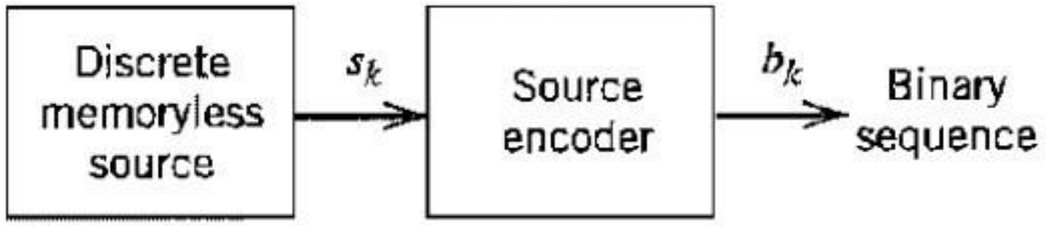
Communication Theory II

Lecture 7: Source Coding Theorem, Huffman Coding

Source Coding Theorem

- An important problem in communication is the efficient representation of data generated by a discrete source.
- The device that performs the representation is called a source encoder.

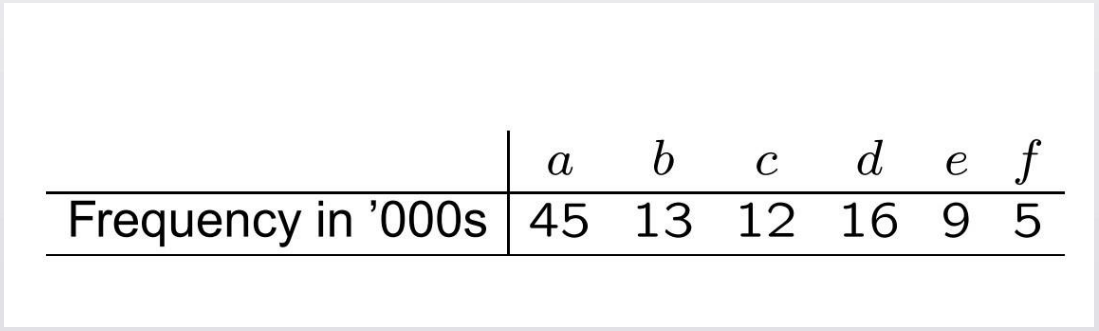


Source encoding.

- A binary code encodes each character as a binary string or codeword.
- We would like to find a binary code that encodes the file using as few bits as possible, ie., compresses it as much as possible.
- In a fixed-length code each codeword has the same length.
- In a variable-length code codewords may have different lengths.

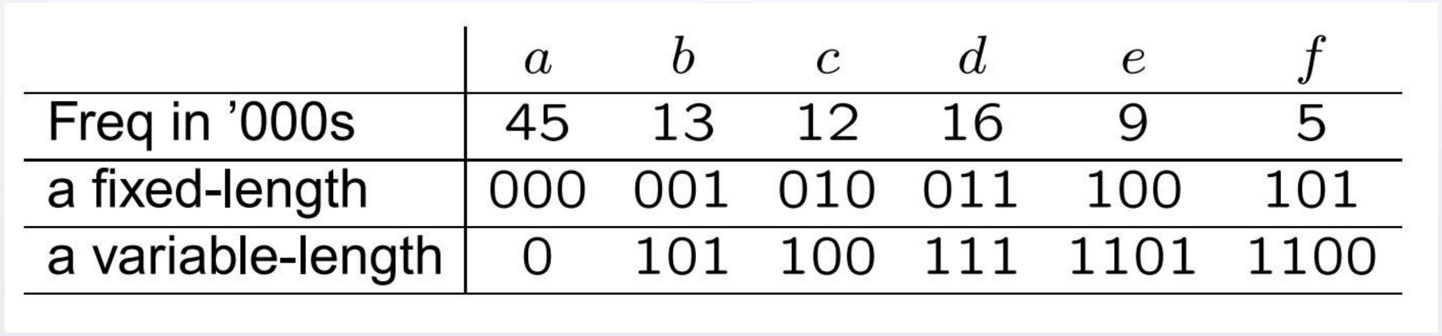
Example

- Suppose that we have a 100, 000 character data file that we wish to store . The file contains only 6 characters, appearing with the following frequencies:



	a	b	c	d	e	f
Frequency in '000s	45	13	12	16	9	5

Examples of fixed and variable length codes



	a	b	c	d	e	f
Freq in ’000s	45	13	12	16	9	5
a fixed-length	000	001	010	011	100	101
a variable-length	0	101	100	111	1101	1100

The fixed length-code requires 300, 000 bits to store the file.

The variable-length code uses only
$(45 \cdot 1+13 \cdot 3+12 \cdot 3+16 \cdot 3+9 \cdot 4+5 \cdot 4) \cdot 1000 = 224, 000$ bits, saving a lot of space!

Can we do better?

