m.
- Because the minimum value of the cdf is zero and the maximum value is one, this exactly partitions the unit interval.

Tag Generation

- Calculate Fx Cumulative frequency for each symbol in sequence.
- Divide the Unit Interval based on Fx for symbols
- For each Symbol In Sequence
  - Identify the Interval in which the symbol resides.
  - Refine that interval using the formula

    low + (high-low)*Fx
- Finally Tag is the midpoint = (low+high)/2

Example 4.3.1

- Consider a three-letter alphabet A = [a1,a2,a3} with P(a1) = 0.7, P(a2) =0.1, and P(a3) =0.2 Find the tag for sequence a1a2a3
- Find cumulative Frequency and then divide the Unit Interval [0,1) based on that

Letter	Probability	Fx
a1	0.7	0.7
a2	0.1	0.8
a3	0.2	1.0

- For Sum of Tag Generation using Arithmetic Coding refer your 'lecture notes'

Decoding Tag

- Decoder has 3 things,
  - Probability model
  - Tag
  - Length of Sequence
- Steps for Decoding

For i=1 to Length of the Sequence

- Identify the interval in which Tag resides
- Symbol for that interval will be the ith symbol in sequence
- Find New Tag = (Old Tag – Low)/ (High – Low) where Low and high are from the above found interval

Algorithm Implementation

- Before we can implement this algorithm, there is one major problem we have to resolve.
- Recall that the numbers in the interval [0, 1) are infinite numbers.
- However, in practice the numbers that can be uniquely represented on a machine is limited by the maximum number of digits or bits E.g. Float with 6 digits

Algorithm Implementation

- As iteration increases, Interval values come closer and closer together.
- This means that in order to represent all the subintervals uniquely we need increasing precision as the length of the sequence increases.
- In a system with finite precision, the two values are bound to converge, and we will lose all information about the sequence from the point at which the two values converged.

Algorithm Implementation

- To avoid that situation, we need to rescale the interval.
- We have to generate tag with scaling
- As the interval becomes narrower, we have three possibilities:

1. The interval is entirely confined to the lower or 1st half of the unit interval [0, 0.5)
2. The interval is entirely confined to the upper or 2nd half of the unit interval [0.5, 1.0)
3. The interval straddles the midpoint of the unit interval. (Not entirely in any half)

Tag Generation with Scaling

- For condition 1 E_1 mapping us used where value of x is converted to 2x, such as $E_1(x)= 2x$
- For condition 2 E_2 mapping us used where value of x is converted to 2(x-0.5), such as $E_2(x)= 2(x-0.5)$
- For Condition 3 Stop Scaling or No Scaling
- For sum of Generation with scaling refer your lecture notes.

Note

- Slides for Chapter 4 – Arithmetic Coding covers only theory part for Sums refer your lecture notes.
- Its compulsory to do the sums with at least 6 decimal places for precision of fractional values
- Prepare the difference between Huffman coding and arithmetic coding

Data Compression

Chapter – 4

Arithmetic Coding

Overview

- Arithmetic coding is another method of generating variable-length codes.
- Arithmetic coding is especially useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly skewed probabilities.
- It is also a very useful approach when, for various reasons, the modeling and coding aspects of lossless compression are to be kept separate

Introduction

- It is more efficient to generate codewords for groups / sequences of symbols rather than generating a separate codeword for each symbol in a sequence.
- In order to find the Huffman codeword for a particular sequence of length m, we need codewords for all possible sequences of length m.
- This fact causes an exponential growth in the size of the codebook.

Introduction

- We need a way of assigning codewords to particular sequences without having to generate codes for all sequences of that length.
- In arithmetic coding a unique identifier or Tag is generated for the sequence to be encoded.
- This tag corresponds to a binary fraction, which becomes the binary code for the sequence.

Coding a Sequence

- In order to distinguish a sequence of symbols from another sequence of symbols we need to tag it with a unique identifier.
- One possible set of tags for representing sequences of symbols are the numbers in the unit interval [0, 1)
- Because the numbers in the unit interval is infinite, it should be possible to assign a unique tag to each distinct sequence of symbols.

Generating a Tag

- The procedure for generating the tag works by reducing the size of the interval in which the tag resides as more and more elements of the sequence are received.
- We start out by first dividing the unit interval into subintervals of the form

  [Fx(i-1), Fx(i)) for i = 1,…, m.
- Because the minimum value of the cumulative density function is zero and the maximum value is one, this exactly partitions the unit interval.

Tag Generation

- Calculate Fx Cumulative frequency for each symbol in sequence.
- Divide the Unit Interval based on Fx for symbols
- For each Symbol In Sequence
  - Identify the Interval in which the symbol resides.
  - Refine that interval using the formula

    low + (high-low)*Fx
- Finally Tag is the midpoint = (low+high)/2

Example

- Consider a three-letter alphabet A = [a1,a2,a3} with P(a1) = 0.7, P(a2) =0.1, and P(a3) =0.2 Find the tag for sequence a1a2a3
- Find cumulative Frequency and then divide the Unit Interval [0,1) based on that

Letter	Probability	Fx
a1	0.7	0.7
a2	0.1	0.8
a3	0.2	1.0

- For Sum of Tag Generation using Arithmetic Coding refer your book

Decoding Tag

- Decoder has 3 things,
  - Probability model
  - Tag
  - Length of Sequence
- Steps for Decoding

For i=1 to Length of the Sequence

- Identify the interval in which Tag resides
- Symbol for that interval will be the ith symbol in sequence
- Find New Tag = (Old Tag – Low)/ (High – Low) where Low and high are from the above found interval

Algorithm Implementation

- Before we can implement this algorithm, there is one major problem we have to resolve.
- Recall that the numbers in the interval [0, 1) are infinite numbers.
- However, in practice the numbers that can be uniquely represented on a machine is limited by the maximum number of digits or bits E.g. Float with 6 digits

Algorithm Implementation

- As iteration increases, Interval values come closer and closer together.
- This means that in order to represent all the subintervals uniquely we need increasing precision as the length of the sequence increases.
- In a system with finite precision, the two values are bound to converge, and we will lose all information about the sequence from the point at which the two values converged.

Algorithm Implementation

- To avoid that situation, we need to rescale the interval.
- We have to generate tag with scaling
- As the interval becomes narrower, we have three possibilities:

1. The interval is entirely confined to the lower or 1st half of the unit interval [0, 0.5)
2. The interval is entirely confined to the upper or 2nd half of the unit interval [0.5, 1.0)
3. The interval straddles the midpoint of the unit interval. (Not entirely in any half)

Tag Generation with Scaling

- For condition 1 E_1 mapping us used where value of x is converted to 2x, such as $E_1(x)= 2x$
- For condition 2 E_2 mapping us used where value of x is converted to 2(x-0.5), such as $E_2(x)= 2(x-0.5)$
- For Condition 3 Stop Scaling or No Scaling
- For sum of Generation with scaling refer your book.

Note

- Slides for Chapter 4 – Arithmetic Coding covers only theory part for Sums refer your book
- Its compulsory to do the sums with at least 6 decimal places for precision of fractional values
- Prepare the difference between Huffman coding and arithmetic coding

DICTIONARY TECHNIQUES

- Dictionary techniques incorporate the structure in the data in order to increase the amount of compression.
- These techniques are both static and adaptive (or dynamic) to build a list of commonly occurring patterns and encode these patterns by transmitting their index in the list.
- They are most useful with sources that generate a relatively small number of patterns quite frequently, such as text sources and computer commands.
- Use static approach in case of prior knowledge else use adaptive approach

- Static Dictionary
  - Ex: to compress the student records at a university
  - Work well only for the applications and data specific situations
  - Digram Coding is common form of static dictionary coding.
  - In Digram coding , dictionary consists of all letters of the source alphabet followed by many pairs of letters called digrams.

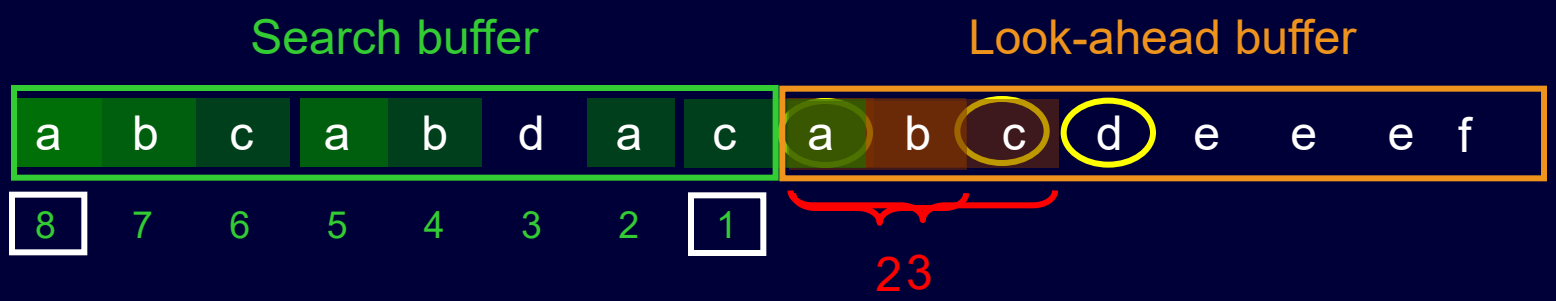
- Digram encoder reads a two character input and searches the dictionary to see if this input exists in the dictionary. If it does, the corresponding index is encoded and transmitted. If note first character of the pair is encoded and the second character in the pair then becomes the first character of the next digram. And the search procedure is repeated .
- Ref: Ex 5.3.1
- Compare table 5.2 and 5.3 and conclude that this technique is suitable of identical source but for variety of source it is not suitable But Adaptive Dictionary based technique is required .

Ziv-Lempel Coding (ZL or LZ)

- Named after J. Ziv and A. Lempel (1977).
- Adaptive dictionary technique.
  - Store previously coded symbols in a buffer.
  - Search for the current sequence of symbols to code.
  - If found, transmit buffer offset and length.
- LZ77 approach , the dictionary is simply a portion of the previously encoded sequence.

- The encoder examines the input sequence through sliding window.
- Window consists of two parts a search buffer that contains a portion of the recently encoded sequence and a look ahead buffer that contains the next portion of the sequence to be encoded.
- Three different possibilities may be encounter during the coding process
  - There is no match for the next character to be encoded in the window
  - There is a match
  - The matched string extends inside the look-ahead buffer

LZ77



Output triplet <offset, length, next>

Transmitted to decoder: 8 3 d 0 0 e 1 2 f

If the size of the search buffer is N and the size of the alphabet is M we need

$$\lceil \log(N+1) \rceil + \lceil \log(N+1) \rceil + \lceil \log M \rceil$$

bits to code a triplet.

*Variation in LZ77 is to u*se a Variable Length Code triplets . Used in >> PKZip, Zip, Lharc, PNG, gzip, ARJ

Drawback with LZ77

- Repetetive patterns with a period longer than the search buffer size are not found.
- If the search buffer size is 4, the sequence
  a b c d e a b c d e a b c d e a b c d e …
  will be expanded, not compressed.

Note : refer Example 5.4.1

LZ78

- Store patterns in a dictionary
- Transmit a tuple <dictionary index, next>
- The index value of 0 is used in the case of no match, and newest entry in the dictionary
- Each new entry into the dictionary is one new symbol concatenated with the existing dictionary entry.
- Ref ex: 5.4.2

LZ78

a b c a b a b c

Output tuple <dictionary index, next>

Transmitted to decoder: 0 a 0 b 0 c 1 b 4 c

Decoded: a b c a b a b c

Dictionary:

1	a
2	b
3	c
4	a b
5	a b c

Strategy needed for limiting dictionary size!

LZW

- Modification to LZ78 by Terry Welch, 1984.
- Applications: GIF (Graphics Interchange Format), v42bis (is a compression standard devised for use over a telephone network), PNG (Portable Network Graphics)
- The UNIX compress command is one of the earlier applications of LZW.
- Patented by UniSys Corp.
- Transmit only the dictionary index.
- The alphabet is stored in the dictionary in advance.

- Welch proposed a technique for removing the necessity of encoding the second element of the pair (i, *c). That is, the encoder would* only send the index to the dictionary. In order to do this, the dictionary has to be prepared with all the letters of the source alphabet.
- The input to the encoder is accumulated in a pattern *p as long as p is contained in the dictionary.*
- If the addition of another letter a results in a pattern p*a (* denotes concatenation) that is not in the dictionary, then the index of p is transmitted to the receiver, the pattern p*a is added to the dictionary, and we start another pattern with the letter a. Ref: example 5.4.3

LZW

Input sequence: a b c a b a b c

Output: dictionary index

Transmitted:

1	2	3	5	5

Decoded:

a	b	c	a b	a b

Encoder dictionary:

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d	9	abc
5	a b		

Decoder dictionary:

1	a	6	bc
2	b	7	ca
3	c	8	aba
4	d		
5	a b		

And now for some applications:
GIF & PNG

GIF

- CompuServe Graphics Interchange Format (1987, 89).
- Features:
  - Designed for up/downloading
  - 1-, 4-, or 8-bit *colour palettes*.
  - Interlace for *progressive decoding*
  - *Transparent colour* for non-rectangular images.
  - Supports multiple images in one file (”animated GIFs”).

GIF: Method

- Compression by LZW.
- Dictionary size 2^{b+1} 8-bit symbols
  - *b* is the number of bits in the palette.
- Dictionary size doubled if filled (max 4096).
- Works well on computer generated images.

GIF: Problems

- Unsuitable for natural images (photos):
  - Maximum 256 colors ($\Rightarrow$ bad quality).
  - Repetetive patterns uncommon ($\Rightarrow$ bad compression).
- LZW patented by UniSys Corp.
- Alternative: PNG

PNG: Portable Network Graphics

- Designed to replace GIF.
- Some features:
  - Indexed *or* true-colour images ($\leq$ 16 bits per plane).
  - Alpha channel.
  - Gamma information.
  - Error detection.
- No support for multiple images in one file.
  - Use MNG for that.
- Method:
  - Compression by LZ77 using a 32KB search buffer.
  - The LZ77 triplets are Huffman coded.
- More information: www.w3.org/TR/REC-png.html

Summary

- Huffman coding
  - Simple, easy, fast
  - Complexity grows exponentially with the block length
  - Statistics built-in in the code
- Arithmetic coding
  - Complexity grows linearly with the block size
  - Easily adapted to variable statistics $\Rightarrow$ used for coding of Markov sources
- Universal coding
  - Adaptive Huffman or arithmetic coder
  - LZ77: Buffer with previously sent sequences <offset,length,next>
  - LZ78: Dictionary instead of buffer <index,next>
  - LZW: Modification to LZ78 <index>

Summary, cont

- Where are the algorithms used?
  - Huffman coding: JPEG, MPEG, PNG, …
  - Arithmetic coding: JPEG, JBIG, MPEG-4, …
  - LZ77: PNG, PKZip, Zip, gzip, …
  - LZW: compress, GIF, v42bis, …

Finally

- These methods work best if the source alphabet is small and the distribution skewed.
  - Text
  - Graphics
- Analog sources (images, sound) require other methods
  - complex dependencies
  - accepted distortion

Chapter 6

Predictive Coding

Context-based Coding

- If in a sequence to be encoded, certain symbols occur with much higher probability than others, more compression can be achieved
- Several ways to achieve this situation:
  - Transform sequence to another one with desired property
  - Use a different probability distribution for each symbol (symbol being encoded has high probability)
- If the transformation or probability distribution is based on the history of the sequence, since history is available to both encoder and decoder, there is no need for transmitting additional information.
- These schemes are called context based coding schemes.

Predictive Coding

- Exp:
  - 1,2,5,7,2,-2,0,-5,-3,-1,1,-2,-7,-4,-2,1,3,4
  - Prediction rule: r=x(n)-(2+x(n-1))
  - Residual sequence: 1,-1,1,0,-7,-4,0,-7,0,0,0,-5,1,0,1,0,-1
- A different kind of prediction is used when encoding nonnumeric data (e.g., text): preceding symbols (context) are used to predict the probability model for the symbol being coded
- Exp: encoding the word *probability*
- If we ignore the first four letters that are coded (prob) the probability of a is 0.06.
- If we use the information that the letter before a is b, this will boost the probability of an a occurring

Prediction with partial match (ppm)

- We would like to use large contexts to determine the probability of the symbol being encoded
- Large context requires estimation and storage of an extremely large number of conditional probabilities
- Instead of estimating these probabilities ahead of time, we can reduce the burden by estimating them as the coding proceeds
- This way we only need to store contexts that have occurred which is a much smaller number than all possible contexts.

Basic algorithm

- The size of the largest context is predetermined
- If the symbol to be encoded has not previously been encountered in this context, an escape symbol is encoded and the algorithm attempts to use the next smaller context
- If the symbol has not occurred in this context either, the size of the context is further reduced
- This process continues until either we obtain a context that has previously been encountered with this symbol or we conclude that the symbol has not been encountered previously in any context

Basic algorithm

- The case when the symbol is considered by itself is called zero-order context
- If a symbol is encountered for the very first time, we would use a model in which all symbols have equal probability (called context order –1)
- Each time a symbol is encountered, the count corresponding to that symbol is updated

Example

- Exp: this_is_the_tithe
- Assumption: the first seven characters have been coded and the various counts are as the following tables
- The word length for arithmetic coding is four

-1 order

Letter	Count	Cum_count
t	1	1
h	1	2
i	1	3
s	1	4
e	1	5
_	1	6

0 order

Letter	Count	Cum_count
t	1	1
h	1	2
i	2	4
s	2	6
_	1	7
<esc>	1	8

Example

first order

Context	Letter	Count	Cum_count
t	h	1	1
	<esc>	1	2
h	i	1	1
	<esc>	1	2
i	s	2	2
	<esc>	1	3
_	i	1	1
	<esc>	1	2
s	_	1	1
	<esc>	1	2

second order

Context	Letter	Count	Cum_count
th	i	1	1
	<esc>	1	2
hi	s	1	1
	<esc>	1	2
is	_	1	1
	<esc>	1	2
s_	i	1	1
	<esc>	1	2
_i	s	1	1
	<esc>	1	2

Example

- The next letter to be coded is _
- The second order context for _ is is
- Cum_count=1 and we have

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{0}{2} \right\rfloor = 0 = 0000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor - 1 = 7 = 0111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

  Transmitted sequence: 0

  l=0000

  u=1111

Example

- The next letter to be coded is t
- The second order context for t is s_
- t has not appeared in this context before
- We code a escape symbol:

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor = 8 = 1000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 15 = 1111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

  Transmitted sequence: 01

  l=0000

  u=1111

- We add an entry in s context for t

Example

- Next we check the first-order context of t which is _
- t has not happened in this context before
- We code another escape symbol:

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor = 8 = 1000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 15 = 1111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

  Transmitted sequence: 011

  l=0000

  u=1111

- Add an entry in _ context for t

Example

- Next we check the zero-order context of t
- The zero order context for t is in the table

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{0}{8} \right\rfloor = 0 = 0000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{8} \right\rfloor - 1 = 1 = 0001$$

- As the 3 MSB of l and u are the same, we shift them out, shift a 0 into the LSBs of l and a 1 into the LSBs of u

Transmitted sequence: 011000

l=0000

u=1111

Example

0 order

Letter	Count	Cum_count
t	2	2
h	1	3
i	2	5
s	2	7
_	1	8
<esc>	1	9

first order

Context	Letter	Count	Cum_count
t	h	2	1
	<esc>	1	2
h	i	1	1
	<esc>	1	2
i	s	2	2
	<esc>	1	3
-	i	1	1
_	t	1	2
	<esc>	1	3
s	_	1	1
	<esc>	1	2

Example

second order

Context	Letter	Count	Cum_count
th	i	1	1
	<esc>	1	2
hi	s	1	1
	<esc>	1	2
is	_	2	2
	<esc>	1	3
s_	i	1	1
	t	1	2
	<esc>	1	3
_i	s	1	1
	<esc>	1	2
_t	h	1	1
	<esc>	1	2

Exclusion principle

- The basic idea behind arithmetic coding is the division of the unit interval into subintervals, each of which represent a particular letter
- The smaller the subinterval, the more bits are required to distinguish it from other subintervals
- If we can reduce the number of symbols, the size of subintervals increase, leading to a reduction in the number of bits required for encoding
- Exclusion principle provides this kind of reduction

