ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO

Lý Thuyết Thông Tin

Sinh viên thực hiện: Nguyễn Quang Huy Hoàng -20207605
Hoàng Hà My -20207644

Ngành Công nghệ thông tin và truyền thông

Giảng viên hướng dẫn: TS. Trịnh Văn Chiến

Khoa: Kỹ thuật máy tính

Trường: Công nghệ thông tin và Truyền thông

HÀ NỘI, 07/2023

Đề Bài

Bài 1:

a) Nhập vào ma trận xác suất kết hợp P(x,y) cỡ MxN với M và N nhập từ bàn phím. Cảnh báo nếu nhập số âm, yêu cầu nhập lại.

b) Tính và hiển thị H(X), H(Y), H(X|Y), H(Y|X), H(X,Y), H(Y)-H(Y|X), I(X;Y)

c) Tính D(P(x)|P(y)) và D(P(y)|P(x))

Bài 2:

a) Nhập vào một chuỗi ký tự không dấu có chiều dài bất kỳ, không phân biệt chữ hoa, chữ thường.

b) Mã hóa Huffman cho chuỗi trên.

c) Mã hóa Shannon-Fano cho chuỗi trên, tính hiệu suất mã hóa, và tính dư thừa.

Tóm tắt công thức bài 1

Entropy (Lượng tin riêng:

$$H(X) = -\sum_{x \in X} p(x) * \log_2(x) \tag{1}$$

Join Entropy (Lượng tin chung):

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) * \log_2 p(x,y) \tag{2}$$

Conditional probability (Xác suất có điều kiện):

$$p(Y|X) = \frac{p(X,Y)}{p(X)} \tag{3}$$

Conditional entropy (Entropy có điều kiện):

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) * \log_2 p(y|x) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) * log_2 \frac{p(x,y)}{p(x)} \tag{4}$$

Mutual information(thông tin tương hỗ):

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) * \log_2 \frac{p(x,y)}{p(x)p(y)} = H(Y) - H(Y|X) \tag{5}$$

Relative entropy (entropy tương đối):

