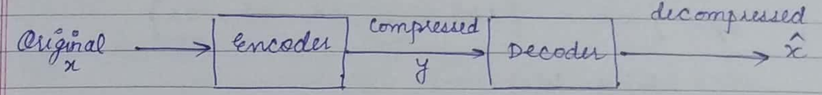
24/ Jan /19

Data Compression

Compression



Compression classification

① Lossless compression ② Lossy compression.

① Lossless – decoded data = original data

- lossless compression ratio < lossy compression.
- eg: LZ1, JBIG (used for digital data)

② Lossy – decoded data ~ original data

- lossy compression ratio > lossless compression
- used where small errors are allowed.
- eg: JPEG, MPEG

Entropy

Entropy is a measurement of information content for a set of message 'S' with probability $p(s)$, with self information -

$$i(s) = \log \frac{1}{p(s)}$$

$$H(s) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$ Conditional entropy.

eg: $p(s) = \{.25, .25, .25, .125, .125\}$

Find $H(s)$

$$H(s) = .25 \log \frac{1}{.25} + .25 \log \frac{1}{.25} + .25 \log \frac{1}{.25} + .125 \log \frac{1}{.125} + .125 \log \frac{1}{.125}$$

$$= 0.5 + 0.5 + 0.5 + 0.375 + 0.375$$
$$= 2.25$$

28/Jan/19

Measures of Performance

* Compression algorithm can be evaluated in a no of different ways.
* We can measure the relative complexity of the algorithm in terms of compression ratio.
* Compression ratio = $\frac{\text{Bits required to represent the data before compression}}{\text{Bits required to represent data after compression}}$.

eg: Suppose storing an image of size (256 x 256) requires 65,536 bytes. After compression it requires 16,384 bytes then the compression ratio is 4:1.

Shanon Fano Coding

* This is a source coding method to construct binary codes.
* Codes should satisfy following conditions-
  1. No sequence of code can be obtained from each other by adding more binary digits to shorter codes. (Prefix property)
  2. T_x of encoded message is efficient. 1 & 0 appear independently.

Procedure -

1) Messages are written in decreasing order of their probabilities.
2) The set of messages is divided into 2 subsets of equal or nearly equal probabilities.
3) Code 0 is assigned to each message in one subset above the partition line.
4) Code 1 is assigned to each message in one subset below the partition line.
5) This procedure continues until each subset contains only one message.

Page :
Date: / /

Example

Apply shanon fano coding procedure for following message -

$[x] = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$

$[P] = \left[\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{4}, \frac{1}{16}, \frac{1}{8}\right]$

take $M = 2$.

Step-1 $[P] = [0.25, 0.125, 0.0625, 0.0625, 0.0625, 0.25, 0.0625, 0.125]$

Message	Probability	Code
X_1	0.25	0
X_6	0.25	0
X_2	0.125	1
X_8	0.125	1
X_3	0.0625	1
X_4	0.0625	1
X_5	0.0625	1
X_7	0.0625	1

divided table in 2 half.

Fig. S1

(Taking upper boundary = 0
Taking below boundary = 1)

{Division of table is done in passes}

Page :
Date: / /

X_1	0.25	0	0			Pass 2
X_6	0.25	0	1			Pass 1
X_2	0.125	1	0	0		Pass 4
X_8	0.125	1	0	1		Pass 3
X_3	0.0625	1	1	0	0	Pass 6
X_4	0.0625	1	1	0	1	Pass 5
X_5	0.0625	1	1	1	0	Pass 7
X_7	0.0625	1	1	1	1	

☆ Efficiency $= \eta = \dfrac{H(X) \quad \{\text{entropy}\}}{\bar{L} \cdot \log_2 M}$

$$\eta = \frac{H(X)}{\bar{L} \log_2 2} = \frac{H(X)}{\bar{L}}$$

$H(X)$ $\eta = 0.5 + 0.375 + 0.5 + 0.375 + 0.25 + 0.25 + 0.25 + 0.25$

$= 2 + 0.750 = 2.750$

$$\bar{L} = \sum_{1}^{n} p_K n_K$$

n_K = code length of message

$= 0.25 \times 2 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 3 + 0.0625 \times 4 \times 4$

$= 2.750$

$$\eta = \frac{0.25 \; 2.750}{2.750}$$

$\eta = 1$

$\eta = 100\%$

29/Jan/19

Page :
Date: / /

Modeling & Coding

* In data compression, when we reconstruct a message the decision whether a compression scheme is to be lossy or lossless.

* The exact compression scheme we use will depend on a no of different factors.

* Some of the most important factor are the characteristics of data that needs to be compressed.

Data Compression algorithm

Modeling	Coding
↓	↓
represent exact information about any redundancy	no of bits required to compress the data

Models

1) Physical model : If we know something about the physics of data generation, we can use that information to construct a model

eg: Speech related application.

Knowledge about the physics of production can be used to construct a physical model for mathematical model for sampled speech.

Page :
Date: / /

2) Probability Model : The simplest statical model for the source is to assume that each letter that is generated by the source is independent of every other letter & each occurance of letter have the same probability. This kind of model is known as probability model.

In this model, we consider only source.

Imp 3) Markov Model : One of the most popular model, represents dependancy in data.

- The model is given by the famous mathematician Markov.
- It is used in lossless compression.
- In this model, we use discrete time frame $\{t_x\}$ and observation frame $\{x_n\}$.
- The probability of observation is given by -

$$P(X_n \mid X_{n-1} \text{ ------ } \mid X_{n-k}) = P(t_{x_n} \mid t_{x_{n-1}} \text{ ------ } \mid t_{x_{n-k}})$$

- In other words, the knowledge of past 'k' symbols is equivalent to the knowledge of entire past history of the process.
- The value taken by the set $\{X_{n-1} \text{ ---- } X_{n-k}\}$ are called the state of process.

- If the size of source alphabet is 'l' then the no of state is l^k.
- First Order of Markov Model is given by -

$$X_n = \rho X_{n-1} + \epsilon_n$$

where

ϵ_n = represents noise

eg: Consider a binary image i.e image has 2 types of pixel - black pixel & white pixel

We know that the appearance of white pixel as the next observation depends on the some extent whether the current pixel is white or black.

S_w = state of white / current pixel white
S_b = state of black / current pixel black.

* Total states = $p(w/b)$, $p(b/w)$, $p(w/w)$, $p(b/b)$
$p(S_w)$, $P(S_b)$

* Entropy $H = \sum_{i=1}^{M} p(S_i) H(S_i)$

* $H(S_w) = -p(b/w) \log[p(b/w)] - p(w/w) \log[p(w/w)]$

* $H(S_b) = -p(w/b) \log[p(w/b)] - p(b/b) \log[p(b/b)]$

3 1 / Feb / 19

Coding - It means assignment of binary sequence to element of an alphabet.

eg:

letter	codeword
a_1	0
a_2	00
a_3	1
a_4	11

Codeword: Individual no. of codes known as codeword.

Alphabet: It is a collection of symbols called letters.

Rate of code: The average no. of bits per symbol is called the rate of code

Uniquely decodable codes: The average length of the code is not only important point is designing a good code.

eg: Suppose we have 4 letters a_1, a_2, a_3, a_4 with following probabilities

letter	
a_1	$P(a_1) = 1/2$.
a_2	$P(a_2) = 1/4$
a_3	$P(a_3) = 1/8$
a_4	$P(a_4) = 1/8$

Code 1	Code 2	Code 3	Code 4	letters
0	0	0	0	a_1
0	1	10	01	a_2
1	00	110	011	a_3
10	11	111	0111	a_4

$\boxed{l = \sum_i P(a_i) n(a_i)}$ average letter

$l_1 = 0.5 \times 1 + 0.25 \times 1 + 0.125 \times 1 + 0.125 \times 2$
$= 1.125$ bits/symbol.

$l_2 = 0.5 \times 1 + 0.25 \times 1 + 0.125 \times 2 + 0.125 \times 2$
$= 1.25$ bits/symbol.

$l_3 = 1 \times 0.5 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 3$
$= 1.75$ bits/symbol

$l_4 = 1 \times 0.5 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 4$
$= 1.875$ bits/symbol.

Code 4 = uniquely decodable.

Tests for unique decodability

- Suppose we have 2 linearly codewords a and b.
- a is k-bit long and b is n bit long and $a < k < n$
- If the k bit of b are identical to those of a then a is called prefix of b.
- n-k bits of b called the dangling suffix.

a = 010
b = 01011
dangling suffix
prefix a

Steps for finding uniquely decodable code

Step-1) Construct a list of all the codewords.

Step-2) Examine all pairs of codewords to check any codeword is prefix of another codeword.

Step-3) Whenever you find such pair at the dangling suffix to the list in previous iteration.

Step-4) Repeat the procedure with current list - if we get a dangling suffix that is a codeword in the list then it is not uniquely decodable. or there is no more unique dangling suffixes - uniquely decodable

eg.1
$a_1 = 0$
$a_2 = 01$
$a_3 = 10$

{0, 01, 10} Not Uniquely decodable.
{0, 01, 10, 1} add dangling prefix to new list.

eg.2
$a_1 = 0$
$a_2 = 01$
$a_3 = 110$
$a_4 = 111$

{0, 01, 110, 111}
{0, 01, 110, 111, 1}
{0, 01, 110, 111, 1, 10}
{0, 01, 110, 111, 1, 10, 11}

{0, 10, 110, 111} → Not uniquely decodable

eg.3
$a_1 = 0$
$a_2 = 01$
$a_3 = 10$

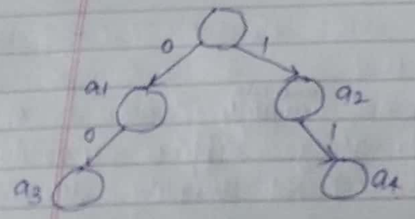
0 0 11 0.
a_1 a_1 a_3
a_1 a_2 a_3

Not uniquely decodable.

