CS3302 Practical 1 160001362

1 a) Given an information source with alphabet A, and a coding K of A. Let $|K(a_i)| = d_i$, then the average length of code words is:

$$L(K) = \sum_{i=1}^{n} d_i * P(a_i)$$

The entropy of an information source can be defined as:

$$H(S) = H(P_1, \ldots, P_n) = -\sum_{i} P_i \log_2 P_i$$

Assumption:

Every binary instantaneous coding K of a source S satisfies

$$L(K) \geq H(S)$$

i.e., no coding can have an average length shorter than the entropy.

Proof:

let $d_i = |K(a_i)|$. By definition using $\log_b b^x = x$:

$$L(K) = \sum_{i=1}^{n} P_i d_i = \sum_{i=1}^{n} P_i \log_2 2^{d_i}$$

$$H(S) - L(K) = \sum_{i=1}^{n} P_i \left(\log_2 \frac{1}{P_i} - \log_2 2^{d_i} \right) \quad \text{(by definition)}$$

$$= \sum_{i=1}^{n} P_i \left(\log_2 \frac{1}{P_i 2^{d_i}} \right) \quad \left(\text{by } \log \frac{x}{y} = \log x - \log y\right)$$

$$= \frac{1}{\log_e 2} \sum_{i=1}^{n} P_i \log_e \frac{1}{P_i 2^{d_i}} \quad \text{(change of base)}$$

$$\log_e x \leq x - 1$$

$$\therefore \quad H(S) - L(K) = \frac{1}{\log_e 2} \sum_{i=1}^{n} p_i \log_e \frac{1}{p_i 2^{d_i}}$$

$$\leq \frac{1}{\log_e 2} \sum_{i=1}^{n} p_i \left(\frac{1}{p_i 2^{d_i}} - 1 \right)$$

$$= \frac{1}{\log_e 2} \left(\sum_{i=1}^{n} \frac{1}{2^{d_i}} - \sum_{i=1}^{n} p_i \right)$$

$$= \frac{1}{\log_e 2} \left(\sum_{i=1}^{n} \frac{1}{2^{d_i}} - 1 \right) \quad \left(\sum_{i=1}^{n} p_i = 1 \right)$$

$$H(S) - L(K) \leq \frac{1}{\log_e 2} \left(\sum_{i=1}^{n} \frac{1}{2^{d_i}} - 1 \right)$$

$\frac{1}{\log_e 2} \approx 1.44$ and $\sum_{i=1}^{n} \frac{1}{2^{d_i}} \leq 1$ as Kraft's inequality states that $\sum_{i=1}^{n} \frac{1}{n^{|K(a_i)|}} \leq 1$ and therefore

$$\left(\sum_{i=1}^{n} \frac{1}{2^{d_i}} - 1 \right) \leq 0$$

$\therefore \quad H(S) - L(K) \leq 0$, which proves that $L(K) \geq H(S)$

therefore the claim by the software supplier is false, as to compress a file to a smaller average length than the entropy of that file would by definition have to involve some information loss, as entropy is a measure of the information content of a source.

1 b)

Symbol	1	2	3	4	5	6
Probability	0.16	0.22	0.12	0.15	0.18	0.17

i) $H(P_1, \ldots, P_n) = -\sum_i P_i \log_2 P_i$

$H(0.16, 0.22, 0.12, 0.15, 0.18, 0.17)$

$$= -0.16 \log_2 0.16 - 0.22 \log_2 0.22 - 0.12 \log_2 0.12 - 0.15 \log_2 0.15 - 0.18 \log_2 0.18 - 0.17 \log_2 0.17$$

$\approxeq 2.56$, Entropy of source with weighted dice

Entropy of source of fair dice:

when all symbols have a probability of $\frac{1}{n}$, the entropy is calculated using

$$H(S) = \log_2 n$$

in this case all symbols (1, 2, 3, 4, 5, 6) have a probability of $\frac{1}{6}$:

$$H(1,2,3,4,5,6) = \log_2 6$$

$$\approxeq 2.58$$

$\therefore$ Entropy of fair dice is greater than entropy of weighted dice as $2.58 > 2.56$.

In fact, entropy of fair dice is also the maximum possible entropy of a source with six symbols as each symbol has a $\frac{1}{6}$ probability of occuring.

In other words, if the entropy is thought of as a numerical measure of the uncertainty of an outcome, the entropy of the fair dice is higher than the entropy of the weighted dice as it is harder to predict which symbol will occur on the fair dice than the weighted dice.

1 b) ii) $S = \{1, 2, 3, 4, 5, 6\}$ → Source alphabet

$C = \{x, Y, Z\}$ → code alphabet

Symbol	2	5	6	1	4	3
Probability	0.22	0.18	0.17	0.16	0.15	0.12
	43	2	5	6	1	
	0.27	0.22	0.18	0.17	0.16	
	561	43	2			
	0.51	0.27	0.22			

Construction of code words:

2	5	6	1	4	3
Z	xx	xY	xZ	Yx	YY
43	2	5	6	1	
Y	Z	xx	xY	xZ	
561	43	2			
x	Y	Z			

Average length of huffman coding:

$$L(K) = \sum_{i=1}^{n} P_i d_i$$

$$= (0.22 \times 1) + (0.18 \times 2) + (0.17 \times 2) + (0.16 \times 2) + (0.15 \times 2) + (0.12 \times 2)$$

$$= 1.78$$

In this case the entropy appears to be larger than the average length of the huffman coding. This is because of a discrepency in units as the entropy was calculated using $\log_2$ where as the coding has three possible code symbols.