Exclusion principle

- Exp: Suppose we want to encode a in sequence *proba* and the contexts are as the following tables
- As a does not occur in two letter context, we issue an escape symbol and reduce the size of the context
- Checking the one-letter context, we see that a does occur with a count of 4 out of total of 21
- By sending escape symbol in context of ob, we have already signaled the decoder that the symbol being coded is not any of the letters previously encountered in the context of ob
- Therefore we can increase the size of the subinterval corresponding to a by temporarily removing l and o from the table

Example

Context	Letter	Count	Cum_count
ob	l	10	10
	o	3	13
	<esc>	2	15
b	l	5	5
	o	3	8
	a	4	12
	r	2	14
	e	2	16
	<esc>	5	21

Context	Letter	Count	Cum_count
b	a	4	4
	r	2	6
	e	2	8
	<esc>	3	11

Burrows-Wheeler Transform(BWT)

- BWT also uses context of the symbols being coded for lossless compression
- BWT requires the entire sequence to be coded be available to the encoder before coding takes place

<u>BWT</u>

- Given a sequence of length N, we create N-1 sequences each of them a cyclic shift of the original sequence
- These N sequences are arranged in lexicographic (dictionary) order
- The encoder then transmits the sequence of length N created by taking the last letter of each sorted cyclically shifted sequence

Burrows-Wheeler Transform(BWT)

- The sequence of last letters (L) and the position of the original sequence in the sorted list are coded and sent to the decoder
- We start with a sequence of length N and end with a representation that contains N+1 elements!
- The sequence has a structure that makes it highly suitable for compression
- A coding method called move-to-front (mtf) is particularly effective in the type of structure exhibited by sequence L

Exp

- Sequence to encode: this_is_the
- 11 characters in the sequence: 11 cyclic permutations of the sequence
- The sequences sorted in lexicographic order are as bellow

0	_is_thethis
1	_thethis_is
2	ethis_is_th
3	hethis_is_t
4	his_is_thet
5	is_is_theth
6	is_thethis_
7	s_is_thethi
8	s_thethis_i
9	thethis_is_
10	this_is_the

L:sshtth_ii_e

Encoding sequence:
L and index 10

BWT decoder

- The first step is to generate sequence F consisting the first elements of each row
- F is simply the sequence L in lexicographic order
- F: __ehhiisstt
- We use L and F to generate a transformation T that will tell us in which order to read elements of L to generate the original sequence

BWT decoder

- Using the index (10) we decode the last character of the original sequence to be e
- Since the rows in the table are the cyclic shift of each other, the character before e is the last element of row starting with e (which is h)
- If we have two identical letters in L, the order in which they occur is the same as the order in which the same letters occur in F

0	_									s
1	_									s
2	e									h
3	h									t
4	h									t
5	i									h
6	i									_
7	s									i
8	s									i
9	t									_
10	t									e

Move-to-Front Coding

- A coding scheme that takes advantage of long runs of identical symbols is the move-to-front (mtf) coding
- In this coding scheme, we start with some initial listing of the source alphabet
- The symbol at the top of the list is assigned number 0, the next one is assigned number one and so on.
- The first time a particular symbol occurs, the number corresponding to its place in the list is transmitted and the symbol is moved to top of the list
- If we have a run of this symbol, we transmit a sequence of 0s

Exp

- L=sshtth_ii_e
- A={_,e,h,i,s,t}

0	1	2	3	4	5
_	e	h	i	s	t

Encoding sequence: 4

0	1	2	3	4	5
s	_	e	h	i	t

Encoding sequence: 403

0	1	2	3	4	5
h	s	-	e	i	t

Encoding sequence: 40350135015

Data Compression

Chapter – 7

Mathematical Preliminaries

for Lossy Coding

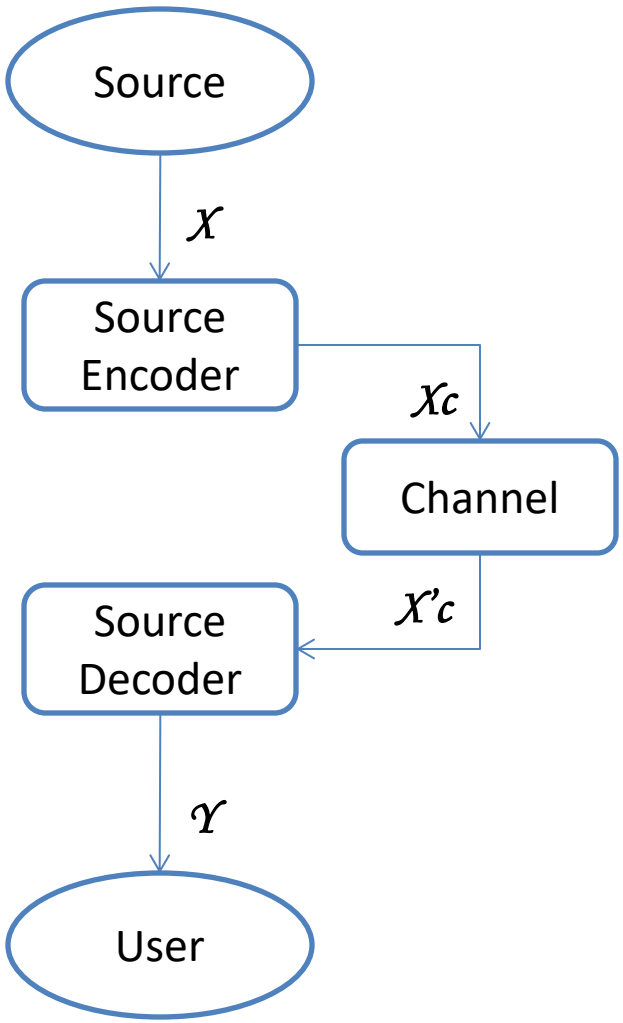
Recap – Lossless compression

- In lossless compression we never had to worry about how the reconstructed sequence would differ from original sequence. As both were Identical.
- However, there is only a limited amount of compression that can be obtained with lossless compression.
- For lossless compression schemes we essentially used only the rate as the performance measure

Lossy compression

- A performance measure is required which will measure the difference between the original and the reconstructed data.
- We refer it as *Distortion* in the reconstructed data.
- So we would like to incur the minimum amount of distortion while compressing to the lowest rate possible.

Block Diagram



- The output of the source is modeled as a random variable X
- The source coder takes the source o/p and produces the compressed representation X_c.
- The channel block represents all transformations. Usually we will take the channel to be identity mapping, which means $X_c = X'_C$
- Source decoder takes compressed representation and produces a reconstruction of the source o/p for the user.

7.3 Distortion Criteria

- Distortion = $\mathcal{X}$ - $\mathcal{Y}$
- Two popular measures of distortion are
  - Squared error measure
  - Absolute difference measure
- They are called Difference distortion measures.
- If $\{\mathcal{X}n\}$ is the source o/p and $\{\mathcal{Y}n\}$ is reconstructed sequence then
- Squared Error measure : $d(x,y) = (x-y)^2$
- Absolute difference measure: $d(x,y) = |x-y|$

MSE

- It is difficult to examine the difference on a term by term basis.
- Therefore a number of avg measures are used to summarize the information in the difference sequence.
- Find average of the squared error measure called mean squared error (mse) and it is often represented by the symbol σ^2 or σ_d^2

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - y_n)^2$$

SNR

- If we are interested in the size of the error relative to the signal, we can find the ratio of the average squared value of the source output and the mse. This is called the *signal-to-noise ratio (SNR).*
  - Where σ^2_x *is the average squared value of the source output, or signal, and* σ^2_d *is the mse.*
  - The SNR is often measured on a logarithmic scale and the units of measurement are *decibels*(abbreviated to dB).

$$SNR = \frac{\sigma_x^2}{\sigma_d^2}$$

$$SNR(dB) = 10\log_{10}\frac{\sigma_x^2}{\sigma_d^2}$$

PSNR

- Sometimes we are more interested in the size of the error relative to the peak value of the signal x_{peak} than with the size of the error relative to the average squared value of the Signal.
- This ratio is called the *peak-signal-to-noise-ratio (PSNR) and is given by*

$$PSNR\ (dB) = 10 \log_{10} \frac{x_{peak}^2}{\sigma_d^2}$$

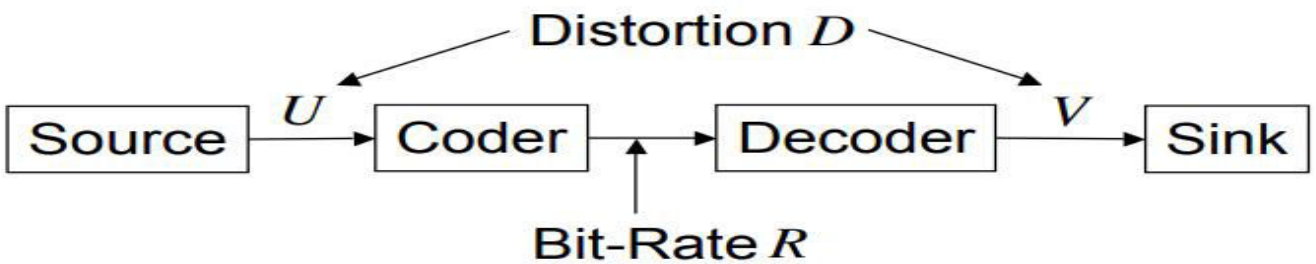
Average Absolute Difference

- Useful for evaluating image compression algorithms

$$d_1 = \frac{1}{N} \sum_{n=1}^{N} |x_n - y_n|$$

7.5 Rate Distortion Theory

- Rate Distortion theory is concerned with the trade-off between distortion and rate in lossy compression schemes.
- Rate distortion theory calculates minimum transmission bit-rate for a given distortion and source.



- Need to define U, V, Coder/Decoder, Distortion D, and Rate R
- Need to establish functional relationship between U, V, D, and R

Definitions

- **Source symbols** are given by the random sequence $\{U_k\}$
  - Each U_k assumes values in the discrete set $\upsilon = \{u_0, u_1, \ldots, u_{M-1}\}$
    - For a binary source: $U = \{0,1\}$
    - For a picture: $U = \{0,1,\ldots,255\}$
  - For simplicity, let us assume U_k to be independent and identically distributed (i.i.d.) with distribution $\{P(u), u \in U\}$

- **Reconstruction symbols** are given by the random sequence $\{V_k\}$ with distribution $\{P(v), v \in \nu\}$
  - Each V_k assumes values in the discrete set $\nu = \{v_0, v_1, \ldots, v_{N-1}\}$
  - The sets υ and ν need not to be the same

Coder / Decoder

- Statistical description of **Coder/Decoder**, i.e. the mapping of the source symbols to the reconstruction symbols, via

$$Q = \{Q(v \mid u), u \in \upsilon, v \in \nu\}$$

- is the conditional probability distribution over the letters of the reconstruction alphabet υ given a letter of the source alphabet ν
- Transmission system is described via

Joint pdf: $P(u,v)$

$$P(u) = \sum_{v \in \nu} P(u,v)$$

$$P(v) = \sum_{u \in \upsilon} P(u,v)$$

$$P(u,v) = P(u) \cdot Q(v \mid u) \qquad \text{(Bayes' rule)}$$

Pdf : probability density function

Distortion

- To determine distortion, we define a non-negative cost function

$$d(u,v), d(.,.) : \upsilon \times \nu \rightarrow [0,\infty)$$

- Examples for d
  - Hamming distance: $d(u,v) = \begin{cases} 0, & \text{for} \quad u \neq v \\ 1, & \text{for} \quad u = v \end{cases}$
  - Squared error: $d(u,v) = |u - v|^2$

- Average **Distortion** $D(Q) = \sum_{u \in \upsilon} \sum_{v \in \nu} \underbrace{P(u) \cdot Q(v \mid u)}_{P(u,v)} \cdot d(u,v)$

Rate

- Shannon average mutual information expressed via entropy

$$I(U;V) = \underset{\substack{\uparrow \\ \text{Source entropy}}}{H(U)} - \underset{\substack{\uparrow \\ \text{Equivocation: conditional entropy}}}{H(U \mid V)}$$

- Equivocation:
  - The conditional entropy (uncertainty) about the source U given the reconstruction V
  - A measure for the amount of missing [quantized] information in the received signal V

Rate Distortion

- The Rate Distortion Function R(D) specifies the lowest rate at which output of a source can be encoded while keeping the distortion less than or equal to D
- $R(D) = \min \{ I(U;V) \}$

Models

- As In case of lossless compression , models play an important role in the design of lossy compression algorithm
- Model in Lossless try exact match
- Model in lossy looks more to the general rather than exact
- An important method for characterizing a particular source is through the use of probability models

7.6.1 Probability Models

- There are four probability models commonly used in the design and analysis of lossy compression system.
- Uniform Distribution
  - If we don't know any thing about the distribution of the source output, except possibly by the range of values.
  - The probability density function for a random variable uniformly distributed between a and b is,

$$f_x(x) = \begin{cases} \dfrac{1}{b-a} & for\ a \leq x \leq b \\ 0 & Otherwise \end{cases}$$

- Gaussian Distribution
  - Most commonly used because It is mathematically tractable and by virtue of the central limit theorem.
  - The probability density function for a random variable with Gaussian distribution and mean μ and variance σ^2 is

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x-\mu)^2}{2\sigma^2}$$

  - Many sources that we deal with have distributions that are quite picked at zero. Example Speech of silence and image
  - In this situation a Gaussian distribution is not very close match to the data.

- Laplacian Distribution
  - A closer match in previous case is Laplacian distribution, which is peaked at zero.
  - The distribution function for a zero means random variable with Laplacian distribution and variance σ^2 is

$$f_x(x) = \frac{1}{\sqrt{2\sigma^2}} \exp \frac{-\sqrt{2}|x|}{\sigma}$$

- Gamma Distribution
  - A distribution which is even more peaked
  - Considerably less tractable than Laplacian distribution is Gamma distribution.
  - The distribution function for a gamma distributed random variable with zero mean distribution and variance σ^2 is

$$f_x(x) = \frac{\sqrt[4]{3}}{\sqrt{8\pi\sigma|x|}} \exp \frac{-\sqrt{3}|x|}{2\sigma}$$

- One way of obtaining the estimate of the distribution of a particular source is to divide the range of output into intervals
- Then find number of values nk that fall in to each interval
- Plot nk/nT, where nT is the total number of source outputs
- This will give us idea of what the input distribution looks like

Data Compression

Chapter – 8

Scalar Quantization

8.2 Introduction

- We are required to represent each source output using one of smaller number of codewords.
- The number of possible distinct source output values is generally much larger than the number of codewords available to represent them.
- The process of representing a large (possibly infinite) set of values with a much smaller set is called *Quantization*.

Quantization

- A simple quantization scheme would be to represent each output of the source with the integer value closest to it. (if the source output is equally close to two integers, we will randomly pick one of them)
- So we would like to incur the minimum amount of distortion while compressing to the lowest rate possible.

Source
-10.0 to 10.0

Infinite Values

2.47 → 2

3.1415926 → 3

Only 21 values {-10,…,0,…,10}

Lossy Compression

- With the compression, we have forever lost the original values of the source output.
- If we are told that the reconstruction values is 3, we can not tell whether the source output was 2.95, 3.16, 3.057932, or any other of an infinite set of values.
- In other words we have lost some information.
- The Set of inputs and outputs of a quantizer can be Scalar or Vector.
  - Scalar quantizer
  - Vector quantizer

8.3 Quantization Problem

In practice, the quantizer consists of two mappings:

- Encoder mapping: (irreversible)
  - Divides the range of values that the source generates into a number of intervals.
  - Each interval is represented by a distinct codeword.
  - When the sample value comes from an analog source, the encoder is called an analog-to-digital (A/D) converter.

- Decoder mapping:
  - Because a codeword represents an entire interval, and there is no way of knowing which value in the interval was actually generated by the source, the decoder puts out a value that, in some sense, best represents all the values in the interval.
  - Midpoint of the interval If the reconstruction is analog, the decoder is often referred to as a digital-to-analog (D/A) converter.
- Figure 8.1 & 8.2

Quantizer

- Construction of the intervals (their location, etc.) can be viewed as part of the design of the encoder.
- Selection of the reconstruction values is part of the design of the decoder.
- The fidelity of the reconstruction depends on both the intervals and the reconstruction values.
- We call this encoder-decoder pair a *Quantizer*.

Distortion

- Distortion – Average squared difference between the quantizer input and output.
- We call this mean squared quantization error (msqe) and denote it by σ_q^2
- The rate of the quantizer is the average number of bits required to represent a single quantizer output.
- We would like to get the lowest distortion for a given rate, or the lowest rate for a given distortion.

Intervals

- Suppose we have an input modeled by a random variable x with probability density function $f_X(x)$.
- If we wished to quantize this source using a quantizer with $\mathcal{M}$ intervals, we would have to specify $\mathcal{M} + 1$ endpoints for the intervals, and a representative value for each of the $\mathcal{M}$ intervals.
- The endpoints of the intervals are known as *decision boundaries*.
- The representative values are called *reconstruction levels*.

- Let us denote,
  - The decision boundaries by $\{b_i\}$ where i=0 to M
  - The reconstruction level by $\{y_i\}$ where i=1 to M
  - The quantization operation by Q(·) Then,
- Q(x) = y_i iif $b_{i-1} < x <= b_i$
- The mean squared quantization error is then given by

$$\sigma_q^2 = \int_{-\infty}^{\infty} (x - Q(x))^2 f_x(x)\,\mathrm{dx}$$

- Given an input probability density function fx(x) and the number of levels M in the quantizer, find the decision boundaries $\{b_i\}$ and the reconstruction levels {yj } so as to minimize the mean squared quantization error given by Equation

$$\sigma_q^2 = \sum_{i=1}^{M} \int_{bi-1}^{bi} (x - y_i)^2 f_x(x)\,\mathrm{dx}$$

- If we use fixed-length codewords to represent the quantizer output, then the size of the output alphabet immediately specifies the rate.
- If the number of quantizer output is M, then the rate given by

$$R = \lceil log_2 M \rceil$$

- For example, if M = 8, then R = 3.

8.4 Uniform Quantizer

- All quantizer are of same size, except possibly for two outer intervals.
- In other words decision boundaries and reconstruction values (mid point of the intervals) are spaced evenly.
- This constant spacing is referred to as the Step Size and is denoted by Δ
- The design of uniform quantizer consist of finding the step size Δ that minimizes the distortion for given input process and number of decision levels.

Example : Quantize the sequence 1.3, 6, 3.45, 6.7, 2.9, 9.9

into a 3 level uniform quantizer.

- for a given source input, the partitions we select and the representation for those partitions will determine the distortion incurred during the quantization process.
- The partitions we select and the binary codes for the partitions will determine the rate for the quantizer.
- Thus, the problem of finding the optimum partitions, codes, and representation levels are all linked.

let the sequence is 1.3, 6, 3.45, 6.7, 2.9, 9.9
Find the sequence of differences
1.3,4.7,-2.55,3.25,-3.8,7
Find Range : [-4,8]
4+8=12 , n=3 level uniform quantizer

q = 12/3 = 4 boundary values

-4,0,4,8 ← boundary

-2,2,6←reconstuction value

Quantize sequence 2, 6, -2, 2, -2, 6

8.6 Nonuniform Quantization

- In order to decrease the average distortion, we can try to approximate the input better in regions of high probability.
- We can do this by making the quantization intervals smaller in those regions that have more probability mass.
- A quantizer that has nonuniform intervals is called a nonuniform quantizer.

Midrise & Midtread

- Quantizer which does not have zero as one of its representation levels is called midrise quantizer.
- Otherwise its called Midtread quantizer. (it is especially useful in situations where it is important that zero value should be presented)
- We use a midrise quantizer if the number of levels is even and midtread quantizer if the number of levels are odd.

Exercise : Quantize the sequence 4.3, 9, 3.5, 6.7, 2.9, 8.8 into a 4 level uniform quantizer.