Prefix Code

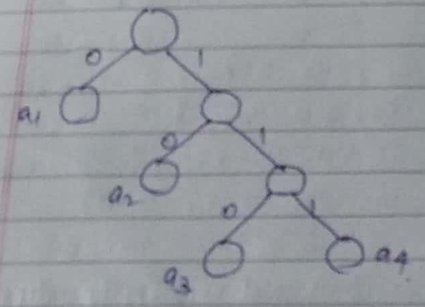
- A code in which no codeword is prefixed to another codeword is called prefix code.
- Also called instantaneous code.

letter	code 1	code 2	code 3	code 4
a_1	0	0	0	0
a_2	1	10	10	01
a_3	00	110	110	011
a_4	11	111	111	0111

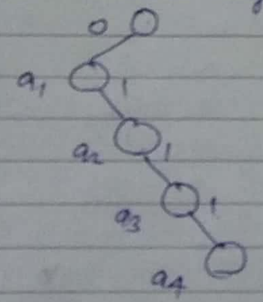


Code 1 - not uniquely decoded.

{ All prefix codes are uniquely decodable & has variable length }



Code 2 - uniquely decoded.



Code not uniquely decoded

- Node that gives rise to another node - Internal
- Node that gives does not give rise to another node - external

4 | Feb | 19 UNIT-2

Huffman Coding

This technique developed by David Huffman as a part of class assignment.

The code generated using this technique are called Huffman codes.

These codes are prefix codes & optimum for given model.

The huffman procedure is based on 2 observation.

① In optimum code symbols that occur more frequently will have shorter codes words in comparison of that symbol occur less frequently.

② In an optimum code the two symbols that occur least frequently will have the same length.

Let $A = \{a_1, a_2, a_3, a_4, a_5\}$ with

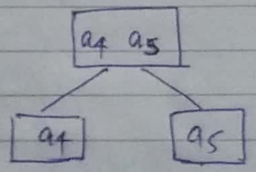
$P(a_1) = P(a_3) = 0.2$

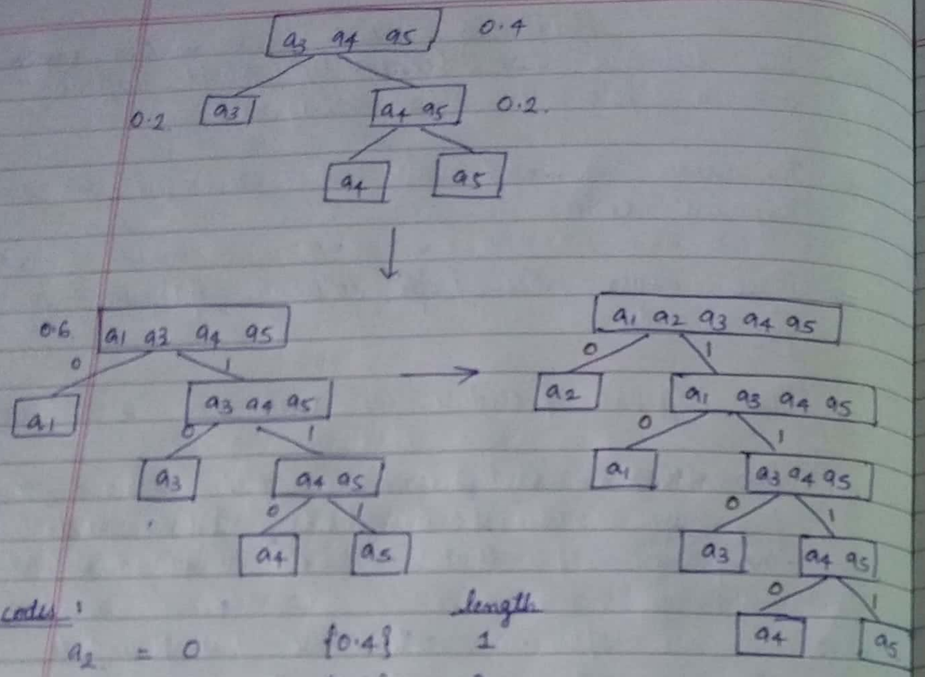
$P(a_4) = P(a_5) = 0.1$

$P(a_2) = 0.4$

Q Find huffman code and calculate entropy.

a_1	0.2	~~a_2~~	~~0.4~~
a_2	0.4	~~a_4 a_5~~	~~0.4~~
a_3	0.2	~~a_1 a_3~~	1
a_4	0.1		
a_5	0.1		





Huffman codes :

			length
a_2	= 0	{0.4}	1
a_1	= 10	{0.2}	2
a_3	= 110	{0.2}	3
a_4	= 1110	{0.1}	4
a_5	= 1111	{0.1}	4

First order Entropy :

$$= 0.2\times1 + 0.2\times2 + 0.2\times3 + 0.1\times4 + 0.4\times4$$
$$= 0.2 + 0.4 + 0.6 + 0.4 + 0.4$$
$$= 2.2$$

ANS

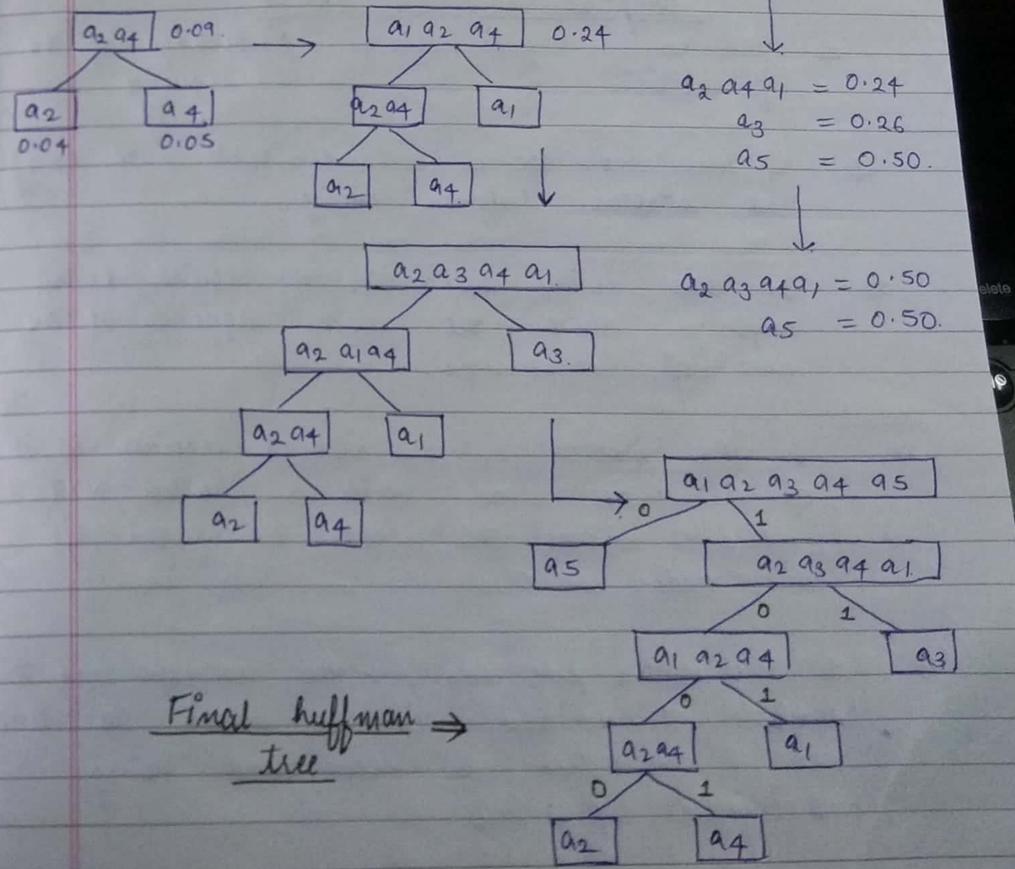
Q A source emits letters from alphabet $A = \{a_1, a_2, a_3, a_4, a_5\}$ with probability -

$P(a_1) = 0.15$ $P(a_3) = 0.26$ $P(a_5) = 0.50$

$P(a_2) = 0.04$ $P(a_4) = 0.05$

Calculate huffman code.

a_1	0.15	a_2	0.04	$a_2\ a_4$	0.09
a_2	0.04	a_4	0.05	a_1	0.15
a_3	0.26	a_1	0.15	a_3	0.26
a_4	0.05	a_3	0.26	a_5	0.50
a_5	0.50	a_5	0.50		



Final huffman tree ⇒

Huffman Code -

a_1	-	101	[3]
a_2	-	1000	[4]
a_3	-	11	[2]
a_4	-	1001	[4]
a_5	-	0	[1]

Entropy - $3 \times 0.15 + 0.04 \times 4 + 0.26 \times 2 + 4 \times 0.05 + 1 \times 0.50$

