Multimedia Communications

Huffman Coding

McMaster University

Optimal codes

- Suppose that $a_i \rightarrow w_i \in C^+$ is an encoding scheme for a source alphabet $A=\{a_1, \ldots, a_N\}$. Suppose that the source letter $a_1, \ldots, a_N$ occur with relative frequencies $f_1, .. f_N$ respectively. The average code word length of the code is defined as:

$$\bar{l} = \sum_{i=1}^{N} f_i l_i$$

where l_i is the length of w_i

- The average number of code letters required to encode a source text consisting of N source letters is $N\bar{l}$
- It may be expensive and time consuming to transmit long sequences of code letters, therefore it may be desirable for $\bar{l}$ to be as small as possible.
- It is in our power to make $\bar{l}$ small by cleverly making arrangements when we devise the encoding scheme.

McMaster University

Optimal codes

- What constraints should we observe?
- The resulting code should be uniquely decodable
- Considering what we saw in the previous chapter, we confine ourselves to prefix codes.
- An encoding scheme that minimizes $\bar{l}$ is called optimal encoding
- The process of finding the optimal code was algorithmized by Huffman.

McMaster University

Optimal codes

- The necessary conditions for an optimal variable-length binary code are:

1. Given any two letters a_j and a_k if $P(a_j) \geq P(a_k)$ then $l_j \leq l_k$
2. The two least probable letters have codewords with the same maximum length
3. In the tree corresponding to the optimum code, there must be two branches stemming from each intermediate node
4. Suppose we change an intermediate node into a leaf node by combining all the leaves descending from it into a composite word of a reduced alphabet. Then, if the original tree was optimal for the original alphabet, the reduced tree is optimal for the reduced alphabet.

Condition #1 and #2

- Condition #1 is obvious
- Suppose an optimum code C exists in which the two code words corresponding to the least probable symbols do not have the same length. Suppose the longer code word is k bits longer than the shorter one.
- As C is optimal, the codes corresponding to the least probable symbols are also the longest.
- As C is a prefix code, none of the code words is a prefix of the longer code.

McMaster University

Condition #2

- This means that, even if we drop the last k bits of the longest code word, the code words will still satisfy the prefix condition.
- By dropping the k bits, we obtain a new code that has a shorter average word length.
- Therefore, C cannot be optimal.

Conditions #3 and #4

- Condition #3: If there were any intermediate node with only one branch coming from that node, we could remove it without affecting the decipherability of the code while reducing its average length.
- Condition #4: If this condition were not satisfied, we could find a code with smaller average code length for the reduced alphabet and then simply expand the composite word of a reduced alphabet. Then, if the original tree was optimal for the original alphabet, the reduced tree is optimal for the reduced alphabet

McMaster University

- Codes generated by Huffman algorithm (explained shortly) meet the above conditions

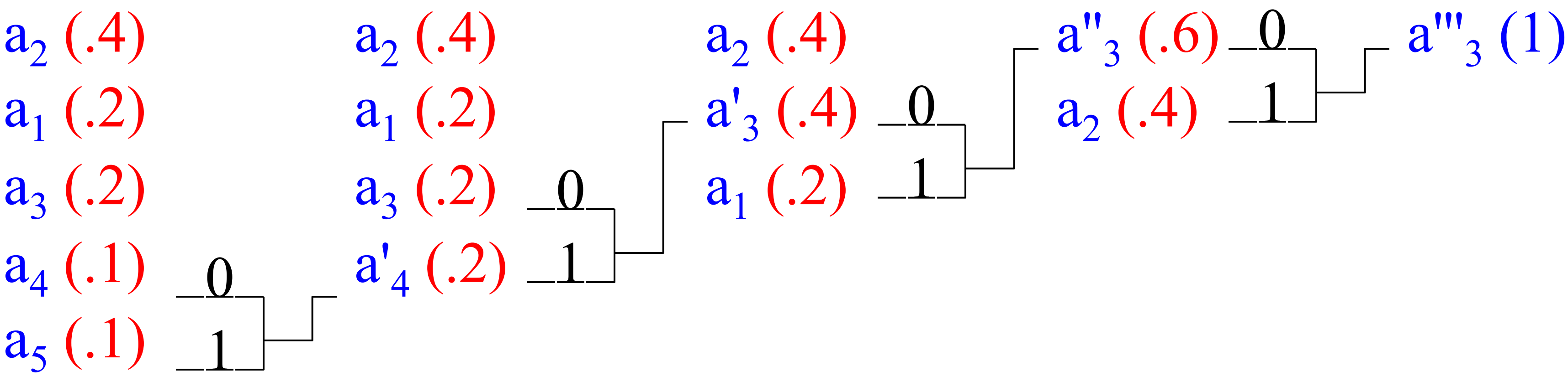
Building a Huffman Code

- The main idea:
  - Let S be a source with alphabet $A=\{a_1,\ldots,a_N\}$ ordered according to decreasing probability.
  - Let S' be a source with alphabet $A'=\{a'_1,\ldots,a'_{N-1}\}$ such that

$$\begin{cases} P(a'_k) = P(a_k) & 1 \le k \le N-2 \\ P(a'_{N-1}) = P(a_N) + P(a_{N-1}) & \end{cases}$$

  - Then if a prefix code is optimum for S', the corresponding prefix code for S is also optimum.
  - We continue this process until we have a source with only two symbols.