2. a) BAABB $\longrightarrow \underbrace{BA}_{110} + \underbrace{ABB}_{101}$

$\therefore$ $K(BAABB) = 110101$

b) 01011111 $\longrightarrow \underbrace{010}_{AAAAA} + \underbrace{111}_{BB} + \underbrace{111}_{BB}$

$\therefore$ $K'(01011111) = AAAAABBBB$

- where K' is the inverse of the coding K

(C) The first part of this solution will demonstrate that any combination of sequences of length $n-4$ can be encoded, while the second part will show why not all symbols from $n-3, \dots, n$ (the last four) can be encoded:

(i) Even sequences of 'B' symbols can be encoded using 'BB' and odd sequences of 'B' can be encoded by appending the previous sequence of 'BB' symbols with the symbol 'BA'.

A single 'A' symbol can be encoded at the end of a sequence of 'A' symbols using 'ABA' or 'ABB', or at the beginning of a sequence of 'A' symbols using 'ABA' or 'BA'.

Two 'A' symbols can be encoded using 'AAB', three can be encoded using 'AAAB', four can be encoded using 'AAAAB' and five can be encoded using 'AAAAA'.

Therefore when 'AAAAA' is used as a prefix one or more times for the previous codes beginning with 'A', any sequence of 'A' symbols ≥ 6 can be encoded, for example:

Sequence	number of A symbols
AAAAA	5
AAAAA ABB	6
AAAAA AAB	7
AAAAA AAAB	8
AAAAA AAAAB	9
AAAAA AAAAA	10
AAAAA AAAAA ABB	11
⋮	⋮

∴ As any odd or even sequence of 'B' symbols can be represented, and as shown above any number of 'A' symbols can be represented, then because the symbols 'AAB', 'AAAB', and 'AAAAB' all end in 'B' symbols and therefore allow sequences of A and B symbols to be chained together, any sequence of symbols can be encoded up to length $n-4$

(continuation of 2.(c))

(ii) Assuming we have source alphabet $S = \{A, B\}$, the 2^4 possible combinations are as follows:

AAAA
AAAB
AABA
AABB
ABAA
ABAB
ABBA
ABBB
BAAA
BAAB
BABA
BABB
BBAA
BBAB
BBBA
BBBB

Immediately we can identify permutations which can not be generated by the coding specified in the question, when acting as the last four characters, for example BBAA or ABAA cannot be generated as there exists only codes ending in odd numbers of A symbols and in addition there is no single A character which could be appended to a code such as 'ABA'.
In contrast, there are sequences of symbols ending in odd or even symbols of B which enable the encoding of permutations such as AABB and BABB, for example:

$AAAAA + BB \longrightarrow AAA\underbrace{AABB}$

$BA + BB \longrightarrow \underbrace{BABB}$

$AAB + BB \longrightarrow A\underbrace{ABBB}$ (example with odd number of B symbols)

∴ This coding can be used to encode all but possibly the last four symbols of any source message.

d) Average length of source alphabet:

$$L(S) = \sum_{i=1}^{n} p_i d_i$$

$$= (0.7 \times 0.7 \times 0.3 \times 3) + (0.7 \times 0.7 \times 0.7 \times 0.3 \times 4) + (0.7 \times 0.7 \times 0.7 \times 0.7 \times 0.7 \times 5)$$
$$+ (0.7 \times 0.7 \times 0.7 \times 0.7 \times 0.3 \times 5) + (0.7 \times 0.3 \times 0.7 \times 3) + (0.7 \times 0.3 \times 0.3 \times 3)$$
$$+ (0.3 \times 0.7 \times 2) + (0.3 \times 0.3 \times 2)$$

$$\approx 3.28 \quad (2dp)$$

Average length of code alphabet:

$$L(K) = \sum_{i=1}^{n} p_i d_i$$

$$= \sum_{i=1}^{n} 3p_i$$

$$= 3 \sum_{i=1}^{n} p_i$$

$$= 3 \times 1$$

$$= 3$$

$$\frac{L(K)}{L(S)} = \frac{3}{3.28} \approx 0.91 \text{ bits per symbol } (2dp)$$

Entropy of the source alphabet:

$$H(S) = H(p_1, \ldots, p_n) = -\sum_i p_i \log_2 p_i$$

$$= -0.7 \log_2 0.7 - 0.3 \log_2 0.3$$

$$\approx 0.88 \quad (2dp)$$

As $\frac{L(K)}{L(S)} = 0.91$, this demonstrates that the code alphabet is more compressed than the original source alphabet, however as 0.91 is greater than the entropy of 0.88, there is still further redundancy which could be compressed, perhaps through the use of a huffman coding rather than a fixed length coding.