Data Compression

Chapter – 8

Mathematical Preliminaries for Lossy Coding

Anokhi Shah

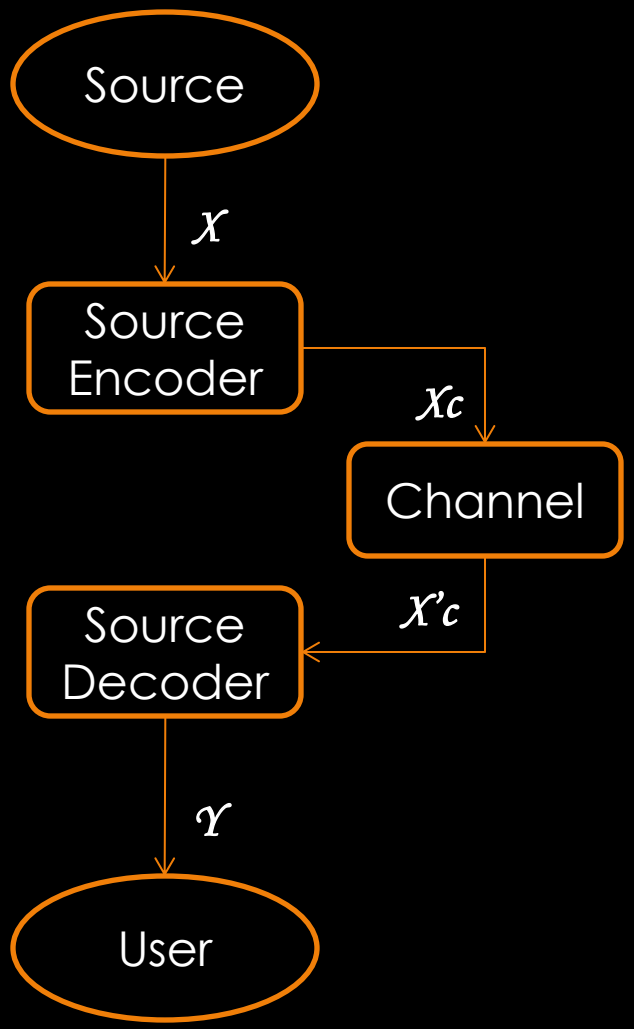
Recap – Lossless compression

- In lossless compression we never had to worry about how the reconstructed sequence would differ from original sequence. As both were Identical.
- However, there is only a limited amount of compression that can be obtained with lossless compression.
- For lossless compression schemes we essentially used only the rate as the performance measure

Lossy compression

- A performance measure is required which will measure the difference between the original and the reconstructed data.
- We refer it as *Distortion* in the reconstructed data.
- So we would like to incur the minimum amount of distortion while compressing to the lowest rate possible.

Block Diagram



- The output of the source is modeled as a random variable X
- The source coder takes the source o/p and produces the compressed representation X_c.
- The channel block represents all transformations. Usually we will take the channel to be identity mapping, which means $X_c = X'_C$
- Source decoder takes compressed representation and produces a reconstruction of the source o/p for the user.

7.3 Distortion Criteria

- Distortion = $\mathcal{X}$ - $\mathcal{Y}$
- Two popular measures of distortion are
  - Squared error measure
  - Absolute difference measure
- They are called Difference distortion measures.
- If $\{\mathcal{X}n\}$ is the source o/p and $\{\mathcal{Y}n\}$ is reconstructed sequence then
- Squared Error measure : $d(x,y) = (x-y)^2$
- Absolute difference measure: $d(x,y) = |x-y|^2$

MSE

- It is difficult to examine the difference on a term by term basis.
- Therefore a number of avg measures are used to summarize the information in the difference sequence.
- Find average of the squared error measure called mean squared error (mse) and it is often represented by the symbol σ^2 or σ_d^2

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - y_n)^2$$

SNR

- If we are interested in the size of the error relative to the signal, we can find the ratio of the average squared value of the source output and the mse. This is called the *signal-to-noise ratio (SNR)*.
- Where σ^2_x *is the average squared value of the source output, or signal, and* σ^2_d *is the mse.*
- The SNR is often measured on a logarithmic scale and the units of measurement are *decibels*(abbreviated to dB).

$$SNR = \frac{\sigma_x^2}{\sigma_d^2}$$

$$SNR(dB) = 10\log_{10}\frac{\sigma_x^2}{\sigma_d^2}$$

PSNR

- Sometimes we are more interested in the size of the error relative to the peak value of the signal x_{peak} than with the size of the error relative to the average squared value of the Signal.
- This ratio is called the *peak-signal-to-noise-ratio (PSNR) and is given by*

$$PSNR\ (dB) = 10 \log_{10} \frac{x_{peak}^2}{\sigma_d^2}$$

Average Absolute Difference

- Useful for evaluating image compression algorithms

$$d_1 = \frac{1}{N} \sum_{n=1}^{N} |x_n - y_n|$$

7.5 Rate Distortion Theory

- Rate Distortion theory is concerned with the trade-off between distortion and rate in lossy compression schemes.
- Rate is defined as Average number of bits used to represent each sample value.
- We will represent the trade-off via rate distortion function R(D)
- R(D) specifies the lowest rate at which the output of a source can be encoded while keeping the distortion less than or equal to D

7.6.1 Probability Models

- There are four probability models commonly used in the design and analysis of lossy compression system.
- Uniform Distribution
  - If we don't know any thing about the distribution of the source output, except possibly by the range of values.
  - The probability density function for a random variable uniformly distributed between a and b is,

$$f_x(x) = \begin{cases} \dfrac{1}{b-a} & for\ a \le x \le b \\ 0 & Otherwise \end{cases}$$

- Gaussian Distribution
  - Most commonly used because It is mathematically tractable and by virtue of the central limit theorem.
  - The probability density function for a random variable with Gaussian distribution and mean μ and variance σ^2 is

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x - \mu)^2}{2\sigma^2}$$

  - Many sources that we deal with have distributions that are quite picked at zero. Example Speech of silence and image
  - In this situation a Gaussian distribution is not very close match to the data.

- Laplacian Distribution
  - A closer match in previous case is Laplacian distribution, which is peaked at zero.
  - The distribution function for a zero means random variable with Laplacian distribution and variance σ^2 is

$$f_x(x) = \frac{1}{\sqrt{2\sigma^2}} \exp \frac{-\sqrt{2}|x|}{\sigma}$$

- Gamma Distribution
  - A distribution which is even more peaked
  - Considerably less tractable than Laplacian distribution is Gamma distribution.
  - The distribution function for a gamma distributed random variable with zero mean distribution and variance σ^2 is

$$f_x(x) = \frac{\sqrt[4]{3}}{\sqrt{8\pi\sigma|x|}} \exp \frac{-\sqrt{3}|x|}{2\sigma}$$

Figure 7.5 page 203

Data Compression

Chapter – 9

Vector Quantization

9.2 Introduction

- In lossless compression we saw that, by taking longer and longer sequences of input samples, it is possible to extract the structure in he source coder output.
- A quantization strategy that works with sequences or blocks of output would provide some improvement in performance over scalar quantization.
- So we wish to generate a representative set of sequences and when given a source output sequence, we would represent it with one of the elements of the representative set.

Vector Quantization

- Group the source output into blocks or vectors.
  - For example we can take a block of $\mathcal{L}$ pixels from an image
  - Treat each pixel value as a component of a vector of size or dimension $\mathcal{L}$
- This vector of source output forms the input to the vector quantizer.
- At both the encoder and decoder of the vector quantizer, we have a set of $\mathcal{L}$ dimensional vectors called codebook.

Code Vector

- So the set is Codebook. And codebook is made up of Code-Vectors
- Code vectors are selected to be representative of the vectors we generate from the source output.
- Each code vector is assigned a binary index.
- Figure 9.1

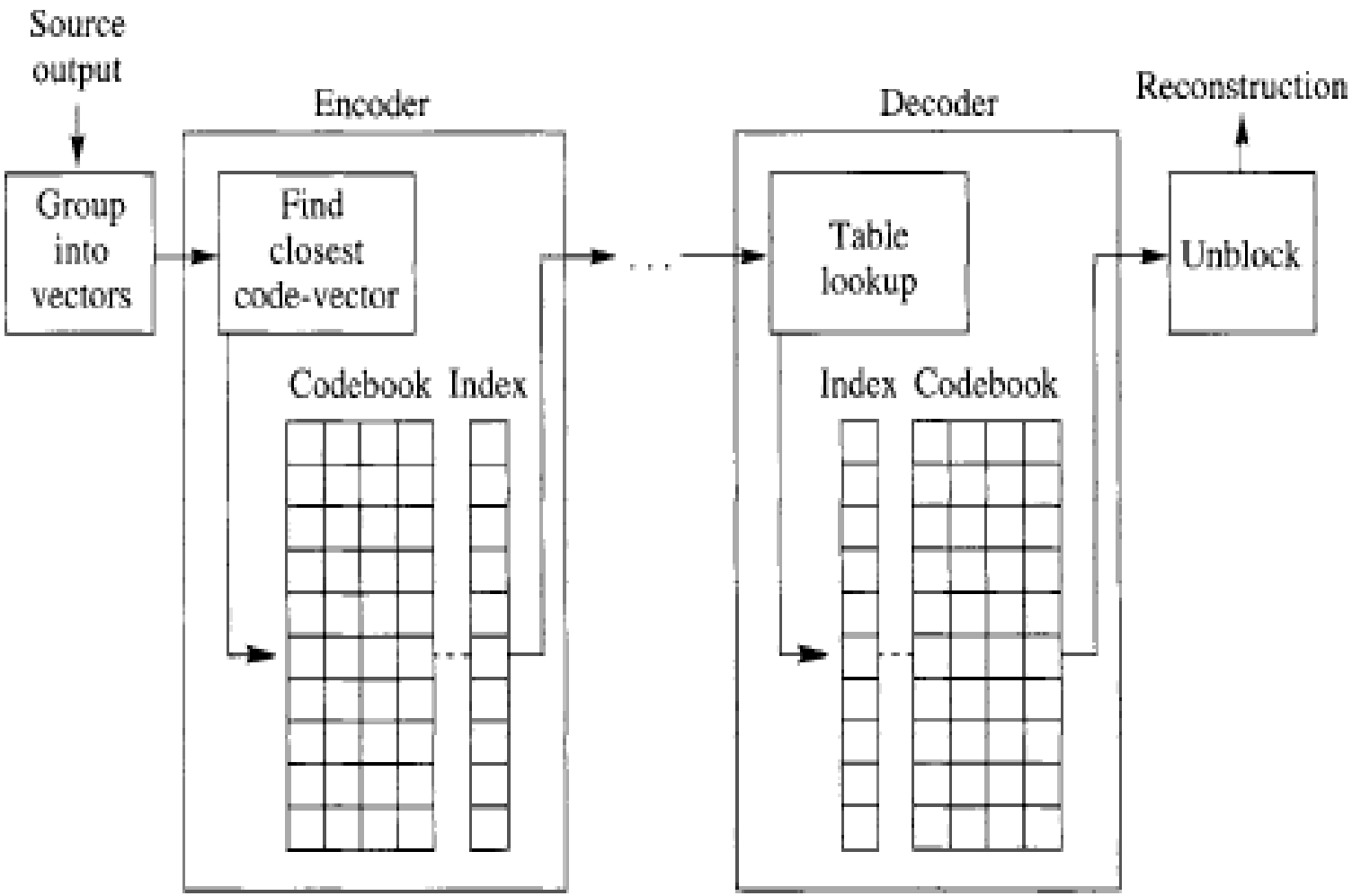


FIGURE 10. 1 **The vector quantization procedure.**

Encoding - Decoding

- At the Encoder, the input vector is compared to each code-vector in order to find the code-vector closest to the input vector.
- In order to inform the decoder about which code-vector was found to be the closest to the input vector, we transmit/store the binary index of code-vector.
- As the decoder has the same codebook, it can retrieve the code vector by performing a lookup in the codebook based on the binary index.

- Although the encoder may have to perform a considerable amount of computations in order to find out the closest reproduction vector to the vector of source outputs.
- The decoding simply consist of Table lookup.
- This makes vector quantization an attractive encoding scheme for applications in which
  - the resources available for decoding are considerably less than the resources available for encoding.
  - Example Multimedia

Advantages

- • Exploit structure in the source samples
- • More flexibility in terms of design

Vector Quantization

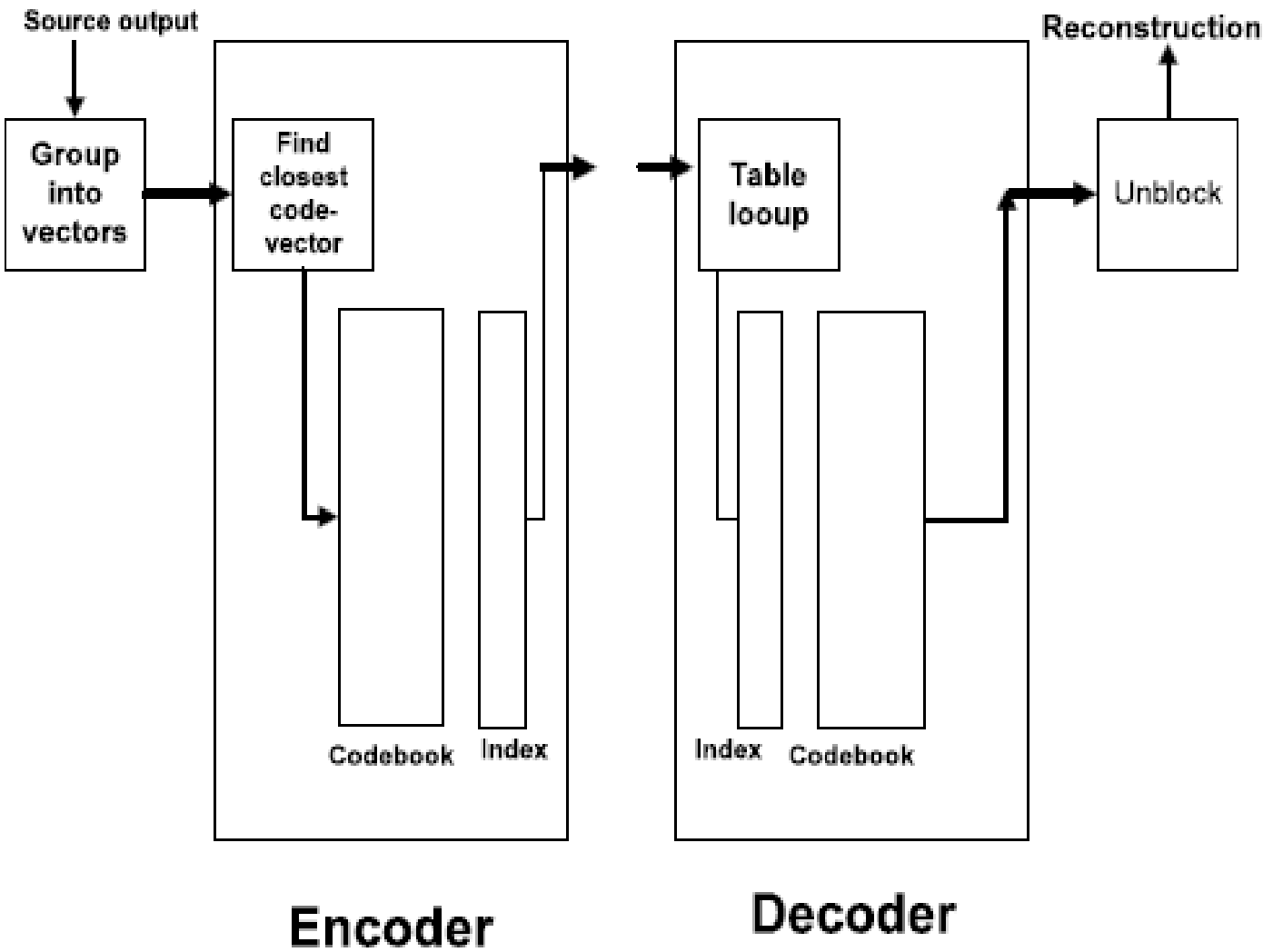
- By grouping source outputs together and encoding them we can extract the source structure and obtain efficient compression.
- Source outputs are grouped into blocks or vectors of length L.
- At the encoder and decoder there is a set of M, L-dimensional vectors called the code-book.
- Vectors in the code-book are called code vectors (levels) or output points.
- Each code vector is assigned a binary index.

Vector Quantization

- Encoder: the input vector is compared to each of the code vectors to find the closest one.
- The binary index of the selected code vector is sent to decoder.
- Decoder has exactly the same codebook and can retrieve the code vector given the binary index.
- If the output of the VQ codebook is fixed-rate coded, then the rate is $r = (\mathrm{Log}_2 M)/L$ bit per sample.
- Distortion measure: mean squared error $\|X - Y_j\|^2 \le \|X - Y_i\|^2$

  $Q(X) = Y_j$
- An important problem in VQ is the design of the code-book

Vector Quantization



Vector Quantization

- Example: 2-D Quantizer
  - A map of the city that is divided into school districts.
  - Each school in its district represents the code vector.
  - Input: Location of child's residence.
  - Output: Rule by which each child is assigned to a school.

- Theorem: Given a signal vector (or data vector), no other coding technique exists that can outperform VQ [Shannon].

Chapter-10

Techniques that transmit information by encoding differences are called Differential encoding techniques.

Differential Coding

- In many sources, the source output does not change a great deal from one sample to the next.
- This means that both the dynamic range and the variance of the sequence of differences $\{d_n = x_n - x_{n-1}\}$ are significantly smaller than that of the source output sequence.
- For correlated sources the distribution of d_n is highly peaked at zero.
- Techniques that transmit information by encoding differences are called differential encoding

Let us assume that both transmitter and receiver start with the same value x_0, Following are the quantization and reconstruction process

$$d_1 = x_1 - x_0$$

$$\hat{d}_1 = Q(d_1) = d_1 + q_1$$

$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - x_1$$

$$\hat{d}_2 = Q(d_2) = d_2 + q_2$$

$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = x_1 + q_1 + d_2 + q_2 = x_2 + q_1 + q_2$$

$$\hat{x}_n = x_n + \sum_{k=1}^{n} q_k$$

We can see that quantization error accumulates as the process continues.

Only information available to the receiver about the sequence $\{x_n\}$ is the reconstructed sequence $\{\hat{x}_n\}$, as this information is also available to the transmitter , we can modify the differencing operation to use the reconstructed value of the previous sample

$d_n = x_n - \hat{x}_{n-1}$

Using this differencing operation let us examine quantization and reconstruction process, assuming $\hat{x}_0 = x_0$

$$d_1 = x_1 - x_0$$
$$\hat{d}_1 = Q(d_1) = d_1 + q_1$$
$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - \hat{x}_1$$
$$\hat{d}_2 = Q(d_2) = d_2 + q_2$$
$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = \hat{x}_1 + d_2 + q_2 = x_2 + q_2$$

$$\hat{x}_n = x_n + q_n$$

No accumulation of quantization noise

3.4 Adaptive Huffman Coding

Problems with Huffman Coding
Huffman coding requires knowledge of the probabilities of the source sequence.
If this knowledge is not available, Huffman coding becomes a two-pass procedure: the statistics are collected in the first pass, and the source is encoded in the second pass.
Theoretically, if we wanted to encode the (K+1)th symbol using the statistics of the first k symbols, we could re-compute the code using the Huffman coding procedure each time a symbol is transmitted.
However, this would not be a very practical approach due to the large amount of computation involved—hence, the adaptive Huffman coding procedures.

Binary Tree

The Huffman code can be described in terms of a binary tree similar to the ones you are drawing till now **but nodes that correspond to Symbols in the alphabet will be denoted by the squares nodes.**

The codeword for a symbol can be obtained by traversing the tree from the root to the leaf corresponding to the symbol, where 0 corresponds to a left branch and 1 corresponds to a right branch.