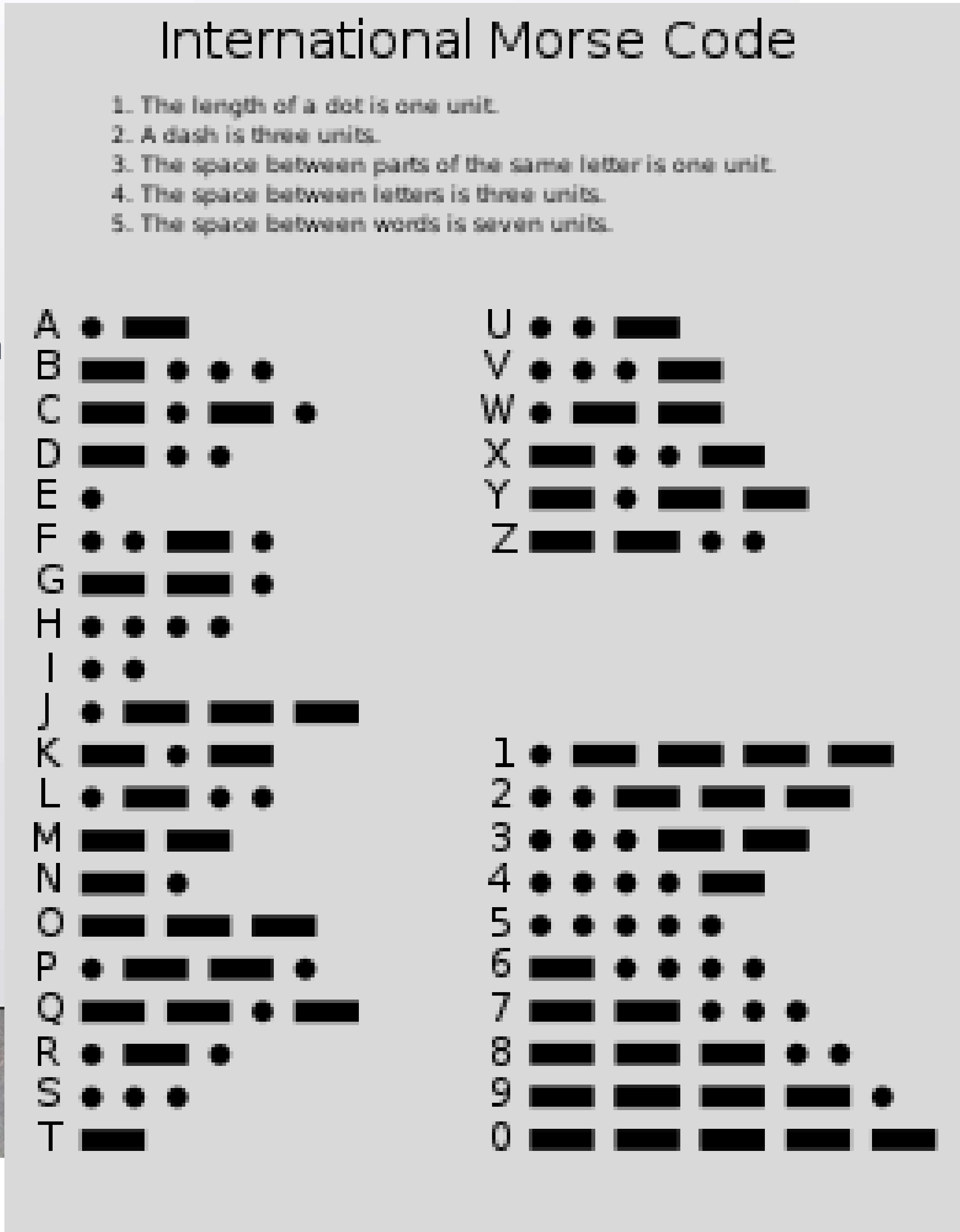
Code

- A code will be a set of codewords,
- e.g., {000, 001, 010, 011, 100, 101} and {0, 101, 100, 111, 1101, 1100}

Example

Morse code is a method of encoding text characters as sequences of two different signal durations, such as short and long signals, dots and dashes, or "dits" and "dahs." It was developed in the 1830s and 1840s by Samuel Morse and Alfred Vail for use with their electric telegraph system. In Morse code, each letter of the alphabet, as well as numbers and some punctuation marks, is represented by a unique combination of short and long signals. The duration of a short signal is typically referred to as a "dot" or a "dit," while the duration of a long signal is called a "dash" or a "dah." The pauses between signals within a letter are short, and the pauses between letters and words are longer.

More probable letters have short codes. E.g. letter E and letter Q, or letter A and letter J.



International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• ▬	U	• • ▬
B	▬ • • •	V	• • • ▬
C	▬ • ▬ •	W	• ▬ ▬
D	▬ • •	X	▬ • • ▬
E	•	Y	▬ • ▬ ▬
F	• • ▬ •	Z	▬ ▬ • •
G	▬ ▬ •		
H	• • • •		
I	• •		
J	• ▬ ▬ ▬		
K	▬ • ▬	1	• ▬ ▬ ▬ ▬
L	• ▬ • •	2	• • ▬ ▬ ▬
M	▬ ▬	3	• • • ▬ ▬
N	▬ •	4	• • • • ▬
O	▬ ▬ ▬	5	• • • • •
P	• ▬ ▬ •	6	▬ • • • •
Q	▬ ▬ • ▬	7	▬ ▬ • • •
R	• ▬ •	8	▬ ▬ ▬ • •
S	• • •	9	▬ ▬ ▬ ▬ •
T	▬	0	▬ ▬ ▬ ▬ ▬

[Morse code - Wikipedia]

Encoding

- replace the characters by the codewords.