(e) A Tunstall code is a code that represents variable length source strings with fixed length codewords, similarly to the coding in the question.

The Sayood textbook provides an algorithm which generates a Tunstall code which satisfies the following two conditions:

> "1. we should be able to parse a source output sequence into sequences of symbols that appear in the codebook."
>
> "2. we should maximise the average number of source symbols represented by each codeword"
>
> (Sayood, 1996)

The second condition satisfies our requirement that our code have the smallest possible average number of code symbols per source symbol among all such codings and is therefore a suitable coding to use in this context.

The algorithm involves calculating the probabilities of symbols in our source alphabet, removing the source symbol with the highest probability, and then creating new source symbols by concatenating the removed source symbol with all previous source symbols (including itself) until we have a maximum of 2^n source symbols. Where 2^n is the maximum number of codewords we can represent, in this case $2^2 = 4$.

The necessary coding of this kind with binary codewords of length 2 can be derived as follows:

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

$$P(A) = 0.7, \quad P(B) = 0.3$$

$$S = \{B, AA, AB\}$$

$$P(B) = 0.3, \quad P(AA) = 0.7^2 = 0.49, \quad P(AB) = 0.7 \times 0.3 = 0.21$$

$$S = \{B, AB, AAA, AAB\}$$

Source sequence	Code
B	00
AB	01
AAA	10
AAB	11

As in part 2.(d), we can calculate the average number of code symbols per source symbol and compare it with the entropy of the source:

$$L(S) = \sum_{i=1}^{n} p_i d_i$$

$$= (0.3 \times 1) + (0.7 \times 0.3 \times 2) + (0.7 \times 0.7 \times 0.7 \times 3) + (0.7 \times 0.7 \times 0.3 \times 3)$$

$$= 2.19$$

$$L(K) = \sum_{i=1}^{n} p_i d_i$$

$$= \sum_{i=1}^{n} 2 p_i$$

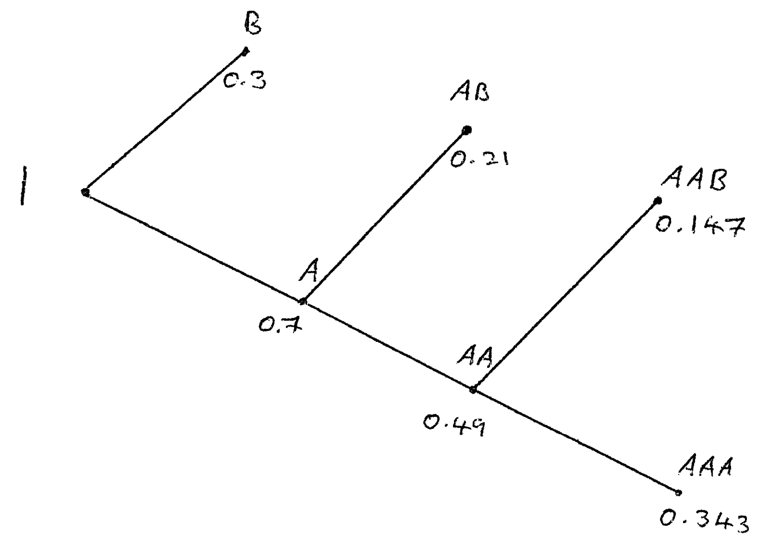
$$= 2 \sum_{i=1}^{n} p_i$$

$$= 2$$

$$\frac{L(K)}{L(S)} = \frac{2}{2.19} = 0.91$$ bits per original source symbol

Again we observe a rate of 0.91 as seen in 2.(d) approaching the entropy $H(S) = 0.88$ (2dp).

To solidify our certainty that $\frac{L(K)}{L(S)}$ is minimised for all such codings of this type, we can observe the coding as a tree of probabilities:



note: leaf nodes are symbols belonging to the source alphabet.

for Tunstall codings represented as such, the school of computer and communication services at the École Polytechnique Fédérale de Lausanne in Switzerland have defined the following:

> "Proof. The sum of the probabilities of the intermediate nodes at depth i is equal to the probability that a leaf has depth strictly greater than i." (EPFL, 2015)

This proves that Tunstall codings give the smallest possible average number of code symbols per source symbol as the higher probability branches (most likely sequences of characters) are assigned more codewords than lower probability branches which tend to be shorter. As all codewords have a fixed length of n bits, this minimises $\frac{L(K)}{L(S)}$ by maximising the denominator $L(S)$.

$\therefore$ This coding has the smallest possible average number of code symbols per source symbol among all such codings.

3.(a) The following solution was implemented using the section on adaptive huffman coding in the Sayood textbook as a reference:

Assuming that no initial statistical information is known by the sender or receiver about the source sequence at the start of the transmission, then the tree relating to the huffman coding will consist of a single node that corresponds to symbols not yet transmitted (NYT) which has a weight of 0.

$$\boxed{0}\ \text{NYT}$$

As symbols are transmitted, the tree will be updated with nodes corresponding to those symbols, whose position in the tree and therefore coding is determined by the frequency and therefore the probability of those symbols occurring.