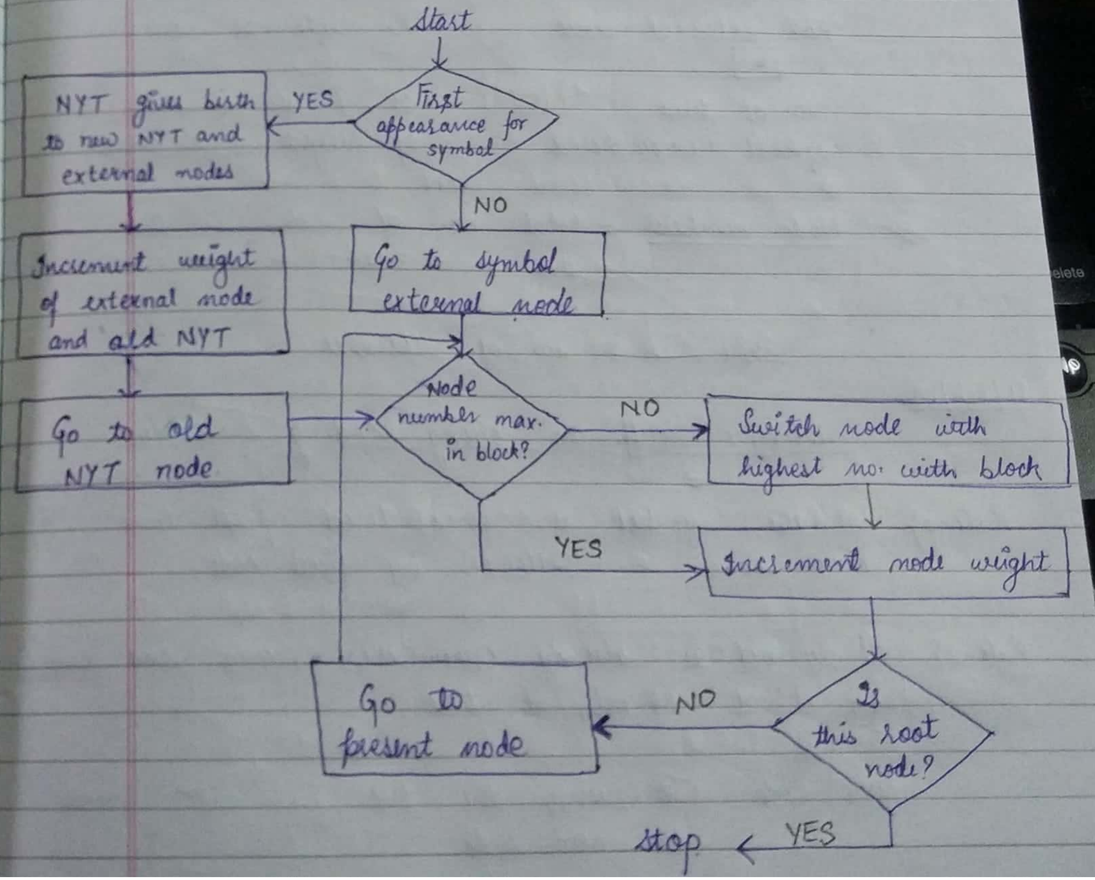
$= 1.83$

5 | Feb | 19

Adaptive huffman coding

- Huffman coding requires knowledge of the probabilities of the source sequence.
- If this knowledge is not available, we can use adaptive huffman coding.
- In this, neither transmitter or receiver knows anything about the source sequence at the start of transmission.
- In this procedure, transmitter & receiver consists of a single node which is called NYT (Not Yet Transmitted) node & weight of this node is 0.
- As transmitter progresses, nodes corresponding to symbols transmitted will be added to the tree & the tree is reconfigured using an update procedure.

- Update procedure requires that the nodes be in fixed order.
- This ordering is preserved by numbering the nodes.
- Largest code given to root node in the tree & smallest code is assigned to NYT node.
- Number from NYT node to the root are assigned in an increasing order from left to right and from lower to upper layer.



Page :
Date: / /

* Tree update (For encoding / For decoding)

* Two parameters
  1. Weight of each leaf
  2. Node number
     (every node have node number)

Height of each leaf

For external node
- No. of time symbol encountered

For internal node
- It consists of sum of weight of its child

* Node number = total no. of nodes in tree
  $= 2m - 1 = 2 \times 26 - 1 = 52 - 1 = 51$

  where m = no. of alphabet.

11/Feb/19

Adaptive huffman coding (Coding Procedure)

Rule-1) If the symbol is encountered first time, then NYT code is followed by fixed code.

Rule-2) If symbol is already encountered, only send code for that symbol, in tree.

NYT Code: Traversing the tree from root node

Page :
Date: / /

Fixed code - For fixed code, we have 2 parameters e and r.

$$m = 2^e + r$$ { Tree update }

m: No. of alphabet in dictionary.

Fixed value.

$$26 = 2^e + r$$
$$26 = 2^4 + 10$$

e = 4
r = 10

Case-1) If $1 \le k \le 2r$; here k is position of symbol in dictionary. { r = 10 }

then -
Symbol is encoded as (e+1) bit binary representation of k-1.

eg: For A = 1-1 = 0 = 00000 { Code for A }

Case-2) If $2r < k$; then e bit binary representation of k-r-1 will be used for encoding symbol.

eg: For V = 22 - 10 - 1 = 11 = 1011

Page :
Date: / /

Example Encode a a r d v a r k.

Case-1 Rule-1 ① a is encountered first time so → NYT + fixed code

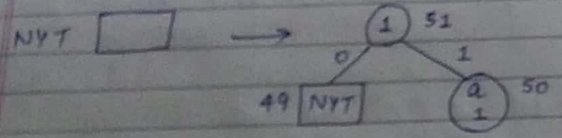
a = NYT code = ____ (nothing since no tree)

a = fixed code = 00000 (encoding)

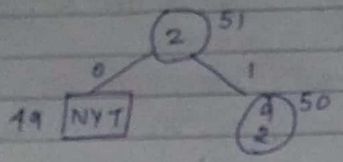
a = NYT code + fixed code (encoding)

a = 00000

Rule-2 ② Tree will be formed now-



② a is encountered next time → **a = 1**

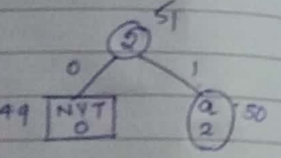


Case-1 Rule-1 ③ For r = 18

bits = e+1 = 4+1 = 5

r = 18-1 = 17

r = 10001 ⇒ NYT + fixed = **01000 1=r**



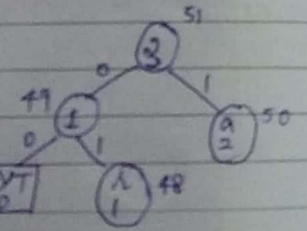
Case-1 Rule-1 ④ For d = NYT + fixed

d = 4

= 4 - 1 = 3

bits = e+1 = 4+1 = 5

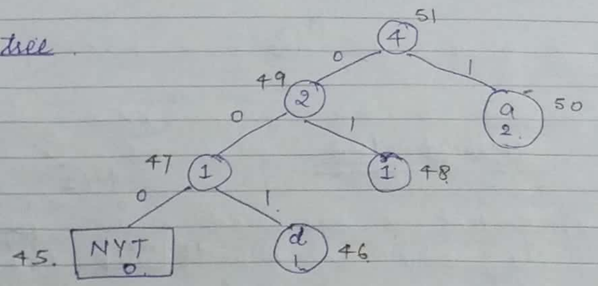
= 00011



Page :
Date: / /

d = 00 + 00011 = 00 00011

⑤ Updated tree



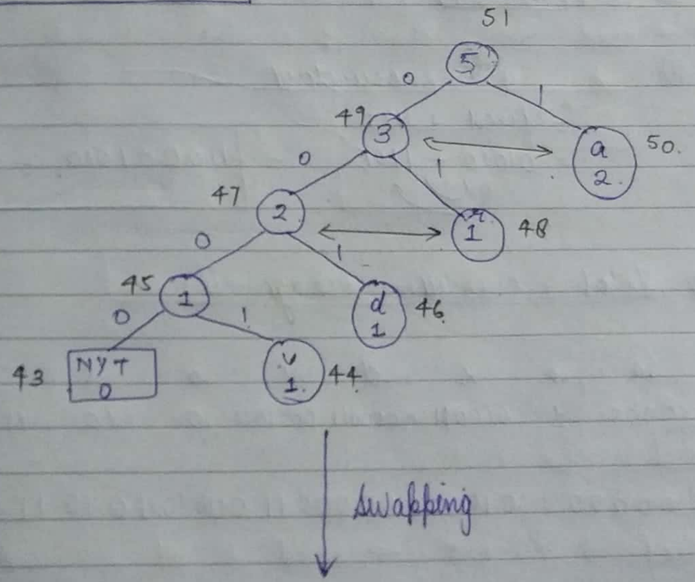
Rule-1 Case-2 v is encountered. - NYT + fixed = 000 + fixed

Fixed : v = 22-10-1 = 11

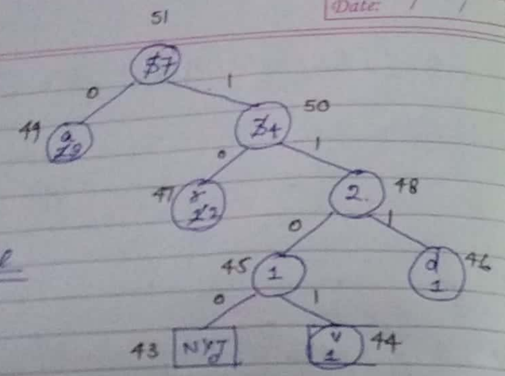
= 1011 [4 bits]

v = 000 1011

⑧



Swapping



Updated tree

Rule-2

(7) a is encountered again

$a = 0$

Rule-2

(8) s is encountered again

$s = 10$

Rule-1 Case-1

(9) k is encountered

= fixed + NYT

= 01010 + 1100 = 11000 1010 = k

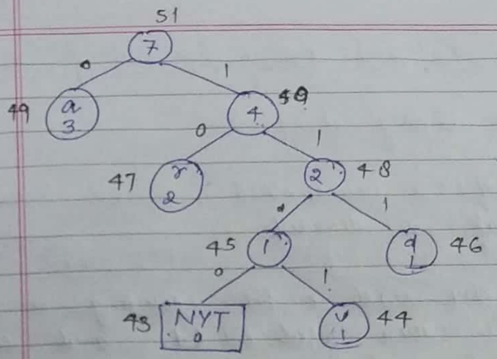
☆ Total encoded message is

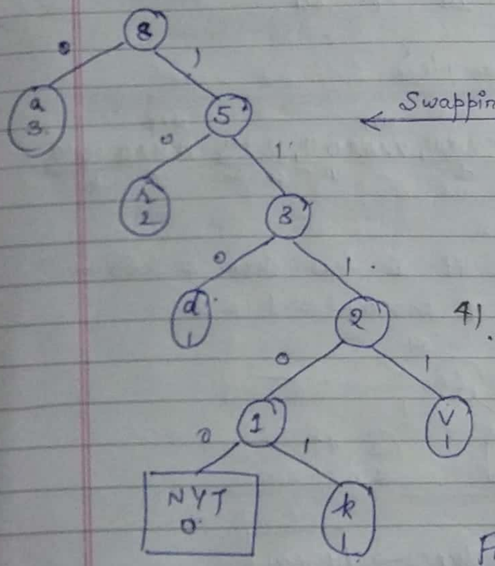
a a s d v a s k

00000 1 01000 00000 11 000 1011 0 10 11000 1010

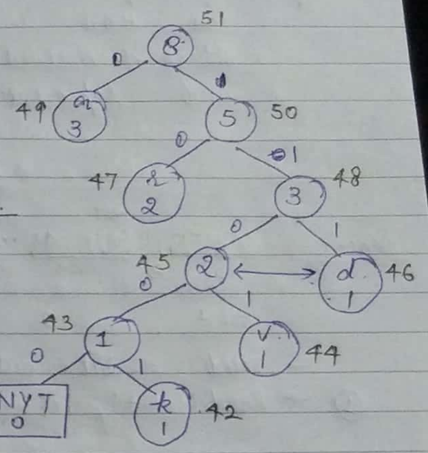
→ 00000 1 01000 00000 11 000 1011 0 10 11 000 1010

ANS





Swapping



Final tree

Adaptive huffman coding procedure for decoding

Step-1) Read the binary string

Step-2) Once a leaf is encountered, symbol corresponding to that leaf is decoded.