$$D(p||q) = \sum_{x \in X} p(x) * \log_2 \frac{p(x)}{q(x)} \tag{6}$$

Code bài 1:

```
import java.util.Scanner;

public class Bai1 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      //Nhập kích thước ma trận xác suất
      System.out.print("Nhập số hàng (M): ");
      int M = sc.nextInt();
      System.out.print("Nhập số cột (N): ");
      int N = sc.nextInt();

      //Khởi tạo ma trận xác suất
      double[][] probability_Matrix = new double[M][N];

      //Nhập các giá trị xác suất
      for (int i =0; i< M; i++)
      {
         for (int j=0; j< N; j++)
         {
            System.out.printf("Nhập P(%d, %d): ", i, j);
            double probability = sc.nextDouble();
            //Kiểm tra xác suất nếu âm yêu cầu nhập lại
            while (probability < 0)
            {
               System.out.println("Không được nhập xác suất
                  âm. Vui lòng nhập lại.");
               System.out.printf("Nhập P(%d, %d): ", i, j);
               probability = sc.nextDouble();
            }
            probability_Matrix[i][j] = probability;
         }
      }
      /*Test
      double[][] matrix_test = {
            {0.08, 0.32, 0.4},
            {0.015, 0.06, 0.075},
            {0.005, 0.02, 0.025}
               };
       */
```

```
double entropyX = cal_Entropy_X(probability_Matrix);
System.out.println("H(X)= "+ entropyX);


double entropyY = cal_Entropy_Y(probability_Matrix);
System.out.println("H(Y)= "+ entropyY);


double conditional_Entropy_X_Given_Y =
   cal_Conditional_Entropy_X_Given_Y(probability_Matrix);
System.out.println("H(X|Y)= " +
   conditional_Entropy_X_Given_Y);

double conditional_Entropy_Y_Given_X =
   cal_Conditional_Entropy_Y_Given_X(probability_Matrix);
System.out.println("H(Y|X)= " +
   conditional_Entropy_Y_Given_X);

double joint_Entropy =
   cal_Joint_Entropy(probability_Matrix);
System.out.println("H(X,Y)= " + joint_Entropy);

double redundancy = cal_Redundancy(entropyY,
   conditional_Entropy_Y_Given_X);
System.out.println("H(Y)  H(Y|X)= " +
   String.format("%.15f", redundancy));

double mutual_info = cal_Mutual_Information(entropyX,
   conditional_Entropy_X_Given_Y);
System.out.println("I(X,Y) = " +
   String.format("%.15f", mutual_info));

double KL_divergence_XY =
   cal_KL_divergence_XY(probability_Matrix);
System.out.println("D(P(x)||P(y)) = " +
   String.format("%.15f", KL_divergence_XY));

double KL_divergence_YX =
   cal_KL_divergence_YX(probability_Matrix);
System.out.println("D(P(y)||P(x)) = " +
   String.format("%.15f", KL_divergence_YX));
```

```
        sc.close();
    }
    //Xây dựng hàm tính log2
    public static double log2(double x) {
        return Math.log(x) / Math.log(2);
    }

    public static double cal_Entropy_X(double[][]
        probability_Matrix){
        int M = probability_Matrix.length;
        int N = probability_Matrix[0].length;
        double entropyX =0;
        for (int i =0;i< M; i++)
        {
            double row_Sum =0;
            for (int j =0; j< N; j++)
            {
                row_Sum += probability_Matrix[i][j];
            }
            entropyX += row_Sum * log2(row_Sum);
        }
        entropyX *= -1;
        return entropyX;
    }

    public static double cal_Entropy_Y(double[][]
        probability_Matrix){
        int M = probability_Matrix.length;
        int N = probability_Matrix[0].length;
        double entropyY =0;

        for (int j =0;j< N; j++)
        {
            double col_Sum =0;
            for (int i =0; i< M; i++)
            {
                col_Sum += probability_Matrix[i][j];
            }
            entropyY += col_Sum * log2(col_Sum);
        }
        entropyY *= -1;
```

```
    return entropyY;
}

public static double
    cal_Conditional_Entropy_X_Given_Y(double[][]
    probability_Matrix){
    int M = probability_Matrix.length;
    int N = probability_Matrix[0].length;
    double conditional_Entropy_X_Given_Y =0;

    for(int j=0; j< N; j++)
    {
        double col_Sum =0;
        for (int i =0; i< M; i++)
        {
            col_Sum += probability_Matrix[i][j];
        }

        for (int i=0; i< M; i++){
            if(probability_Matrix[i][j] > 0)
            {
                conditional_Entropy_X_Given_Y -=
                    (probability_Matrix[i][j]) *
                    log2(probability_Matrix[i][j] / col_Sum);
            }
        }
    }
    return conditional_Entropy_X_Given_Y;
}

public static double
    cal_Conditional_Entropy_Y_Given_X(double[][]
    probability_Matrix){
    int M = probability_Matrix.length;
    int N = probability_Matrix[0].length;
    double conditional_Entropy_Y_Given_X =0;

    for(int i=0; i< M; i++)
    {
        double row_Sum =0;
        for (int j =0; j< N; j++)
```

```
            {
                row_Sum += probability_Matrix[i][j];
            }
            for (int j=0; j< N; j++){
                if(probability_Matrix[i][j] > 0)
                {
                    conditional_Entropy_Y_Given_X -=
                        (probability_Matrix[i][j]) *
                        log2(probability_Matrix[i][j] / row_Sum);
                }
            }
        }
        return conditional_Entropy_Y_Given_X;
    }

    public static double cal_Joint_Entropy(double[][]
        probability_Matrix){
        int M = probability_Matrix.length;
        int N = probability_Matrix[0].length;
        double joint_Entropy = 0;
        for (int i=0; i< M; i++)
        {
            for (int j=0; j< N;j++)
            {
                if (probability_Matrix[i][j] > 0){
                    joint_Entropy -= probability_Matrix[i][j] *
                        log2(probability_Matrix[i][j]);
                }
            }
        }

        return joint_Entropy;
    }

    public static double cal_Redundancy(double entropyY,
        double conditional_Entropy_Y_Given_X) {
        return entropyY - conditional_Entropy_Y_Given_X;
    }

    public static double cal_Mutual_Information(double
        entropyX, double conditional_Entropy_X_Given_Y) {
```

```
        return entropyX - conditional_Entropy_X_Given_Y;
    }

    public static double cal_KL_divergence_XY(double[][]
        probability_Matrix){
        int M = probability_Matrix.length;
        int N = probability_Matrix[0].length;
        double divergence_XY =0;

        for (int i =0; i< M; i++)
        {
            double row_Sum =0;
            double col_sum =0;
            for(int j =0; j< N; j++){
                row_Sum += probability_Matrix[i][j];
                col_sum += probability_Matrix[j][i];
            }
            divergence_XY += row_Sum * log2(row_Sum/ col_sum);
        }

        return divergence_XY;
    }
    public static double cal_KL_divergence_YX(double[][]
        probability_Matrix){
        int M = probability_Matrix.length;
        int N = probability_Matrix[0].length;
        double divergence_YX =0;

        for (int i =0; i< M; i++)
        {
            double row_Sum =0;
            double col_Sum =0;
            for(int j =0; j< N; j++){
                row_Sum += probability_Matrix[i][j];
                col_Sum += probability_Matrix[j][i];
            }
            divergence_YX += col_Sum * log2(col_Sum/ row_Sum);
        }
        return divergence_YX;
    }
}
```