Building a Huffman Code



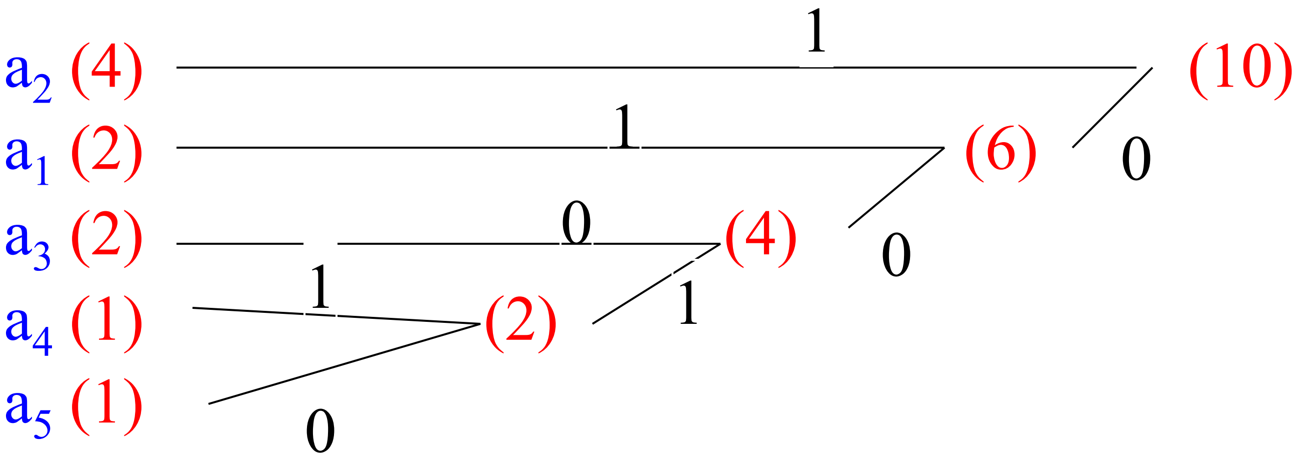
a_2	(.4)	1
a_1	(.2)	01
a_3	(.2)	000
a_4	(.1)	0010
a_5	(.1)	0011

average length = 2.2 bits

entropy = 2.122 bits

- We can use a tree diagram to build the Huffman code
- Assignment of 0 and 1 to the branches is arbitrary and gives different Huffman codes with the same average codeword length
- Sometimes we use the counts of symbols instead of their probability
- We might draw the tree horizontally or vertically

Building a Huffman Code



a_2 : 1

a_1 : 01

a_3 : 000

a_4 : 0010

a_5 : 0011

Huffman Codes

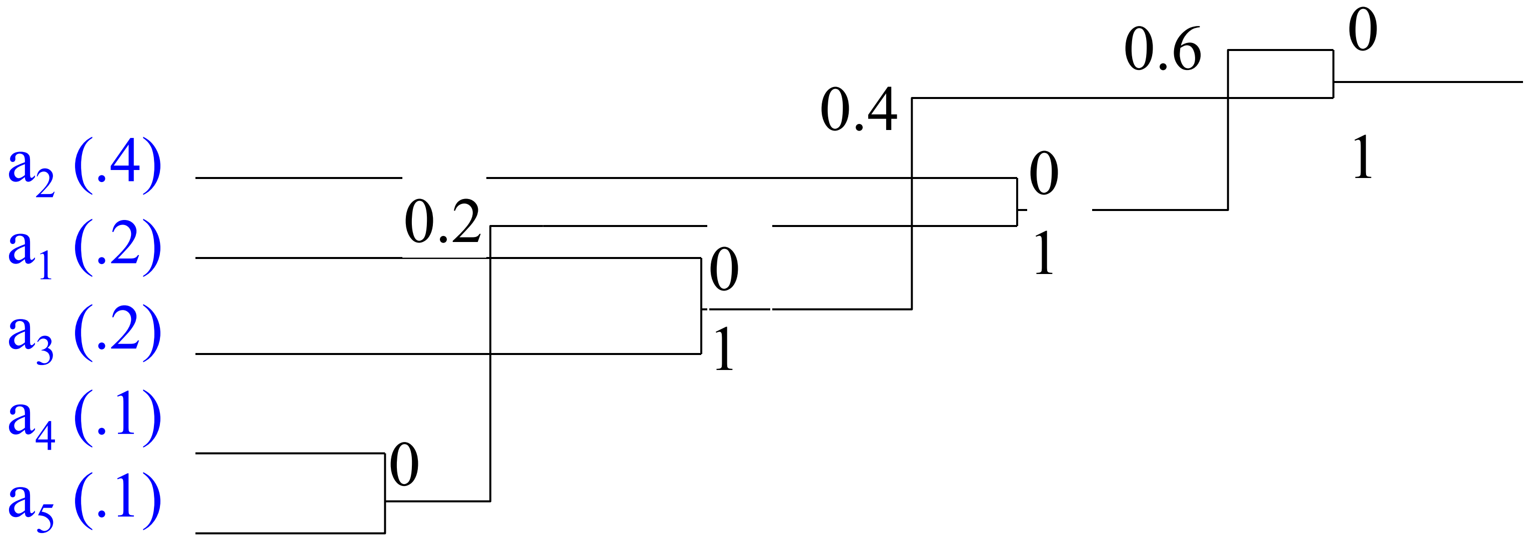
- Since the Huffman code which gives the optimal codes meets the Conditions #1, #2, #3 and #4, these conditions are also sufficient conditions.
- It can be shown that not only does Huffman's algorithm always give a "right answer", but also, every "right answer".
  - For the proof see section 4.3.1 in "Information theory and data compression" by D. Hankerson.

McMaster University

Minimum Variance Huffman Codes

- According to where in the list the combined source is placed, we obtain different Huffman codes with the same average length (same compression performance).
- In some applications we do not want the code word lengths to vary significantly from one symbol to another (example: fixed-rate channels).
- To obtain a minimum variance Huffman code, we always put the combined symbol as high in the list as possible.