Step-3) If leaf is NYT - read e bit. resulting number is less than r then read e+1 bit and convert into decimal +1. if e>8 then ~~read~~ Calculate $k+r+1$ and represent in e bit.

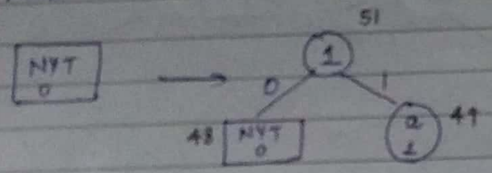
Example

Decode - 00000 | 1 | 00001 | 00011 | 010 | 11 | 0 | 10 | 00 | 1010
(NYT before 00001, 00011, 010, 11; NYT before 00 | 1010)
a q r d v a r x t

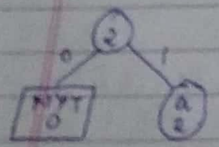
e=4 , r=10

→ Read e bit but it is less than r, So -

① → Read e+1 bit : 00000+1 = 1 = a

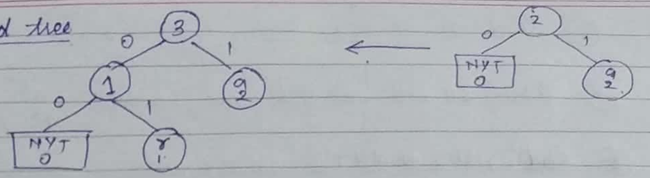


② Read 1 → from tree → 1 = a



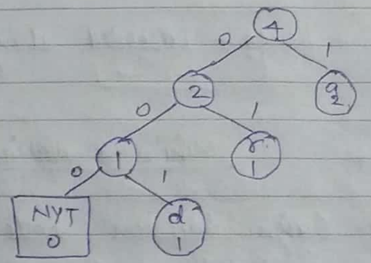
③ Read 0 → NYT → read e bit = 2 < 10
→ Read e+1 bit = r = 18

Updated tree

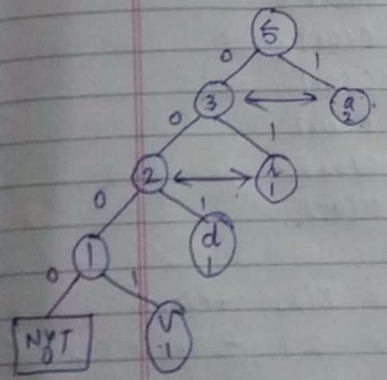


④ Read 00 → NYT → Read e bit → 0001 < 10.
Read e+1 bit → 00011+1 = 4 = d

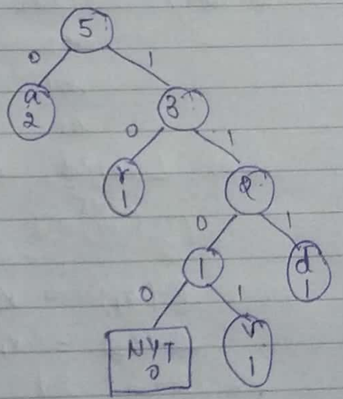
Updated tree



⑤ Read 0 → Read 0 → Read 0 → NYT
Read e bit = 1011 = 11 < 10 ×
= k+r+1 = 11+10+1 = 22 = v



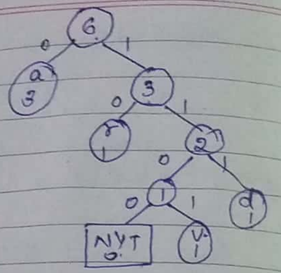
Swapping we get



⑥ Read 0 → a = 0+1

⑦ Read 10 → 1 = 10

⑧ Read 1100 → NYT
Read 0101 < r
Read 01010 + 1 = 11 = k



[Make tree for k & balance]

Hence decoded string is → a a r d v a r k

ANS

Minimum Variance huffman code

eg: Design huffman code with minimum variance.

$A = \{a_1, a_2, a_3, a_4, a_5\}$

$P(a_1) = P(a_3) = 0.2$

$P(a_4) = P(a_5) = 0.1$

$P(a_2) = 0.4$

Solution:

Letter	Probability	Codeword
a_2	0.4	$C(a_2)$
a_1	0.2	$C(a_1)$
a_3	0.2	$C(a_3)$
a_4	0.1	$C(a_4)$
a_5	0.1	$C(a_5)$

$C(a_4) = \alpha_1 * 0$
$C(a_5) = \alpha_1 * 1$

$P(a_4') = P(a_4) + P(a_5)$
$= 0.1 + 0.1 = 0.2$

{ New probability should be taken to maximum hight }

Table-2)

Letter	Probability	Codeword	
a_2	0.4	$C(a_2)$	
a_4'	0.2	$C(a_4')$	α_1
a_1	0.2	$C(a_1)$	
a_3	0.2	$C(a_3)$	

$C(a_1) = \alpha_2 * 0$
$C(a_3) = \alpha_2 * 1$

$P(a_1') = P(a_1) + P(a_2) = 0.2 + 0.2$
$= 0.4$

Table-3)

Letter	Probability	Codeword	
a_1'	0.4	$C(a_1')$	α_2
a_2	0.4	$C(a_2)$	
a_4'	0.2	$C(a_4')$	α_1

$C(a_2) = \alpha_3 * 0$
$C(a_4') = \alpha_3 * 1$
$\alpha_1 \Rightarrow \alpha_3 * 1$

$P(a_2') = 0.4 + 0.2 = 0.6$

Table-4)

Letter	Probability	Codeword		
a_2'	0.6	$C(a_2')$	α_3	0
a_1'	0.4	$C(a_1')$	α_2	1

$C(a_1) = \alpha_2 * 0$
$C(a_1) = 10$

$C(a_3) = \alpha_2 * 1$
$C(a_3) = 11$

$C(a_2) = \alpha_3 * 0$
$C(a_2) = 00$

$C(a_4) = \alpha_3 * 0$
$C(a_4) = 0.1 * 0$
$C(a_4) = 010$

$C(a_5) = \alpha_1 * 1$
$C(a_5) = 011$

14/Feb/19

Page :
Date: / /

GOLOMB CODES

It is used to design encoding of integer values.

coding scheme = Unary code + different code
(q → quotient) (r → remainder)

Both $n > 0$ & $m > 0$.

$q = \lfloor \frac{n}{m} \rfloor$ $\quad$ $r = n - qm$

Unary code

- no. of 1's followed by 0
- no of 0's followed by 1.

eg: Unary code of 5 ⇄ 11110
00001
can be

Different Code of r

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

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

Page :
Date: / /

Example

- Design golomb codes for $m = 5$, $n = 0, 1, \ldots, 15$.

ANS) $\lceil \log_2 m \rceil = \lceil \log_2 5 \rceil = 3$

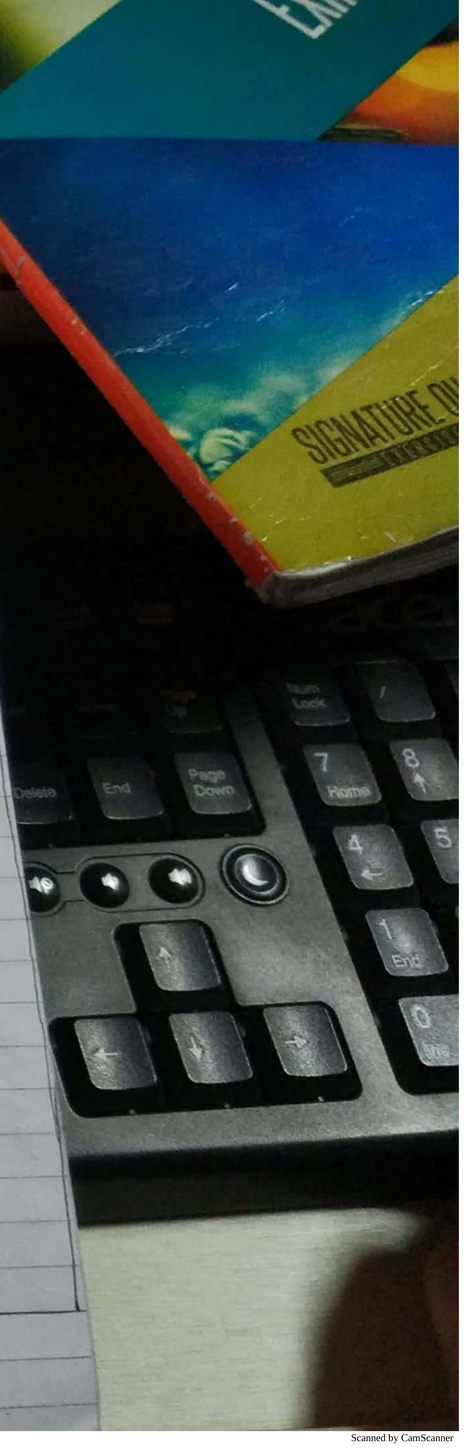
$\lfloor \log_2 m \rfloor = \lfloor \log_2 5 \rfloor = 2$

For different code of r

$2^{\lceil \log_2 m \rceil} - m = 2^3 - 5 = 8 - 5 = 3$ {2 bits} (First (0,1,2))

$r + 2^{\lceil \log_2 m \rceil} - m = r + 3$ {3 bits}

n	q	r	Codeword
0	0	0	0 00
1	0	1	0 01
2	0	2	0 10
3	0	3	0 110
4	0	4	0 111
5	1	0	1000
6	1	1	1001
7	1	2	10 10
8	1	3	10 110
9	1	4	10 111
10	2	0	11000
11	2	1	11001
12	2	2	110 10
13	2	3	110 110
14	2	4	110 111
15	3	0	1110 00



Rice Code

Rice code can be viewed as adaptive golomb code.