Example: Γ = {a, b, c, d}

If the code is C1 = {a= 00, b = 01, c = 10, d = 11}.
then "**bad"** is encoded into **010011**

If the code is C3 = {a = 1, b = 110, c = 10, d = 111}
then "**bad"** is encoded into **1101111**

Decoding

- Given an encoded message, decoding is the process of turning it back into the original message

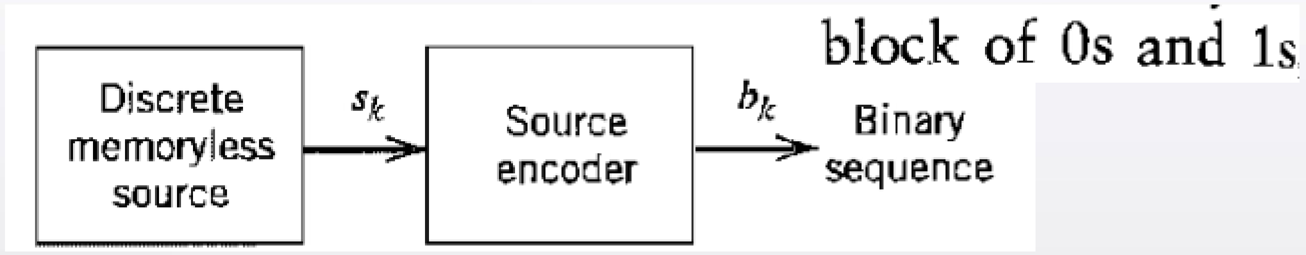
C1 = {a= 00, b = 01, c = 10, d = 11}.

For example relative to C1, **010011** is uniquely decodable to "**bad**".

C3 = {a = 1, b = 110, c = 10, d = 111}

But, relative to C3, **1101111** is not uniquely decipherable since it could have encoded either "**bad**" or "**acad**".

Codeword length and entropy



Average codeword length

$$\bar{L} = \sum_{k=0}^{K-1} p_k l_k$$

average number of bits per source symbol

Encoded length

Probability of S_k

Coding efficiency

Minimum possible value of $\bar{L}$

$$\eta = \frac{L_{\min}}{\bar{L}}$$

With $\bar{L} \geq L_{\min}$, we clearly have $\eta \leq 1$. The source encoder is said to be *efficient* when η approaches unity.

Given a discrete memoryless source of entropy $H(\mathcal{S})$, the average code-word length $\bar{L}$ for any distortionless source encoding scheme is bounded as

$$\bar{L} \geq H(\mathcal{S})$$

Codeword length and entropy

Shannon's source coding theorem

This means that the average number of bits per source symbol must be greater than or equal to the entropy of the source. It implies that to achieve efficient compression, the average number of bits per symbol should be close to the entropy value

entropy provides a theoretical lower bound on the average number of bits per symbol in an optimal coding scheme, and it serves as a fundamental measure of the information content of the source.

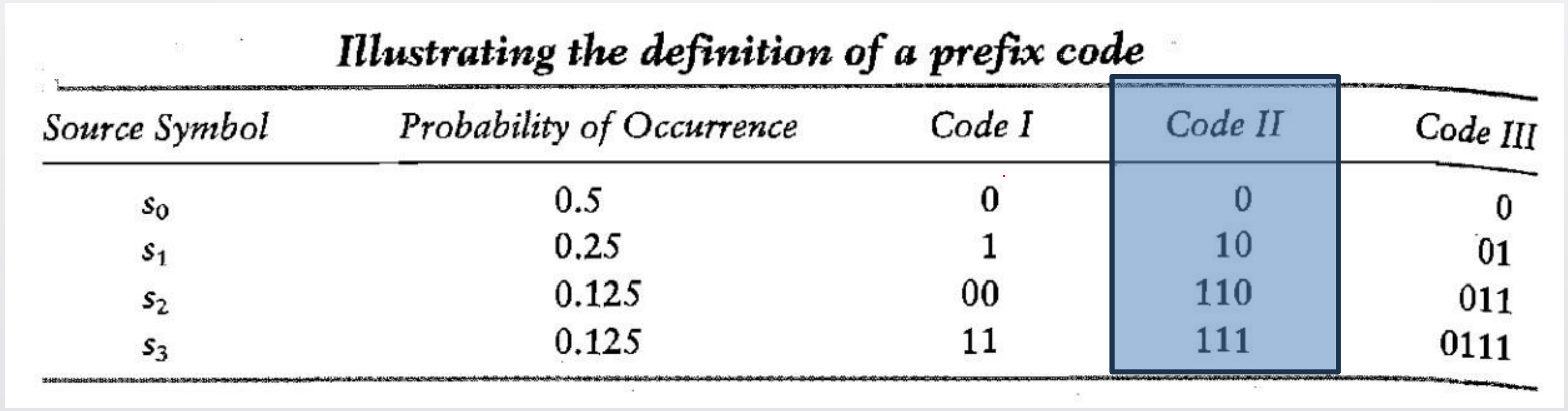
$$\eta = \frac{H(\mathcal{S})}{\overline{L}}$$

Data compression

- Data compression reduces data size while maintaining essential information.
- Entropy measures the average information or uncertainty in a source.
- Entropy indicates compressibility and redundancy within the data.
- The average number of bits per symbol should be close to entropy for efficient compression.
- Coding schemes assign shorter codes to more frequent symbols and longer codes to less frequent ones.
- Shannon's source coding theorem establishes a lower bound: average bits per symbol ≥ entropy.
- Practical compression algorithms like Huffman coding aim to achieve close-to-entropy compression.