For adaptive Huffman code works**, we add two other parameters to the binary tree**

1. **Weight** (Written as a number inside node)
   - **1.1. Weight of Internal Node –** Sum of the Weight of its offspring
   - **1.2. Weight of External / Leaf Node –** number of times the symbol corresponding to the leaf has been encountered

2. **Node Number**
   - The node number y_i is a unique number assigned to each internal and external node.
   - If we have an alphabet of size n, then the 2n — 1 internal and external nodes can be numbered as $y_1, y_2, \ldots, y_{2n-1}$ such that if x_j is the weight of node y_j, we have $x_1 \le x_2 \le \ldots\ldots \le x_{2n-1}$
   - Furthermore, the nodes y_{2j-1} and y_{2j} are offspring of the same parent node, or siblings, for $1 \le j < n$, and the node number for the parent node is greater than y_{2j-1} and y_{2j}
   - These last two characteristics are called the **sibling property**, and any tree that possesses this property is a **Huffman tree**
   - **Conclusion :** For Huffman Tree Root node will have the max node number which is 2n-1 and Node numbers are given from Top to Bottom and Right to Left in descending order

NYT
In the adaptive Huffman coding procedure, neither transmitter nor receiver knows anything about the statistics of the source sequence at the start of transmission.
The tree at both the transmitter and the receiver consists of a **single node** that corresponds to all symbols **not yet transmitted (NYT)** and has a **weight of zero**.
As transmission progresses, nodes corresponding to symbols transmitted will be added to the tree, and the tree is reconfigured using an update procedure.
Before the beginning of transmission, a fixed code for each symbol is agreed upon between transmitter and receiver.

Fix Code

If the source has an alphabet of size **m**, then pick **e** and **r** such that

$\mathbf{m = 2^e + r}$ and $\mathbf{0 < r < 2^e}$.

For English alphabet **m** will be **26**

Convert m to binary. Binary of 26 = 11010

Find the max power of 2 in binary. For 26 max power is 16 viz 2^4

$\mathbf{m = 2^e + r}$**,** we have **m=26** and **e=4** so **r = 26-16=10**

The letter a_k is encoded as,

- if $1 \leq k \leq 2r$ then e+1 bit binary of k-1
- else e bit binary of k-r-1

Example

Encode b: b is a2, k=2 which is between 1 and 20 so find 5 bit binary of 1.

Code for b = 00001

Encode x: x is a24, k=24 which is not between 1 and 20 so find 4 bit binary of 13. Code for x = 1101

Method
When a symbol is encountered for the first time, the code for the NYT node is transmitted, followed by the fixed code for the symbol.
A node for the symbol is then created, and the symbol is taken out of the NYT list.
Both transmitter and receiver start with the same tree structure.
The updating procedure used by both transmitter and receiver is identical.
Therefore, the encoding and decoding processes remain synchronized.

Conclusion
The update procedure requires that the nodes be in a fixed order. This ordering is preserved by numbering the nodes.
The largest node number is given to the root of the tree, and the smallest number is assigned to the NYT node.
Root node will have node number 2n-1. The weight of root node will be sum of weight of all nodes in tree.
The numbers from the NYT node to the root of the tree are assigned in increasing order from left to right, and from bottom to top **OR** The numbers from the Root node to the NYT node are assigned in decreasing order from top to bottom and from right to left
The set of nodes with the same weight in the same or lower depth level makes up a block

Adaptive Huffman Coding – Update procedure

Example 3.4.1 Encoding message [a a r d v a r k], alphabet size – 26

Following codes are generated based on message to be encoded & alphabet size

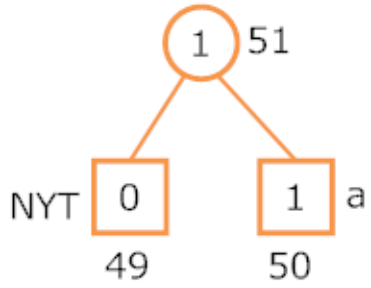
a	00000
d	00011
k	01010
r	10001
v	1011

Total nodes possible: 2n-1 = 51

Root (*Highest*) node number = 51

Initially tree will have only one node NYT numbered 51 and weight zero

a
Read Symbol : a First Appearance of the Symbol? Yes Send Code for NYT Node : NYT node is root, we don't have code for root node Send Index in the NYT List : Code for a 00000 Call update procedure
Update Procedure First Appearance of the Symbol? Yes NYT gives birth to new NYT and External node for 'a' Increment weight of external and Old NYT node. W(a/50) = 1, W(oldNYT/51) =1 Goto old NYT node. No = 51 Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000

a a

First Appearance of the Symbol? No

Code is the path from the root node to the corresponding node (a/50) : 1

Call update procedure

Update Procedure

First Appearance of the Symbol? No

Goto External node of the symbol. No = 50

Node number max in block? Yes (Block {50})

Increment node weight. W(a/50) = 2

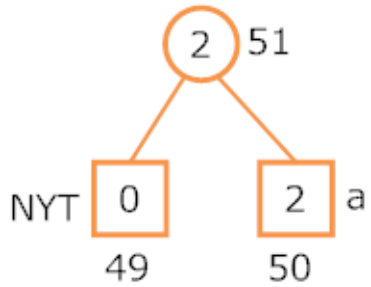
Root node? No

Goto Parent node. No = 51

Node number max in block? Yes (Block {51})

Increment node weight. W(51) = 2

Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000 1

a a r

Read Symbol : r

First Appearance of the Symbol? Yes

Send Code for NYT Node : NYT node is 49, Code 0

Send Index in the NYT List : Code for r 10001

Call update procedure

Update Procedure

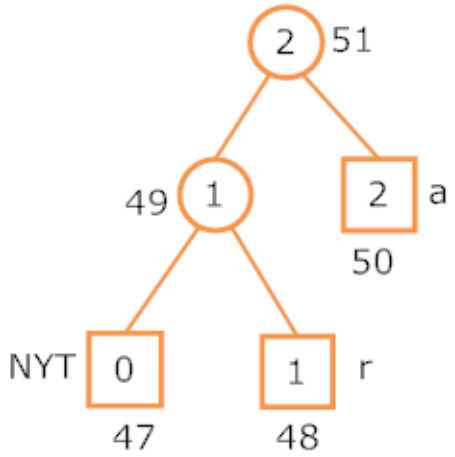
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for 'r'

Increment weight of external and Old NYT node. W(r/48) = 1, W(oldNYT/49) =1

Goto old NYT node. No = 49

Root node? No

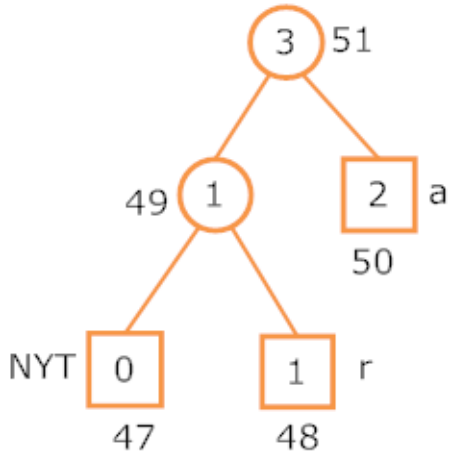


Goto Parent node. No = 51

Node number max in block? Yes (Block {51,50})

Increment node weight. W(51) = 3

Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000 1 0 10001

a a r d

Read Symbol : d

First Appearance of the Symbol? Yes

Send Code for NYT Node : NYT node is 47, Code 00

Send Index in the NYT List : Code for d 00011

Call update procedure

Update Procedure

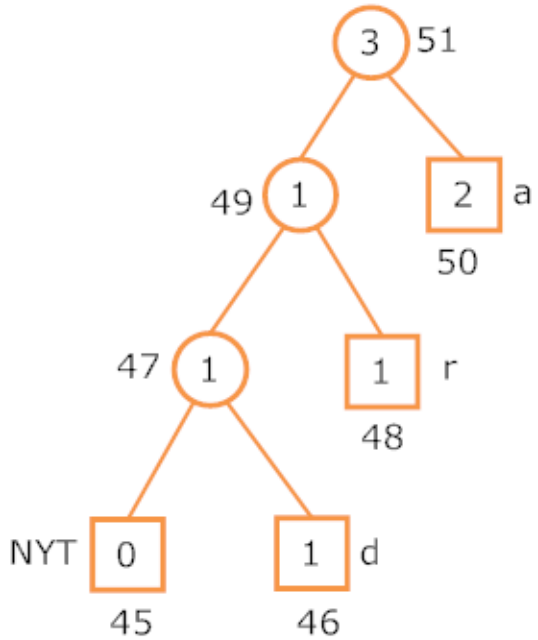
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for 'd'

Increment weight of external and Old NYT node. W(d/46) = 1, W(oldNYT/47) =1

Goto old NYT node. No = 47

Root node? No



Goto Parent node. No = 49

Node number max in block? Yes (Block {49, 48, 47, 46})

Increment node weight. W(49) = 2

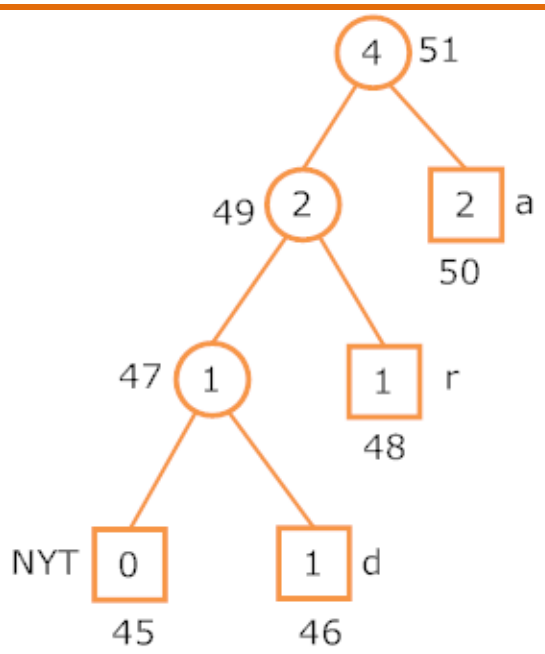
Root node? No

Goto Parent node. No = 51

Node number max in block? Yes (Block {51})

Increment node weight. W(51) = 4

Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000 1 0 10001 00 00011

a a r d v

Read Symbol : v

First Appearance of the Symbol? Yes

Send Code for NYT Node : NYT node is 45, Code 000

Send Index in the NYT List : Code for v 1011

Call update procedure

Update Procedure

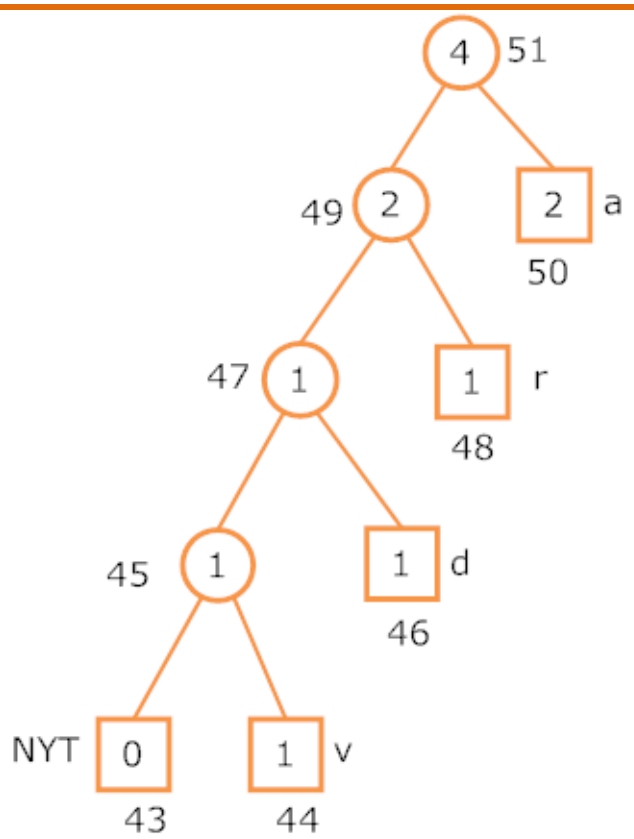
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for 'v'

Increment weight of external and Old NYT node. W(v/44) = 1, W(oldNYT/45) =1

Goto old NYT node. No = 45

Root node? No

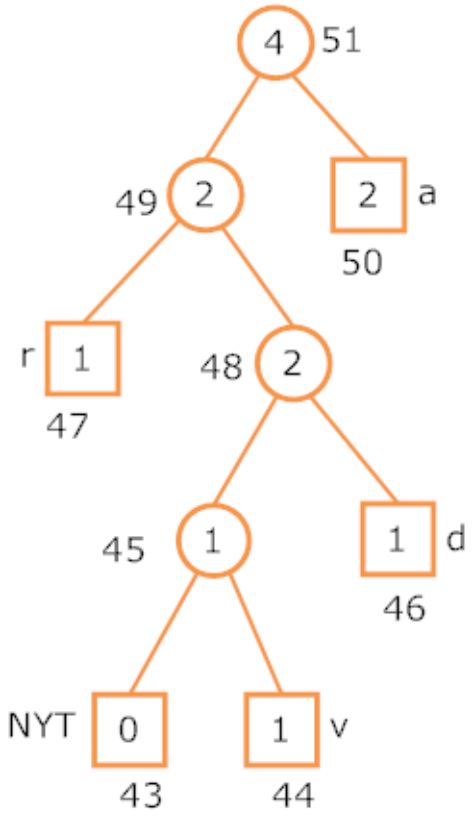


Goto Parent node. No = 47

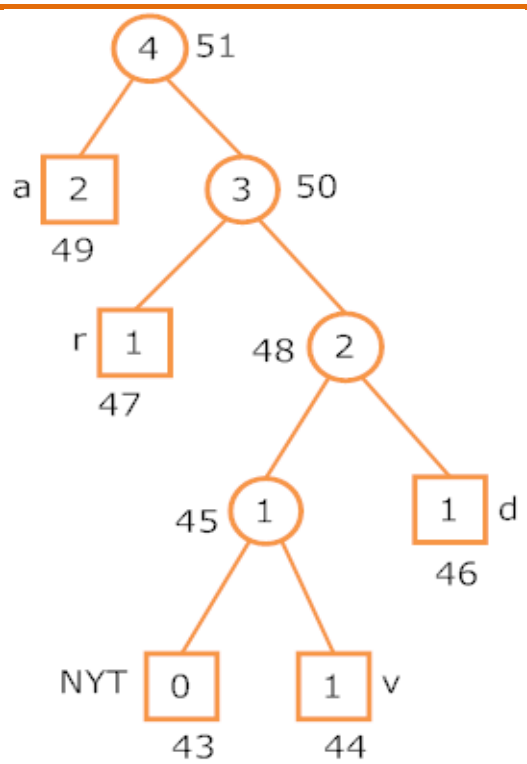
Node number max in block? No (Block {**48**, 47, 46, 45, 44})

Switch node with highest numbered node in block. Switch with 48, ‘r’ will be 47, No =48

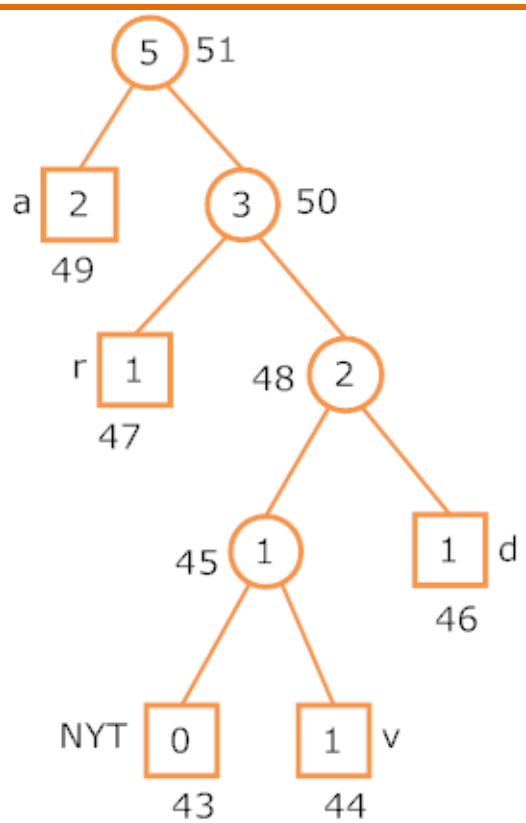
Increment node weight. W(48) = 2



Root node? No
Goto Parent node. No = 49
Node number max in block? No (Block {**50**, 49, 48})
Switch node with highest numbered node in block. Switch with 50, ‘a’ will be 49, No =50
Increment node weight. W(50) = 3



Root node? No
Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 5
Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000 1 0 10001 00 00011 000 1011

a a r d v a

First Appearance of the Symbol? No

Code is the path from the root node to the corresponding node (a/49) : 0

Call update procedure

Update Procedure

First Appearance of the Symbol? No

Goto External node of the symbol. No = 49

Node number max in block? Yes (Block {49,48})

Increment node weight. W(a/49) = 3

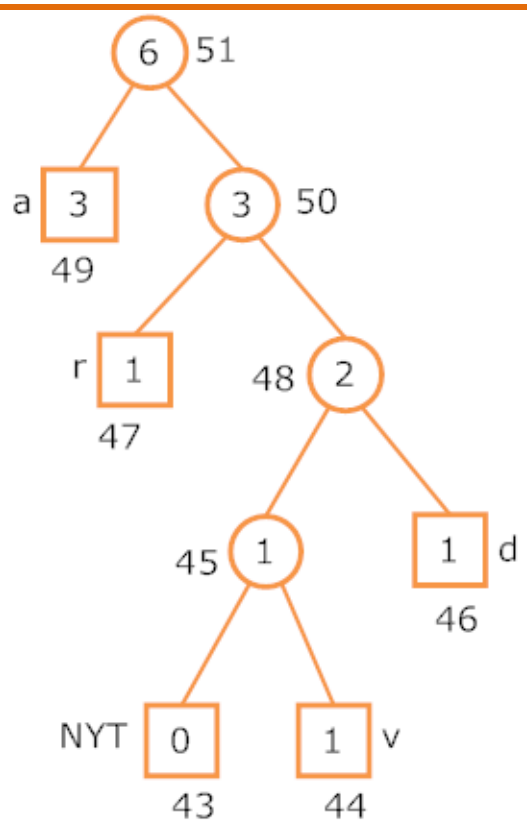
Root node? No

Goto Parent node. No = 51

Node number max in block? Yes (Block {51})

Increment node weight. W(51) = 6

Root node? Yes Stop



Is this the last symbol to read? No, continue...

Output till now 00000 1 0 10001 00 00011 000 1011 0

Can you complete the rest of the message? i.e. r and k

Adaptive Huffman Coding – Update procedure

Example 3.4.1 Decode 00000101000100000110001011 0, alphabet size - 26

Following codes are generated based on alphabet size

	k	1<=k <=2r	K-1	K-r-1	bits	Code
a	1	Yes	0		5	00000
b	2	Yes	1		5	00001
c	3	Yes	2		5	00010
d	4	Yes	3		5	00011
e	5	Yes	4		5	00100
f	6	Yes	5		5	00101
g	7	Yes	6		5	00110
h	8	Yes	7		5	00111
i	9	Yes	8		5	01000
j	10	Yes	9		5	01001
k	11	Yes	10		5	01010
l	12	Yes	11		5	01011
m	13	Yes	12		5	01100
n	14	Yes	13		5	01101
o	15	Yes	14		5	01110
p	16	Yes	15		5	01111
q	17	Yes	16		5	10000
r	18	Yes	17		5	10001

s	19	Yes	18		5	10010
t	20	Yes	19		5	10011
u	21	No		10	4	1010
v	22	No		11	4	1011
w	23	No		12	4	1100
x	24	No		13	4	1101
y	25	No		14	4	1110
z	26	No		15	4	1111

e = 4, r =10

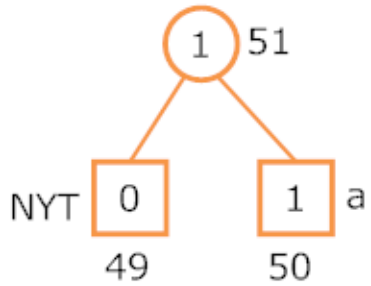
Total nodes possible: 2n-1 = 51

Root (*Highest*) node number = 51

Initially tree will have only one node NYT numbered 51 and weight zero

00000101000100000110001011 0
Goto root of the tree. No = 51 External node? Yes NYT node? Yes Read e bits : 0000 Set P as decimal of read bits. P = D(0000) =0 P < r ? Yes 0 < 10 Read one more bit. 00000, P = D(00000) =0 Decode the P+1 element from the NYT list. P+1 = 1, In list for k=1 symbol is 'a' Call update procedure
Update Procedure First Appearance of the Symbol? Yes NYT gives birth to new NYT and External node for 'a' Increment weight of external and Old NYT node. W(a/50) = 1, W(oldNYT/51) =1 Goto old NYT node. No = 51 Root node? Yes

Stop



Is this the last bit? No, continue...