Kết quả chạy:

Đầu vào:

$$\begin{bmatrix} 0.1 & 0.05 & 0.05 & 0.06 \\ 0.08 & 0.03 & 0.1 & 0.04 \\ 0.08 & 0.09 & 0.1 & 0.06 \\ 0.05 & 0.04 & 0.02 & 0.05 \end{bmatrix}$$

Kết quả:

Bai1

```
/Users/quanghoang/Library/Java/JavaVirtualMachines/openjdk-19/Contents/Home/bin/java -javaagent
Nhập số hàng (M): 4
Nhập số cột (N): 4
Nhập P(0, 0): 0.1
Nhập P(0, 1): 0.05
Nhập P(0, 2): 0.05
Nhập P(0, 3): 0.06
Nhập P(1, 0): 0.08
Nhập P(1, 1): 0.03
Nhập P(1, 2): 0.1
Nhập P(1, 3): 0.04
Nhập P(2, 0): 0.08
Nhập P(2, 1): 0.09
Nhập P(2, 2): 0.1
Nhập P(2, 3): 0.06
Nhập P(3, 0): 0.05
Nhập P(3, 1): 0.04
Nhập P(3, 2): 0.02
Nhập P(3, 3): 0.05
H(X)= 1.95612775622607
H(Y)= 1.9794623904035775
H(X|Y)= 1.9003921266894845
H(Y|X)= 1.9237267608669923
H(X,Y)= 3.8798545170930616
H(Y) - H(Y|X)= 0.055735629536585
I(X,Y) = 0.055735629536585
D(P(x)||P(y)) = 0.029674373794831
D(P(y)||P(x)) = 0.030061274667934
```

Bài 2

0.1 Mã hoá Huffman

0.1.1 Lý thuyết

Mã hóa Huffman là một thuật toán rất phổ biến để mã hóa dữ liệu. Mã hóa Huffman là một thuật toán tham lam, giảm thời gian truy cập trung bình của mã càng nhiều càng tốt.

Phương pháp này tạo ra các chuỗi bit có độ dài thay đổi được gọi là mã theo cách sao cho ký tự xuất hiện thường xuyên nhất có độ dài mã ngắn nhất. Đây là một cách tối ưu để giảm thiểu thời gian truy cập trung bình của các ký tự. Nó cung cấp mã tiền tố và do đó đảm bảo nén dữ liệu không mất dữ liệu và tránh sự mơ hồ.