Prefix Code

- A code is called a **prefix code** if no codeword is a prefix of any other code word.



Illustrating the definition of a prefix code

Source Symbol	Probability of Occurrence	Code I	Code II	Code III
s_0	0.5	0	0	0
s_1	0.25	1	10	01
s_2	0.125	00	110	011
s_3	0.125	11	111	0111

- A prefix code has the important property that it is always uniquely decodable.

In a prefix code, each symbol is represented by a unique binary codeword. The key characteristic of a prefix code is that no codeword is a prefix (initial segment) of another codeword. This property guarantees unambiguous decoding because when we encounter a sequence of bits during decoding, we can determine the corresponding symbol without the need for lookahead or further examination.

Prefix Code- example

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

If we encounter the bit sequence "10" during decoding, we can be certain that it corresponds to symbol B because "10" is a complete codeword. We don't need to look ahead or check for any additional bits.

Symbol A: 0 Symbol B: 10 Symbol C: 100

In this case, if we encounter the bit sequence "10," we can't determine the symbol yet because "10" could either represent symbol B or be the prefix of another codeword. The code lacks the prefix property, leading to ambiguity during decoding.

Prefix codes, such as Huffman coding, are widely used in various data compression applications because they provide efficient and reliable compression while ensuring clear and unambiguous decoding.

Optimum Source Coding Problem

The objective is to find a binary prefix code that **minimizes the average number of bits required to encode symbols from a given alphabet, based on their frequency distribution.**

- Start with the given alphabet A and its corresponding frequency distribution f(ai).
- Sort the symbols in A based on their frequencies in non-decreasing order. The symbol with the lowest frequency will have the longest code, and the symbol with the highest frequency will have the shortest code.

- The problem: Given an alphabet *A* = {a$_1$, . . . , a$_n$} with frequency distribution *f*(a$_i$) find a binary prefix code *C* for *A* that minimizes the number of bits

$$B(C) = \sum_{i=1}^{n} f(a_i) L(C(a_i))$$

needed to encode a message of $\sum_{i=1}^{n} f(a_i)$ characters, where $(C(a_i))$ is the codeword for encoding a_i , and $L(C(a_i))$ is the length of the codeword $C(a_i)$

Huffman Code

- Huffman developed a nice greedy algorithm for solving this problem and producing a minimum cost (optimum) prefix code. The code that it produces is called a Huffman code.

Huffman Code

Step 1: Pick two letters x, y from alphabet A with the smallest frequencies and create a subtree that has these two characters as leaves. (greedy idea)
Label the root of this subtree as z.

Step 2: Set frequency $f(z) = f(x) + f(y)$.
Remove x, y and add z creating new alphabet
$A' = A \cup \{z\} - \{x, y\}$.
Note that $|A'| = |A| - 1$.

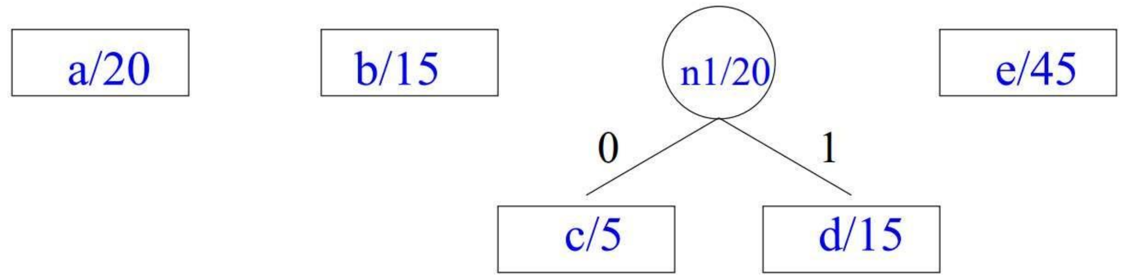
Repeat this procedure, called *merge*, with new alphabet A' until an alphabet with only one symbol is left.

The resulting tree is the Huffman code.

Example of Huffman Coding

Let $A = \{a/20, b/15, c/5, d/15, e/45\}$ be the alphabet and its frequency distribution.

In the first step Huffman coding merges c and d.

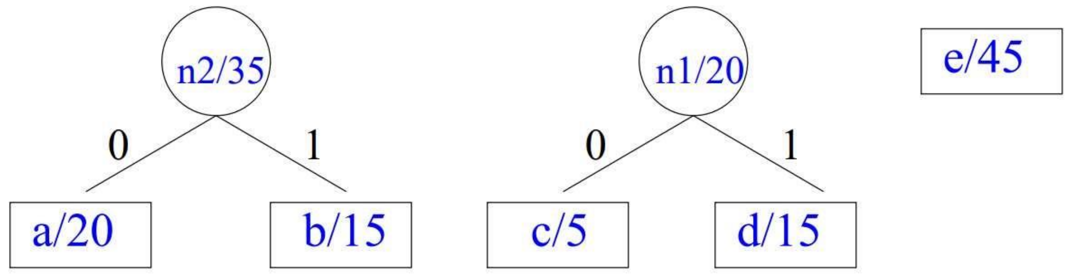


assigning '0' for the left branch and '1' for the right branch

Alphabet is now $A_1 = \{a/20, b/15, n1/20, e/45\}$.

