AIM

This experiment aims to test the C++ code for the Shannon's encoder and decoder and verify the output with the manual calculation.

INTRODUCTION

Shannon coding, named after Claude Shannon, is a fundamental concept in information theory, often used in data compression. Shannon's pioneering work in 1948 laid the groundwork for coding techniques that could effectively reduce the number of bits needed to represent information without loss.

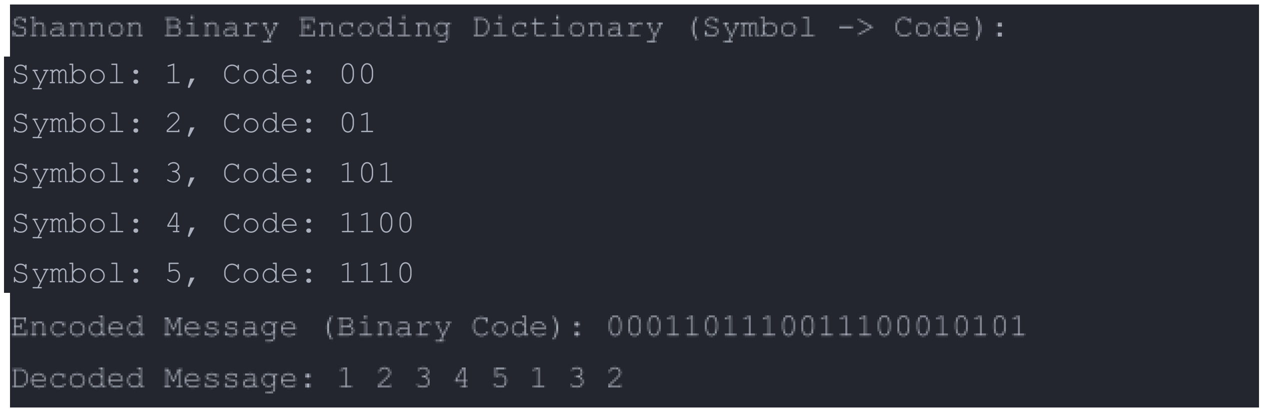
This coding method relies on the probability of occurrence of symbols in a dataset. Shannon coding aims to assign shorter codes to more frequent symbols and longer codes to less frequent ones. This process reduces the overall bitlength of a message, optimizing storage and transmission efficiency.

In Shannon coding, a symbol's code length is determined by the symbol's probability, as higher-probability symbols require fewer bits, and lower-probability symbols require more bits. The entropy, or average amount of information per symbol, sets the theoretical limit of the minimum average code length required for encoding a source. Shannon coding is often compared with other compression algorithms, such as Huffman coding, which builds on Shannon's theory to create more efficient coding schemes.