Sequence of non-negative integers and is divided in different blocks which is known as "j integer piece".

Standard of this code is given by CCSDS which is a space data standerd.

Compresses data from space.

By default the size is 16 bit.

Rice code can be divided into 2 parts -

- Preprocessor (modeling)
- Binary coder (coding)

(1) Preprocessor

- It removes a correlation from input and generate a sequence of non-negative integers

Step-1) In given sequence $\{y_i\}$ generate a predicted value $\{\hat{y}_i\}$

Step-2) Calculate $\boxed{d_i = y_i - \hat{y}_i}$ modeling.

Step-3) Let y_{max} and y_{min} be the largest and smallest values of sequence.

Step-4) Calculate $\boxed{T_i = \min\{y_{max} - \hat{y}_i, \hat{y} - y_{min}\}}$ $y_{min} = 0$ always

Step-5) Sequence d_i can be converted into sequence of non-negative integer

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

(2) Binary Coder

In binary coder, coded block is transmitted by identifier that indicates the particular option for coding.

(1) Fundamental sequence (Unary code)

(2) Split sample option

⇓

If the number is k bit long, then we apply m^{th} split option, we take it as a m least significant bit of k, followed by unary code representing (k-m) and use as MSB.

eg: $n = 23$ $k = 8$ bit $m = 3$ split sample.

$= 00010111$

(k-m) MSB → 2 → Unary code of 2 = 110

LSB

⇒ LSB + Unary code

⇒ 111 110

(3) Second extension option

This is applicable for low entropy and many values of x_i are zero.

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

encoded by unary code.

(4) Zero block option

When multiple blocks of x_i are zero, then apply ROS code.

This code is used when last five or more blocks in a segment are zero.

21/Feb/19

Example

Q) Encode by rice code, sequence has 16 values using J=8 and 1 split sample option.

32, 33, 35, 39, 37, 38, 39, 40, 40, 40, 40, 40, 39, 40, 40, 41, 40

{By default J=16 if not given} (m=1)

Ans)

Table $\hat{y}_i = y_{i-1}$ (Preprocessing)

y_i	32	33	35	39	37	38	39	40	40	40	40	39	40	40	41	40
$\hat{y}_i$	0 (By default)	32	33	35	39	37	38	39	40	40	40	40	39	40	40	41
d_i $y_i - \hat{y}_i$	32	1	2	4	-2	1	1	1	0	0	0	-1	1	0	1	-1
T_i	0	9	8	6	2	4	3	2	1	1	1	1	2	1	1	0
x_i	32	2	4	8	3	2	2	2	0	0	0	1	5	0	2	1

★ $T_i = \min\{y_{max} - \hat{y}_i \to \hat{y}_i - y_{min}\}$ $\left\{\begin{matrix} y_{max} = 41 \\ y_{min} = 0 \end{matrix}\right\}$ Taken from $\hat{y}_i$ entry

★ Assume prediction of 0 for the 1st element of the sequence.

★ $$x_i = \begin{cases} 2d_i & 0 \le d_i \le T_i \\ 2|d_i| - 1 & -0T_i \le d_i < 0 \quad \text{(negative values)} \\ T_i + |d_i| & \text{Otherwise.} \end{cases}$$

Binary Coder [k=6 ; Based on largest no]

	Numbers	Binary representation	code (LSB + Unary code)	
Block 1.	32.	100000	0111111111111111110	
	2.	000010	010	
	4	000100	0110	
	8	001000	011110	because m=1, 1 bit is considered for LSB
	3	000011	110	
	2.	000010	010	
	2	000010	010	
	2	000010	010	
Block 2.	0	000000	00	
	0	000000	00	
	0	000000	00	
	1	000001	10	
	2	000010	010	
	0	000000	00	
	2	000010	010	
	1	000001	10	

ANS

18 / Feb / 19

Page :
Date: / /

Tunstall Code

Step-1) Used to design that application where code should be of equal length.

Step-2) n bit tunstall code have total no. of max 2^n codes.

eg: Design 3 bit tunstall code :

$A = \{A, B, C\}$

$P(A) = 0.6$ $P(B) = 0.3$ $P(C) = 0.1$

Max. codes $= 2^n = 2^3 = 8$

Highest probability letter should be at the top.

Letter	Probability
A	0.6
B	0.3
C	0.1

} Code book

Discard higher probability & concatenate with others (himself) in main table.

Letter	Probability
B	0.3
C	0.1
AA	$0.6 \times 0.6 = 0.36$
AB	0.18
AC	0.06

} Discarding AA

Page :
Date: / /

#	Letter	Probability		
	B	0.3	0	000
	C	0.1	1	001
	AB	0.18	2	010
	AC	0.06	3	011
	AAA	0.216	4	100
	AAB	0.108	5	101
	AAC	0.036	6	110

- Each codeword represent different no. of letters
- Advantage - ① Error in the codeword donot propagate

UNIT-3 26/Feb/19

Page :
Date: / /

Image Compression

Arithmetic coding

* In arithmetic coding, a unique identifier called tag is generated for the sequence to be encoded.
* This tag corresponds to binary fraction which becomes the binary code for the sequence.
* Arithmetic coding approach is divided into 2 phase.
  1) Unique identifier or tag
  2) Unique binary code.
* The tag belongs to interval [0, 1).
* For generating tag, we will use $X(a_i^0) = i^0$, $a_i \in A$ where
  $A = \{a_1, a_2 \text{ ---- } a_n\}$
* A is alphabet for discrete source and X is the random variable

$$P(X = i^0) = P(a_i^0)$$
$$F_X(i^0) = \sum_{K=1}^{P} P(X = k)$$

* tag values [0, 1]
  - lower limit l_0
  - upper limit u_0

Page :
Date: / /

$X = \{x_1, x_2 \text{ ---- } x_n\}$.

Lower limit :
$$l^n = l^{n-1} + (u^{n-1} - l^{n-1}) F_X(x_n - 1)$$

Upper limit :
$$u^n = l^{n-1} + (u^{n-1} - l^{n-1}) F_X(x_n)$$

where n is length of sequence.

eg: encode "456"

$n = 3$ → $n=1$, $n=2$, $n=3$.

Final Tag Formula

$$T_X(x) = \frac{u^n + l^n}{2}$$

Example ($a_1 a_2 a_3 a_4$)

(1) Encode sequence "1321" for $F_X(K) = 0 \quad K \le 0$
$F_X(1) = 0.8$, $F_X(2) = 0.82$, $F_X(3) = 1$
$F_X(K) = 1, \quad K > 3$.

$l^0 = 0$
$u^0 = 1$
$n = 4$

Solution)
$l^0 = u^0 * (u^0 - l^{-1}) F_X(x_0 - 1)$
$x_1 = 1$, $x_2 = 3$, $x_3 = 2$, $x_4 = 1$

* $l^1 = l^0 + (u^0 - l^0) F_X(x_1 - 1)$
  $l^1 = 0 + (1 - 0) F_X(1 - 1)$
  $= F_X(0)$
  $l^1 = 0$

New tag [0, 0.8]
$l^1 = 0$
$u^1 = 0.8$

* $u^1 = l^0 + (u^0 - l^0) F_X(x_1)$
  $= 0 + (1 - 0) F_X(1)$
  $u^1 = 0.8$

* $l^2 = l^1 + (u^1 - l^1) F_X(x_2 - 1)$
  $= 0 + (0.8 - 0) F_X(2)$
  $= 0.8 \times 0.82$
  $l^2 = 0.656$

$u^2 = 0 + (0.8-0)\times 1$

$\boxed{u^2 = 0.8}$

$l^3 = l^2 + (u^2 - l^2)\, F_X(X_3 - 1)$
$= 0.656 + (0.8 - 0.656)\, F_X(2-1)$
$= 0.656 + (0.8 - 0.656)\, 0.8$

$\boxed{l^3 = 0.7712}$

$u^3 = 0.656 + (0.8 - 0.656)\, 0.82$

$\boxed{u^3 = 0.77408}$

$l^4 = 0.7712 + (0.77408 - 0.7712)\, F_X(X_4 - 1)$

$\boxed{l^4 = 0.7712}$

$u^4 = 0.7712 + (0.77408 - 0.7712)\, 0.8$

$\boxed{u^4 = 0.773504}$

Final tag value

$$T_X(x) = \frac{0.7712 + 0.773504}{2} = \frac{1.544704}{2}$$

$\boxed{T_X(x) = 0.772352}$

(2) Decode the tag 0.772352 upto 4 values. Values of F(x) is in previous question.

Solution) $l^0 = 0 \quad u^0 = 1$

- $l^n = l^{n-1} + (u^{n-1} + l^{n-1})\, F_X(X_n - 1)$
- $u^n = l^{n-1} + (u^{n-1} + l^{n-1})\, F_X(X_n)$

\# $l^1 = l^0 + (u^0 - l^0)\, F_X(X_1 - 1)$
$l^1 = 0 + (1-0)\, F_X(X_1 - 1)$
$l^1 = F_X(X_1 - 1)$

\# $u^1 = F_X(X_1)$ { put values & check if range is satisfied }

If $n=1$

$u^1 = F_X(1-1) = F_X(0)$

$\boxed{l^1 = 0}$

$u^1 = F_X(1) \Rightarrow \boxed{0.8 = u1}$

Since $[0, 0.8] \in [0,1]$; $n=1$ satisfies the equation. as $0.772352 \in [0, 0.8]$ sequence no. is taken correct.