Example of Huffman Coding – Continued

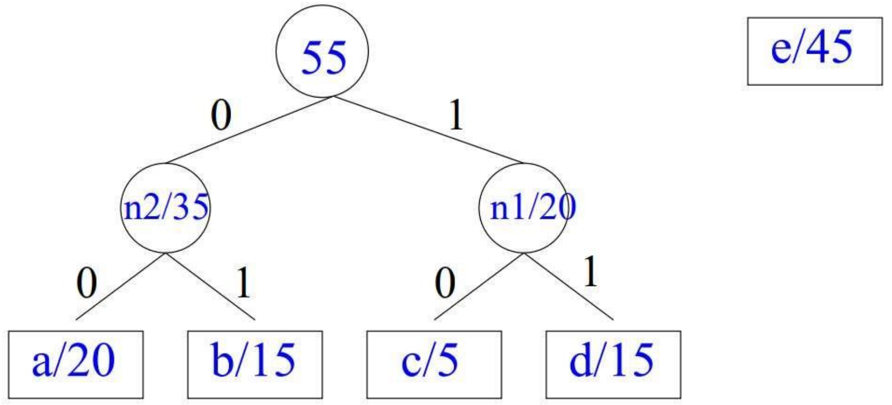
Alphabet is now $A_1 = \{a/20, b/15, n1/20, e/45\}$.
Algorithm merges a and b
(could also have merged $n1$ and b).



New alphabet is $A_2 = \{n2/35, n1/20, e/45\}$.

Example of Huffman Coding – Continued

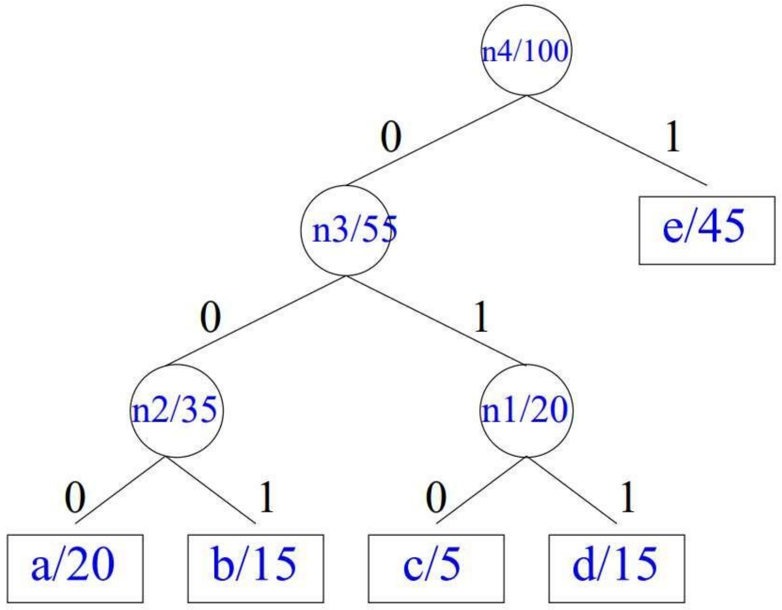
Alphabet is $A_2 = \{n2/35, n1/20, e/45\}$.
Algorithm merges $n1$ and $n2$.



New alphabet is $A_3 = \{n3/55, e/45\}$.

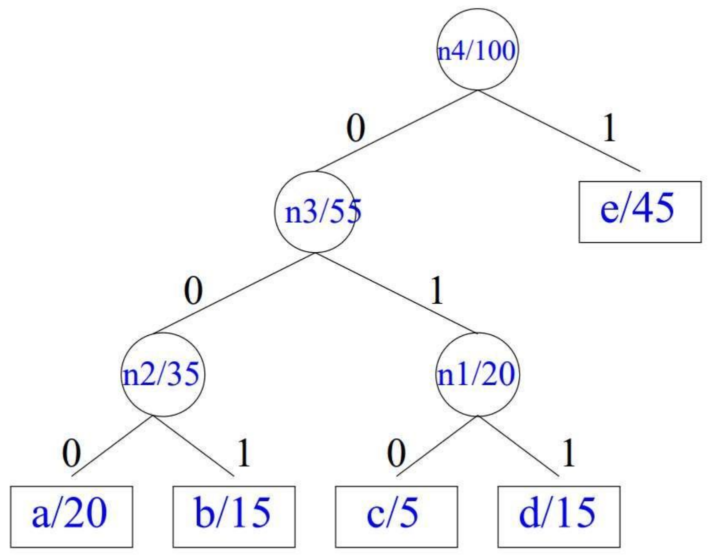
Example of Huffman Coding – Continued

Current alphabet is $A_3 = \{n3/55, e/45\}$.
Algorithm merges e and $n3$ and finishes.



Example of Huffman Coding – Continued

Huffman code is obtained from the Huffman tree.