Output till now a

00000 1010001000001100010110

Goto root of the tree. No = 51

External node? No

Read bit and goto corresponding node. Bit – 1, Node No. 50

External node? Yes

NYT node? No

Decode element corresponding to node. Node no 50 is for ‘a’. Decoded symbol is ‘a’

Call update procedure

Update Procedure

First Appearance of the Symbol? No

Goto External node of the symbol. No = 50

Node number max in block? Yes (Block {50})

Increment node weight. W(a/50) = 2

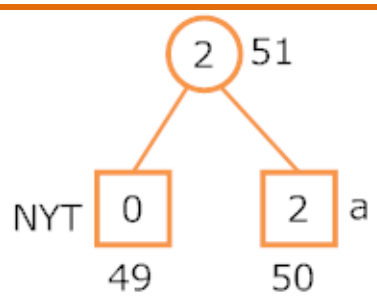
Root node? No

Goto Parent node. No = 51

Node number max in block? Yes (Block {51})

Increment node weight. W(51) = 2

Root node? Yes Stop



Is this the last bit? No, continue...

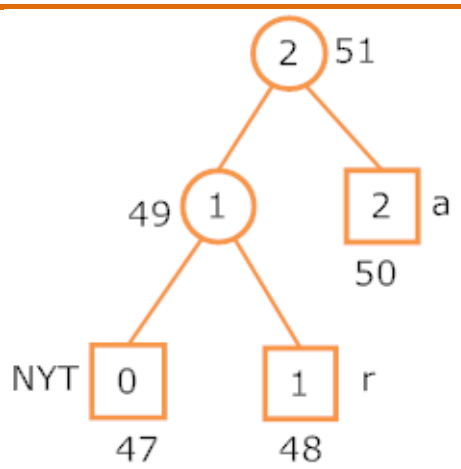
Output till now a a

000001 010001000001100010110

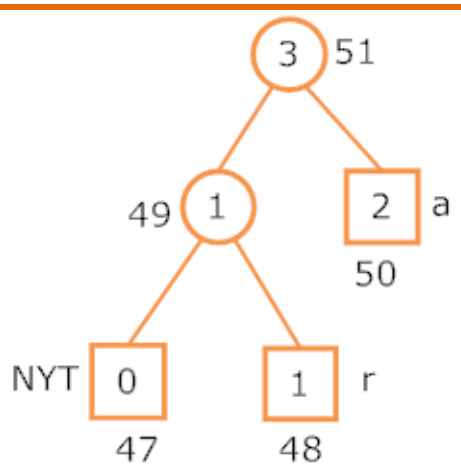
Goto root of the tree. No = 51
External node? No
Read bit and goto corresponding node. Bit – 0, Node No. 49
External node? Yes
NYT node? Yes
Read e bits : 1000
Set P as decimal of read bits. P = D(1000) = 8
P < r ? Yes 8 < 10
Read one more bit. 10001, P = D(10001) =17
Decode the P+1 element from the NYT list. P+1 = 18, In list for k=18 symbol is ‘r’
Call update procedure

Update Procedure

First Appearance of the Symbol? Yes
NYT gives birth to new NYT and External node for ‘r’
Increment weight of external and Old NYT node. W(r/48) = 1, W(oldNYT/49) =1
Goto old NYT node. No = 49
Root node? No



Goto Parent node. No = 51
Node number max in block? Yes (Block {51,50})
Increment node weight. W(51) = 3
Root node? Yes Stop



Is this the last bit? No, continue...

Output till now a a r

000001010001 000001100010110

Goto root of the tree. No = 51

External node? No

Read bit and goto corresponding node. Bit – 0, Node No. 49

External node? No

Read bit and goto corresponding node. Bit – 00, Node No. 47

External node? Yes

NYT node? Yes

Read e bits : 0001

Set P as decimal of read bits. P = D(0001) = 1

P < r ? Yes 1 < 10

Read one more bit. 00011, P = D(00011) =3

Decode the P+1 element from the NYT list. P+1 = 4, In list for k=4 symbol is 'd'

Call update procedure

Update Procedure

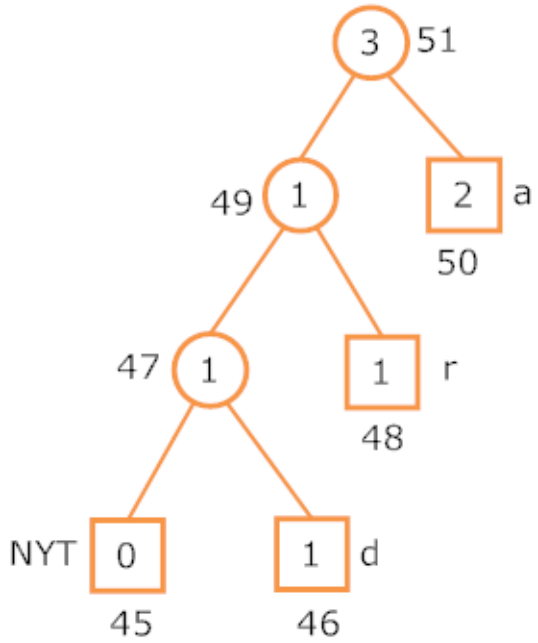
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for 'd'

Increment weight of external and Old NYT node. W(d/46) = 1, W(oldNYT/47) =1

Goto old NYT node. No = 47

Root node? No



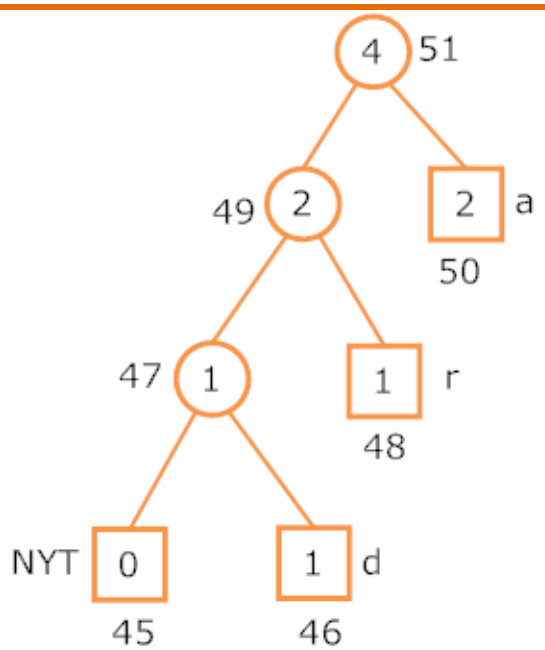
Goto Parent node. No = 49

Node number max in block? Yes (Block {49, 48, 47, 46})

Increment node weight. W(49) = 2

Root node? No

Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 4
Root node? Yes Stop



Is this the last bit? No, continue...

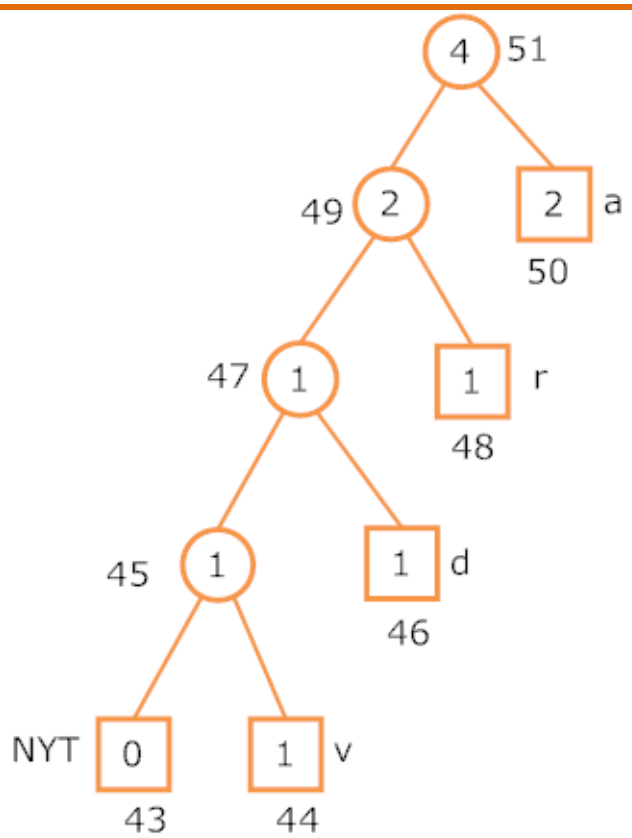
Output till now a a r d

0000010100010000011 00010110

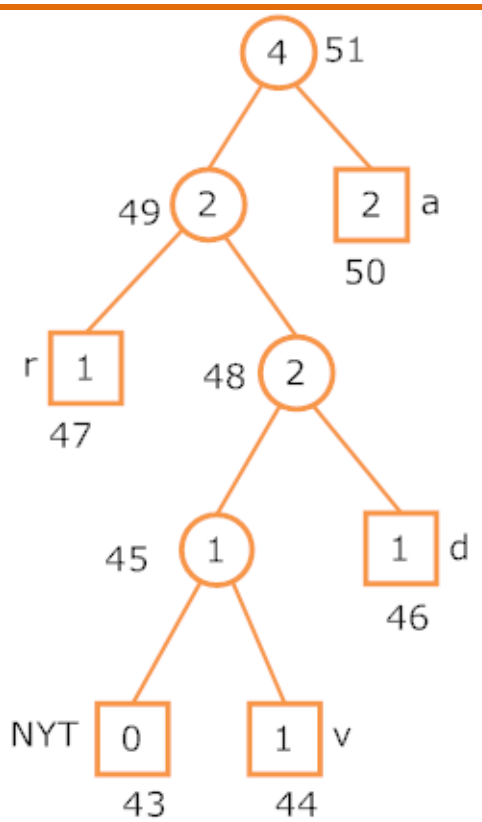
Goto root of the tree. No = 51
External node? No
Read bit and goto corresponding node. Bit – 0, Node No. 49
External node? No
Read bit and goto corresponding node. Bit – 00, Node No. 47
External node? No
Read bit and goto corresponding node. Bit – 000, Node No. 45
External node? Yes
NYT node? Yes
Read e bits : 1011
Set P as decimal of read bits. P = D(1011) = 11
P < r ? No 11 < 10
Add r to P. P = P + r = 11 + 10 = 21
Decode the P+1 element from the NYT list. P+1 = 22, In list for k=22 symbol is 'v'
Call update procedure

Update Procedure
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for ‘v’
Increment weight of external and Old NYT node. W(v/44) = 1, W(oldNYT/45) =1
Goto old NYT node. No = 45
Root node? No



Goto Parent node. No = 47
Node number max in block? No (Block {**48**, 47, 46, 45, 44})
Switch node with highest numbered node in block. Switch with 48, ‘r’ will be 47, No =48
Increment node weight. W(48) = 2



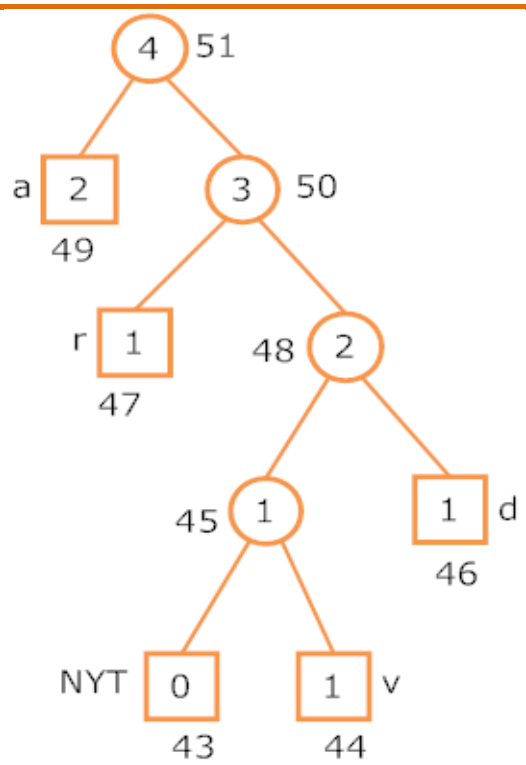
Root node? No

Goto Parent node. No = 49

Node number max in block? No (Block {**50**, 49, 48})

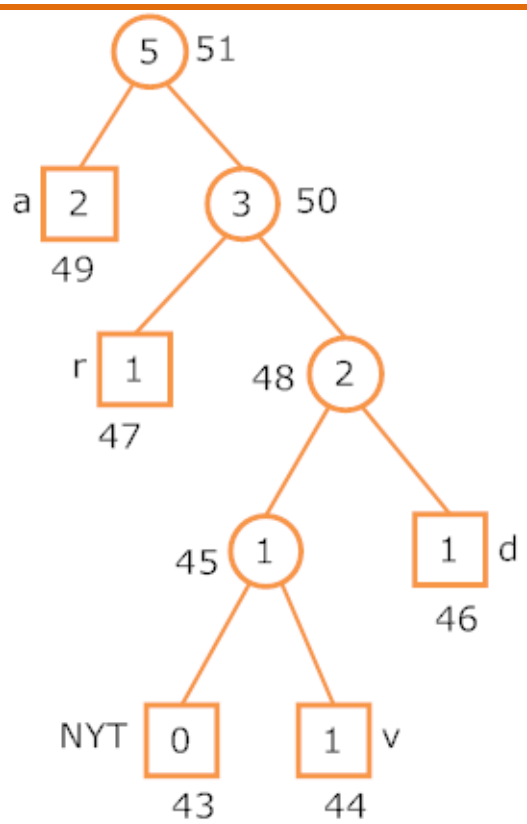
Switch node with highest numbered node in block. Switch with 50, ‘a’ will be 49, No =50

Increment node weight. W(50) = 3



Root node? No

Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 5
Root node? Yes Stop



Is this the last bit? No, continue...

Output till now a a r d v

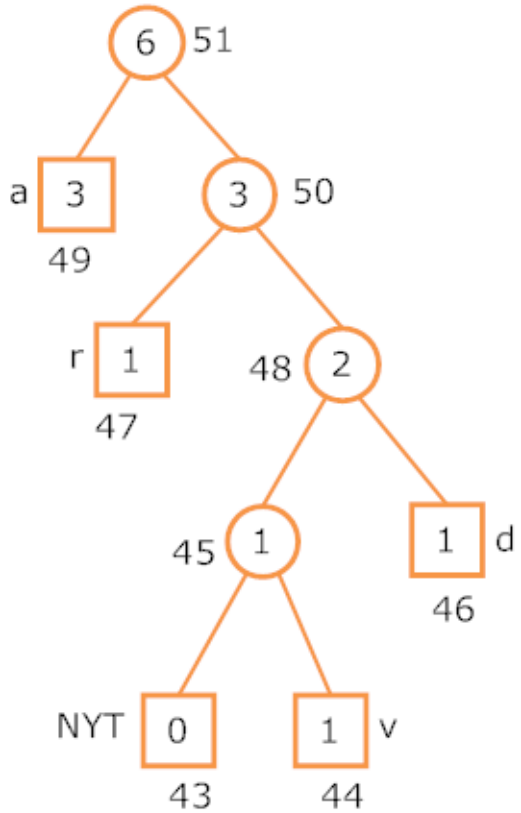
00000101000100000110001011 0

Goto root of the tree. No = 51
External node? No
Read bit and goto corresponding node. Bit – 0, Node No. 49
External node? Yes
NYT node? No
Decode element corresponding to node. Node no 49 is for ‘a’. Decoded symbol is ‘a’
Call update procedure

Update Procedure
First Appearance of the Symbol? No
Goto External node of the symbol. No = 49
Node number max in block? Yes (Block {49,48})
Increment node weight. W(a/49) = 3
Root node? No
Goto Parent node. No = 51
Node number max in block? Yes (Block {51})

Increment node weight. W(51) = 6

Root node? Yes Stop



* Is this the last bit? Yes, Stop

Output till now a a r d v a

* Encoding was performed for only a a r d v a, so decoding is also performed till that only. For your practise you can complete the whole sum.

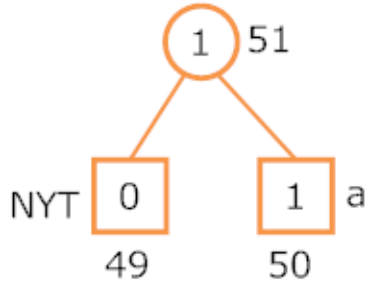
Adaptive Huffman Coding – Update procedure

Example 3.4.1 Encoding message [a a r d v a r k], alphabet size – 26

Total nodes possible: 2n-1 = 51

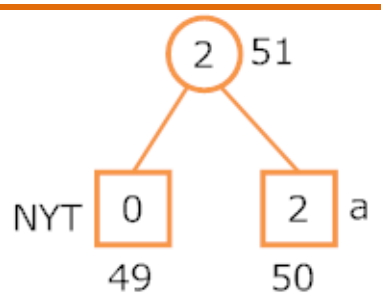
Root (*Highest*) node number = 51

Initially tree will have only one node NYT numbered 51 and weight zero

a
First Appearance of the Symbol? Yes NYT gives birth to new NYT and External node for 'a' Increment weight of external and Old NYT node. W(a/50) = 1, W(oldNYT/51) =1 Goto old NYT node. No = 51 Root node? Yes Stop


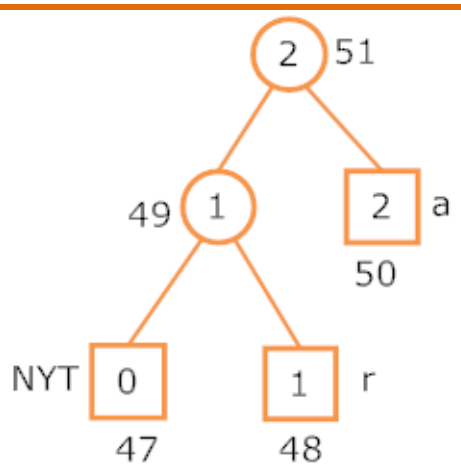
a a

First Appearance of the Symbol? No
Goto External node of the symbol. No = 50
Node number max in block? Yes (Block {50})
Increment node weight. W(a/50) = 2
Root node? No
Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 2
Root node? Yes Stop

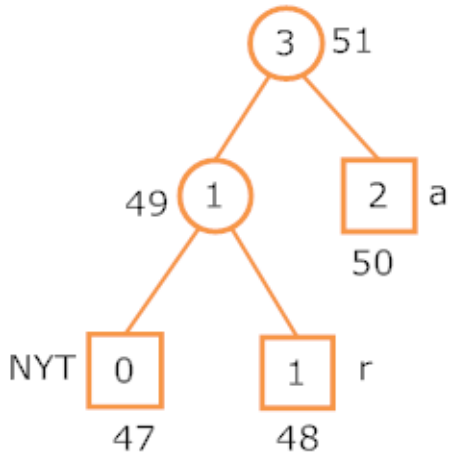


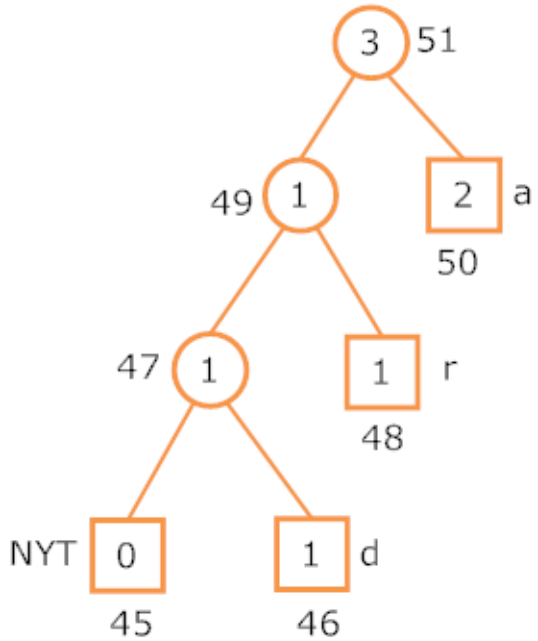
a a r

First Appearance of the Symbol? Yes
NYT gives birth to new NYT and External node for 'r'
Increment weight of external and Old NYT node. W(r/48) = 1, W(oldNYT/49) =1
Goto old NYT node. No = 49
Root node? No