\# $l^2 = l^1 + (u^1 - l^1)\, F_X(X_2 - 1)$
$= 0 + (0.8 - 0)\, F_X(X_2 - 1)$
$l^2 = 0.8\, F_X(X_2 - 1)$

\# $u^2 = 0 + 0.8\, F_X(X_2)$
$u^2 = 0.8\, F_X(X_2)$

If $n=1$ ✗

$u^2 = 0.8\, F_X(1-1)$
$= 0.$

$u^2 = 0.8\, F_X(0)$
$= 0.8 \times 0.8$
$= 0.64$

$0.772352 \notin [0, 0.64]$ ✗

If $n=2$ ✗

$l^2 = 0.8\, F_X(2-1)$
$= 0.8 \times 0.8$
$= 0.64$

$u^2 = 0.8 \times 0.82$
$= 0.656$

$0.772352 \notin [0.64, 0.656]$

If $n=3$ ✓

$u^2 = 0.8\, F_X(2)$
$= 0.8 \times 0.82$

$\boxed{l^2 = 0.656}$

$u^2 = 0.8 \times 1$

$\boxed{u^2 = 0.8}$

$0.772352 \in [0.656, 0.8]$

$l^3 = l^2 + (u^2 - l^2)\ Fx\ (X_3 - 1)$
$= 0.656 + (0.8 - 0.656)\ Fx\ (X_3 - 1)$

$\boxed{l3 = 0.656 + 0.144\ Fx (X_3 - 1)}$

$\boxed{u^3 = 0.656 + 0.144\ Fx\ (X_3)}$

If $n = 1$ X

$l^3 = 0.656 + 0.144 \times 0$
$= 0.656$

$u_3 = 0.656 + 0.144 \times 0.8$
$= 0.7712.$

$0.772352 \notin [0.656, 0.7712]$

If $n = 2$ ✓

$l^3 = 0.656 + 0.144 \times 0.8$
$\boxed{l^3 = 0.7712}$

$u_3 = 0.656 + 0.144 \times 0.82$
$\boxed{u_3 = 0.77408}$

$0.772352 \in [0.7712, 0.77408]$

$l^4 = 0.7712 + (0.77408 - 0.7712)\ Fx\ (X_4 - 1)$
$l^4 = 0.7712 + 0.00288\ Fx\ (X_4 - 1)$

$u^4 = 0.7712 + 0.00288\ Fx\ (X_4)$

If $n = 1$ ✓

$l^4 = 0.7712 + 0$
$\boxed{l^4 = 0.7712}$

$u^4 = 0.7712 + 0.00288 \times 0.8$
$\boxed{u^4 = 0.773504}$

Thus the sequence becomes ⇒ 1321 ANS

27/Feb/19

☆ Tag Generation with Scaling Method

- It is also called binary method
- Code sequence is in binary format.
- In scaling method, the tag is encoded in binary value.
- The tag lies in between 0 to 1.
- This process is also known as Incremental encoding
- The tag interval is divided into 2 parts.

[0, 1]

[0, 0.5] — Lower half → Send '0' bit

[0.5, 1] — Upper half → Send '1' bit

Rescaling factor: $E(x) = 2x$ $\quad E(x) = 2(x - 0.5)$

Example

Encode sequence 1321 ($a_1 a_3 a_2 a_1$) into binary value

$P(a_1) = 0.8$
$P(a_2) = 0.02$
$P(a_3) = 0.18$

[0, 1]
$l^0 = 0$
$u^0 = 1$

$F_1(x) = 0.8$
$F_2(x) = 0.8 + 0.02 = 0.82$
$F_3(x) = 0.82 + 0.18 = 1.00$

$l^1 = l^0 + (u^0 - l^0) F_x(1-1)$
$= l^0 + (u^0 - l^0) F_x(0)$
$= 0 + (1-0) \times 0$
$\boxed{l^1 = 0}$

$u^1 = l^0 + (u^0 - l^0) F_x(1)$
$= 0 + (1-0) \times 0.8$
$\boxed{u^1 = 0.8}$

tag value $[0, 0.8]$

Note: We will proceed further as tag value = [0, 0.8] do not lie in any half.

If it would have been then -
- We will send the code.
- Apply rescaling factor until tag value continue to lie in the new range.

$l^2 = l^1 + (u^1 - l^1) F_x(3-1)$
$= 0 + [0.8 - 0] 0.82$
$\boxed{l^2 = 0.656}$

$u^2 = 0 + [0.8 - 0] \times 1$
$\boxed{u^2 = 0.8}$

tag value $[0.656, 0.8]$

→ Lies in upper half ⇒ $[0.5, 1]$

→ So we will send 1 as code

Now we apply rescaling factor -

$l^2 = 2(X - 0.5)$
$= 2(0.656 - 0.5)$
$= 2[0.156)$
$= 0.312$

$u^2 = 2(X - 0.5) = 2(0.8 - 0.5) = 2 \times 0.3 = 0.6$

$[0.312, 0.6]$ → does not lie in any range.
→ we proceed further for l^3 & u^3.

$l^3 = 0.656 + (0.8 - 0.656) F(1)$
$= 0.656 + (0.8 - 0.656) - 0.8$
$= 0.312 + (0.6 - 0.312) 0.8$
$\boxed{l^3 = 0.5424}$

$u3 = 0.312 + (0.6 - 0.312) 0.82$.
$\boxed{u^3 = 0.54816}$

$[0.5424, 0.54816]$

Send 1

Lies in upper half & rescaling it we get

$l^3 = 2(0.5424 - 0.5) = 0.0848$
$u3 = 2[0.54816 - 0.5] = 0.09632$

Send 0

Lies in ~~upper~~ lower half & rescaling

$l^3 = 2 \times 0.0848 = 0.1696$
$u3 = 2 \times 0.09632 = 0.19264$

Send 0

Lies in lower half & rescaling

$l^3 = 0.3392$
$u3 = 0.38528$

Send 0

Lies in lower half & rescaling

$l^3 = 0.6784$
$u3 = 0.77056$

Send 1.

Lies in upper half & rescaling

$l3 = 0.3568$
$u3 = 0.54112$

} does not lie in any half.

$l4 = 0.3568 + (0.54112 - 0.3568)\ Fx(0)$
$= 0.3568$

$u4 = 0.3568 + (0.54112 - 0.3568)\ 0.8$
$= 0.504256$

It does not lie in any range

Sending bit = 110001

→ We have to send receiver a stopping bit.
→ So, either we select $l4$ or $u4$ and convert it into binary and after decimal append this value to get final value.

→ We select $l4$.

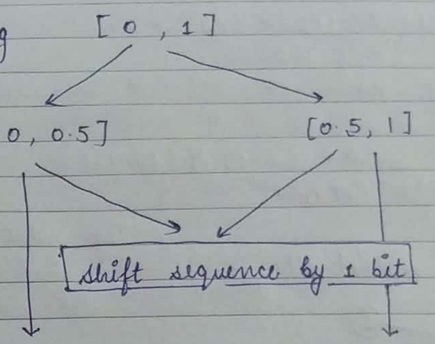
$0.3568 \times 2 = 0.7136 = 0.$
$0.7136 \times 2 = 1.4272 = 1$
$0.4272 \times 2 = 0.8544 = 0$
$0.8544 \times 2 = 1.7088 = 1$
$0.7088 \times 2 = 1.4176 = 1$
$0.4176 \times 2 = 0.8352 = 0$
$0.8352 \times 2 = 1.6704 = 1$

(0101101)

Sending bit = 110001.0101101....

28/Feb/19

Tag Degeneration with Scaling Method



Difference b/w two smallest interval will be taken

$2^{-K} <$ Difference

Rescaling factor is same as encoding

$E(x) = 2x$

$E(x) = 2(x - 0.5)$

Example

Q- 1100011000000 ---- 0

Decode with tag degeneration scaling method

$P(a_1) = 0.8$
$P(a_2) = 0.02$ (upto 4 values)
$P(a_3) = 0.18$

Solution)

$F(0) = 0$
$F(1) = 0.8$ [$P(a_1)$]
$F(2) = 0.8 + 0.02 = 0.82.$ ($P(a_1) + P(a_2)$)
$F(3) = 0.18 + 0.82 = 1.00.$ ($P(a_1) + P(a_2) + P(a_3)$)

→ $2^{-K} <$ difference. (smallest value).
$2^{-K} < 0.02$

$K = 6$

} so we will select 6 bit in the starting and move accordingly.

Selecting 110001

$110001 = 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 0\times2^{-5} + 1\times2^{-6}$
$= 0.765625$
(First tag value)

- $l^1 = 0 + (1-0)\, F_x(x_n - 1)$
  $l^1 = F_x(x_n - 1)$
- $u^1 = F_x(x_n)$

} Tag value lies in this range.

If $x_n = 1$ ← (1)

$l^1 = F_x(1-1) = F(0) = 0$
$u^1 = F_x(1) = F(1) = 0.8$

} Tag lies in this interval ✓

l_1 u_1
$[0, 0.8]$

Sequence ~~Tag~~ Value = 1

It does not lies in any half so -

- $l^2 = 0 + (0.8-0)\, F_x(x_n - 1)$
  $= 0.8\, F_x(x_n - 1)$
- $u^2 = 0.8\, F_x(x_n)$

If $x_n = 1$

$l^2 = 0.8\, F(0) = 0$
$u^2 = 0.8 \times 0.8 = 0.64$

} Tag value doesnot lie in this interval.

If $x_n = 2$

$l^2 = 0.8 \times 0.8 = 0.64$
$u^2 = 0.8 \times 0.82 = 0.656$

} Tag value doesnot lie in this interval.

If $x_n = 3$ ← (3)

$l^3 = 0.656$
$u^3 = 0.8 \times 1 = 0.8$

} Tag value lies in this range.

→ Tag value lies in upper half. $[0.656, 0.8]$

→ Shift value by 1 bit ⇒ New tag = 100011

→ Rescaling factor ~~$E(x) = 2(x - 0.5)$~~
~~$= 2(0.765625 - 0.5)$~~
~~$E(x) = 0.53125$~~

$l^3 = 2(0.656 - 0.5) = 0.312$.
$u^3 = 2(0.8 - 0.5) = 0.6$.

