Rapport du mini-projet multimédias

Compression des textes

Enseignante:Mme Fekih Tayssir

Membre de groupe:

- **Godhbani Ranim**
- **Dkhili Fares**

Section: ICE3

Introduction

Le présent rapport décrit le développement et l'implémentation d'une application de codage et de décodage de texte, en utilisant les algorithmes Huffman, Run-Length Encoding (RLE) et Fano-Shannon. L'objectif principal de ce projet est de fournir une interface conviviale permettant à l'utilisateur de coder et de décoder du texte avec différents algorithmes, ainsi que d'analyser les performances de chaque algorithme en termes de compression et de décompression.

Description du Projet

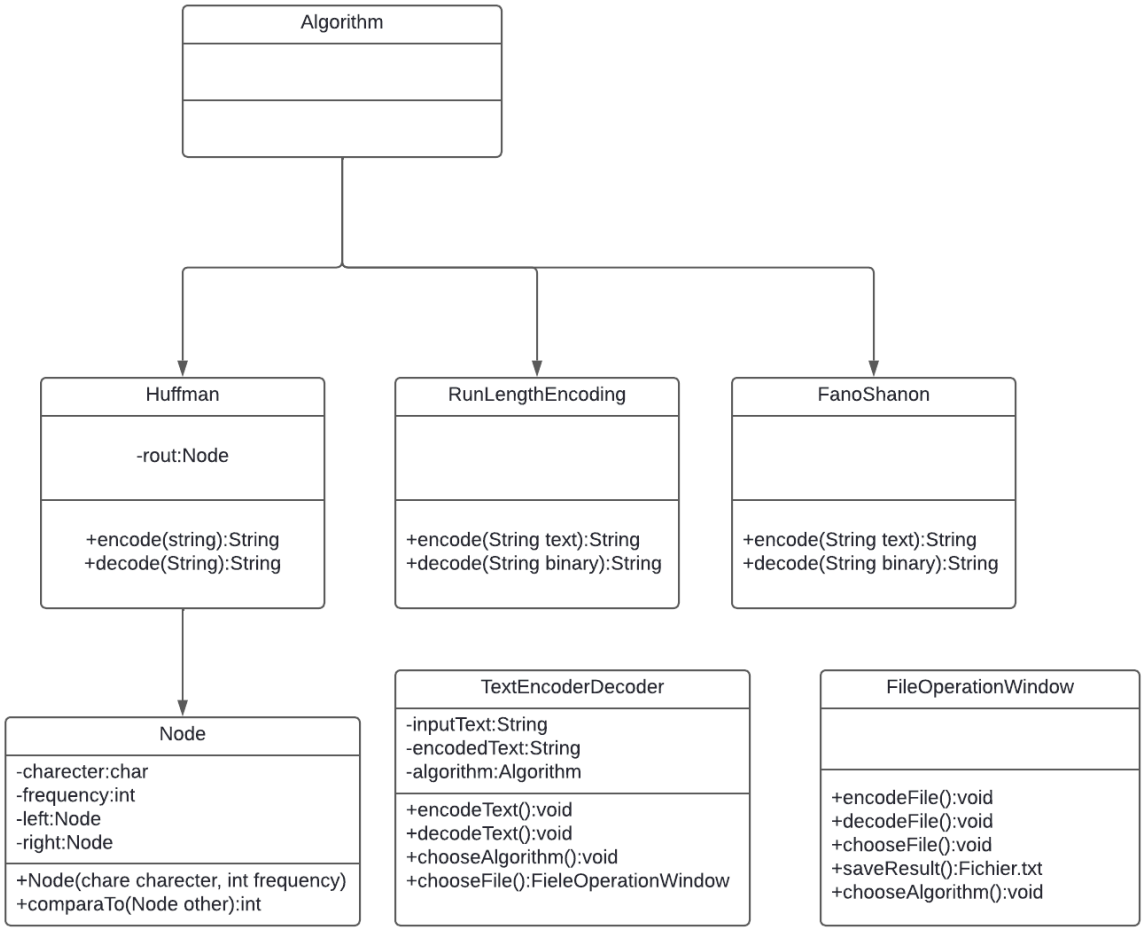
Le projet consiste en une application Java qui propose une interface graphique permettant à l'utilisateur d'entrer du texte, de sélectionner l'algorithme de codage/décodage souhaité et d'afficher le résultat du traitement. L'application permet également de coder et de décoder des fichiers texte.