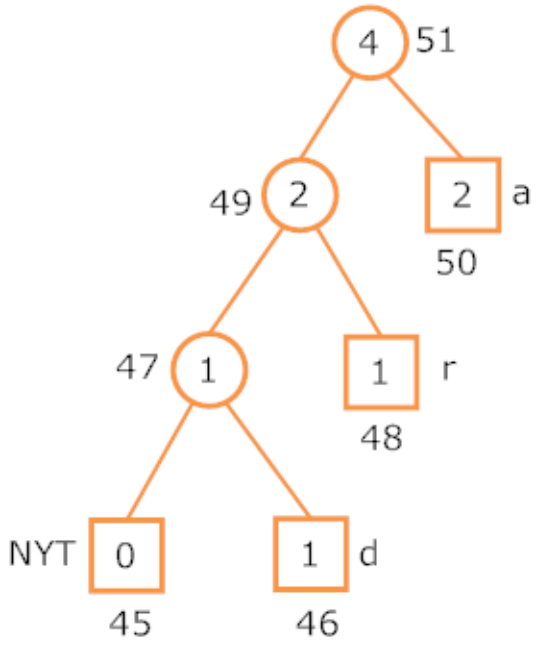


Goto Parent node. No = 51
Node number max in block? Yes (Block {51,50})
Increment node weight. W(51) = 3
Root node? Yes Stop



a a r d
First Appearance of the Symbol? Yes NYT gives birth to new NYT and External node for 'd' Increment weight of external and Old NYT node. W(d/46) = 1, W(oldNYT/47) =1 Goto old NYT node. No = 47 Root node? No

Goto Parent node. No = 49 Node number max in block? Yes (Block {49, 48, 47, 46}) Increment node weight. W(49) = 2 Root node? No Goto Parent node. No = 51 Node number max in block? Yes (Block {51}) Increment node weight. W(51) = 4

Root node? Yes Stop



a a r d v

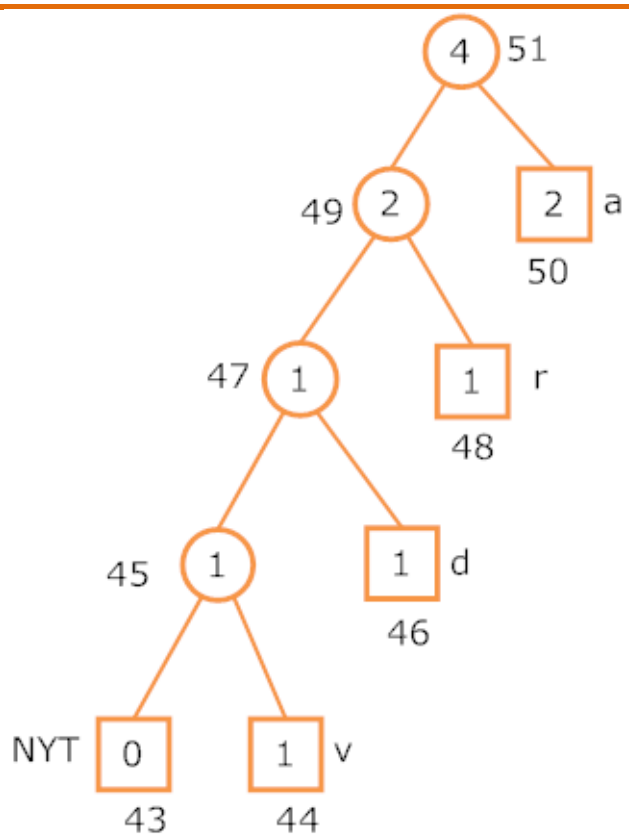
First Appearance of the Symbol? Yes

NYT gives birth to new NYT and External node for 'v'

Increment weight of external and Old NYT node. W(v/44) = 1, W(oldNYT/45) =1

Goto old NYT node. No = 45

Root node? No

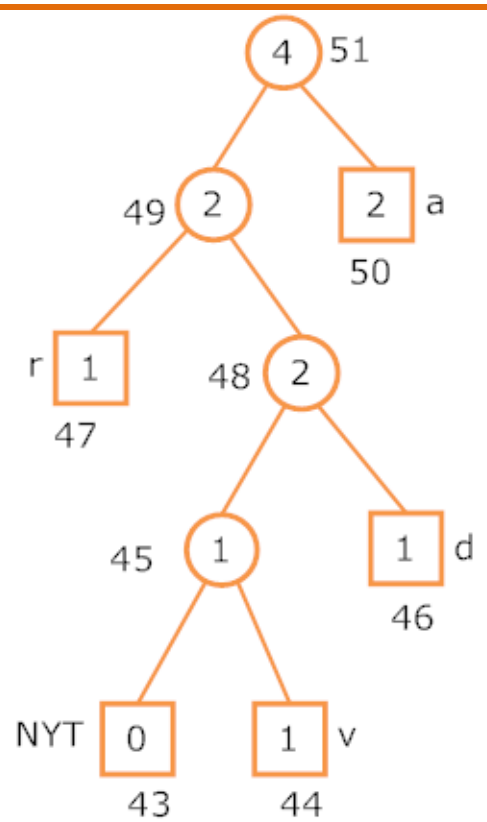


Goto Parent node. No = 47

Node number max in block? No (Block {**48**, 47, 46, 45, 44})

Switch node with highest numbered node in block. Switch with 48, 'r' will be 47, No =48

Increment node weight. W(48) = 2



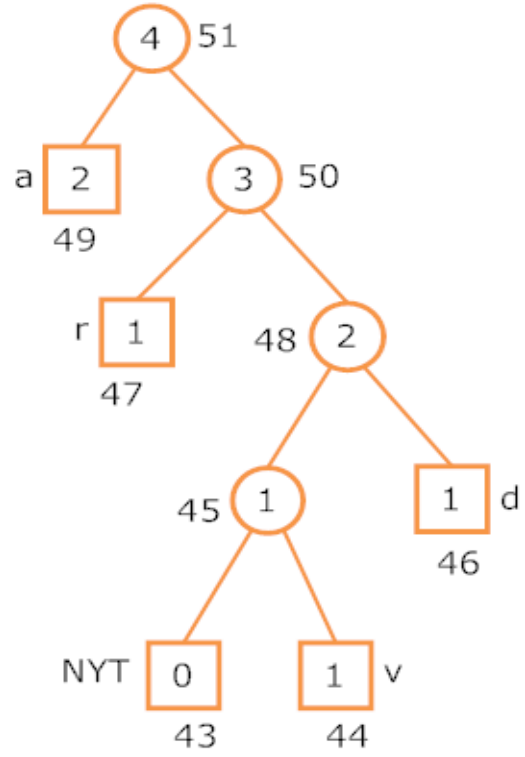
Root node? No

Goto Parent node. No = 49

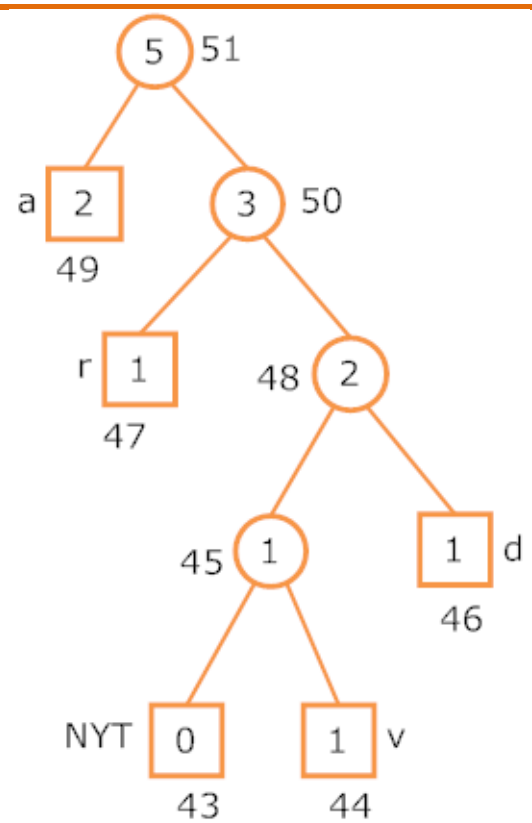
Node number max in block? No (Block {**50**, 49, 48})

Switch node with highest numbered node in block. Switch with 50, ‘a’ will be 49, No =50

Increment node weight. W(50) = 3

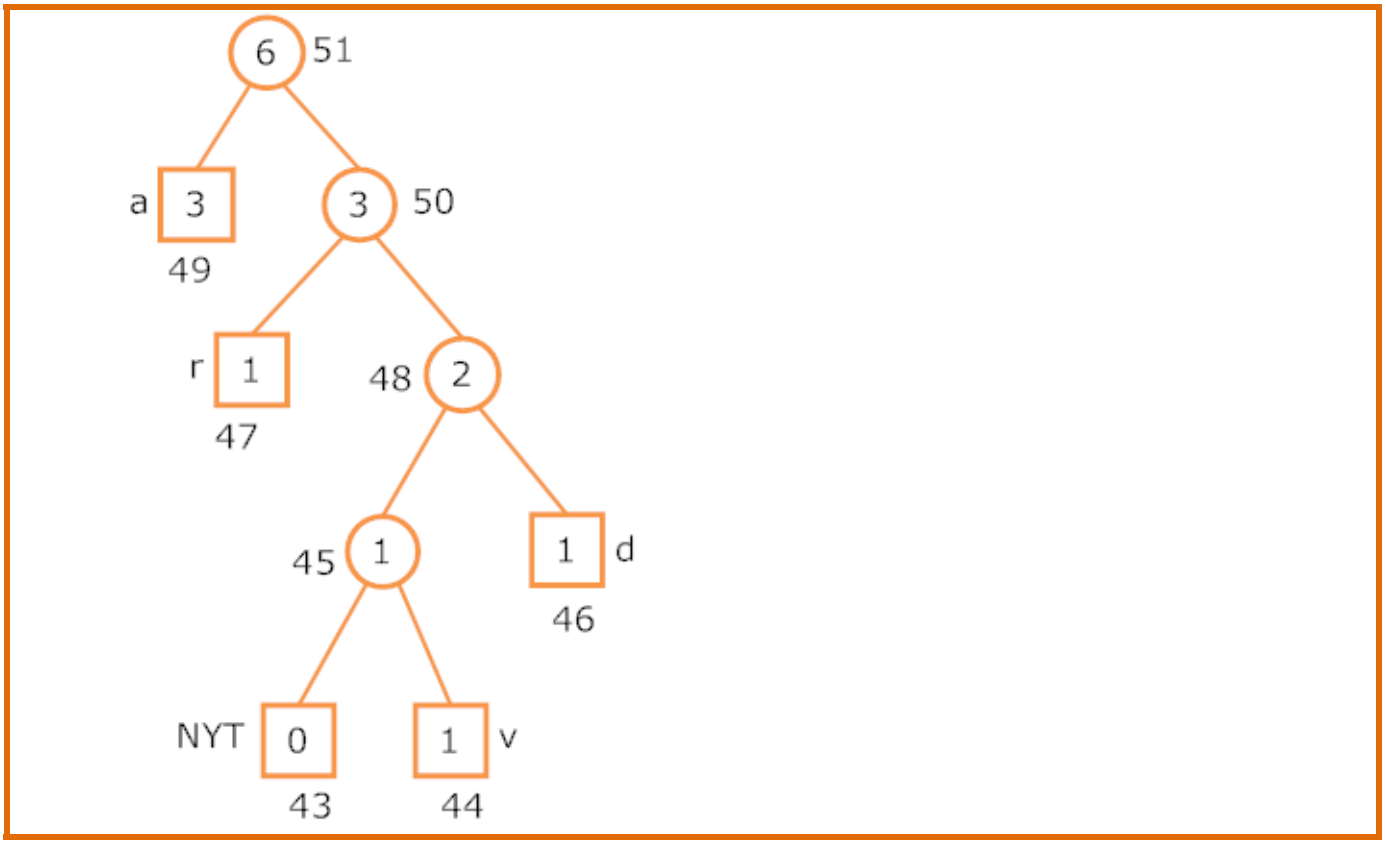


Root node? No
Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 5
Root node? Yes Stop



a a r d v a

First Appearance of the Symbol? No
Goto External node of the symbol. No = 49
Node number max in block? Yes (Block {49,48})
Increment node weight. W(a/49) = 3
Root node? No
Goto Parent node. No = 51
Node number max in block? Yes (Block {51})
Increment node weight. W(51) = 6
Root node? Yes Stop



Can you complete the rest of the message? i.e. r and k

PPMB - "sessions" (largest context size 2)

Count array for context order 2

Context	Letter	Count	PPMB	Cum Count
se	s	1	0	0
	<esc>	1	1	1
	Total		1	
es	s	1	0	0
	<esc>	1	1	1
	Total		1	
ss	i	1	0	0
	<esc>	1	1	1
	Total		1	
si	o	1	0	0
	<esc>	1	1	1
	Total		1	
io	n	1	0	0
	<esc>	1	1	1
	Total	2	1	
on	s	1	0	0
	<esc>	1	1	1
	Total		1	

Count array for context order 1

Context	Letter	Count	PPMB	Cum Count
s	e	1	0	0
	s	1	0	0
	i	1	0	0
	<esc>	1	3	3
	Total		3	
e	s	1	0	0
	<esc>	1	1	1
	Total		1	
i	o	1	0	0
	<esc>	1	1	1
	Total		1	
o	n	1	0	0
	<esc>	1	1	1
	Total		1	
n	s	1	0	0
	<esc>	1	1	1
	Total		1	

Count array for context order 0

Letter	Count	Cum Count
s	~~1~~ ~~2~~ ~~3~~ 4	4
e	1	5
i	1	6
o	1	7
n	1	8

Count array for context order -1

Letter	Count	Cum Count
s	1	1
e	1	2
i	1	3
o	1	4
n	1	5

PPMC - "abacadabc" (largest context size 2)

Count array for context order 2

Context	Letter	Count	PPMC	Cum Count
ab	a	1	1	1
	c	1	1	2
	<esc>	1	2	4
	Total		4	
ba	c	1	1	1
	<esc>	1	1	2
	Total		2	
ac	a	1	1	1
	<esc>	1	1	2
	Total		2	
ca	d	1	1	1
	<esc>	1	1	2
	Total		2	
ad	a	1	1	1
	<esc>	1	1	2
	Total	2	2	
da	b	1	1	1
	<esc>	1	1	2
	Total		2	

Count array for context order 1

Context	Letter	Count	PPMC	Cum Count
a	b	~~1~~ 2	2	2
	c	1	1	3
	d	1	1	4
	<esc>	1	3	7
	Total		7	
b	a	1	1	1
	c	1	1	2
	<esc>	1	2	4
	Total		4	
c	a	1	1	1
	<esc>	1	1	2
	Total		2	
d	a	1	1	1
	<esc>	1	1	2
	Total		2	

Count array for context order 0

Letter	Count	Cum Count
a	~~1~~ ~~2~~ ~~3~~ 4	4
b	1	5
c	~~1~~ 2	7
d	1	8

Count array for context order -1

Letter	Count	Cum Count
a	1	1
b	1	2
c	1	3
d	1	4

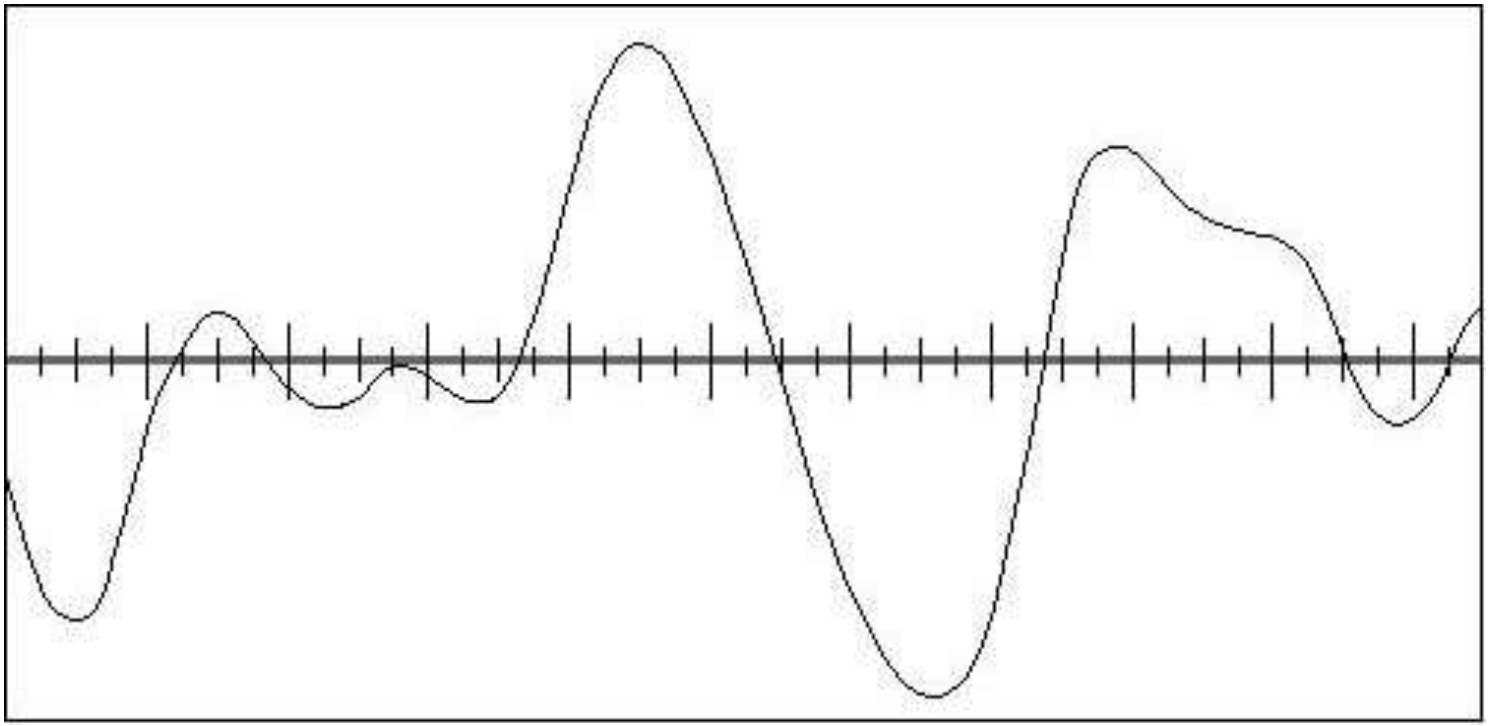
Speech Compression

Introduction

- Use of multimedia in personal computers
- Requirement of more disk space
- Also telephone system requires compression
- Topics to be covered:
  - Digital audio concepts
  - Lossless compression of sound
  - Lossy compression of sound