C++ PROGRAM

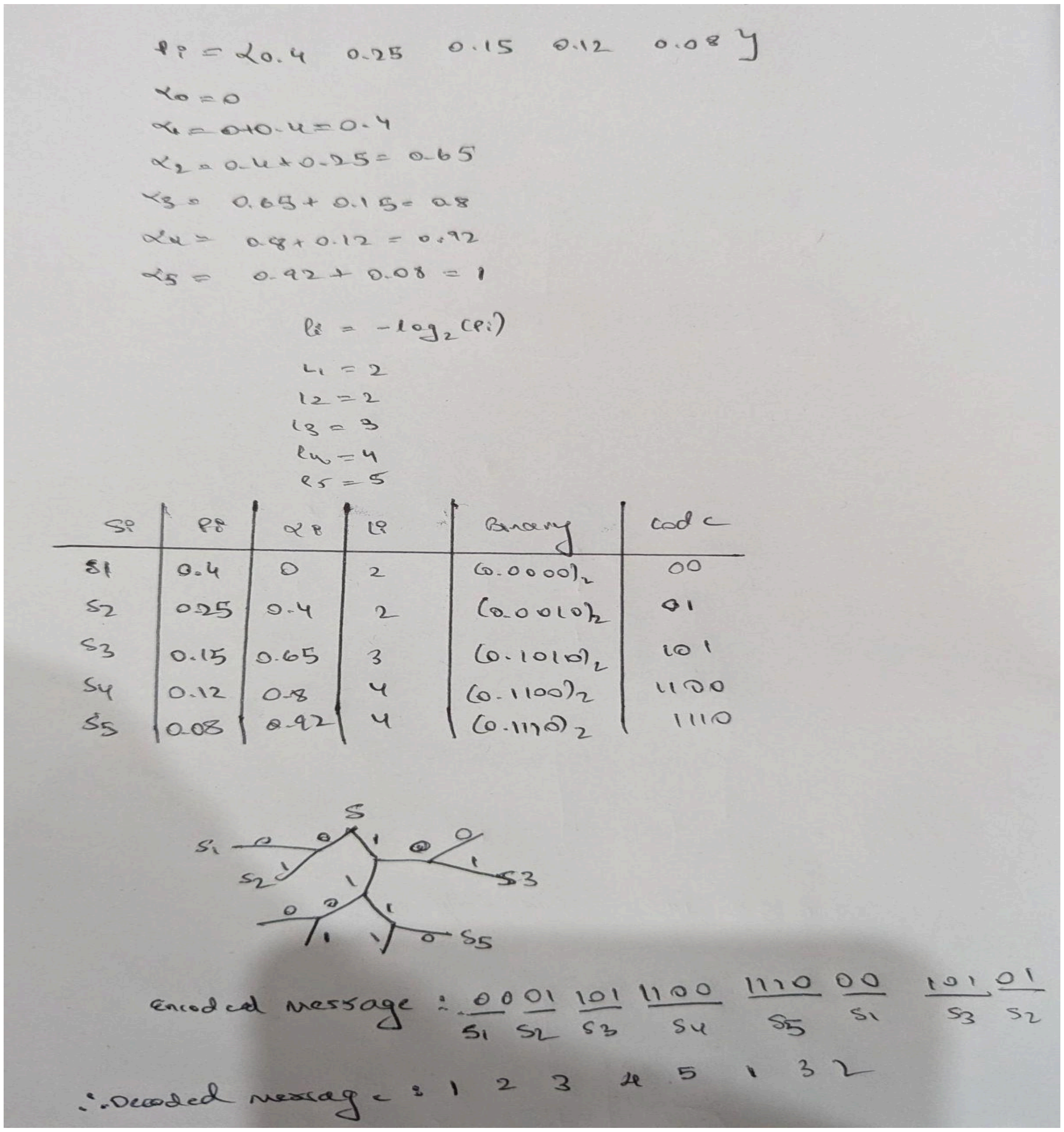
```
#include
<bits/stdc++.h> using
namespace std;

// Function to calculate cumulative probabilities
vector<double> calculateCumulativeProbabilities(vector<double> probabilities) {
    vector<double> cumulative_probabilities(probabilities.size(), 0);
    cumulative_probabilities[0] = 0; // First element starts at 0
    for (size_t i = 1; i < probabilities.size(); i++) {
        cumulative_probabilities[i] = cumulative_probabilities[i - 1] + probabilities[i -
1];
    }
    return cumulative_probabilities;
}

// Function to convert fractional cumulative probability to binary
string convertToBinary(double cumulative_prob, int code_length) {
    string binary_code = "";
    double frac = cumulative_prob;
    for (int i = 0; i < code_length; i++) {
        frac *= 2;
        if (frac >= 1) {
            binary_code += '1';
            frac -= 1;
        } else {
            binary_code += '0';
        }
    }
    return binary_code;
}

int main() {
    // Step 1: Define symbols and their probabilities
    vector<int> symbols = {1, 2, 3, 4, 5};
    vector<double> probabilities = {0.4, 0.25, 0.15, 0.12, 0.08};


    // Step 2: Sort symbols and probabilities in descending order
    vector<pair<double, int>> symbol_prob_pairs;
    for (size_t i = 0; i < symbols.size(); i++) {
        symbol_prob_pairs.push_back(make_pair(probabilities[i], symbols[i]))

    }
    sort(symbol_prob_pairs.rbegin(), symbol_prob_pairs.rend());
    for (size_t i = 0; i < symbols.size(); i++) {
        probabilities[i] = symbol_prob_pairs[i].first;
        symbols[i] = symbol_prob_pairs[i].second;
    }

    // Step 3: Calculate cumulative probabilities
    vector<double> cumulative_prob =
calculateCumulativeProbabilities(probabilities);

    // Step 4: Generate Shannon binary codes
    map<int, string> shannonDict;
    for (size_t i = 0; i < symbols.size(); i++) {
        int code_length = ceil(-log2(probabilities[i])); // Code length = -log2(p_i)
        string binary_code = convertToBinary(cumulative_prob[i], code_length);
        shannonDict[symbols[i]] = binary_code;
    }

    // Display Shannon binary encoding dictionary
    cout << "Shannon Binary Encoding Dictionary (Symbol -> Code):\n";
    for (const auto& entry : shannonDict) {
        cout << "Symbol: " << entry.first << ", Code: " << entry.second << endl;
    }

    // Step 5: Encode a message using Shannon binary encoding
    vector<int> message = {1, 2, 3, 4, 5, 1, 3, 2};
    string encodedMessage = "";
    for (int symbol : message) {
        encodedMessage += shannonDict[symbol];
    }

    cout << "Encoded Message (Binary Code): " << encodedMessage << endl;

    // Step 6: Decode the binary code back to the original message
    vector<int> decodedMessage;
    string temp = "";
    for (char bit : encodedMessage) {
        temp += bit;
        for (const auto& entry : shannonDict) {
            if (entry.second == temp) {
                decodedMessage.push_back(entry.first);
                temp = ""; // Reset temp for the next symbol
                break;
            }
            }
        }

        // Display the decoded message
        cout << "Decoded Message: ";
        for (int symbol : decodedMessage) {
            cout << symbol << " ";
        }
        cout << endl;

        return 0;
    }
```