Fonctionnalités Principales

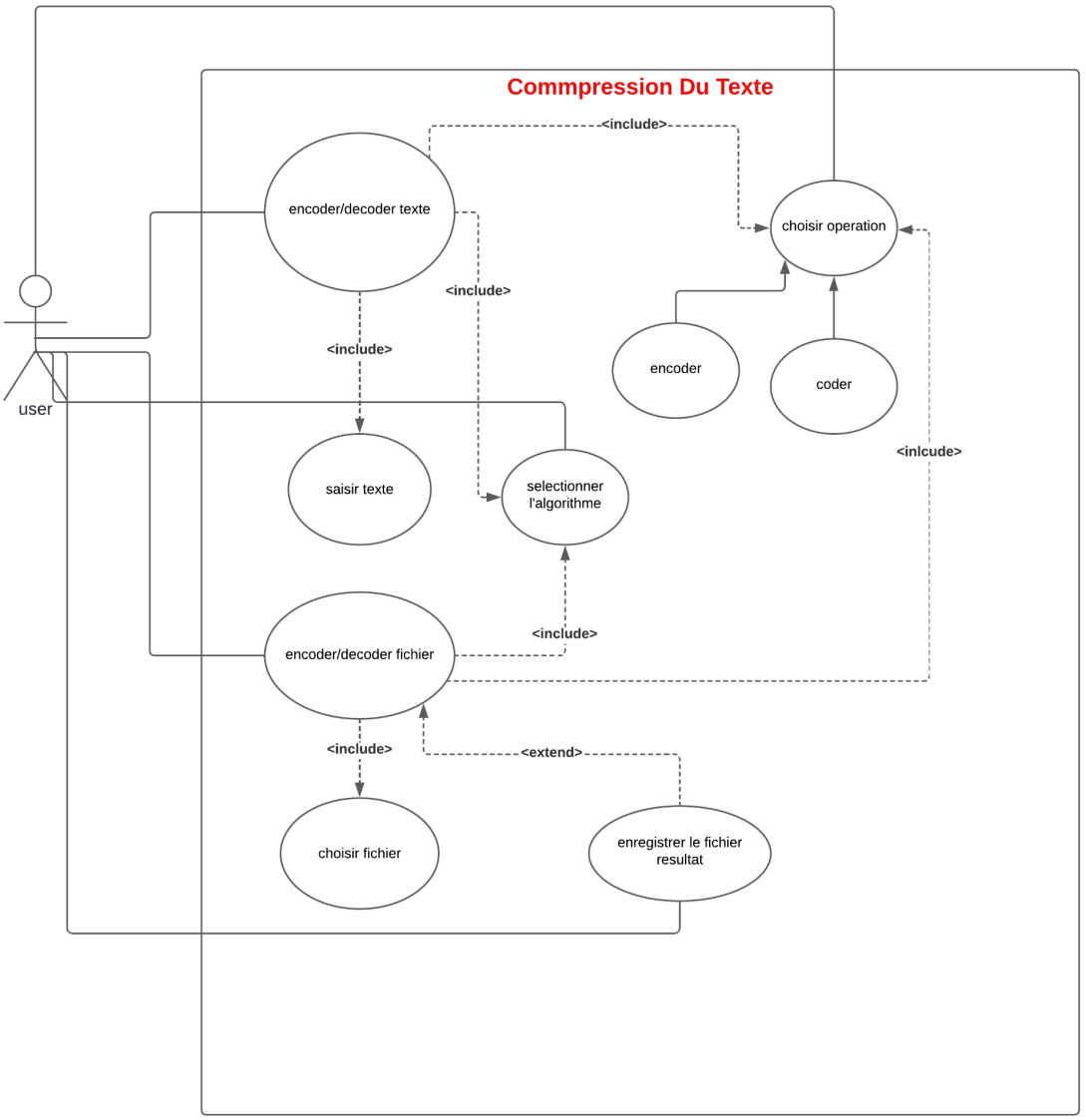
1. Interface Utilisateur Intuitive : L'application propose une interface utilisateur conviviale comprenant des zones de texte pour l'entrée et le résultat, ainsi qu'une sélection d'algorithmes et des boutons pour coder, décoder et opérer sur des fichiers.
2. Algorithme de Codage et de Décodage : Les algorithmes Huffman, Run-Length Encoding (RLE) et Fano-Shannon sont implémentés pour coder et décoder le texte saisi par l'utilisateur.
3. Opération sur des Fichiers : L'application permet de choisir un fichier texte à encoder ou décoder, et offre la possibilité de sauvegarder le résultat dans un nouveau fichier.
4. Analyse des Performances : l'application calcule la longueur binaire du texte encodé, le taux de compression et le nombre de bits utilisés par caractère.