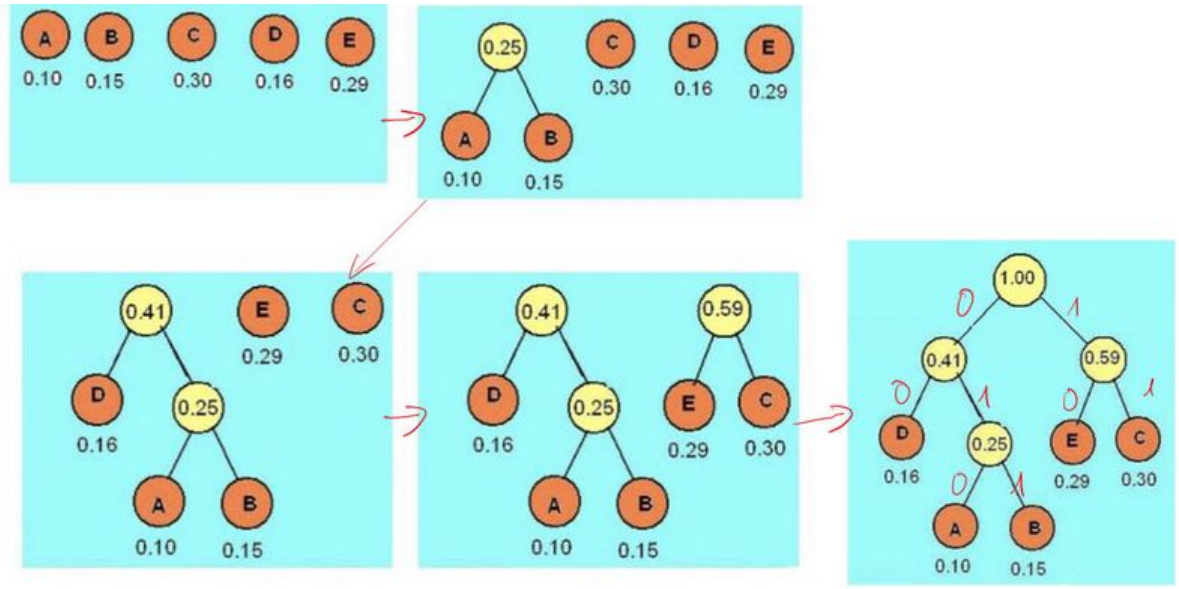
0.1.2 Các bước mã hoá

Bước 1: Hợp nhất D ký hiệu với xác suất nhỏ nhất để tạo ra một biểu tượng mới có xác suất là tổng của D ký hiệu trên.

Bước 2: Gán D ký hiệu trên với các số 0, 1,..., D-1 và quay lại Bước1.

Bước 3: Lặp lại các bước 1, 2 đến khi tổng xác suất là 1.

0.1.3 Mô tả quá trình mã hoá



Bộ mã tối ưu tương ứng là:

A	B	C	D	E
010	011	11	00	10

Hàng đợi ưu tiên: Trong mỗi bước của thuật toán xây dựng cây Huffman, ta luôn phải chọn ra hai gốc có trọng số nhỏ nhất. Để làm việc này ta sắp xếp các gốc vào một hàng đợi ưu tiên theo tiêu chuẩn trọng số nhỏ nhất. Một trong các cấu trúc dữ

liệu thuận lợi cho tiêu chuẩn này là cấu trúc đống (với phần tử có trọng số nhỏ nhất nằm trên đỉnh của đống).

0.2 Mã hoá Shannon-Fano

0.2.1 Lý thuyết

Thuật toán Shannon Fano là một kỹ thuật mã hóa entropy được sử dụng để nén dữ liệu không mất dữ liệu. Nó sử dụng xác suất xuất hiện của một ký tự và gán một mã có độ dài thay đổi duy nhất cho mỗi ký tự đó.