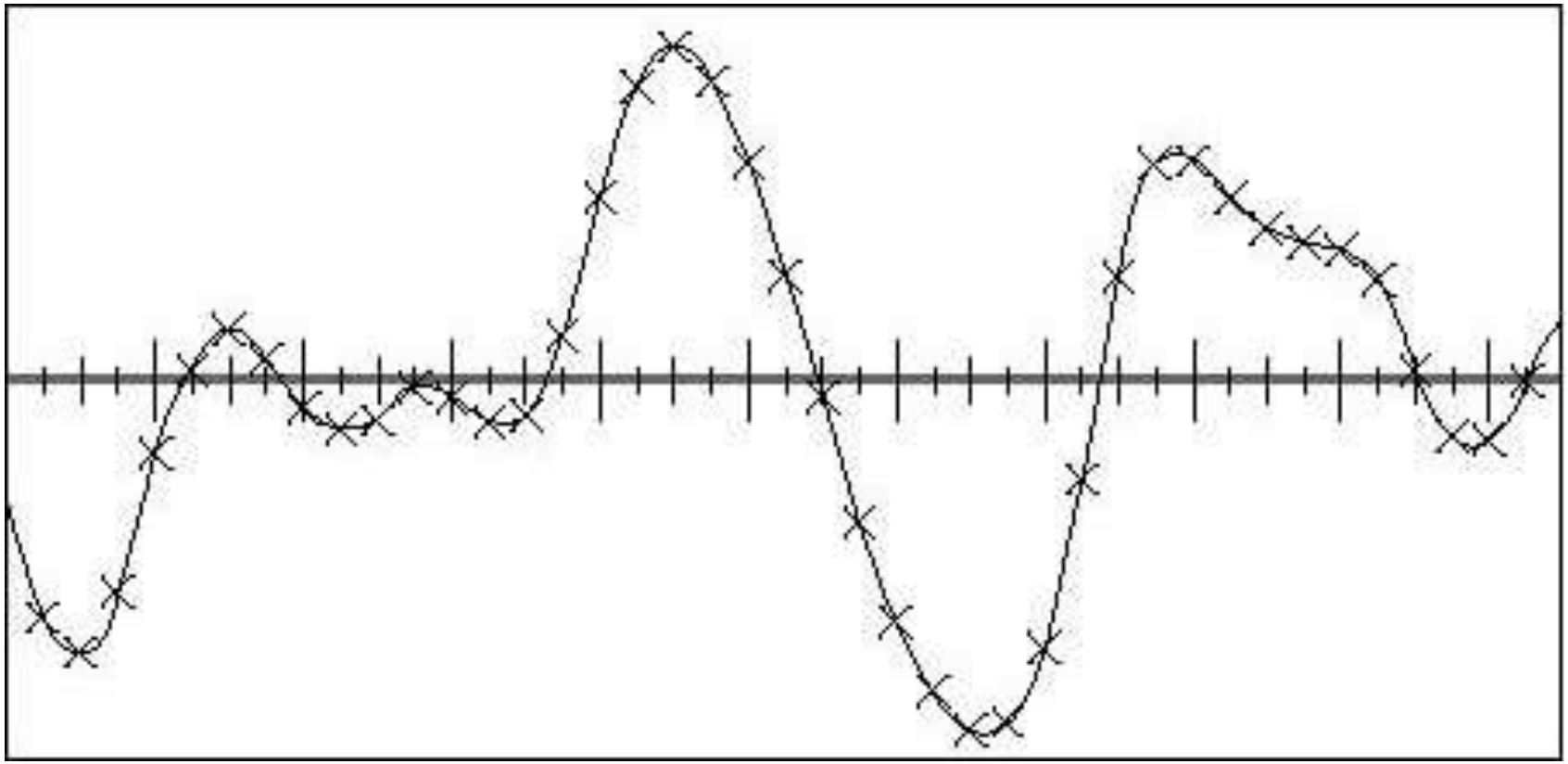
Digital Audio Concepts

- Need to convert sound from analog to digital
- Typical audio waveform



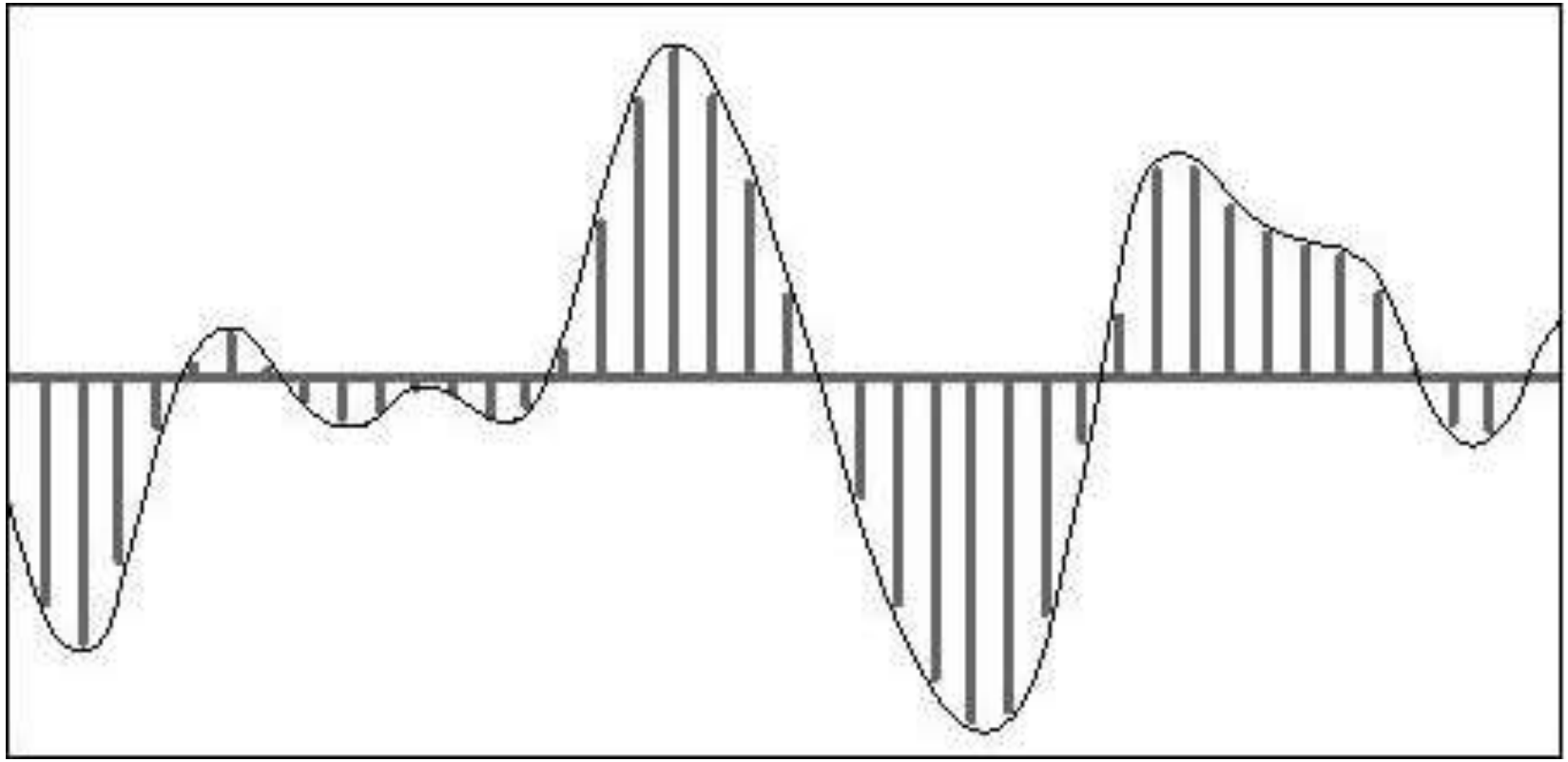
- Mic → generate voltage according to air pressure
- Human ear frequency range upto 20kHz
- It is measured in dB

- Sampling consists of taking measurements of the input signal at regular times, converting them to an appropriate scale, and storing them
- A typical audio waveform being sampled at 8KHz



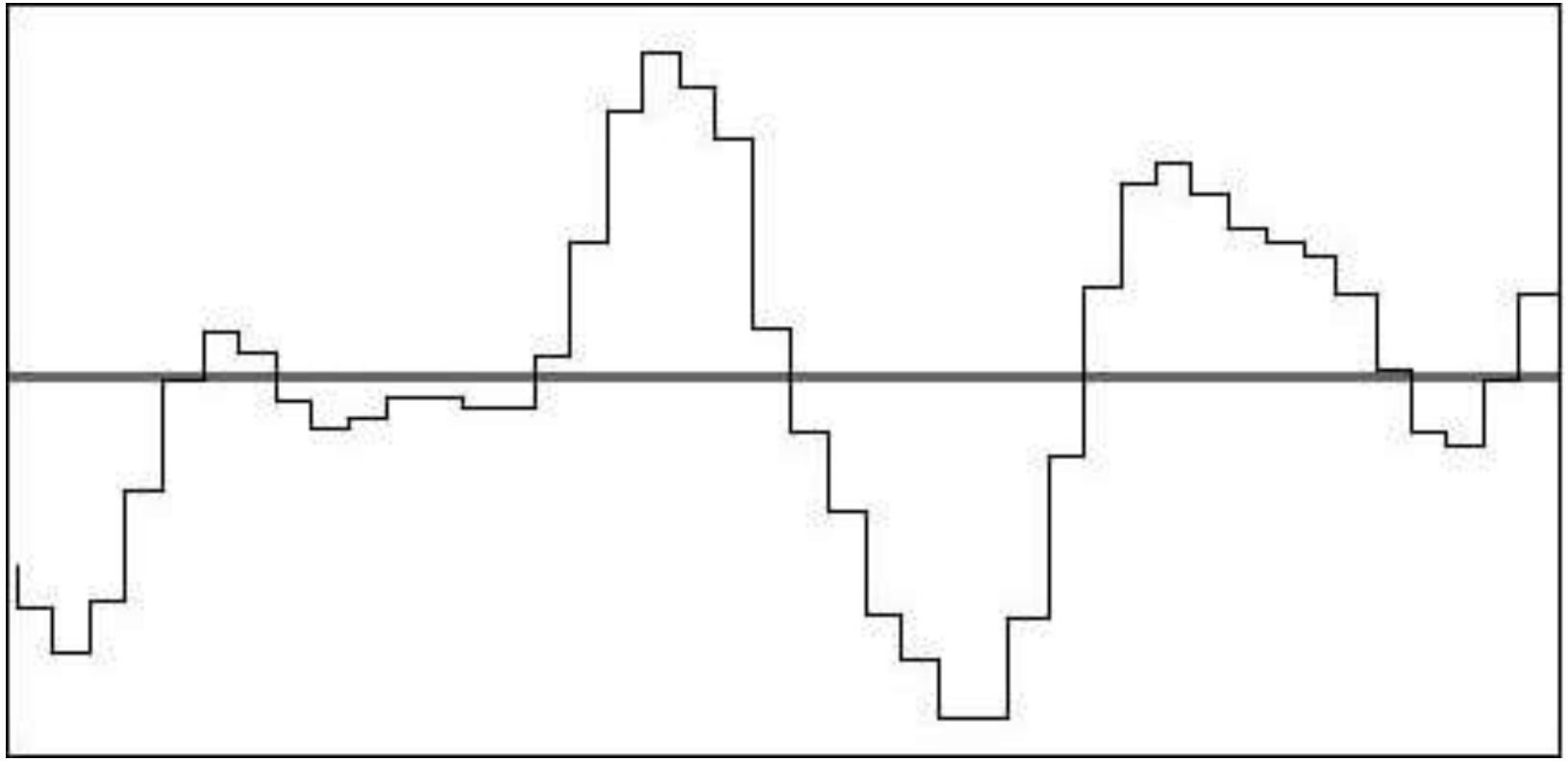
- First step is ADC
- It takes a given voltage and scales it to an appropriate digital measurement
- 8-bit ADC

- Sample voltages overlaid with the input AC signal



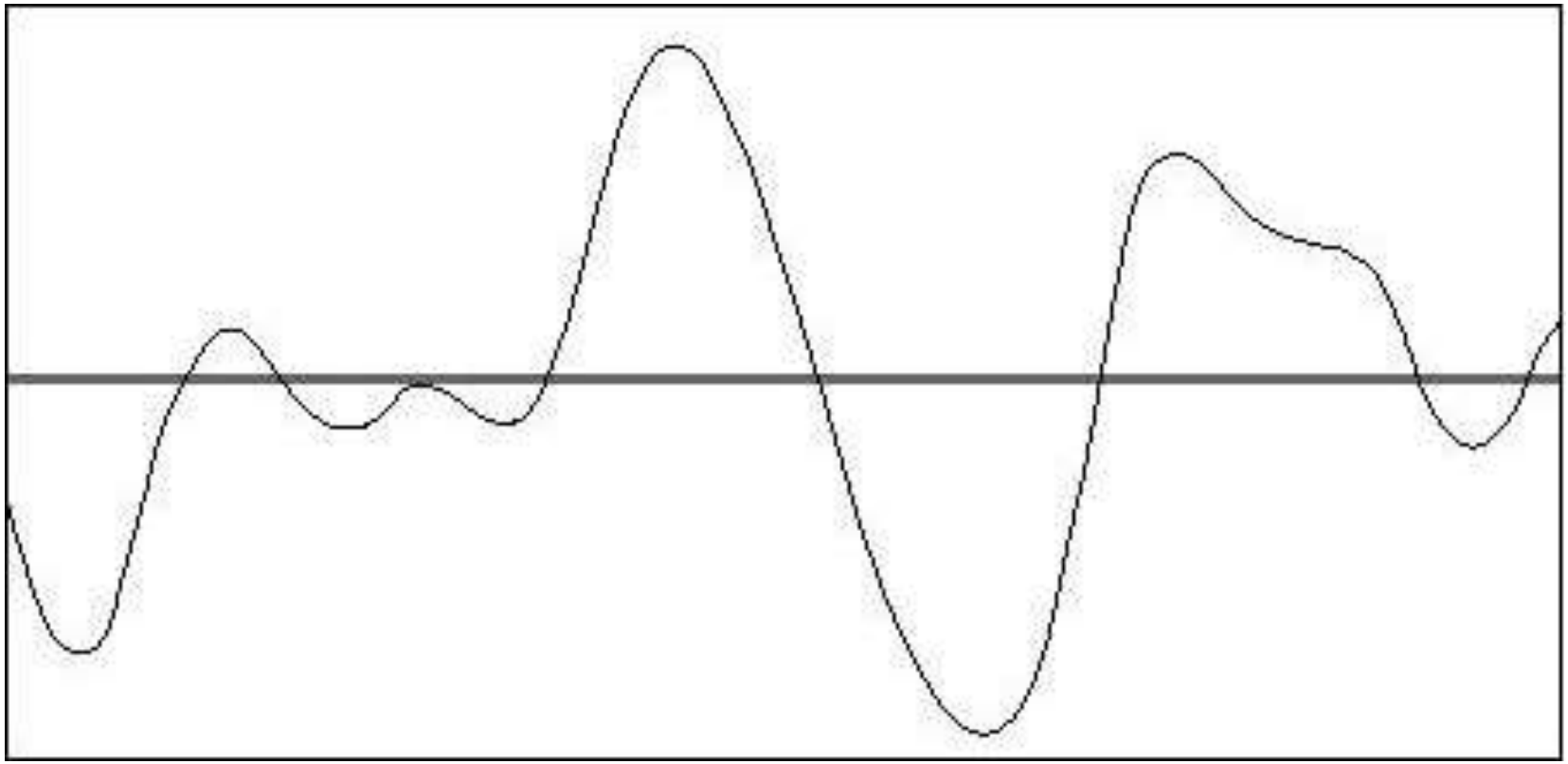
- For playback, DAC is used
- The DAC is responsible for taking a digital value and converting it to a corresponding analog signal
- We may not get exact output but it must be identical

- DAC output



- To smooth the curve, a low-pass filter is used
- It passes the lower frequencies and blocks higher frequencies

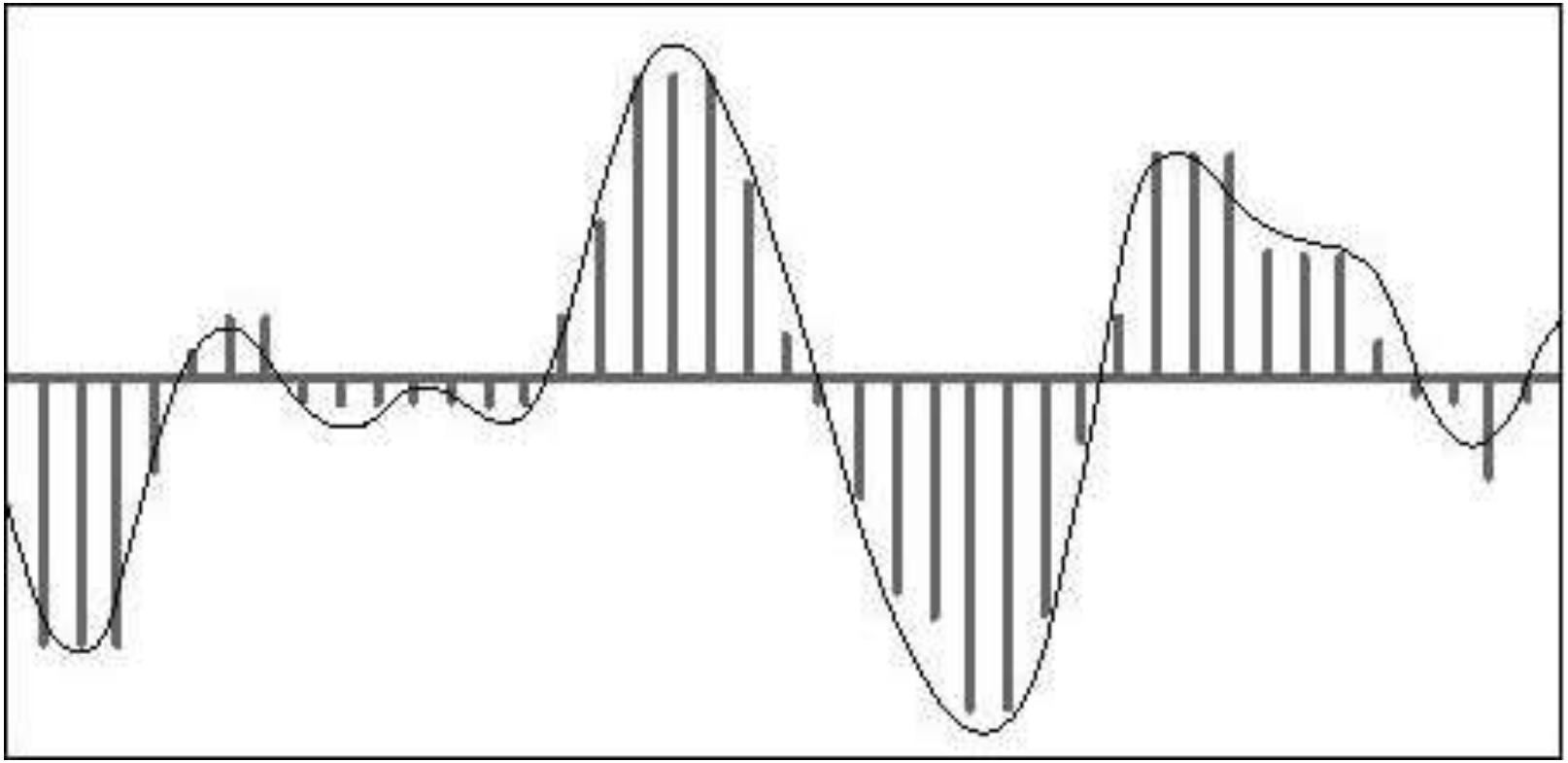
- Sampling at a much higher rate



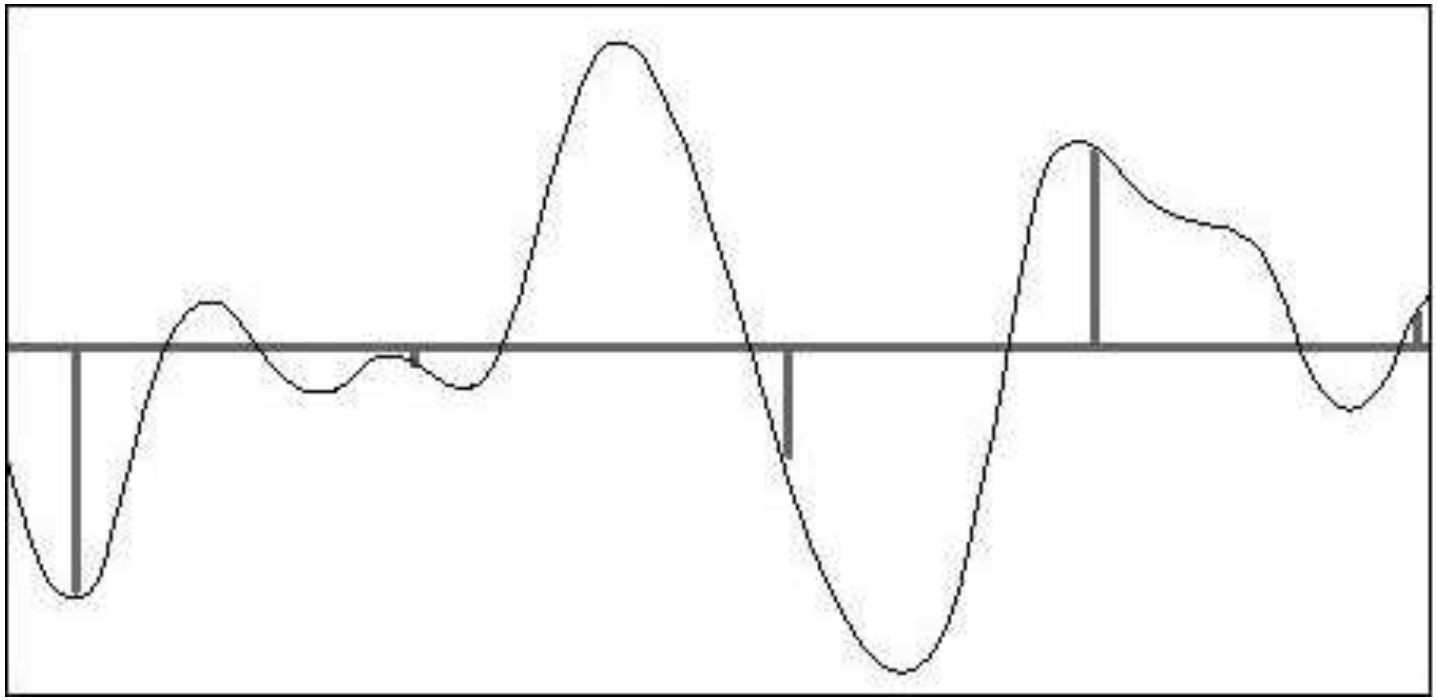
Sampling Variables

- The ***sample resolution*** is simply a measure of how accurately the digital sample can measure the voltage it is recording
- Quantization error
- Depending on quantization error, your output has some distortions
- Crash of drums in an orchestra → 500mv
- Delicate violin solo may never go outside 5mv