Conception:Les diagramme UML

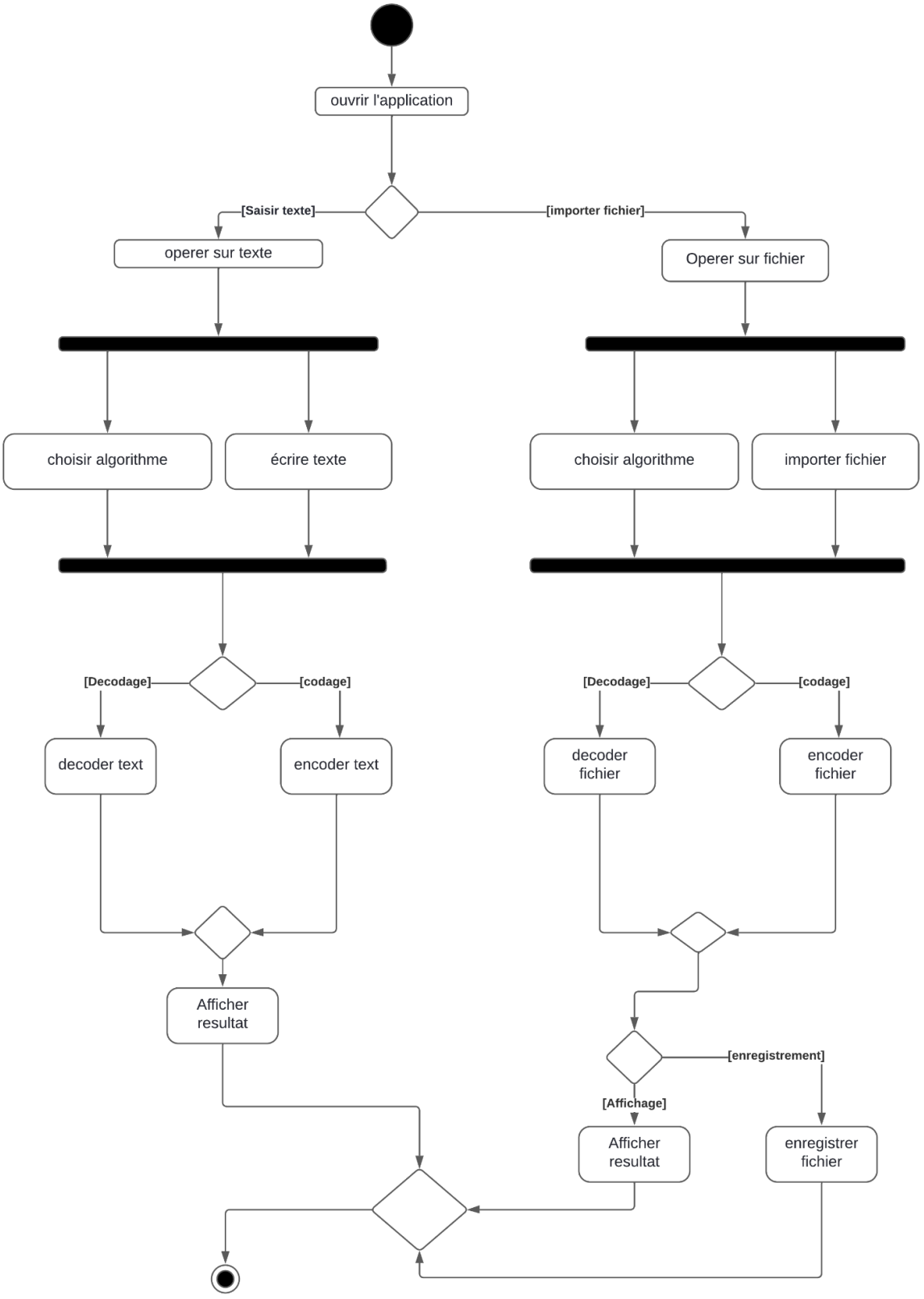
1) Diagramme de classe



2) Diagramme de cas d’utilisation



3) Diagramme d'activité:



Implémentation

Architecture Logicielle

L'application est développée en Java, utilisant le paradigme de programmation orientée objet pour modéliser les algorithmes de codage et de décodage ainsi que l'interface utilisateur.

lien du site de développement des diagramme UML

Lucid.app

l'IDE utilisé pour l'implémentation des code Java:

Eclipse: 

Algorithmes de Codage et de Décodage

1. Huffman : L'algorithme de codage Huffman utilise un arbre binaire pour représenter les caractères du texte, attribuant des codes binaires de longueurs variables aux caractères en fonction de leur fréquence d'apparition.

```
package codage;

import java.util.HashMap;

import java.util.PriorityQueue;

public class Huffman {

private static class Node implements Comparable<Node> {

char character;

int frequency;

Node left, right;

Node(char character, int frequency) {

this.character = character;

this.frequency = frequency;

}

@Override

public int compareTo(Node other) {
```

```
return this.frequency - other.frequency;
}
}
private static Node root; // Declare root as a class-level variable
public static String encode(String text) {
HashMap<Character, Integer> frequencyMap = new HashMap<>();
for (char c : text.toCharArray()) {
frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
}
PriorityQueue<Node> pq = new PriorityQueue<>();
for (char c : frequencyMap.keySet()) {
pq.offer(new Node(c, frequencyMap.get(c)));
}
while (pq.size() > 1) {
Node left = pq.poll();
Node right = pq.poll();
Node parent = new Node('\0', left.frequency + right.frequency);
parent.left = left;
parent.right = right;
pq.offer(parent);
}
root = pq.poll(); // Assign root after building the Huffman tree
HashMap<Character, String> codeMap = new HashMap<>();
buildCodeMap(root, "", codeMap);
StringBuilder encoded = new StringBuilder();
for (char c : text.toCharArray()) {
encoded.append(codeMap.get(c));
}
return encoded.toString();
```

```
}
private static void buildCodeMap(Node node, String code, HashMap<Character, String> codeMap) {
if (node != null) {
if (node.left == null && node.right == null) {
codeMap.put(node.character, code);
}
buildCodeMap(node.left, code + "0", codeMap);
buildCodeMap(node.right, code + "1", codeMap);
}
}
public static String decode(String binary) {
StringBuilder decoded = new StringBuilder();
Node current = root;
for (char bit : binary.toCharArray()) {
if (bit == '0') {
current = current.left;
} else {
current = current.right;
}
if (current.left == null && current.right == null) {
decoded.append(current.character);
current = root;
}
}
return decoded.toString();
}
}
```