Minimum Variance Huffman Codes



a$_2$ (.4) 00
a$_1$ (.2) 10
a$_3$ (.2) 11
a$_4$ (.1) 010
a$_5$ (.1) 011

average length = 2.2 bits
entropy = 2.122 bits

Huffman Codes

- The average length of a Huffman code satisfies

$$H(S) \le \bar{l} \le H(S) + 1$$

- The upper bound is loose. A tighter bound is

$$\bar{l} \le \begin{cases} H(S) + P_{max} & P_{max} < 0.5 \\ H(S) + P_{max} + 0.086 & P_{max} \ge 0.5 \end{cases}$$

McMaster University

Extended Huffman Codes

- If the probability distribution is very skewed (large P_{max}), Huffman codes become inefficient.
- We can reduce the rate by grouping symbols together.
- Consider the source S of independent symbols with alphabet $A=\{a_1,\ldots,a_N\}$.
- Let us construct an extended source $S^{(n)}$ by grouping n symbols together
- Extended symbols in the extended alphabet

$$A^{(n)} = \underbrace{A \times A \times \ldots \times A}_{\text{(n times)}}$$

Extended Huffman Codes: Example

Huffman code (n = 1)

a_1	.95	0
a_3	.03	10
a_2	.02	11

R = 1.05 bits/symbol

H = .335 bits/symbol

Huffman code (n = 2)

a_1a_1	.9025	0
a_1a_3	.0285	100
a_3a_1	.0285	101
a_1a_2	.0190	111
a_2a_1	.0190	1101
a_3a_3	.0009	110000
a_3a_2	.0006	110010
a_2a_3	.0006	110001
a_2a_2	.0004	110011

R = .611 bits/symbol

Rate gets close to the entropy only for n > 7.

McMaster University

Extended Huffman Codes

- We can build a Huffman code for the extended source with a bit rate $R^{(n)}$ which satisfies

$$H(S^{(n)}) \leq R^{(n)} \leq H(S^{(n)}) + 1$$

- But $R = R^{(n)}/n$ and, for i.i.d. sources, $H(S) = H(S^{(n)})/n$, so

$$H(S) \leq R \leq H(S) + \frac{1}{n}$$

- As $n \to \infty$, $R \to H(s)$.
  - Complexity (memory, computations) also increases (exponentially).
  - Slow convergence for skewed distributions.

McMaster University

Nonbinary Huffman Codes

- The code elements are coming from an alphabet with m>2 letters
- Observations
  1. The m symbols that occur least frequently will have the same length
  2. The m symbols with the lowest probability differ only in the last position
- Example: ternary Huffman code for a source with six letters
  - First combine three letters with the lowest probability, giving us a reduced alphabet with 4 letters,
  - Then combining three lowest probability gives us an alphabet with only two letters
  - We have three values to assign and only two letters, we are wasting one of the code symbols

Nonbinary Huffman Codes

- Instead of combining three letters at the beginning we could have combined two letters, into an alphabet of size 5,
- If we combine three letters from this alphabet we end up in alphabet with a size of 3.
- We could combine three in the first step and two in the second step. Which one is better?
- Observation:
  - all combine letters will have codewords of the same length
  - Symbols with the lowest probability will have the longest codeword
- If at some stage we are allowed to combine less than m symbols the logical place is the very first stage

• In the general case of a code alphabet with m symbols (m-ary) and a source with N symbols, the number of letters combined in the first phase is: N modulo (m-1)

Adaptive Huffman Codes

- Two-pass encoders: first collect statistics, then build Huffman code and use it to encode source.
- One-pass (recursive) encoders:
  - Develop the code based on the statistics of the symbols already encoded.
  - The decoder can build its own copy in a similar way.
  - Possible to modify code without redesigning entire tree.
  - More complex than arithmetic coding.

Adaptive Huffman Codes

- Feature: no statistical study of the source text need to be done before hand
- The encoder keeps statistics in the form of counts of source letters, as the encoding proceeds, and modifies the encoding according to those statistics
- What about the decoder? if the decoder knows the rules and conventions under which the encoder proceeds, it will know how to decode the next letter
- Besides the code stream, the decoder should be supplied with the details of how encoder started and how the encoder will proceed in each situation

McMaster University

Adaptive Huffman Codes

- In adaptive Huffman coding, the tree and corresponding encoding scheme change accordingly
- Two versions of the adaptive Huffman will be described:
  1. Primary version
  2. Knuth and Gallager method

Primary version

- The leaf nodes (source letters) are sorted non-increasingly
- We merge from below in the case of a tie
- When two nodes are merged they are called siblings
- When two nodes are merged, the parent node will be ranked on the higher of the two sibling nodes.
- In the labeling of the edges (branches) of the tree, the edge going from parent to highest sibling is labeled zero
- At start all letters have a count of one
- Update after count increment: the assignment of leaf nodes to source letters is redone so that the weights are in non-increasing order

Performance

- Adaptive Huffman codes: respond to locality
- Encoder is "learning" the characteristics of the source. The decoder must learn along with the encoder by continually updating the Huffman tree so as to stay in synchronization with the encoder.
- Another advantage: they require only one pass over the data.
- Of course, one-pass methods are not very interesting if the number of bits they transmit is significantly greater than that of the two-pass scheme. Interestingly, the performance of these methods, in terms of number of bits transmitted, can be better or worse than that of static Huffman coding.
- This does not contradict the optimality of the static method as the static method is optimal only over all methods which assume a time-invariant mapping.