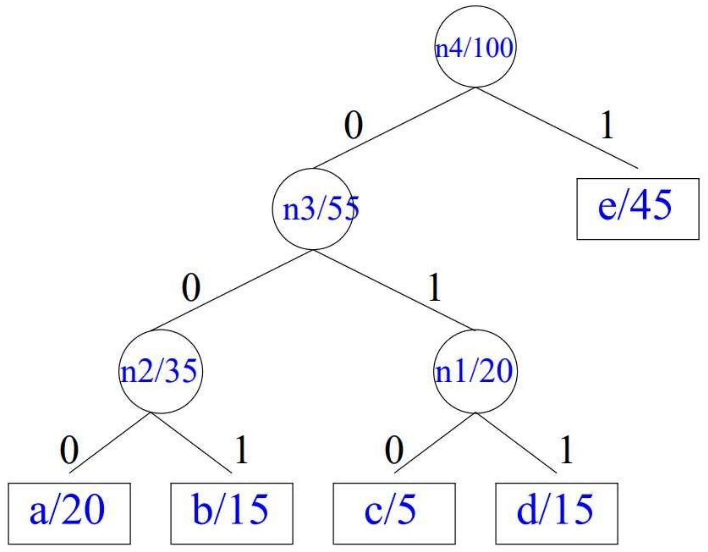
Huffman code is

$a = 000$, $b = 001$, $c = 010$, $d = 011$, $e = 1$.

This is the optimum (minimum-cost) prefix code for this distribution.

Example of Huffman Coding

Huffman code is obtained from the Huffman tree.



Huffman code is
$a = 000, b = 001, c = 010, d = 011, e = 1.$
This is the optimum (minimum-cost) prefix code for this distribution.

Calculate average code length and entropy

$$\bar{L} = \sum_{k=0}^{K-1} p_k l_k$$

Encoded length

Probability of S_k

$$H = \sum_1^N p_i log \frac{1}{p_i}$$

Huffman Coding

- Is coding unique?

  - The Huffman encoding process is not unique due to two variations in the process.

    The first variation is in the assignment of '0' and '1' to the last two source symbols during the splitting stage. However, the resulting differences are trivial.

    The second variation occurs when the probability of a combined symbol equals another probability in the list. Different placements can result in code words of different lengths, but the average code-word length remains the same.

Huffman Coding- variance of the average code-word length

Average codeword length

$$\sigma^2 = \sum_{k=0}^{K-1} p_k (l_k - \overline{L})^2$$

length of the code word

probability

• When a combined symbol is moved as high as possible during the Huffman coding process, the resulting Huffman code tends to have a significantly smaller variance compared to when it is moved as low as possible.

• Based on this observation, it is reasonable to choose the former Huffman code (combined symbol moved as high as possible) over the latter (combined symbol moved as low as possible) to reduce the variability in code-word lengths.

Huffman Coding

Weaknesses

- Data with uniform probabilities (equal frequencies), the overhead of storing the Huffman tree or codebook can outweigh the benefits of compression. In such cases, the compression achieved by Huffman coding might not be significant.

- Sensitivity to input distribution: The effectiveness of Huffman coding heavily depends on the frequency distribution of the input data. If the distribution changes significantly, the entire Huffman tree must be recomputed, which might not be practical in real-time scenarios or streaming data.

- Encoding and decoding complexity: Constructing the Huffman tree and encoding data can be computationally intensive, especially for large alphabets or data streams. While decoding is generally efficient, the encoding process requires traversing the tree for each character to obtain its corresponding code

Huffman Coding

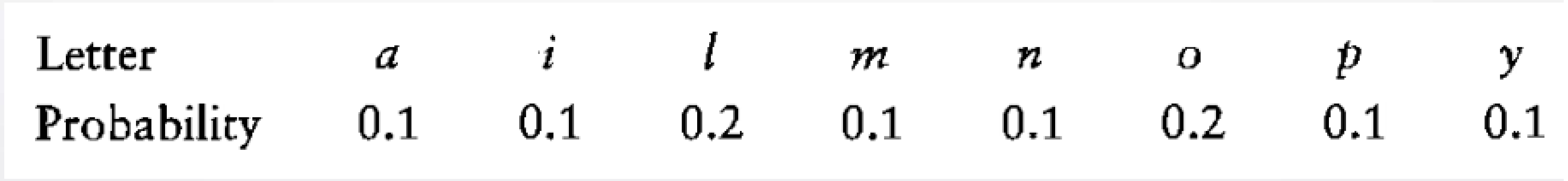
Suppose we have the following string of characters that we want to encode using Huffman code.

"ABRACADABRA"

What is the bit length ratio of Huffman code, to the following fixed-length (3 bits per character) coding?

A -> 000
B -> 001
C -> 010
D -> 011
R-> 100

Example



Letter	*a*	*i*	*l*	*m*	*n*	*o*	*p*	*y*
Probability	0.1	0.1	0.2	0.1	0.1	0.2	0.1	0.1

Compute two different Huffman codes for this alphabet. In one case, move a combined symbol in the coding procedure as high as possible, and in the second case, move it as low as possible. Hence, for each of the two codes, find the average code-word length and the variance of the average code-word length over the ensemble of letters.