An initial fixed code for each symbol must be agreed upon in order to be able to identify said symbol the first time it is transmitted from sender to receiver. The ~~processing~~ for determining such a code is as follows:

Assuming a source alphabet $S = \{a_1, a_2, \ldots, a_m\}$ of size m, the equality $m = 2^e + r$ must hold where $0 \leq r < 2^e$, in this case $e = 2$ and $r = 0$.

Each character's encoding would usually be computed by finding the $(e+1)$-bit binary representation of $K-1$ for each a_K in the source alphabet, however as $r=0$, the necessary condition of $1 \leq K \leq 2r$ does not hold and so the alternate method of a_K being encoded as an ~~e~~ e-bit binary representation of $K-r-1$ is used.

using the given source and code alphabets:

$$S = \{a, b, c, d\} \qquad K = \{0, 1\}$$

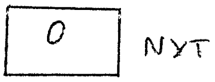
the fixed codes for the source alphabet are as follows:

$$m = 2^e + r$$
$$m = 2^{(2)} + 0$$
$$m = 4$$

source symbol	code symbol
$a = a_1$	$K - r - 1 = 1 - 0 - 1 = 0 \longrightarrow 00$
$b = a_2$	$K - r - 1 = 2 - 0 - 1 = 1 \longrightarrow 01$
$c = a_3$	$K - r - 1 = 3 - 0 - 1 = 2 \longrightarrow 10$
$d = a_4$	$K - r - 1 = 4 - 0 - 1 = 3 \longrightarrow 11$

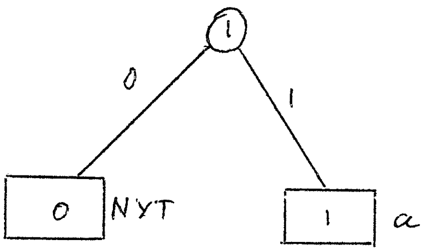
These fixed codes are stored in the NYT list, and when they are sent, are preceded with the NYT symbol (aside from the very first character transmitted which by definition must not have yet been transmitted), and are then removed from the list. A node for the symbol is also created in both trees with an initial weight of 1.

The encoding for "aaabcbacda" is as follows:



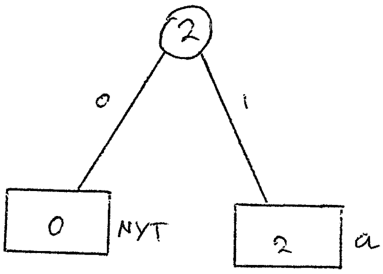
input: "a"

output: "00" → fixed code for 'a'



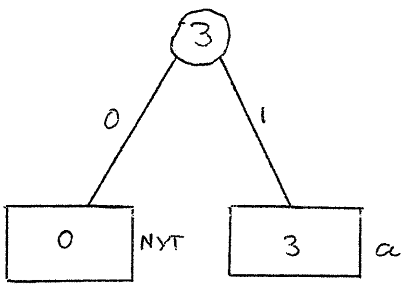
input: "aa"

output: "001"



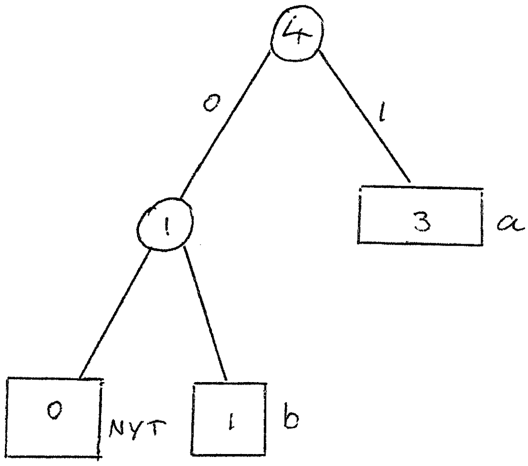
input: "aaa"

output: "0011"



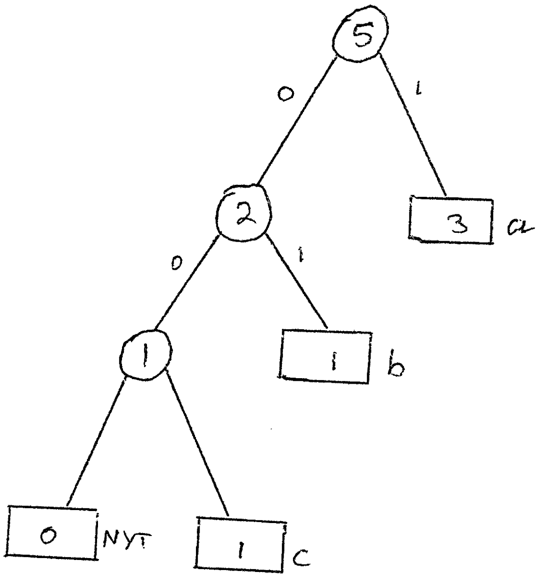
input: "aaab"