Primary version

- The Huffman tree and associated encoding scheme are expected to settle down eventually to the fixed tree and scheme that might have arisen from counting the letters in a large sample of source text
- The advantage of adaptive Huffman encoding can be quite important in situations that the source nature changes
- Exp: a source text consists of a repeated 10,000 times, b repeated 10,000 times, c repeated 10,000 and d repeated 10,000.
- Non-adaptive Huffman: probability ¼, four binary codes each 2 bits

Primary version

- Adaptive Huffman: encode almost all a's by a single digit. The b's will be coded by two bits each, c's and d's with three bits each
- It is wasting the advantage over non-adaptive Huffman
- Source of problem: when the nature of the source text changes, one or more of the letters may have built up such a hefty counts that it takes a long time for the other letters to catch up
- Solution: periodically multiply all the counts by some fraction and round down (of course the decoder must know when and how much the counts are scaled down)

Knuth & Gallager method

- Problem with primary method: updating the tree
- The tree resulting from an application of Huffman's algorithm belong to a special class of diagrams called binary tree
- A binary tree with n leaf nodes has 2n-1 nodes and 2n-2 nodes other than the root
- Theorem: Suppose that T is a binary tree with n leaf nodes with each node y_i assigned a non-negative weight x_i. Suppose that each parent is weighted with the sum of the weights of its children. Then T is a Huffman tree (obtainable by some instance of Huffman's algorithm) if and only if 2n-2 non-root nodes of T can be arranged in a sequence $y_1 y_2 \ldots y_{2n-2}$ with the properties that
  1. $x_1 \le x_2 \le \ldots\ldots \le x_{2n-2}$
  2. y_{2k-1} and y_{2k} are siblings

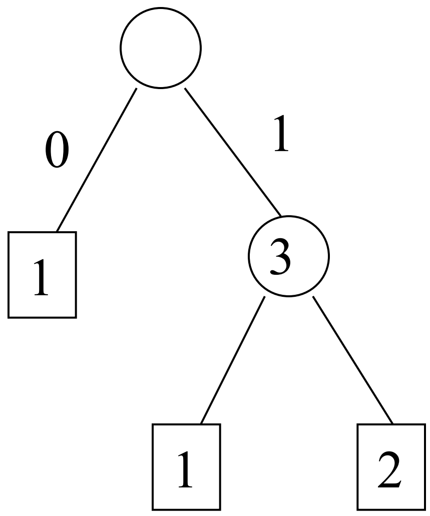
McMaster University

Knuth & Gallager method

- Knuth and Gallager proposed to manage the Huffman tree at each stage in adaptive Huffman encoding by ordering the nodes $y_1 y_2 \ldots y_{2n-2}$ so that he weight on y_k is non-decreasing with k and so that y_{2k-1} and y_{2k} are siblings
- This arrangement allows updating after a count increment in order of n operations, while redoing the whole tree from scratch requires order of n^2 operations.

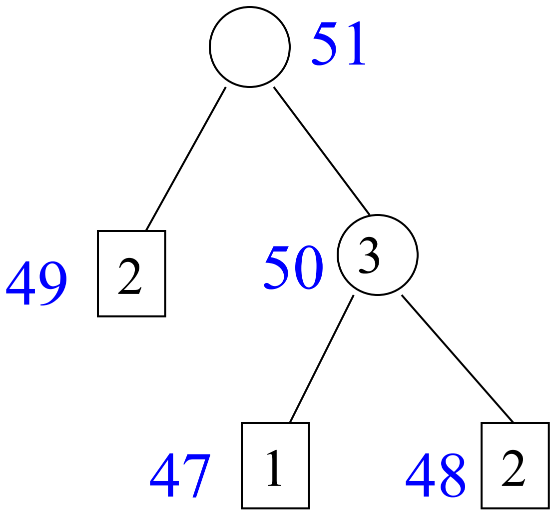
Knuth & Gallager method

- Squares denote external nodes and correspond to the symbols
- Codeword for a symbol can be obtained by traversing the tree from root to the external node corresponding to the symbol, 0 corresponds to a left branch and 1 corresponds to a right branch
- Weight of a node: a number written inside the node
  - For external nodes: number of times the symbol corresponding to the node has been encountered
  - For internal nodes: sum of the weight of its off springs



Knuth & Gallager method

- Node number: a unique number assigned to each internal and external node
- Block: set of nodes with the same weight
- Node interchange: entire subtree being cut off and re-grafted in new position