Nếu c là một ký tự:

Xác suất (c) = Tần số(c) / tổng các tần số

0.2.2 Các bước mã hoá

Bước 1: Các ký tự được sắp xếp theo thứ tự giảm dần của xác suất

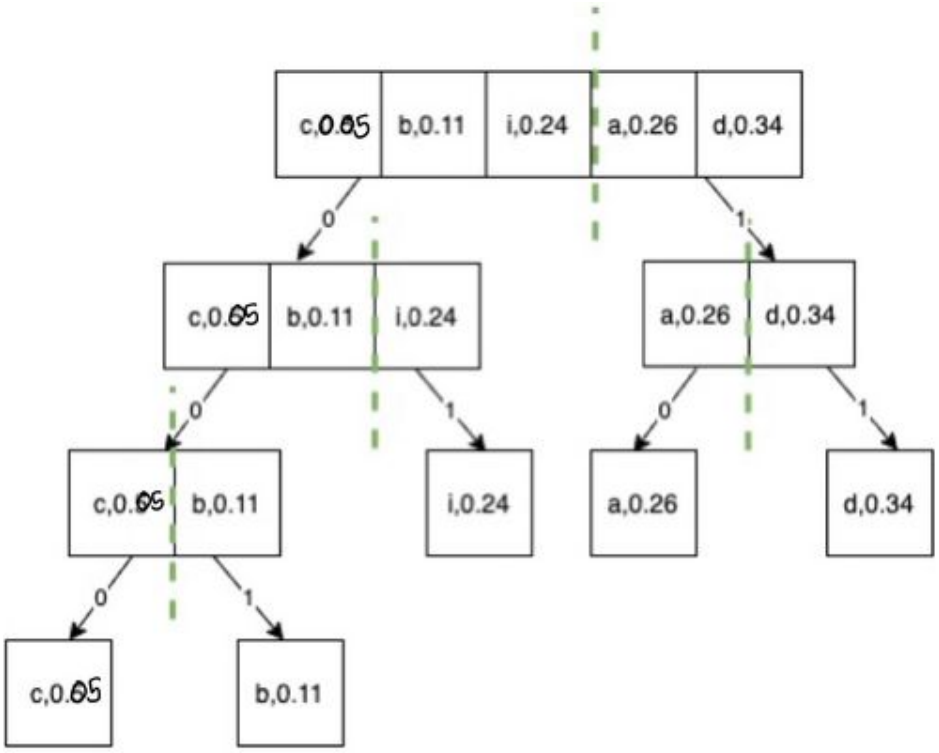
Bước 2: Chia các ký tự thành 2 tập X và Y với xác suất tương đương.

Bước 3: Tập X gán nhãn 1, tập Y gán nhãn 0

Bước 4: Lặp lại Bước 2 và Bước 3 đến khi không chia được nữa

Bước 5: Ghi lại các từ mã

0.2.3 Mô tả quá trình mã hoá



Bộ mã tối ưu tương ứng là:

c	b	i	a	d
000	001	01	10	11

0.2.4 Công thức tính hiệu suất mã và tính dư thừa

Hiệu suất mã hóa Shannon-Fano:

$$\eta = \frac{H}{L} \tag{7}$$

$$0 \leq \eta \leq 1 \tag{8}$$

H là lượng tin riêng:

$$H = -\sum_{i=1}^{n} p_i * log_2(p_i) \tag{9}$$

L là chiều dài mã trung bình:

$$L = \sum_{i=1}^{n} p_i * l_i \tag{10}$$

Tính dư thừa:

$$R = 1 - \eta \tag{11}$$

Mã nguồn bài 2

```
package Bai2;
import java.util.*;
public class Symbol {
   public char c;
   public int frequency;
   public String code;
   public Symbol left;
   public Symbol right; // Các node con trái, phải
   public Symbol(char c, int frequency) {
      this.c = c;
      this.frequency = frequency;
      this.code = "";
   }
}


public class string_Encoding
{
   public static double calculate_Efficiency(List<Symbol>
      symbols) {
      double entropy = 0;
      int total_Frequency = 0;
      double avg_Length = 0;
      for (Symbol symbol : symbols) {
         total_Frequency += symbol.frequency;
      }
      for (Symbol symbol : symbols) {
         double probability = (double) symbol.frequency /
            total_Frequency;
         entropy += -probability * log2(probability);
         avg_Length += probability * symbol.code.length();
      }
      return entropy / avg_Length;
   }
   static double log2(double n) {
      return Math.log(n) / Math.log(2);
   }
   static void fano_Shannon(List<Symbol> symbols, int beg,
      int end) {
```

```
    if (beg == end)
        return;
    int y = beg;
    int z = end;
    int sum_left = 0;
    int sum_right = 0;
    while (y <= z) {
        if (sum_left <= sum_right) {
            sum_left += symbols.get(y).frequency;
            y++;
        } else {
            sum_right += symbols.get(z).frequency;
            z--;
        }
    }
    for (int h = beg; h < y; h++) {
        symbols.get(h).code += "0";
    }
    for (int h = y; h <= end; h++) {
        symbols.get(h).code += "1";
    }

    fano_Shannon(symbols, beg, y - 1);
    fano_Shannon(symbols, y, end);
}
static void huffman_Encoding(List<Symbol> symbols) {
    // Tạo hàng đợi ưu tiên từ danh sách các ký tự
    PriorityQueue<Symbol> priority_Queue = new
        PriorityQueue<>((a,
        b)->Integer.compare(a.frequency, b.frequency));
    for (Symbol symbol : symbols) {
        priority_Queue.offer(symbol);
    }
    // Xây dựng cây Huffman bằng cách gộp các nút có tần
        số nhỏ nhất
    while (priority_Queue.size() > 1) {
        Symbol left = priority_Queue.poll();
        Symbol right = priority_Queue.poll();
        Symbol merged = new Symbol('\0', left.frequency +
            right.frequency);
        merged.code = "";
```

```
            merged.left = left;
            merged.right = right;
            priority_Queue.offer(merged);
        }
        // Gán mã hóa Huffman cho từng ký tự
        if (!priority_Queue.isEmpty()) {
            assign_Huffman_Codes(priority_Queue.peek(), "");
        }
    }
    static void assign_Huffman_Codes(Symbol symbol, String
        code) {
        if (symbol.left != null) {
            assign_Huffman_Codes(symbol.left, code + "0");
        }
        if (symbol.right != null) {
            assign_Huffman_Codes(symbol.right, code + "1");
        }
        if (symbol.left == null && symbol.right == null) {
            symbol.code = code;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nhập chuỗi: ");
        String input_String = sc.nextLine();
        input_String = input_String.toLowerCase();

        // Khởi tạo bảng tần số là một đối tượng HashMap
        Map<Character, Integer> frequency_table = new
            HashMap<>();
        // Chuyển chuỗi đầu vào thành một mảng các ký tự
        char[] characters = input_String.toCharArray();
        // Duyệt qua từng ký tự trong mảng characters
        for (char c : characters) {
            // Kiểm tra xem ký tự đãcó trong bảng tần số chưa
            if (frequency_table.containsKey(c)) {
                // Nếu ký tự đãcó trong bảng tần số, tăng giá
                    trị tần số lên 1
                int frequency = frequency_table.get(c);
                frequency_table.put(c, frequency + 1);
            } else {
```

```
            // Nếu ký tự chưa có trong bảng tần số, thêm ký
               tự vào bảng và gán giá trị tần số là 1
            frequency_table.put(c, 1);
        }
    }
    List<Symbol> symbol_List = new ArrayList<>();
    for (Map.Entry<Character, Integer> entry :
        frequency_table.entrySet()) {
        symbol_List.add(new Symbol(entry.getKey(),
            entry.getValue()));
    }
    // Sắp xếp symbol_List theo tần số giảm dần
    Collections.sort(symbol_List, (a, b) ->
        Integer.compare(b.frequency, a.frequency));
    // Áp dụng mã hoá Huffman cho symbol_List
    huffman_Encoding(symbol_List);
    // In ký tự, tần số và mã hóa theo Huffman
    System.out.println("Ký tự, tần số và mã hóa theo
        Huffman:");
    for (Symbol symbol : symbol_List) {
        System.out.println(symbol.c + " : " +
            symbol.frequency + " : " + symbol.code);
    }
    // Tạo bảng mã hóa từ danh sách các ký tự
    Map<Character, String> encoding_Table_Huffman = new
        HashMap<>();
    for (Symbol symbol : symbol_List) {
        encoding_Table_Huffman.put(symbol.c, symbol.code);
    }

    // Mã hoá chuỗi đầu vào bằng cách thay thế mỗi ký tự
        bằng mã hóa tương ứng từ bảng mã hóa
    StringBuilder encoded_String_Huffman = new
        StringBuilder();
    for (char c : characters) {
        String code = encoding_Table_Huffman.get(c);
        encoded_String_Huffman.append(code);
    }
    // In chuỗi đãđược mã hoá
    System.out.println("Chuỗi đãđược mã hoá huffman:");
    System.out.println(encoded_String_Huffman.toString());
```

```
        // Áp dụng mã hoá Shannon-Fano cho symbol_List
        fano_Shannon(symbol_List, 0, symbol_List.size() - 1);

        // In ký tự, tần số và mã hóa theo Shannon-Fano
        System.out.println("Ký tự, tần số và mã hóa theo
           Shannon-Fano:");
        for (Symbol symbol : symbol_List) {
           System.out.println(symbol.c + " : " +
              symbol.frequency + " : " + symbol.code);
        }
        // Tạo bảng mã hóa từ danh sách các ký tự
        Map<Character, String> encoding_Table_Shannon = new
           HashMap<>();
        for (Symbol symbol : symbol_List) {
           encoding_Table_Shannon.put(symbol.c, symbol.code);
        }

        // Mã hoá chuỗi đầu vào bằng cách thay thế mỗi ký tự
           bằng mã hóa tương ứng từ bảng mã hóa
        StringBuilder encoded_String_Shannom = new
           StringBuilder();
        for (char c : characters) {
           String code = encoding_Table_Shannon.get(c);
           encoded_String_Shannom.append(code);
        }

        // In chuỗi đãđược mã hoá
        System.out.println("Chuỗi đãđược mã hoá
           shannon-Fano:");
        System.out.println(encoded_String_Shannom.toString());
        // Tính hiệu suất mã hoá Shannon-Fano
        double efficiency = calculate_Efficiency(symbol_List);
        double redundancy = 1 - efficiency;

        System.out.println("Hiệu suất mã hoá: " + efficiency);
        System.out.println("Tính dư thừa: " + redundancy);
        sc.close();
    }
}
```