} Finish here.

→ Does not lie in any half.

Sequence = 13

Selecting 100011

$100011 = 0.546875$ (Second tag value)

- $l^3 = 0.312 + (0.6 - 0.312)\, F_x(x_n - 1)$
  $= 0.312 + 0.288\, F_x(x_n - 1)$
- $u^3 = 0.312 + 0.288\, F_x(x_n)$

If $x_n = 1$

$l^3 = 0.312$.
$u^3 = 0.5424$

} Tag lies in this interval. does not.

If $x_n = 2$ ← (2)

$l^3 = 0.5424$
$u^3 = 0.54816$

} Tag lies in this half. & it lies in upper half.

Page :
Date: / /

Sequence = 132

→ Shift value = 000110.

→ Rescaling factor ⇒ $l^3 = 0.0848$, $u^3 = 0.09632$ } It lies in ~~upper~~ lower half.

→ Shift again = 001100.

→ Rescaling factor ⇒ $l^3 = 0.1696$, $u^3 = 0.19264$ } Lies in lower half.

→ Shift again = 011000

→ Rescaling factor ⇒ $l^3 = 0.3392$, $u^3 = 0.38528$ } Lies in lower half.

→ Shift again = 110000.

→ Rescaling factor ⇒ $l^3 = 0.6784$, $u^3 = 0.77056$ } Lies in upper half.

→ Shift again = 100000.

→ Rescaling factor ⇒ $l^3 = 0.3568$, $u^3 = 0.54112$ } Does not lies in any half. & tag also doesnot lie.

Selecting 100000

100000 = 0.5 (Third tag value).

$l^4 = 0.3568 + 0.18432\ F_x(X_n - 1)$

$u^4 = 0.3568 + 0.18432\ F_x(X_n)$

Page :
Date: / /

(4)

If $X_n = 1$

$l^4 = 0.3568$, $u^4 = 0.504736$ } Tag lies in this range

Sequence = 1321

It does not lies in any half. } Stop here }

$l^5 = 0.3568 + 0.147936\ F_x(X_n - 1)$

$u^5 = 0.3568 + 0.147936\ F_x(X_n)$

If $X_n = 1$

$l^5 = 0.3568$, $u^5 = 0.4751488$ } Tag doesnot lies in this interval.

If $X_n = 2$

$l^5 = 0.4751488$, $u^5 = 0.47810752$ } Tag does not lies in this interval.

If $X_n = 3$

$l^5 = 0.47810752$, $u^5 = 0.504736$ } Tag Lies in this half.

4/March/19

Page :
Date: / /

JBIG and JBIG-2

- It is an image compression technique.

- Dictionary Technique
- Context-Based Algorithm
- Burrows-Wheeler Transform (BWT)
- Move to Front encoding

(A) Dictionary Technique

It is lossless compression technique.

It is used for long phrases & long sentences

It is of two type - Static dictionary
- Dynamic dictionary. (adaptive)

i. Static dictionary - no change in knowledge.

- Technique : Diagram coder

ii. Dynamic dictionary - knowledge keeps on changing.

- Technique : LZ-77 / LZ1 { encoding & decoding }

LZ-78 / LZ2.

LZ-W.

Page :
Date: / /

Example
Diagram Coder

Q- A = {a, b, c, d, r}

Based on the knowledge about the source we build dictionary.

Code	Entry
000	ra
001	b
010	c
011	d
100	r
101	ab
110	ac
111	ad

Encode the sequence.

abracadabra

Solution: ① Initially we will have to select 2 symbols together.

ab = 101
ra = No code → split
r = 100
ac = 110
ad = 111
ab = 101
ra = No code → split
r = 100
a = 000.

abracadabra ⇒ 101 100 110 111 101 100 000

ANS

Dynamic Dictionary

① LZ-77 (LZ1 / Sliding Window Technique)

Window size — Search Buffer | Look ahead buffer

Encoded pattern is kept here

Symbols to be encoded

We will use a triplet -

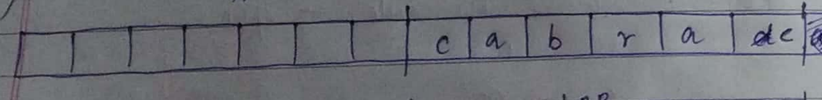
<o , l , c>

- o → Offset: It is a distance between pointer from search buffer to look ahead buffer
- l → length of match: No. of alphabet letters matched.
- c → codeword: The one whose code is not matched.

Example

Q- c a b r a c a d a b r a r r a r r a d. . Use LZ-77 to encode, given the window size = 13 and look ahead buffer = 6.

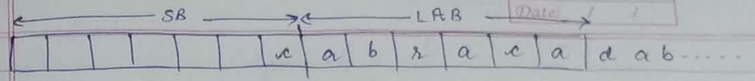
Solution) Search buffer = 13 - 6 = 7



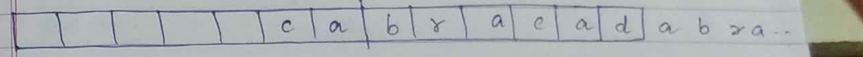
① Offset is 0 initially

$c = \langle 0, 0, c(c) \rangle$ { Codeword of c triplet }

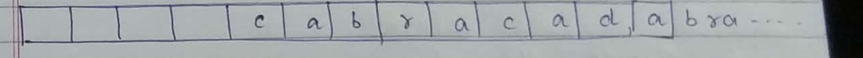
5 / March / 19



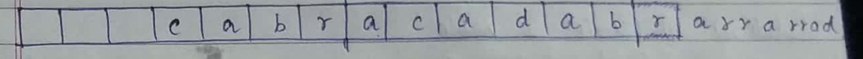
② $a = \langle 0, 0, c(a) \rangle$



③ $b = \langle 0, 0, c(b) \rangle$

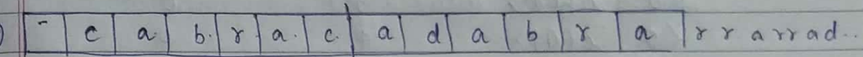


④ $r = \langle 0, 0, c(r) \rangle$

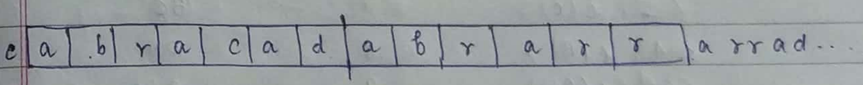


⑤ $a = \langle 3, 1, c(c) \rangle$

Now both a and c will shift to search buffer.

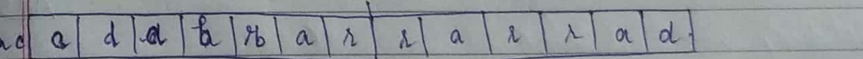


⑥ $a = \langle 2, 1, c(d) \rangle$



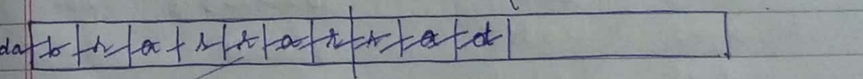
⑦ $a = \langle 7, 4, c(r) \rangle$

⑧



~~$r = \langle 3, 2, c(d) \rangle$~~ { r a r → r a & r a → r a } ⑤ matches

⑨



$r = \langle 3, 5, c(d) \rangle$

5/March/19

① LZ-78 (LZ2) Encoding

It uses two tuples $\langle i, c \rangle$ → codeword
(i: index value)

Note:- For newest entry - index value is '0'.

Example

Q: Encode the string by LZ-78.
A B C D A B C A B C D A A B C A B C E

Solution) ① Create a table

Encoded output	Index	Entry
<0, c(A)>	1	A
<0, c(B)>	2	B
<0, c(C)>	3	C
<0, c(D)>	4	D
<1, c(B)>	5	AB
<3, c(A)>	6	CA
<2, c(C)>	7	BC
<4, c(A)>	8	DA
<5, c(C)>	9	ABC
<9, c(E)>	10	ABCE

ANS

Decoding

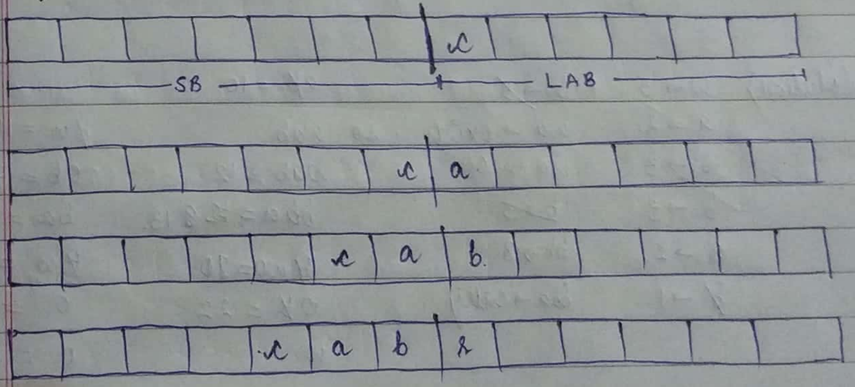
Q- Decode the following sequence by LZ-78 (LZ2).
<0, c(A)>, <0, c(B)>, <0, c(C)>, <0, c(D)>, <1, c(B)>, <3, c(A)>, <2, c(C)>, <4, c(A)>, <5, c(C)>, <9, c(E)>.

Encoded O/P	Index	Entry
<0, c(A)>	1	A
<0, c(B)>	2	B
<0, c(C)>	3	C
<0, c(D)>	4	D
<1, c(B)>	5	AB
<3, c(A)>	6	CA
<2, c(C)>	7	BC
<4, c(A)>	8	DA
<5, c(C)>	9	ABC
<9, c(E)>	10	ABCE

Sequence: A B C D AB CA BC DA ABC ABCE ANS

Q- Decode using LZ-77 / LZ-1 the following sequence.
<0, 0, c(a)>, <0, 0, c(a)>, <0, 0, c(b)>, <0, 0, c(A)>, <3, 1, c(c)>, <2, 1, c(d)>, <7, 4, c(A)>, <3, 5, c(d)>, for the window size = 13 and look ahead buffer = 6.