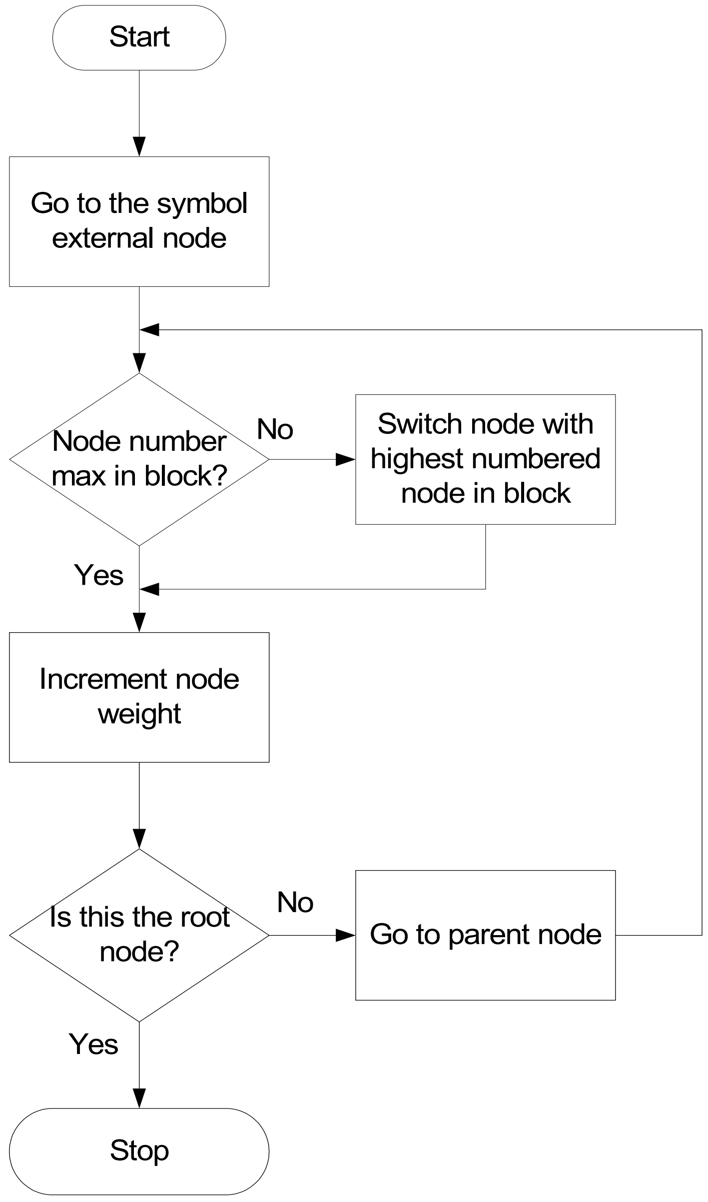


Adaptive Huffman Codes

- Version 1: encoder is initialized with source alphabet with a count of one
  - Send the binary code for the symbol (traverse the tree)
  - If the node is the root, increase its weight and exit
  - If the node has the highest node number in its block, increment its weight, update its parent.
  - If the node does not have the highest node number, swap it with the node with highest number in the block, increment its weight, update its parents.

Adaptive Huffman Codes

- Update a node:
  - If the node is the root, increase its weight and exit
  - If the node has the highest node number in its block, increment its weight, update its parent.
  - If the node does not have the highest node number, swap it with the node with highest number in the block, increment its weight, update its parents.


Start
Go to the symbol external node
Node number max in block?
No
Switch node with highest numbered node in block
Yes
Increment node weight
Is this the root node?
No
Go to parent node
Yes
Stop