Steps:-

1. Generate Huffman code
2. calculate average code length
3. calculate variance of the average code-word length

Example

Letter	S_0	S_1	S_2	S_3	S_4
Probability	0.55	0.15	0.15	0.1	0.05

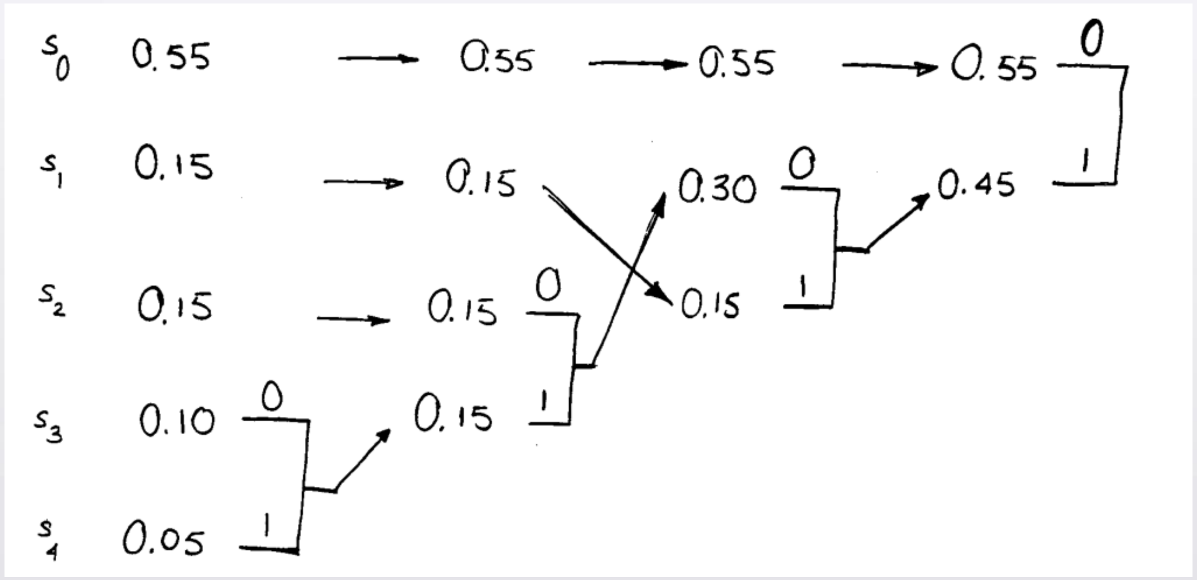
Compute two different Huffman codes for this alphabet. In one case, move a combined symbol in the coding procedure as high as possible, and in the second case, move it as low as possible. Hence, for each of the two codes, find the average code-word length and the variance of the average code-word length over the ensemble of letters.