Solution)



			c	a	b	r	a	c					
	c	a	b	r	a	c	a	d					
c	a	b	r	a	c	a	d	a	b	r	a	r	
cabrac	a	d	a	b	r	a	r	r	a	r	r	a	d

Final Sequence: cabracadabrarrarrad

ANS

7/March/19

③ LZW encoding: It is just index based.

eg: w a b b a ♯ w a b b a ♯ w a b b a ♯ w a bb a ♯ w o o ♯ w o o ♯ w o o

Initial LZW dictionary:

Index	Entry
1	♯
2	a
3	b
4	o
5	w

(♯ → represents blank)

Solution)

w → 5.	w → 5	a♯ = 10	
a → 2.	wa → exists, so	wab	♯w = 11
b → 3	wa → 6	wab = 12	ab = 7
b → 3	b → 3	bba = ♯♯ 13.	ba = 9
a → 2.	bb → 8	♯wo = 20	♯w = 11
♯ → 1	bba → entry	o♯ = 22	o = 4
		w	o = 4

New table

Index	Entry
1	♯
2	a
3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10.	a♯
11	♯w
12.	wab.
13.	bba
14	a♯w.
15.	wabb
16	ba♯
17	♯wa.
18	abb.
19.	ba♯w
20	wo
21	oo
22.	o♯
23.	♯wo
24	oo♯
25.	♯woo

Code: 5 2 3 3 2 1 6 8 1 0 12 13

Q- Decode the sequence.

3	1	4	6	8	4	2	1	2	5	10	6	11	13	6
r	a	t	at	ata	t	b	a	b	ra	tb	at	ba	bat	

Initial table
1 - a
2 - b
3 - r
4 - t

Solution>

Index	Entry
1	a
2	b
3	r
4	t
5	ra
6	at
7	ta
8	ata
9	atat
10	tb
11	ba
12	ab
13	abr

Sequence: r a t a t a t b t b a b r a t b at b a b r at

Read by self { File and Image compression in UNIX ; Graphical Interchange format (GIF) }

(Basic algo with <ESC> symbol)

(B) Predictive Coding / Context Based Algorithm

PPM Basic: If the distribution or transformation is based on the history of sequence and make some prediction, then we use that history to determine a sequence in predictive manner. Such scheme is known as predictive coding.

PPM (Predictive with Partial Match)

1) In this one only need to store those context that have occured in a sequence which is being encoded.

2) At the beginning of encoding, we will need to code the letters, that have not occured previously in the context.

3) Escape symbol <ESC> is used to signal that the letters to be encoded has not been in the context.

Basic Algorithm:

(a) If the symbol has not occured in the context then -
<ESC> encoded.

(b) Attempt to use the next smaller context.

(c) Each time symbol is encountered, the count corresponding to that symbol is updated, in all the tables.

Page :
Date: / /

Example

Encode the following -

"this ♯ is ♯ the ♯"

Solution) ① Assume we have already encoded initial 7 symbols.

② Context upto 2nd order should be maintained.
[-1, 0, 1, 2] → Total 4 tables to be made.

Assume most highes order context is 2 i.e. from -1 to 2 (-1, 0, 1, 2)

③ Table Creation
Count vacory for (-1) Order context.

letters	count	cumulative count
t	1	1
h	1	2
i	1	3
s	1	4
♯	1	5
e	1	6

Here we consider only distinct symbols from string

Total count = 6

④ 0th Order Context table designing

Since we have assumed 7 symbols already encoded we will use "this ♯ is" string.

Page :
Date: / /

letters	count	cumulative count
t	1	1
h	1	2
i	1	4
s	2	6
♯	✗2	~~7~~ 8
<ESC>	1	~~8~~ 9

Here table is formed using string assumed as already encoded.

Total Count = 89

⑤ 1st Order Context table designing

	Previous Symbol/context	letter	count	cumulative count	
①	t	h	1	1	total count = 2
	<ESC>		1	2	
②	h	i	1	1	total count = 2
	<ESC>		1	2	
③	i	s	2	2	total count = 3
	<ESC>		1	3	
④	s	♯	✗2	✗2	total count = 23
	<ESC>		1	23	
⑤	♯	i	1	1	total count = 2
	<ESC>		1	2	

Page :
Date: / /

⑥ 2nd Order Context table designing

	context	letter	count	cumulative count	
①	th	i	1	1	total count = 2
	<ESC>		1	2	
②	hi	s	1	1	total count = 2
	<ESC>		1	2	
③	is	␣	~~1~~ 2	~~1~~ 2	total count = ~~2~~ 3
	<ESC>		1	~~2~~ 3	
④	s␣	i	1	1	total count = 2
	<ESC>		1	2	
⑤	␣i	s	1	1	total count = 2
	<ESC>		1	2	

FORMULA: $l^n = l^{n-1} + \left[\frac{(u^{n-1} - l^{n-1} + 1) \times \text{Cumulative Count}(x_{n-1})}{\text{total\_count}} \right]$

$$u^n = l^{n-1} + \left[\frac{(u^{n-1} - l^{n-1} + 1) \times \text{Cumu\_Count}(x_n)}{\text{total\_count}} \right] - 1$$

"this␣is␣is␣this␣test␣"

11/March/19

Page :
Date: / /

Assume the word length for arithmetic coding is 6 bit.

$[0, 1) \rightarrow [l^0, u^0)$

Initially; $l^0 = 000000 = 0$

$u^0 = 111111 = 63$

For blank '␣' we check 2nd Order Context

Context of ␣ = "i␣"

Cumu_Count = 1

Total_count = 2.

- $l' = l^0 + \frac{(u^0 - l^0 + 1) \text{ Cum\_Count}(x_{n-1})}{\text{total\_count}}$

$= 0 + \frac{(63 - 0 + 1) \times \text{Cum\_Count}(1-1)}{2}$

$l' = 0$ {000000}

- $u' = l^0 + \left[\frac{(u^0 - l^0 + 1) \text{ Cum\_count}(x_n)}{\text{total\_count}} \right] - 1$

$= 0 + \left[\frac{(63 - 0 + 1) \times 1}{2} \right] - 1$

$u' = 32 - 1$

$= 31$

$= 011111$

Compare l^0 & u^0 and check the number of matching bits and send it.

$l^1 = 000000$
$u^1 = 011111$ } → Send = 0

$l^1 = 000000$
$u^1 = 111111$

After discarding add the same MSB to LSB.

New $\begin{bmatrix} l^1 = 000000 \\ u^1 = 111111 \end{bmatrix} \begin{matrix} = 0 \\ = 63 \end{matrix}$ { Update the value of $ in cum_count = 2 in 2nd order context table }

Now checking 1st order table for $. Since context 's' was already in 2nd order table for $, we will not calculate l & u.

Simply update 1st order table.

Now checking 0th order table for $, and updating $.

Next symbol is 't'
Context of 't' = s$

For s$, no 't' is present in 2nd order table, so we will encode <ESC> symbol.

Cum_count = 2
Total_count = 2

$l^2 = 0 + \frac{(63-0+1) \times Cum\_count(2-1)}{2} = 32.$

$\boxed{l^2 = 32} = 100000$

$u^2 = 0 + \left[\frac{(63-0+1) \times Cum\_count(2)}{2}\right] - 1$

$= \frac{64 \times 2}{2} - 1.$

$\boxed{u^2 = 63} = 111111$

⇒ $l^2 = 100000$
$u^2 = 111111$ } Send = 1

$\begin{bmatrix} l^2 = 000000 = 0 \\ u^2 = 111111 = 63 \end{bmatrix}$

→ Update 2nd order table for <ESC>.

Add row →

Updated 2nd order Table

④	s$	i	1	1.
		t	1	2.
		<ESC>	1	3.

total count = 3

→ Update 1st order table

But again entry of t is not there.
We will calculate l and u and encode <ESC> symbol.
{ Row ④ is considered } { where 't' is present as a context }.

Cum_count = 2.
Total_count = 2.

→ $l^2 = 32 = 100000.$
$u^2 = 63 = 111111$ } $\begin{bmatrix} l^2 = 000000 = 0 \\ u^2 = 111111 = 63. \end{bmatrix}$ Send = 1

12 / March / 19

Page :
Date: / /

Burrows - Wheeler Transform (BWT) encoding

BWT encoding is a lossless compression technique

It is log sorting algorithm.

It requires entire sequence to be encoded.

The available code present before complete sequence

There are following steps-

① We have given sequence of length N.

② Create (N-1) other sequences where cyclic shift of original sequence takes places.

③ N sequence arranged in lexicographic order

④ Encoder transmit sequence of length N by only taking place the last letter of each sorted cyclic shift.

There are 2 requirements:

* L - ~~length of sequence~~ encoded sequence
* Index position - position of original sequence.

Page :
Date: / /

Example "t h i s ␣ i s ␣ t h e"

N = 11

Permutation of original sequence

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

Total unique symbols - ␣ , t , h , i , s , e

Arranging lexiographically - { ␣ , e , h , i , s , t }

Lexicographic table

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

L: s s h t t h ƒ i i ƒ e.

index position = 10 (where original message is present)

ANS

BWT Decoding

By using the sequence L & taking the index value of original situation, we have 2 values -

- F → this is the first decoding symbol
- L → it is the last letter.

Example: Decode the following sequence by BWT.

L: s s h t t h ƒ i i ƒ e

index: 10

Solution:

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

F is a permutation of sequence L in lexiographic order.

# index = 10.
F[10] = t

# index = 4. {value of t in L?
F[4] = h.

# index = 5 {value of h in L?
F[5] = i.

# index = 7 {value of i in L?
F[7] = s

# index = 0
F[0] = ƒ

# index = 9
F[9] = t

# index = 6
F[6] = ƒ i

# index = 3
F[3] = h.

# index = 8.
F[8] = ƒ s.

# index = 2
F[2] = e.

# index = 1.
F[1] = ƒ.

index =.

Decoded sequence

" t h i s ƒ i s ƒ t h e "

ANS