SIMULATION OUTPUT



```
Shannon Binary Encoding Dictionary (Symbol -> Code):
Symbol: 1, Code: 00
Symbol: 2, Code: 01
Symbol: 3, Code: 101
Symbol: 4, Code: 1100
Symbol: 5, Code: 1110
Encoded Message (Binary Code): 00011011100111000010101
Decoded Message: 1 2 3 4 5 1 3 2
```

MANUAL CALCULATION



$P_i = \{0.4 \quad 0.25 \quad 0.15 \quad 0.12 \quad 0.08\}$

$\alpha_0 = 0$

$\alpha_1 = 0+0.4 = 0.4$

$\alpha_2 = 0.4+0.25 = 0.65$

$\alpha_3 = 0.65+0.15 = 0.8$

$\alpha_4 = 0.8+0.12 = 0.92$

$\alpha_5 = 0.92+0.08 = 1$

$l_i = -\log_2(P_i)$

$l_1 = 2$

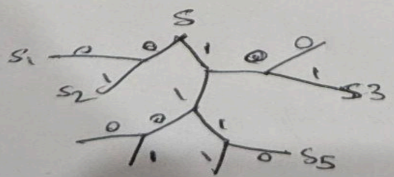
$l_2 = 2$

$l_3 = 3$

$l_4 = 4$

$l_5 = 5$

S_i	P_i	α_i	l_i	Binary	code
S1	0.4	0	2	$(0.0000)_2$	00
S2	0.25	0.4	2	$(0.0010)_2$	01
S3	0.15	0.65	3	$(0.1010)_2$	101
S4	0.12	0.8	4	$(0.1100)_2$	1100
S5	0.08	0.92	4	$(0.1110)_2$	1110



Encoded message : $\underbrace{00}_{S1}\ \underbrace{01}_{S2}\ \underbrace{101}_{S3}\ \underbrace{1100}_{S4}\ \underbrace{1110}_{S5}\ \underbrace{00}_{S1}\ \underbrace{101}_{S3}\ \underbrace{01}_{S2}$

$\therefore$ Decoded message : 1 2 3 4 5 1 3 2

RESULTS & DISCUSSION

The implementation of Shannon's encoder and decoder in C++ effectively demonstrates how entropy-based coding can achieve efficient data compression. By assigning shorter binary codes to more frequent symbols and longer codes to less common ones, the encoder optimizes the overall message length. The encoded message length closely aligns with the source's entropy, reducing redundancy while retaining all information.