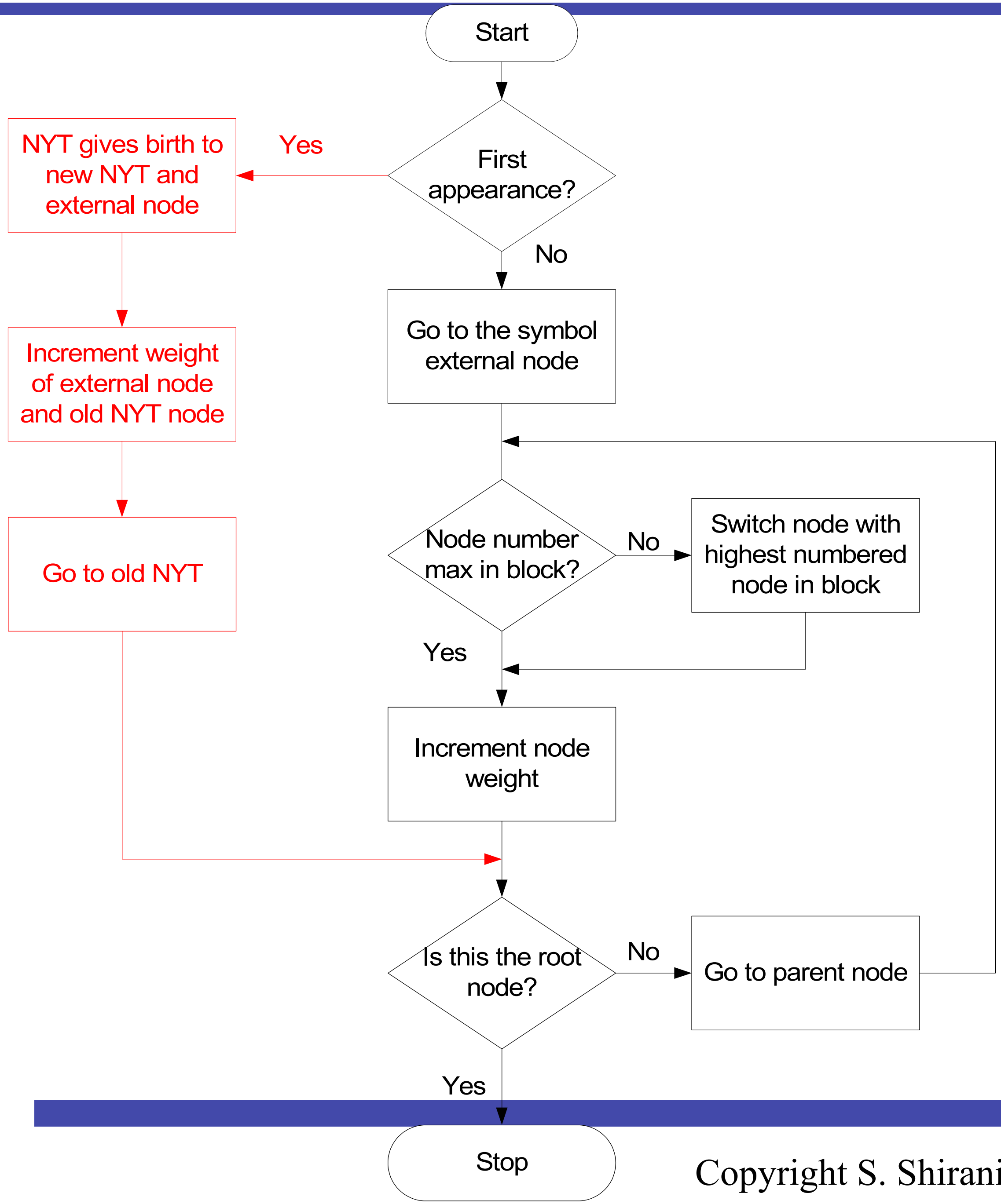
Adaptive Huffman Codes

- Version 2: no initialization is required
- At the beginning the tree consists of a single node called Not Yet Transmitted (NYT) with a weight of zero.
- If the symbol to be coded is new:
  - An NYT and a fixed length code for the symbol is transmitted.
  - In the tree, NYT is converted to a new NYT and an external node for the symbol. The weight of the new external node is set to one. The weight of the parent (old NYT) is set to one. The parent of the old NYT is updated.
- If the symbol already exists:
  - Send the binary code for the symbol (traverse the tree)
  - Update the node

Adaptive Huffman Codes

- Update a node:
  - If the node is the root, increase its weight and exit
  - If the node has the highest node number in its block, increment its weight, update its parent.
  - If the node does not have the highest node number, swap it with the node with highest number in the block (as long as the node with the higher number is not the parent of the node being updated), increment its weight, update its parents.

McMaster University


Start
First appearance?
Yes
NYT gives birth to new NYT and external node
Increment weight of external node and old NYT node
Go to old NYT
No
Go to the symbol external node
Node number max in block?
No
Switch node with highest numbered node in block
Yes
Increment node weight
Is this the root node?
No
Go to parent node
Yes
Stop

Adaptive Huffman Codes

- Fixed length coding:
  - If the n symbols in the alphabet, we find e and r such that:

$$n = 2^e + r$$

$$0 \le r < 2^e$$

If $1 \le k \le 2r$ then a_k is coded as the (e+1)-bit binary representation of k-1