output: "0011001" ⟶ NYT code + fixed code for b



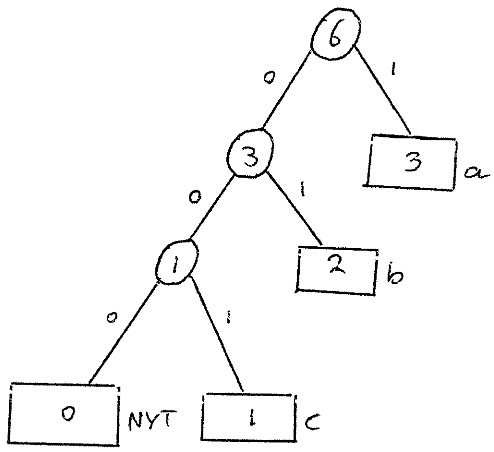
input: "aaabc"

output: "00110010010" ⟶ NYT code + fixed code for c



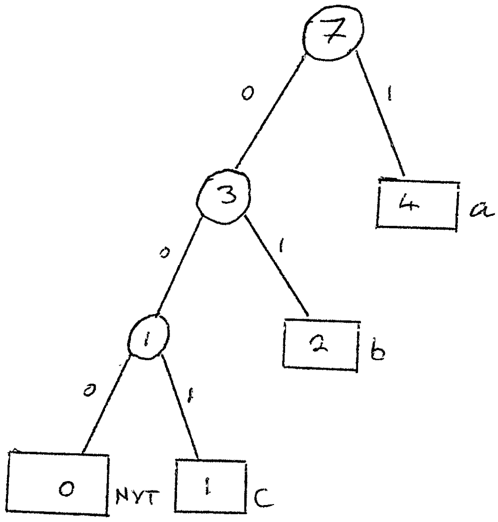
input: "aaabcb"

output: "0011001001001" ⟶ code for b



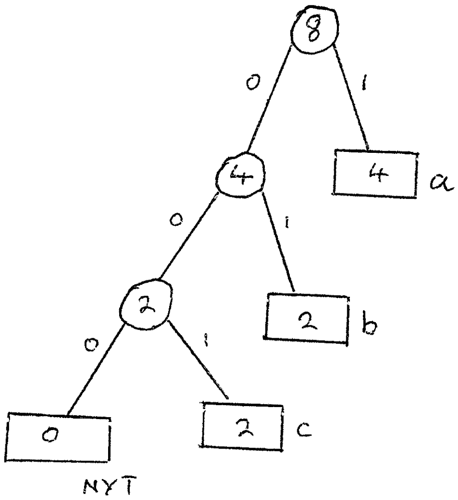
input: "aaabcba"

output: "00110010010011" ⟶ (code) symbol for a



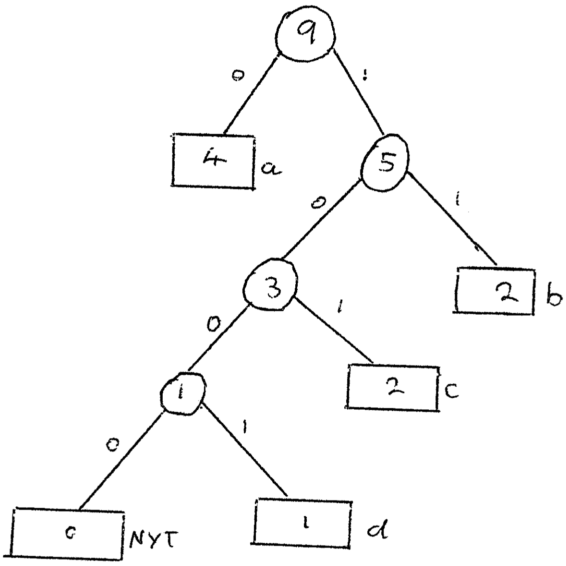
input: "aaabcbac"

output: "00110010010011001" ⟶ code for c



input: "aaabcbacd"

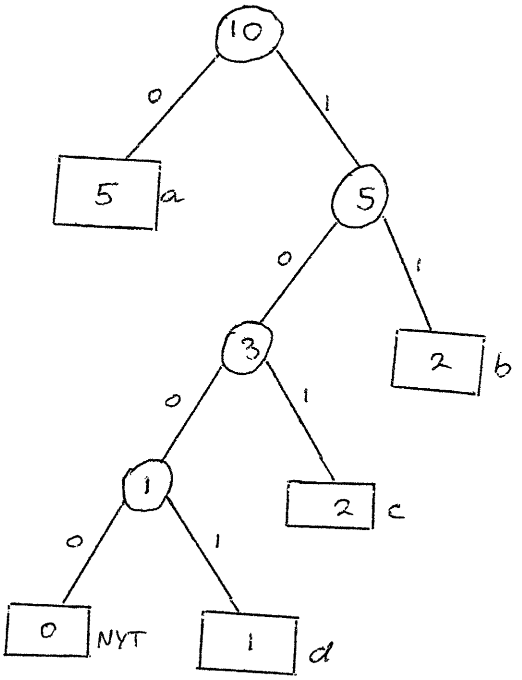
output: "0011001001001100100011" ⟶ NYT code + fixed code for d



input: "aaabcbacda"

output: "00110010010011001000110" → code for a

final state of code tree:



final encoding:

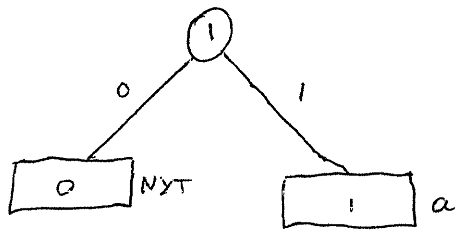
$\therefore$ K(aaabcbacda) = 00110010010011001000110

(b) The process for decoding a message is similar to encoding a message as the same initial tree is used with a single NYT node. The first symbol to be decoded must therefore come from the NYT list.

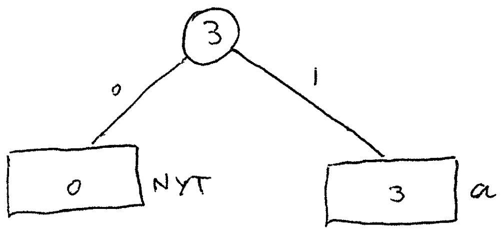
The value of e is 2, so therefore we read data 2 bits at a time.

Input : "00"
output: "a" (retrieved from NYT list)

The tree is then updated in the same way as before:

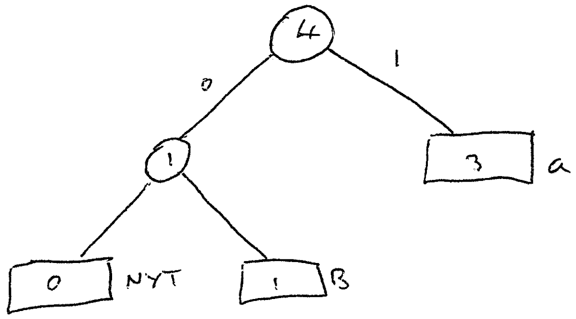


Input: "0011"
output: "a aa" (tree is traversed to 'a' node)



Input : "0011001" (Initially reads 2 bits but because the first bit is the NYT node a third bit is read as e-bits must be read following an NYT signal).

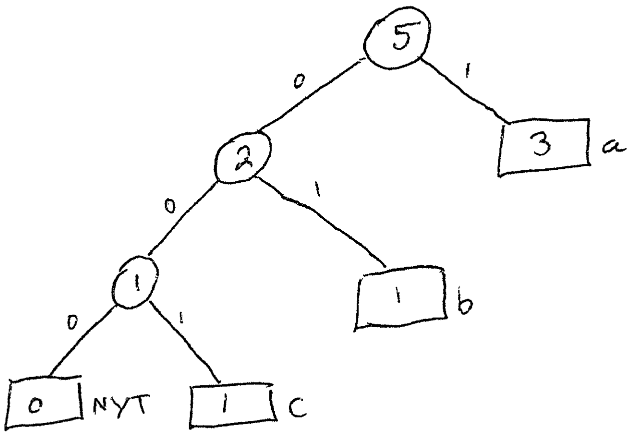
output: "aaab" ('01' is looked up in NYT table which returns a 'b')



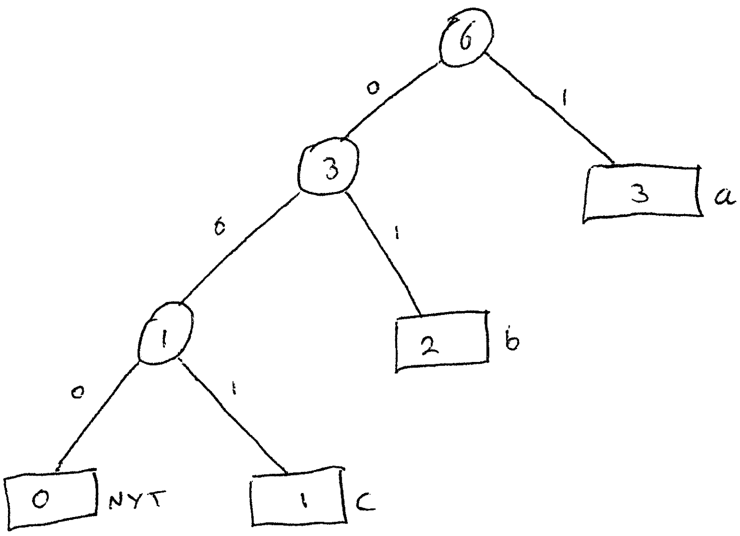
Input: "001100100"
output: "aaab"

no additional characters are output yet as an NYT node has been reached by decoding the characters 00