The decoder ensures the integrity of the process by accurately reconstructing the original message, validating the reliability of the prefix-free codes generated. A comparison between the original input size and the compressed message length highlights significant space savings, underscoring the suitability of this method for storage and transmission. Additionally, the implementation emphasizes the critical role of accurate probability distributions in achieving optimal compression, as deviations in these probabilities can impact encoding efficiency.

APPLICATION

Shannon coding, based on the principles of entropy and information theory, finds applications in various fields where efficient data compression and transmission are essential. Some key applications include:

1. Data Compression

File Compression: Used in algorithms for compressing text files, such as in utilities like ZIP or RAR, to minimize storage space.

Multimedia Compression: Forms the theoretical foundation for audio (e.g., MP3) and video (e.g., MPEG) compression techniques, reducing file sizes without significant loss of quality.

2. Communication Systems

Error-Free Data Transmission: Helps in encoding messages for transmission over networks or channels, ensuring minimal redundancy while preserving information.

Channel Capacity Analysis: Assists in determining the maximum data rate that can be achieved over a communication channel without errors.

3. Storage Systems

Efficient Data Storage: Enables optimal use of storage resources by reducing the amount of data to be saved, applicable in databases and large-scale data centers.

4. Cryptography and Security

Key Compression: Shannon's principles aid in minimizing the size of cryptographic keys while maintaining their security properties.

Data Masking: Ensures efficient coding schemes for secure communication and obfuscation.

5. Machine Learning and Artificial Intelligence

Feature Encoding: Assists in compressing and encoding features for models, especially in natural language processing and image recognition tasks.

Data Preprocessing: Provides theoretical insights into lossless compression of datasets for training.

CONCLUSION

The implementation of Shannon's encoder and decoder in C++ demonstrates the practicality of entropy-based compression techniques. By allocating code lengths proportional to the probabilities of symbols, the approach effectively reduces redundancy, offering a compact representation of data while preserving accuracy during decoding. The results confirm that Shannon coding achieves near-optimal compression by approaching the theoretical minimum code length for a given dataset.

Although the method's efficiency depends on precise probability distributions, it represents a pivotal advancement in the field of data compression. This project highlights the dual significance of Shannon coding in both theoretical exploration and practical application, providing a robust framework for developing more advanced algorithms tailored for data storage and transmission.

Aim:

Write a C program for the generation of Pseudo Noise (PN) Sequence using Shift register.

Introduction:

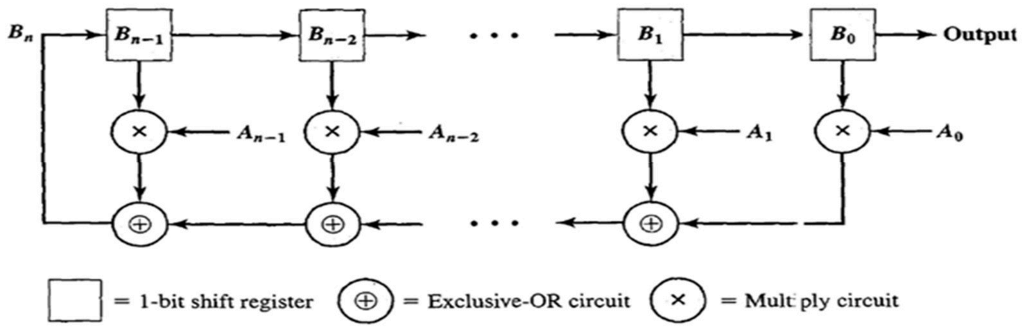
A pseudo-noise (PN) or pseudorandom sequence is a binary sequence with an autocorrelation that resembles, over a period, the autocorrelation of a random binary sequence. Its autocorrelation also roughly resembles the autocorrelation of bandlimited white noise. Although it is deterministic a pseudo noise sequence has many characteristics that are similar to those of random binary sequences, such as having a nearly equal number of 0s and 1s, very low correlation between shifted versions of the sequence, very low cross correlation between any two sequences, etc. The PN sequence is usually generated using sequential logic circuits. A feedback shift register, consists of consecutive stages of two state memory devices and feedback logic. Binary sequences are shifted through the shift registers in response to clock pulses, and the output of the various stages are logically combined and fed back as the input to the first-stage. ‘1,y-hen the feedback logic consists of exclusive-OR gates, which is usually the case, the shift register is called a linear PN sequence generator.

