|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Information Theory & Coding Techniques** | **Semester:** | **V** |
| **Date of Performance:** | **30 / 07 / 2024** | **Batch:** | **B - 1** |
| **Faculty Name:** | **Prof. Makarand Kulkarni** | **Roll Number:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_ / 25** |

**Experiment No.: 2**

**Title:** **To implement Shannon-Fano algorithm using MATLAB**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| * To develop the Shannon-Fano algorithm using MATLAB. * To validate the mathematical results with the simulated results obtained from this MATLAB code. |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Use basic concept of probability theory, information theory and source coding in communication. |

|  |
| --- |
| **Theory:** |
| Data Compression, also known as source coding, is the process of encoding or converting data in such a way that it consumes less memory space. Data compression reduces the number of resources required to store and transmit data. It can be done in two ways- lossless compression and lossy compression. Lossy compression reduces the size of data by removing unnecessary information, while there is no data loss in lossless compression.  Shannon Fano Algorithm is an entropy encoding technique for lossless data compression of multimedia. Named after Claude Shannon and Robert Fano, it assigns a code to each symbol based on their probabilities of occurrence. It is a variable length encoding scheme, that is, the codes assigned to the symbols will be of varying length. |

|  |
| --- |
| **Stepwise – Procedure:** |
| **Steps of algorithm –**   1. Create a list of probabilities or frequency counts for the given set of symbols so that the relative frequency of occurrence of each symbol is known. 2. Sort the list of symbols in decreasing order of probability, the most probable ones to the left and least probable to the right. 3. Split the list into two parts, with the total probability of both the parts being as close to each other as possible. 4. Assign the value 0 to the left part and 1 to the right part. 5. Repeat the steps 3 and 4 for each part, until all the symbols are split into individual subgroups. |

|  |
| --- |
| **Observations:** |
| **SIMULATION –**   1. **Code of Shannon-Fano:**   import math  def shannon\_fano(symbols, probabilities):      sorted\_indices = sorted(range(len(probabilities)), key=lambda i: probabilities[i], reverse=True)      sorted\_symbols = [symbols[i] for i in sorted\_indices]      sorted\_probabilities = [probabilities[i] for i in sorted\_indices]      codes = [''] \* len(symbols)      def assign\_codes(start, end):          if start == end:              return          total = sum(sorted\_probabilities[start:end+1])          cumulative = 0          split = start          for i in range(start, end+1):              cumulative += sorted\_probabilities[i]              if cumulative >= total / 2:                  split = i                  break          for i in range(start, split+1):              codes[sorted\_indices[i]] += '0'          for i in range(split+1, end+1):              codes[sorted\_indices[i]] += '1'          assign\_codes(start, split)          assign\_codes(split+1, end)      assign\_codes(0, len(symbols) - 1)      codebook = {symbols[i]: codes[i] for i in range(len(symbols))}      return codebook  def calculate\_entropy(probabilities):      entropy = -sum(p \* math.log2(p) for p in probabilities if p > 0)      return entropy  def calculate\_average\_codeword\_length(codebook, probabilities, symbols):      avg\_length = 0      for i, symbol in enumerate(symbols):          avg\_length += probabilities[i] \* len(codebook[symbol])      return avg\_length  num\_messages = int(input("Enter the number of symbols: "))  symbols = []  probabilities = []  for i in range(num\_messages):      symbol = input(f"Enter symbol {i+1}: ")      probability = float(input(f"Enter probability for {symbol}: "))      symbols.append(symbol)      probabilities.append(probability)  if abs(sum(probabilities) - 1.0) > 1e-6:      print("Warning: Probabilities do not sum to 1. Please check your input.")  codebook = shannon\_fano(symbols, probabilities)  entropy = calculate\_entropy(probabilities)  avg\_codeword\_length = calculate\_average\_codeword\_length(codebook, probabilities, symbols)  print("\nShannon-Fano Codes:")  for symbol in codebook:      code\_length = len(codebook[symbol])      print(f"Symbol: {symbol}, Code: {codebook[symbol]}, Number of Bits: {code\_length}")  print(f"\nEntropy of the source: {entropy:.4f} bits")  print(f"Average codeword length: {avg\_codeword\_length:.4f} bits")  code\_efficiency = (entropy/avg\_codeword\_length)\*100  print(f"\nCode Efficiency: {code\_efficiency}")   1. **Output of Code:**      1. **Calculations:** |

|  |
| --- |
| **Post Lab Subjective Type Questions:** |
| 1. **What is the concept of source coding? What are the different techniques in it?**   Source coding refers to the process of efficiently representing information (data) from a source. This involves compressing the data to reduce redundancy and to store or transmit it more efficiently. The main goal is to minimize the number of bits used to represent the data without losing any information (lossless compression) or by allowing some loss of information (lossy compression).  Techniques in Source Coding:   1. **Huffman Coding:**   Description: A variable-length coding technique where shorter codes are assigned to more frequent symbols and longer codes to less frequent symbols.  Application: Widely used in file compression formats such as ZIP and in multimedia codecs.   1. **Run-Length Encoding (RLE):**   Description: A simple form of compression where sequences of the same data value are stored as a single data value and count.  Application: Useful for data with many repeated elements, such as simple graphic images.   1. **Arithmetic Coding:**   Description: A method that represents the entire message as a single number, a fraction n, which lies between 0 and 1.  Application: Used in applications requiring high compression efficiency, like video and image compression standards (JPEG, H.264).   1. **Lempel-Ziv-Welch (LZW) Coding:**   Description: A dictionary-based compression technique that replaces repeated occurrences of data with references to a dictionary containing the data.  Application: Used in GIF images and some implementations of the ZIP file format.   1. **Shannon-Fano Coding:**   Description: A top-down approach where the source symbols are arranged in order of frequency, and a binary tree is constructed based on cumulative probabilities.  Application: One of the early algorithms used for lossless data compression.   1. **Delta Encoding:**   Description: Stores the difference between consecutive data points rather than the actual values.  Application: Used in situations where data points are close in value, such as time-series data.   1. **Compare the different source coding techniques.**  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Technique** | **Compression Ratio** | **Complexity** | **Lossless/Lossy** | **Applications** | | **Huffman Coding** | Moderate | Moderate | Lossless | File compression (ZIP), multimedia codecs | | **Run-Length Encoding** | Low | Low | Lossless | Simple graphics, fax machine | | **Arithmetic Coding** | High | High | Lossless | Video and image compression (JPEG) | | **LZW Coding** | Moderate | Moderate | Lossless | GIF images, ZIP file format | | **Shannon-Fano Coding** | Moderate | Moderate | Lossless | Early data compression applications | | **Delta Encoding** | Variable | Low | Lossless | Time-series data, video compression |   kk |

|  |
| --- |
| **Conclusion:** |
| The Shannon-Fano algorithm was successfully developed using MATLAB, and the simulated results closely matched the theoretical expectations, validating the accuracy and effectiveness of the implementation. This confirms the reliability of the algorithm for efficient source coding in practical applications. |

**Signature of faculty in-charge with date:**