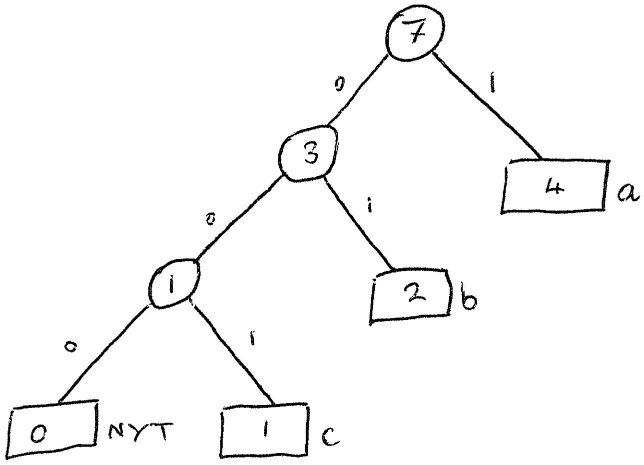
Input: "00110010010"
output: "aaabc" (c is retrieved from NYT list)



Input: "0011001001001"
output: "aaabcb" (node b is reached by following 01)

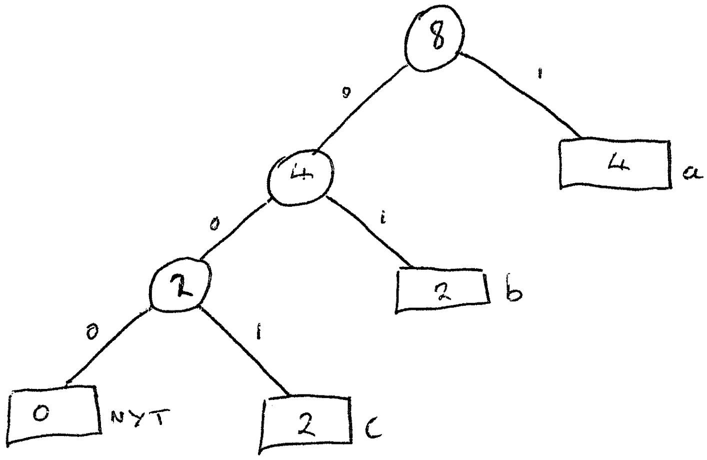


Input: "001100100100110"
output: "aaabcba" (a is output because of '1' bit, traversal then begins down left hand side of tree due to '0' bit)



Input: "0011001 0010011001"

output: "aaabcbac" (last three bits of input, '001', traverse to node 'c')

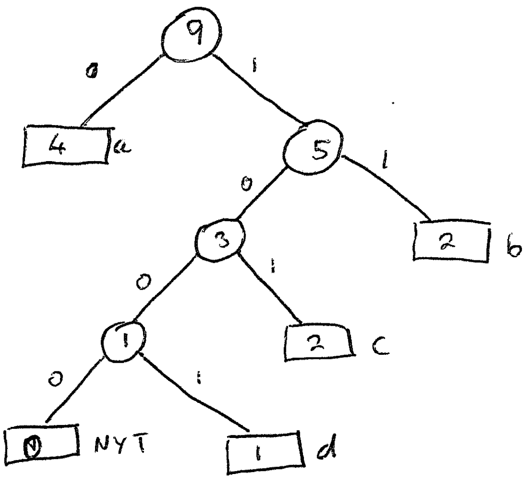


Input: "0011001001001100100"

output: "aaabcbac" (output remains unchanged. '00' causes traversal twice down left hand side of tree).

Input: "00110010010011001000011" (0 bit traverses to NYT node so an additional e-bits must be read)

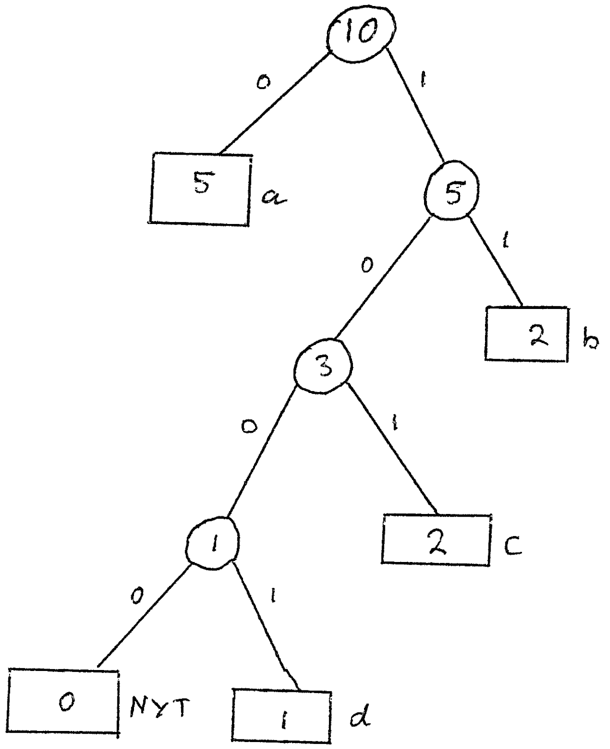
output: "aaabcbacd" ('11' is looked up in NYT table which returns a 'd')



Input : "0011001001001100100011<u>0</u>" (last bit of input)

output : "aaabcbacda" (due to readjustment of the tree, decoding '0' results in an 'a' as the final output character)

final state of tree:



As demonstrated above, an identical tree has been generated by the receiver of the message.