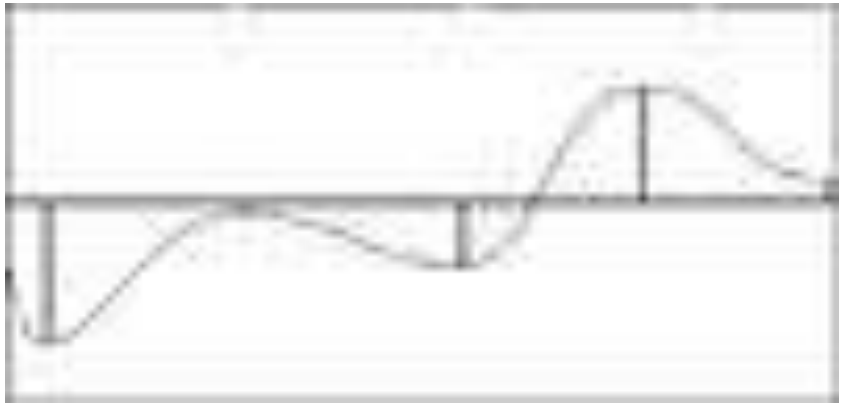
- Quantization error when sampling a waveform



- ***Sampling Rate*** → Time interval for reading
- Nyquist Theoram
- Slower sampling rate



- If sampling rate is much slower, output is not desired
- Output of slower sampling rate



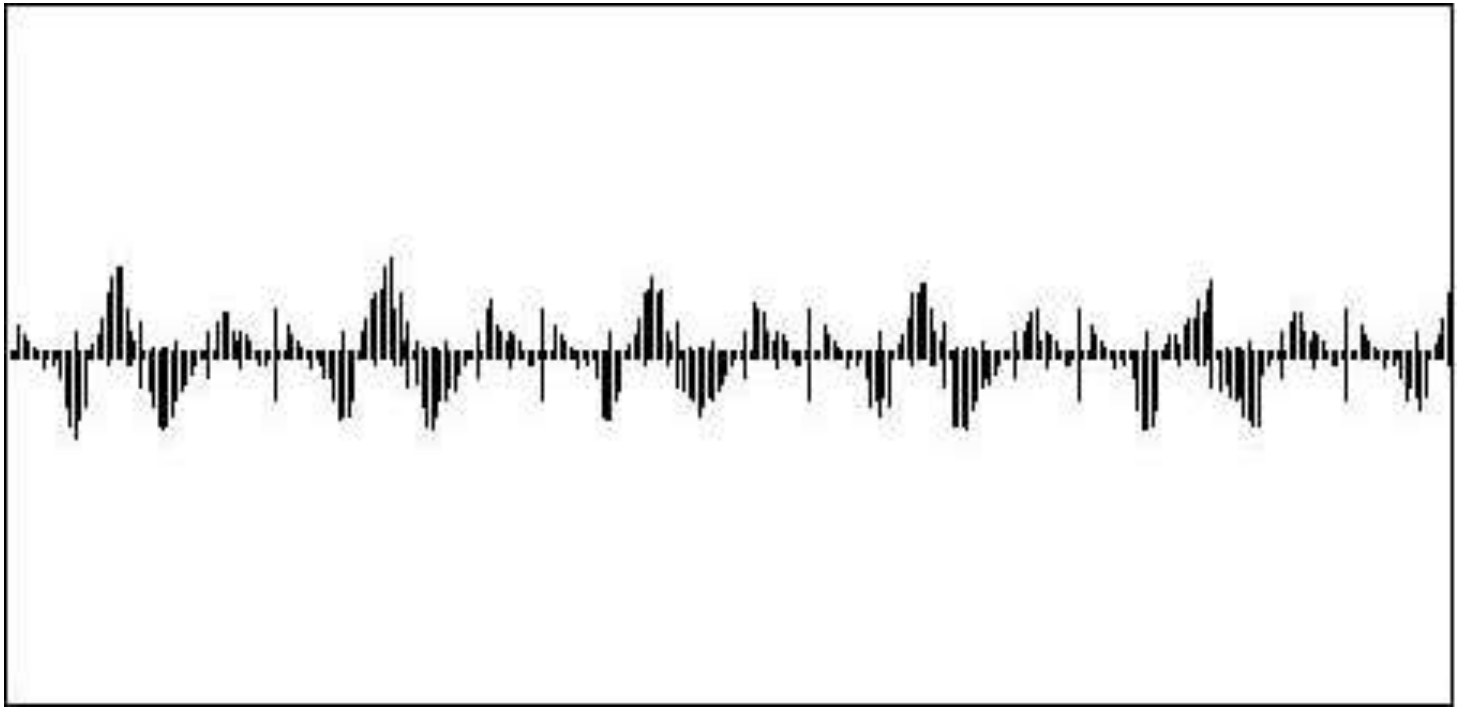
- Human ear capacity → 20kHz
- So sampling rate must be 40kHz or more
- In CD, 44kHz using 10-bit samples
- Digital phone's sampling rate is 8kHz
- Quality of sound
  - Mathematically very hard to measure
  - Based on listeners

PC Based Sound

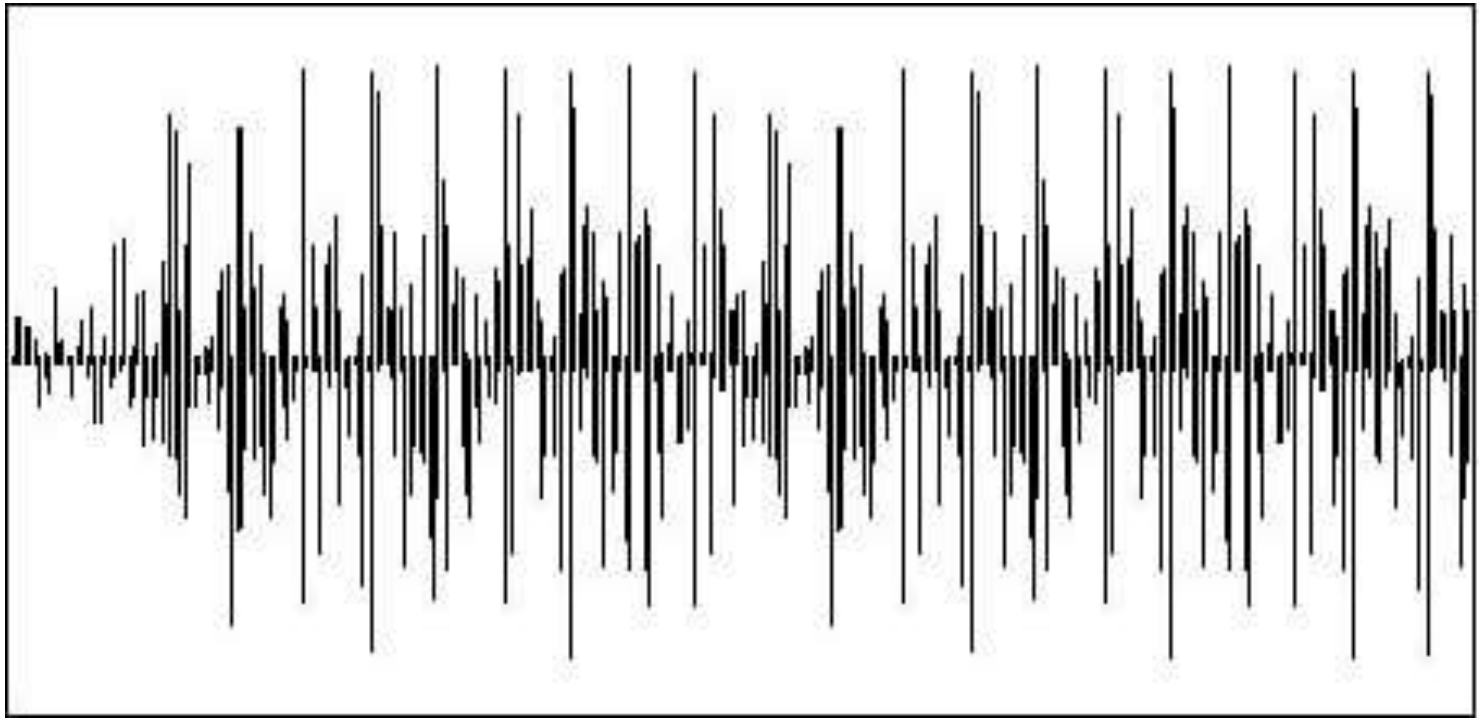
- Early PCs → no sound
- PC speakers has only one bit sound → beep or buzz
- Sounds require higher disk space so higher compression ratio and higher processing power needed.
- Cost of processing is decreasing fast compare to cost of transmission lines

Lossless Compression of Sound

- Generally not used for sound
- Bandwidth is fix for transmission lines
- A sound sample that is easy to compress



- A sound sample that is difficult to compress



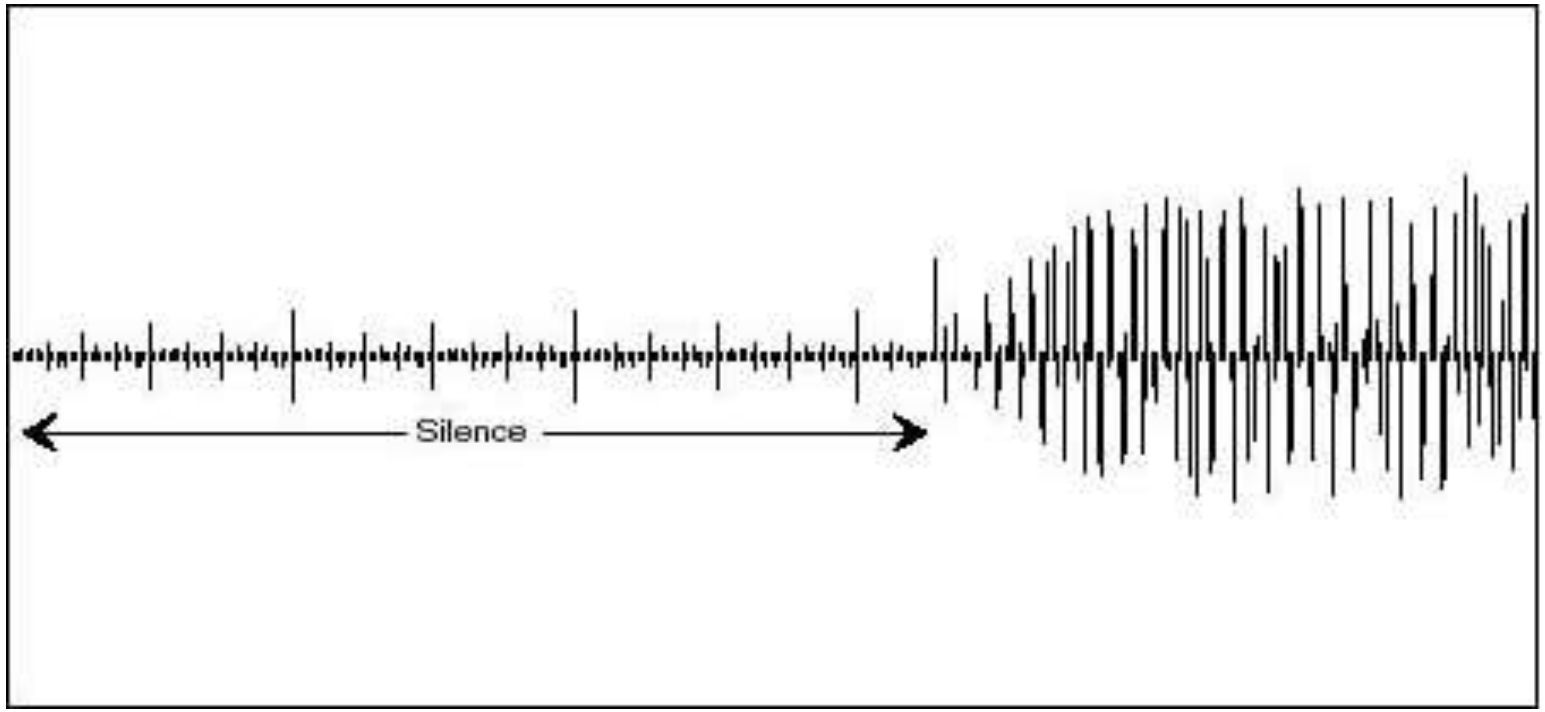
- Lossless compression can not give batter results

Lossy Compression

- Certain loss in fidelity
- Lossy compression followed by lossless compression
- Lossy compression frequently smooths out the data, which makes it even more suitable for lossless compression

Silence Compression

- lossy technique
- A sound sample with a long sequence of silence

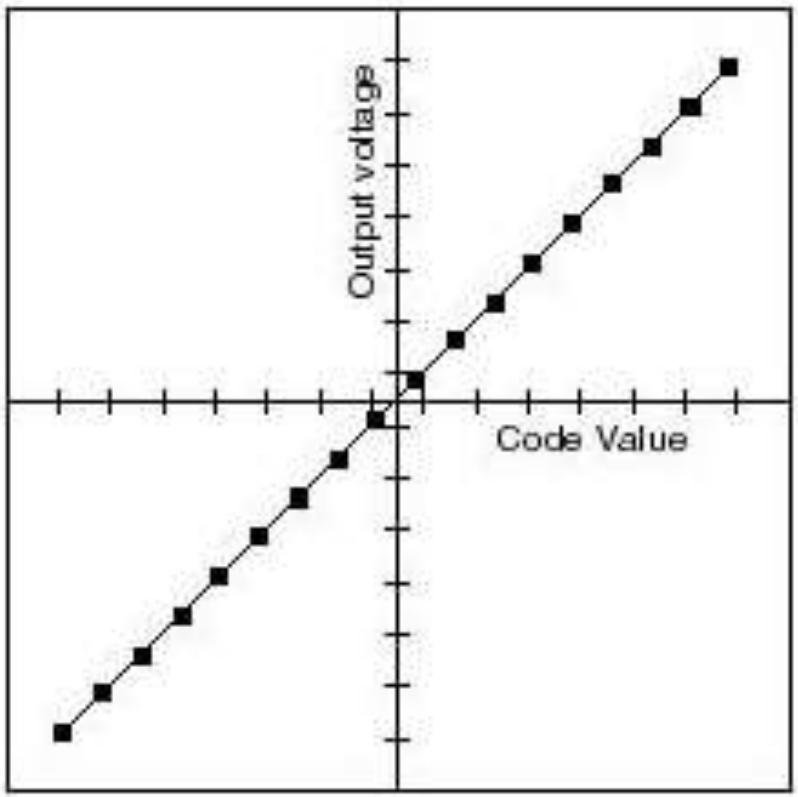


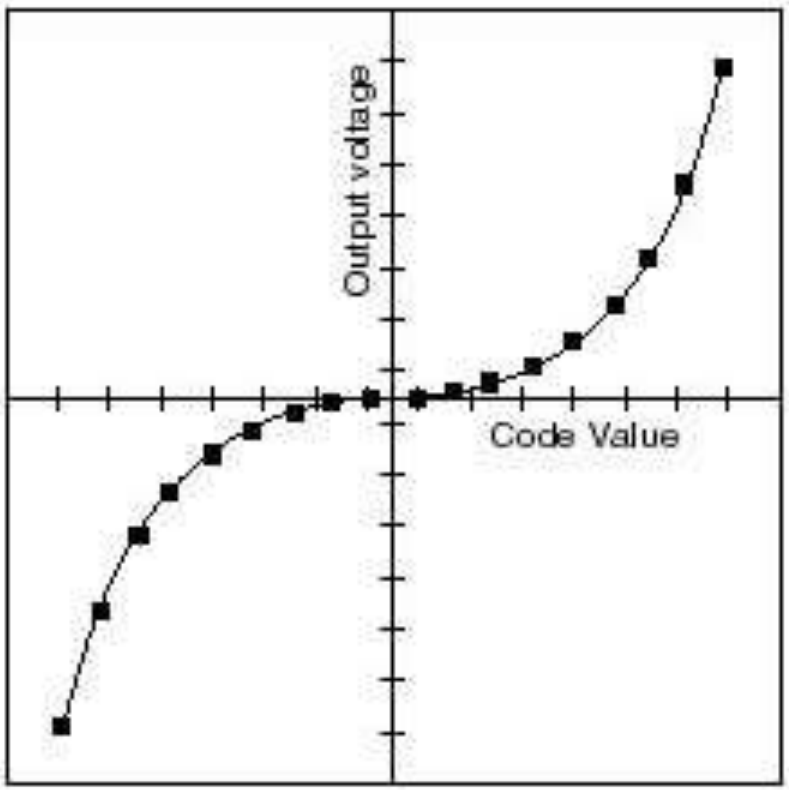
- Parameters of silence compression prog.
  - Threshold value of silence
  - Run of silence
  - Threshold for recognizing the start of a run of silence (generally 4)
  - Threshold for recognizing the stop of silence (generally 2 or 3)
- Mainly it converts noisy silence to pure silence
- It provide an excellent way to remove redundancy from sound files

Companding

- We need higher sampling resolution to cover all sound samples.
- But it generate higher data rate
- And if it is lower, small values are considered as silence
- To solve this, telecommunication industry use non-linear matched set of ADCs and DACs.
- Normal ADC (used in PCs) uses linear conversion schemes

- Linear and non-linear conversion





- Normal ADC and DAC
  - Code change from 0 to 1, voltage change from 0mv to 1mv
  - Code change from 100 to 101, voltage change from 100mv to 101mv
- Today's digital telecommunication equipment are using ***companding codec***
- It does not use standard linear function but uses exponential function
- Resolution of smaller codec values are much finer then larger ones

- Non-linear ADC and DAC
  - Code change from 0 to 1, the difference would be 1mv
  - Code change from 100 to 101, the difference would be 10mv
- This will give effective range of 13-bit codecs by using only 8-bit samples
- Example

Input code	0	1	2	3	4	5	6	7
Output value	0	13	28	44	62	81	103	127

- This algorithm does an excellent job of compressing sound without damaging quality
- Both compression and decompression can take place via lookup table, so faster processing
- Amount of compression is known in advance and does not depend on input file
- Algorithm can be tuned to any desired compression ratio