If k>2r, a_k is coded as the e-bit binary representation of k-r-1

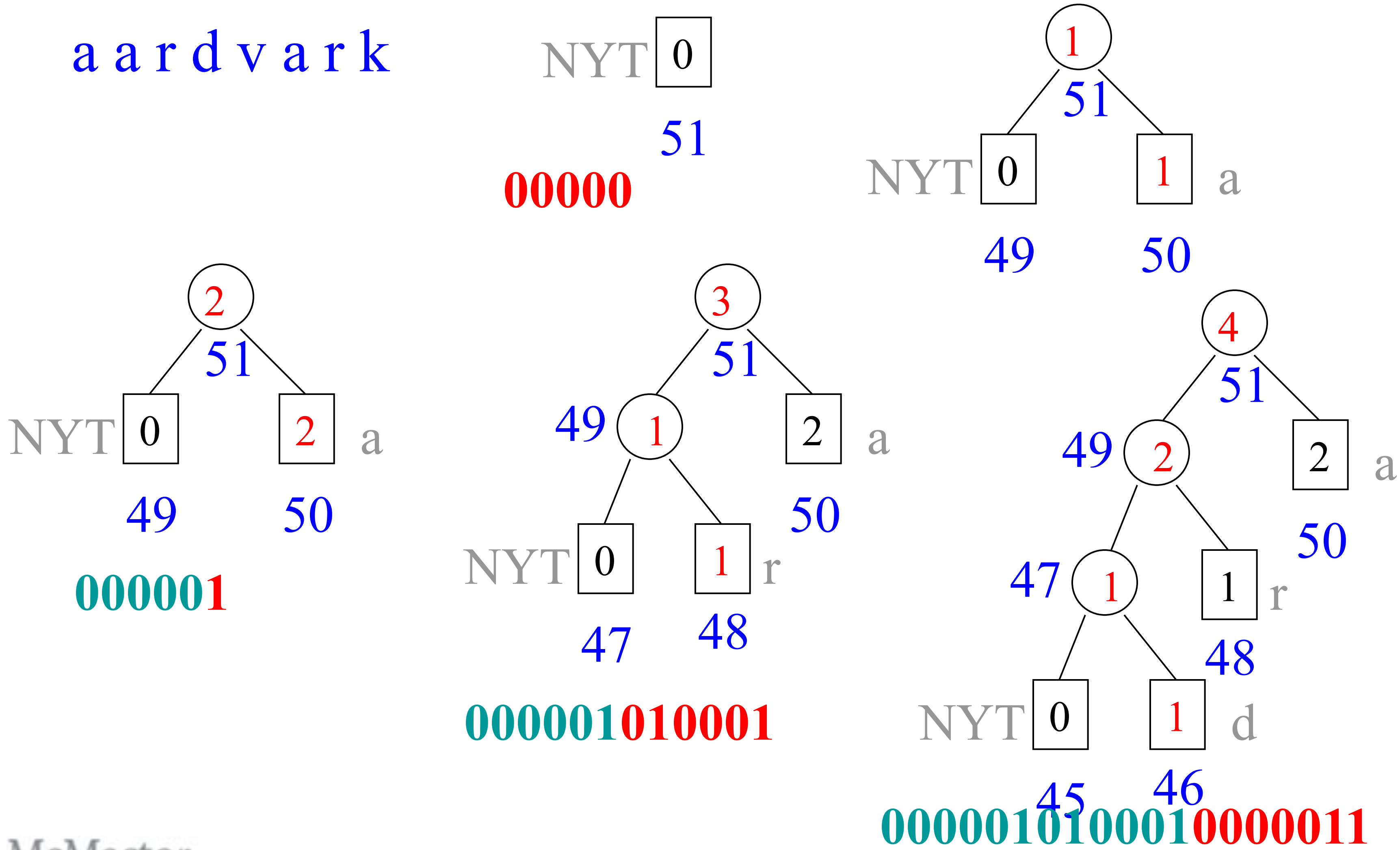
Exp: n=26, e=4, r=10

a2: 00001

a22: 1011

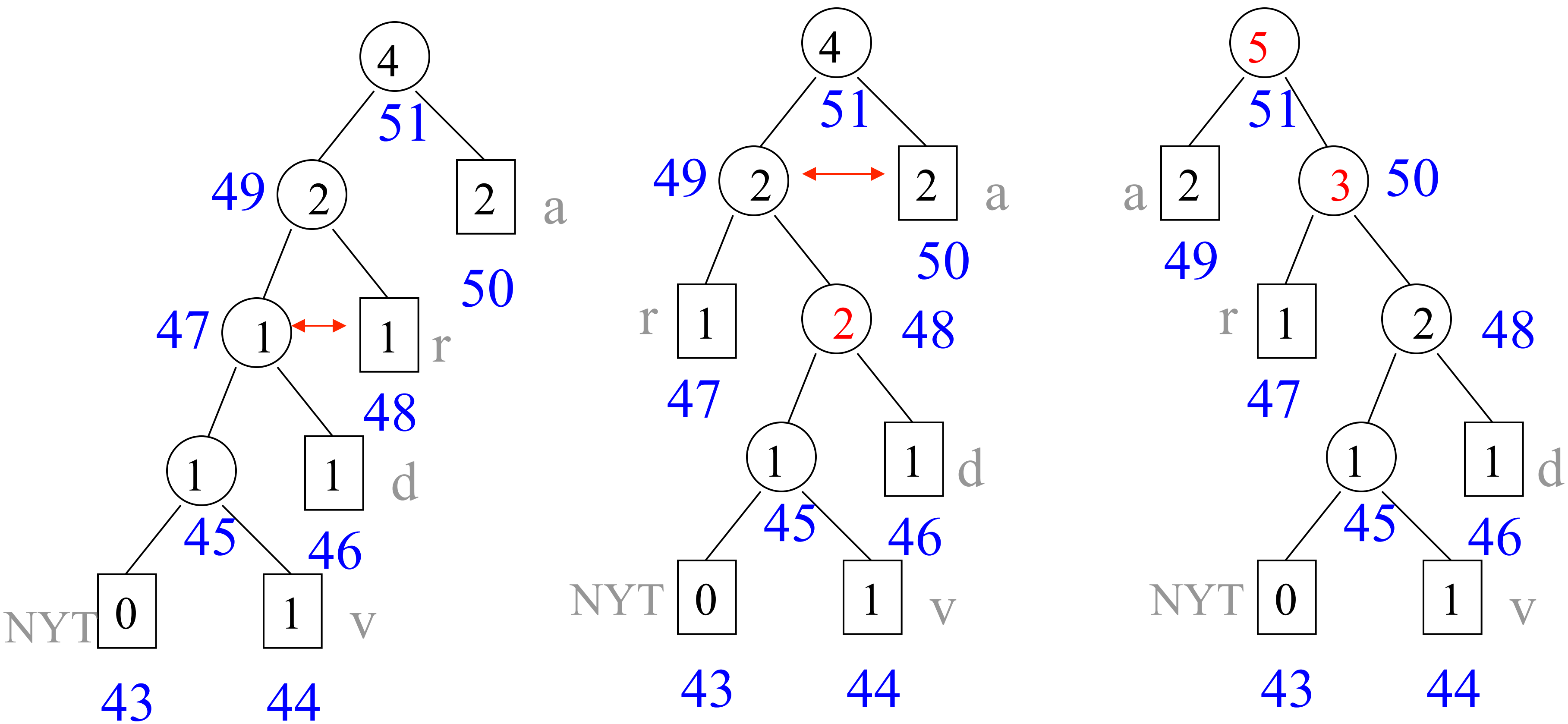
Adaptive Huffman Codes

a a r d v a r k



Adaptive Huffman Codes

a a r d v a r k



McMaster University

Golomb Codes

- Golomb-Rice 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.
- Unary code: simple codes for this situation
- Unary code of an integer n is n 1s followed by a 0.
- Exp: 4 -> 11110

Golomb Codes

- Golomb codes: has a parameter m
- An integer n is represented by two numbers q and r:

$$q = \left\lfloor \frac{n}{m} \right\rfloor$$

$$r = n - qm$$

  - q is unary coded
  - r can take values 0,1,2, .., m-1
  - m a power of two: use the $\log_2 m$-bit representation of r
  - Not a power of two: use the $\lceil \log_2 m \rceil$-bit representation of r