Based on the final decoded output, we can state that:

$\therefore$ $K'(00110010010011001000110) = \underline{\underline{aaabcbacda}}$

where K' is assumed to be the decoding function.

(C) The adaptive huffman encoding algorithm used to solve questions 3.(a) and 3.(b) can be broken down (for the sender) into two distinct phases:

(i) Encoding Procedure
(ii) update Procedure

As the updating of the tree occurs just after a character has been encoded (for example when the weighting of a node changes, resulting in a swap) this solution assumes that the update procedure therefore has no effect on the encoding procedure itself. in terms of complexity.

worst case complexity of encoding Procedure:

$O(n)$

Justification:

Binary trees have $2n-1$ nodes. Assuming that the tree has been fully updated with all characters from the not-yet-transmitted list, then in the worst case when the leaf node of the character to be transmitted is the last character reached (algorithm has performed a full tree traversal while searching for the leaf node of character) then $2n-1$ nodes will have been traversed resulting in $O(2n-1)$ complexity, simplifying to $O(n)$.

Another scenario resulting in this worst case complexity could be when only the NYT (not yet transmitted) node exists in the tree and the final character from the source alphabet needs to be encoded.

In order to work out the fixed code of the character to transmit, as discussed previously, the calculation $K-r-1$ must be performed. The value of r for this encoding is known to be 0, however to determine the value of K we must search the source alphabet linearly for the index K of the source character in question. for example:

$$S = \{a_1, a_2, \dots, a_n\}$$

for any a_K, must search list until a matching source character is found, and assuming there is no sophisticated sorting, this requires a linear search of complexity $O(n)$

4(a)

Input Symbol	Sequence	list
m	1	emnoy
o	1, 3	menoy
n	1, 3, 3	omeny
e	1, 3, 3, 3	nomey
y	1, 3, 3, 3, 4	enomy
m	1, 3, 3, 3, 4, 4	yenom
o	1, 3, 3, 3, 4, 4, 4	myeno
n	1, 3, 3, 3, 4, 4, 4, 4	omyen
e	1, 3, 3, 3, 4, 4, 4, 4, 4	nomye
y	1, 3, 3, 3, 4, 4, 4, 4, 4, 4	enomy
m	1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4	yenom
o	1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4	myeno
n	1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4	omyen
e	1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4	nomye
y	1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4	enomy

$\therefore$ moneymoneymoney $\longrightarrow$ 1, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4

(b) $P(m) = \frac{3}{15} = 0.2$, $P(o) = \frac{3}{15} = 0.2$, $P(n) = \frac{3}{15} = 0.2$,

$P(e) = \frac{3}{15} = 0.2$, $P(y) = \frac{3}{15} = 0.2$

Entropy of original source:

$$H(S) = -\sum_i p_i \log_r p_i$$, in this case $r = 5$ as there are five source symbols

$$H(0.2, 0.2, 0.2, 0.2, 0.2)$$

$$= -(0.2)\log_5(0.2) - (0.2)\log_5(0.2) - (0.2)\log_5(0.2) - (0.2)\log_5(0.2) - (0.2)\log_5(0.2)$$

$$= 1$$

Entropy of translated source:

$P(1) = \frac{1}{15}$, $P(2) = \frac{0}{15} = 0$, $P(3) = \frac{3}{15} = 0.2$, $P(4) = \frac{11}{15}$

$P(0) = \frac{0}{15} = 0$

As $\log 0$ is undefined, we ignore the zero probabilities for the entropy calculation:

$$H\left(\frac{1}{15}, \frac{3}{15}, \frac{11}{15}\right)$$

$$= -\left(\frac{1}{15}\right)\log_5\left(\frac{1}{15}\right) - \left(\frac{3}{15}\right)\log_5\left(\frac{3}{15}\right) - \left(\frac{11}{15}\right)\log_5\left(\frac{11}{15}\right)$$

$$\approx 0.45 \text{ (2dp)}$$

(c) This pre-processing step could be useful in compressing natural language text as natural languages tend to have a few characters (or words) that occur very frequently (such as e and t in English) and so these frequent characters would likely be moved to the front of the transform list, resulting in a larger number of smaller numbers (e.g. 0, 1, 2) in the output, resulting in less than the original 26 characters (in the case of a source made up of the latin alphabet), and therefore potentially a lower entropy and more compression when the message is encoded using a scheme such as Huffman encoding or arithmetic encoding.

Bibliography

- Sayood, K. (1996). Introduction to data compression (4th ed.). Morgan Kaufmann.

- Variable to Fixed Length Source Coding – Tunstall coding (P.1, Tech). (2015). Lausanne: École Polytechnique Fédérale de Lausanne - School of computer and communication sciences