Kết quả chạy:

Đầu vào:

Chuỗi ký tự: Nguyen Quang Huy Hoang

Kết quả:

```
/Users/quanghoang/Library/Java/JavaVirtualMachines/openjdk-19/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/
Nhập chuỗi: Nguyen Quang Huy Hoang
Ký tự, tần số và mã hóa theo Huffman:
n : 4 : 00
  : 3 : 011
u : 3 : 110
g : 3 : 101
a : 2 : 1001
h : 2 : 1111
y : 2 : 010
q : 1 : 11100
e : 1 : 11101
o : 1 : 1000
Chuỗi đã được mã hoá huffman:
00101110010111010001111100110100100101011111111001001111111000100100101
Ký tự, tần số và mã hóa theo Shannon-Fano:
n : 4 : 0000
  : 3 : 011010
u : 3 : 110011
g : 3 : 101100
a : 2 : 10011010
h : 2 : 11111011
y : 2 : 0101100
q : 1 : 111001101
e : 1 : 111011110
o : 1 : 10001111
Chuỗi đã được mã hoá shannon-Fano:
000010110011001101011001110111100000011010111001101110011001101000001011000110101111101111001101011000110101111101110001111100110100000101100
Hiệu suất mã hoá: 0.49185093734880375
Tính dư thừa: 0.5081490626511962
```