Steps:-

1. Generate Huffman code
2. calculate average code length
3. calculate variance of the average code-word length

Example

1. Compute the Huffman code for this source, moving a "combined" symbol as low as possible.



s_0	0
s_1	1 1
s_2	1 0 0
s_3	1 0 1 0
s_4	1 0 1 1

The average code-word length is therefore

$$\bar{L} = \sum_{k=0}^{4} p_k l_k$$

$$= 0.55(1) + 0.15(2) + 0.15(3) + 0.1(4) + 0.05(4)$$

$$=1.9$$

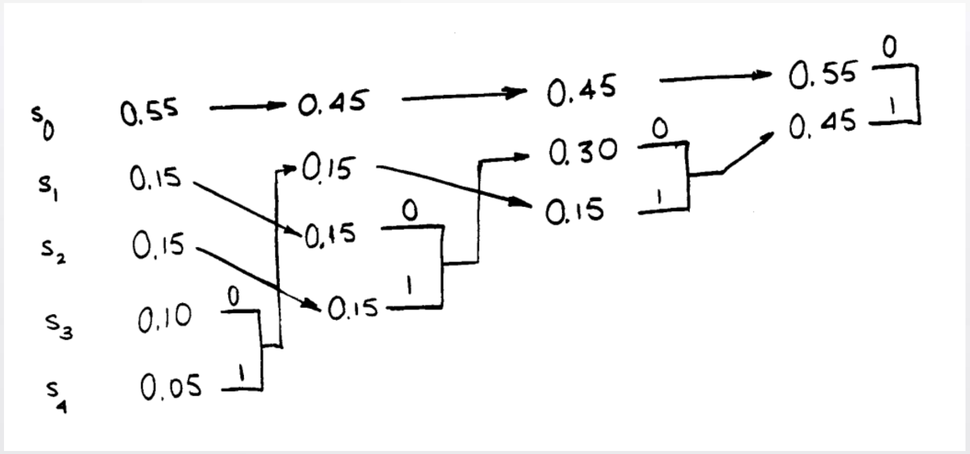
The variance of L is

$$\sigma^2 = \sum_{k=0}^{4} p_k (l_k - \bar{L})^2$$

$$= 0.55(-0.9)^2 + 0.15(0.1)^2 + 0.15(1.1)^2 + 0.1(2.1)^2 + 0.05(2.1)^2$$

$$= 1.29$$

1. Compute the Huffman code for this source, moving a "combined" symbol as high as possible.



s_0	0
s_1	1 0 0
s_2	1 0 1
s_3	1 1 0
s_4	1 1 1

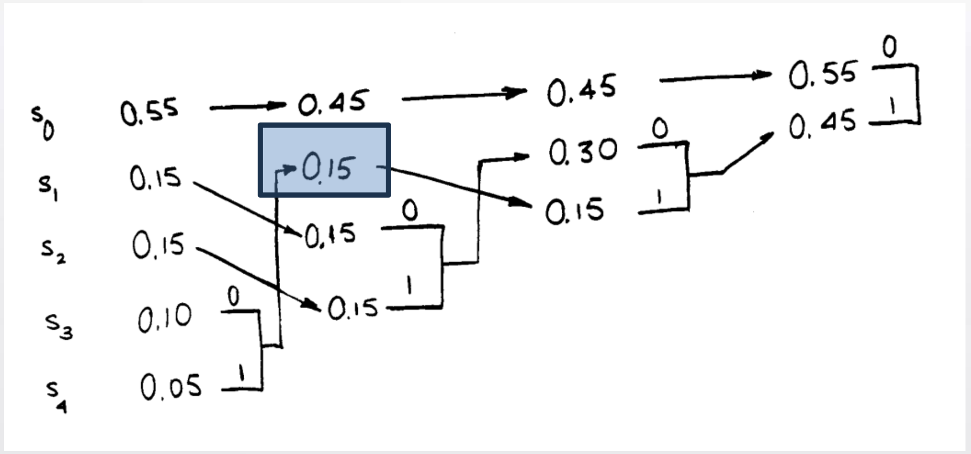
The average code-word length is

$$\bar{L} = 0.55(1) + (0.15 + 0.15 + 0.1 + 0.05)(3)$$
$$= 1.9$$

The variance of $\bar{L}$ is

$$\sigma^2 = 0.55(-0.9)^2 + (0.15 + 0.15 + 0.1 + 0.05)(1.1)^2$$
$$= 0.99$$

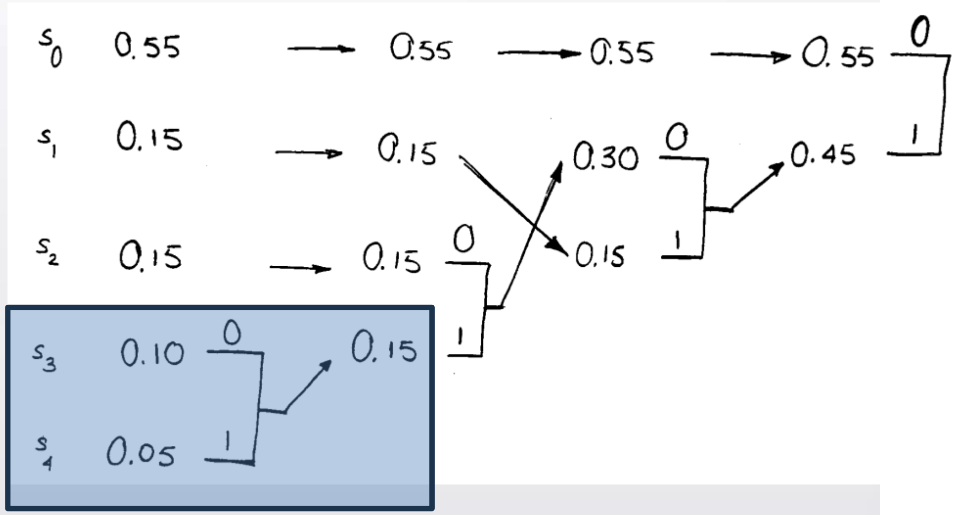
Comparison



s_0	0
s_1	1 0 0
s_2	1 0 1
s_3	1 1 0
s_4	1 1 1

Average code length = 1.9

variance of the average code-word length = 0.99



s_0	0
s_1	1 1
s_2	1 0 0
s_3	1 0 1 0
s_4	1 0 1 1

Average code length = 1.9

variance of the average code-word length = 1.29

Both has same average code length but different variance values.

Based on this observation, it is reasonable to choose the former Huffman code (combined symbol moved as high as possible) over the latter (combined symbol moved as low as possible) to reduce the variability in code-word lengths.

In Huffman coding, the tree construction process involves iteratively combining symbols with the lowest probabilities until all symbols are combined into a single tree. During this process, there is flexibility in the order of combining the symbols, which can result in different code-word lengths.

For the former Huffman code (combined symbol moved as high as possible):
When the combined symbol (formed by merging two least probable symbols) is placed higher up in the tree, it will have a shorter code length than if it were placed lower down. This is because, in a binary tree, the depth of the nodes determines the code length. Placing the combined symbol higher up means it will be closer to the root, resulting in a shorter code length.

For the latter Huffman code (combined symbol moved as low as possible):
When the combined symbol is placed lower down in the tree, it will have a longer code length compared to if it were placed higher up, as it will be farther from the root in the binary tree structure.

Since the former Huffman code places combined symbols higher up in the tree, it tends to result in shorter code lengths for the most probable symbols. This leads to a Huffman code with smaller variance in code-word lengths because the difference between the longest and shortest code lengths is minimized. Consequently, the average code-word length remains closer to the optimal value, resulting in a more efficient compression scheme.

In summary, choosing the former Huffman code with combined symbols moved as high as possible reduces the variability in code-word lengths, resulting in a more balanced and efficient encoding, compared to the latter Huffman code where combined symbols are moved as low as possible.

Example

A discrete memoryless source has an alphabet of seven symbols whose probabilities of occurrence are as described here:

Symbol	s_0	s_1	s_2	s_3	s_4	s_5	s_6
Probability	0.25	0.25	0.125	0.125	0.125	0.0625	0.0625

Compute the Huffman code for this source, moving a “combined” symbol as high as possible. Explain why the computed source code has an efficiency of 100 percent