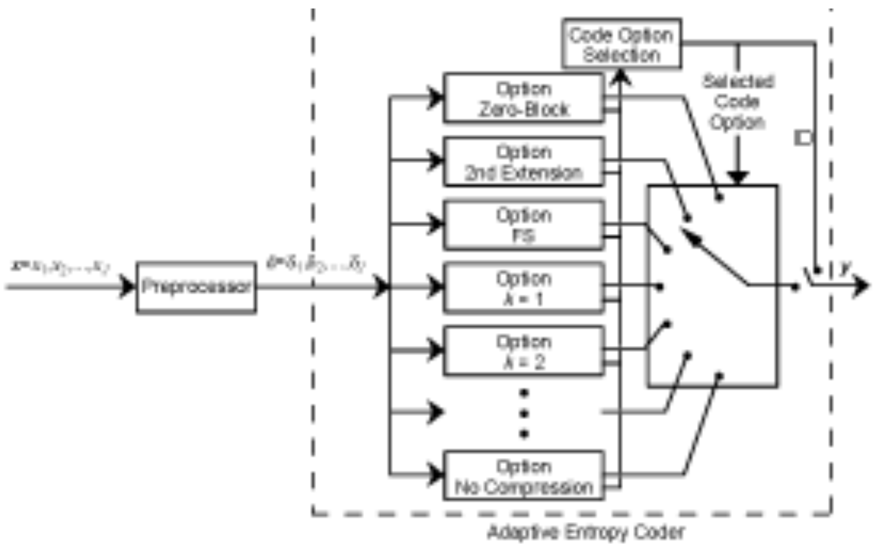
McMaster University

Golomb Codes

- The number of bits can be reduced if we use $\lfloor \log_2 m \rfloor$ bit representation of r for the first $2^{\lceil \log_2 m \rceil} - m$ values and $\lceil \log_2 m \rceil$ bit binary representation of $r + 2^{\lceil \log_2 m \rceil} - m$ for the rest of values
- Exp: m=5, $\lfloor \log_2 5 \rfloor = 2$ $\lceil \log_2 5 \rceil = 3$
- First 8-5=3 values of r (0,1,2) will be represented by 2 bits binary representation of r, and the next two values (3,4) will be represented by the 3-bit representation of r+3 (6,7)
- 14 -> 110111

Rice codes

- Rice coding has two steps: preprocessing and coding
- Preprocessing: generates a sequence of nonnegative integers where smaller values are more probable than larger values



Rice codes

$\{y_i\}$

$prediction: \hat{y}_i$

$\hat{y}_i = y_{i-1}$

$d_i = y_i - y_{i-1}$

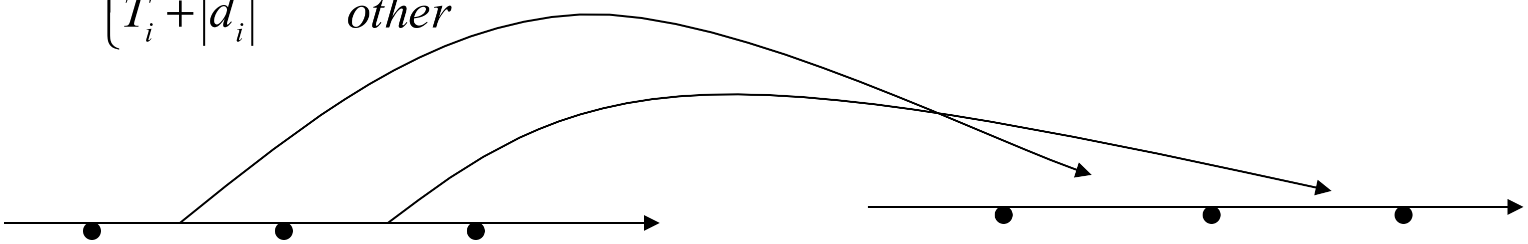
$y_{\min} < y_i < y_{\max}$

$T_i = \min\{y_{\max} - \hat{y}_i, \hat{y}_i - y_{\min}\}$

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

Rice codes

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



- The preprocessed sequence is divided into segments with each segment being further divided into blocks of size J (e.g., 16)
- Each block is coded using one of the following options (the coded block is transmitted along with an identifier indicating the option used):

Rice codes Options

1. Fundamental sequence: unary code (n is coded as n 0s followed by a 1)
2. Split sample options:
   - There is a parameter k
   - A m-bit number n code consists of k least significant bits of n followed by the unary code of the m-k most significant bits
3. Second extension option: the sequence is divided into consecutive pairs of samples. Each pair is used to obtain an index: $\gamma = \frac{1}{2}(x_i + x_{i+1})(x_i + x_{i+1} + 1) + x_{i+1}$, the index is coded using a unary code.
4. Zero block option: used when one or more blocks are all zero. Number of zero blocks is transmitted using a code. See Table 3.17

McMaster University

Rice codes

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

McMaster University

Variable Length Codes & Error Propagation

A 00
B 01
C 10
D 11

A B C D A B: 00 01 10 11 00 01

00 01 10 10 00 01

A B C C A B

A 0
B 10
C 110
D 111

A B C D A B: 0 10 110 111 0 10

0 10 110 0 110 10

A B C A C B

Tunstall Codes

- In Tunstall code, all the codewords are of equal length. However, each codeword represents a different number of letters.

Sequence	Codeword
AAA	00
AAB	01
AB	10
B	11

Tunstall code

- Design of an n bit Tunstall code for a source with an alphabet size of N
- Start with the N letters of the source alphabet in codebook
- Remove the entry in the codebook with highest probability and add the N string obtained by concatenating this letter with every letter in the alphabet (including itself)
- This increases the size of codebook from N to N+(N-1)
- Calculated the probabilities of the new entries
- Select the entry with the highest probability and repeat until the size of the code book reaches 2^n

Tunstall code

- Exp:
- S={A,B,C}, P(A)=0.6, P(B)=0.3, P(c)=0.1, n=3 bits

Sequence	P
A	0.6
B	0.3
C	0.1

Sequence	P
B	0.3
C	0.1
AA	0.36
AB	0.18
AC	0.06

Sequence	P
B	000
C	001
AB	010
AC	011
AAA	100
AAB	101
AAC	110