PN Properties:

The sequence of numbers be random in some well-defined statistical sense. The following two criteria are used to validate that a sequence of numbers is random:

- **Uniform distribution:** The distribution of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same.
- **Correlation property:** If a period of the sequence is compared term by term with any cycle shift of itself, the number of terms that are the same differs from those that are different by at most 1

General Block diagram of PN sequence generator:

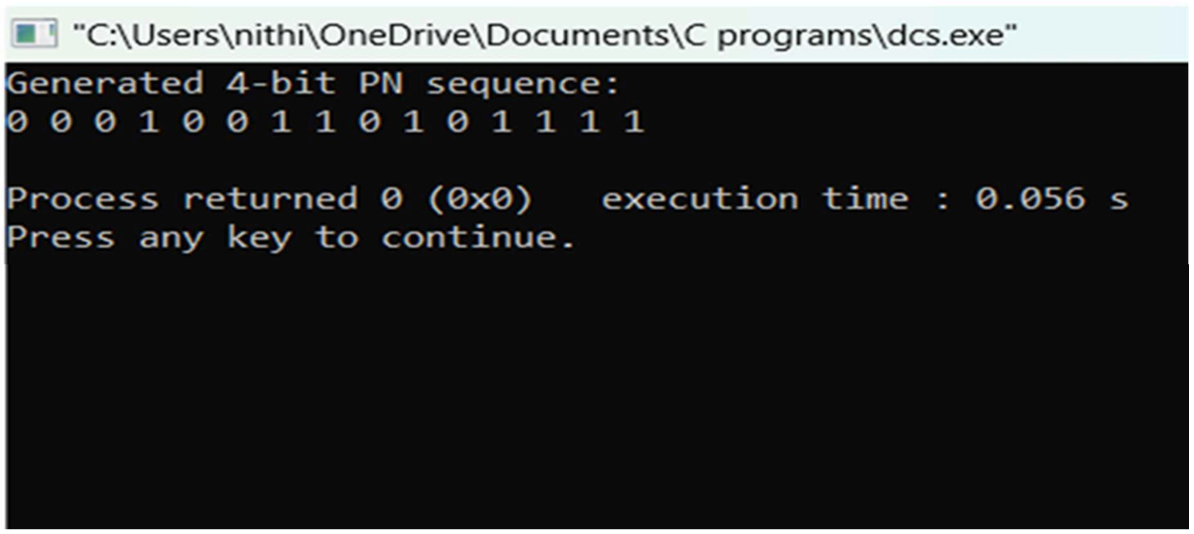


C Program:

```
#include<stdio.h>
int main (){
int n =4;
int L = (1 << n) - 1;
   int shift_register[4] = {1, 0, 0, 0};
   int pn_sequence[L];
   int feedback_taps[4] = {0, 0, 1, 1};
   printf("Generated 4-bit PN sequence:\n");
   for (int i = 0; i < L; i++) {
      pn_sequence[i] = shift_register[3];
      printf("%d ", pn_sequence[i]);
      int feedback = 0;
      for (int j = 0; j < n; j++) {
         feedback ^= (shift_register[j] * feedback_taps[j]);
      }
      for (int j = n - 1; j > 0; j--) {
         shift_register[j] = shift_register[j - 1];
      }
      shift_register[0] = feedback;
```

```
    }
    printf("\n");
    return 0;
}
```

Simulation Output:

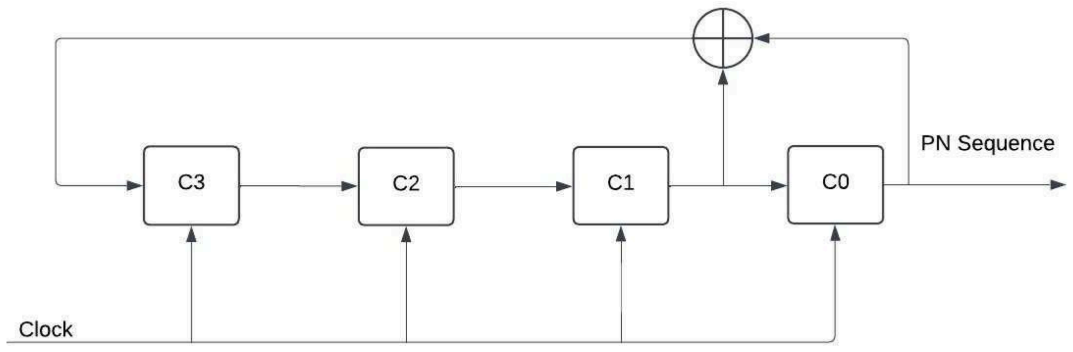


"C:\Users\nithi\OneDrive\Documents\C programs\dcs.exe"

```
Generated 4-bit PN sequence:
0 0 0 1 0 0 1 1 0 1 0 1 1 1 1

Process returned 0 (0x0)   execution time : 0.056 s
Press any key to continue.
```

Manual Calculation:

Block Diagram:



Truth Table:

Initial state = 1000

Polynomial = $x^2 + x^3$ [0011]

n = 4

L= 2^n-1 = 15

State	C3	C2	C1	C0	PN output
Initial =0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	1	0	0	1	1
4	1	1	0	0	0
5	0	1	1	0	0
6	1	0	1	1	1
7	0	1	0	1	1
8	1	0	1	0	0
9	1	1	0	1	1
10	1	1	1	0	0
11	1	1	1	1	1
12	0	1	1	1	1
13	0	0	1	1	1
14	0	0	0	1	1
15=0	1	0	0	0	0

PN output = 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1

Results and discussion:

PN output = 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1

Discussion:

The PN (Pseudo-Noise) sequence generated by the above code is a maximal-length sequence using a 4-bit Linear Feedback Shift Register (LFSR). This sequence exhibits key properties like pseudo- randomness, balance, and periodicity, making it essential for applications in spread-spectrum communication, cryptography, and error correction. The LFSR operates by shifting bits through a register and generating a feedback bit based on the XOR of specific tapped positions. The taps [0, 0, 1, 1] ensure the sequence length is 2n−1=152^n - 1 = 15, cycling through all possible non-zero states. The code's logic correctly implements the feedback and shift operations, producing a deterministic yet random-like sequence. The PN sequence's balance and run-length properties demonstrate its suitability for encoding and spreading signals in communication systems.

Advantages:

- Pseudo-Randomness
- Interference Resistance
- Privacy and Security
- Low Cross-Correlation
- Efficient Bandwidth Utilization
- High Noise Immunity

Applications:

- Used in Direct Sequence Spread Spectrum (DSSS)
- Used in Frequency Hopping Spread Spectrum (FHSS)
- Used for pulse compression and to distinguish between signals in high-resolution radar systems
- Used for Error Detection and Correction
- Used in Random Number Generation

Conclusion:

The above code successfully implements a 4-stage Linear Feedback Shift Register (LFSR) to generate a 4-bit PN (Pseudo-Noise) sequence with a length of 2^n - 1 = 15. It accurately demonstrates the principles of feedback and shifting, with feedback calculated from specific tapped positions to ensure a maximal-length sequence. The generated PN sequence exhibits pseudo-randomness, balance, and periodicity, which are essential for applications in communication systems, cryptography, and signal processing. This implementation highlights the efficiency of LFSRs in generating deterministic yet random-like sequences, showcasing their practical importance in modern engineering and technology.