2. Run-Length Encoding (RLE) : Cet algorithme compresse les données en remplaçant les séquences répétitives de caractères par un symbole de caractère suivi du nombre de répétitions.

```
package codage;

public class RunLengthEncoding {

public static String encode(String text) {

StringBuilder encoded = new StringBuilder();

int count = 1;

for (int i = 1; i < text.length(); i++) {

if (text.charAt(i) == text.charAt(i - 1)) {

count++;

} else {

encoded.append(count).append(text.charAt(i - 1));

count = 1;

}

}

encoded.append(count).append(text.charAt(text.length() - 1));

return encoded.toString();

}

public static String decode(String binary) {

StringBuilder decoded = new StringBuilder();

for (int i = 0; i < binary.length(); i += 2) {

int count = Integer.parseInt(String.valueOf(binary.charAt(i)));

char character = binary.charAt(i + 1);

for (int j = 0; j < count; j++) {

decoded.append(character);

}

}

return decoded.toString();

}
```

```
}
```

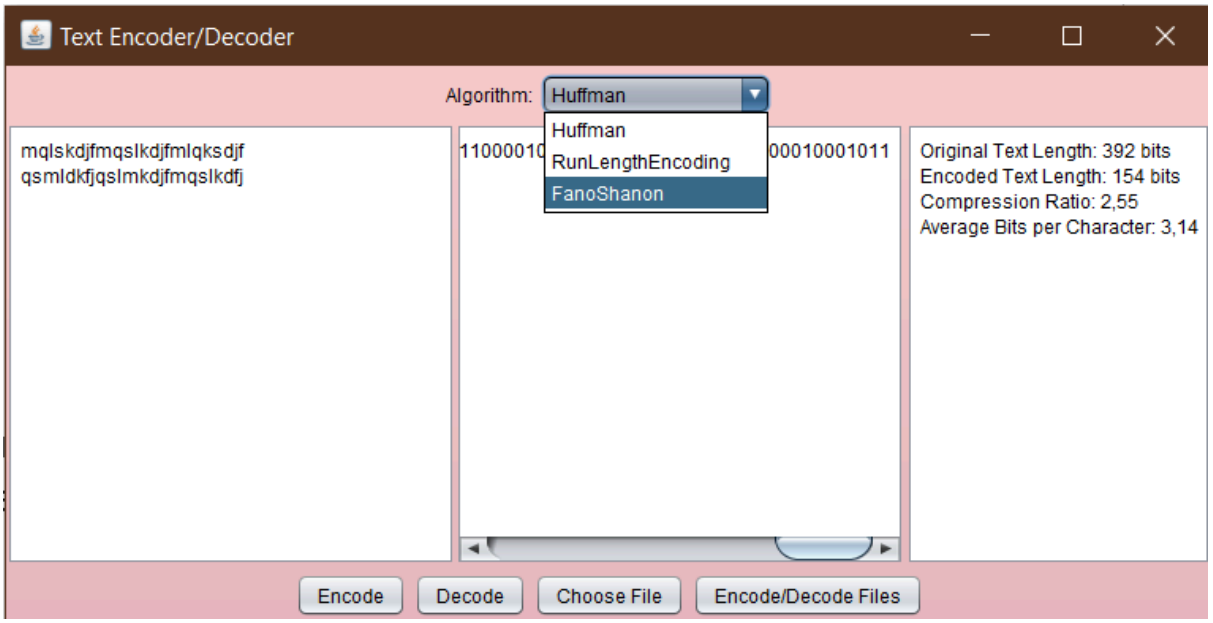
3. Fano-Shannon : L'algorithme de codage Fano-Shannon est similaire à Huffman mais utilise une approche différente pour construire l'arbre de codage.

```
package codage;

import java.util.HashMap;

public class FanoShanon {

public static String encode(String text) {

HashMap<Character, Integer> frequencyMap = new HashMap<>();

for (char c : text.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

}

StringBuilder encoded = new StringBuilder();

for (char c : text.toCharArray()) {

int frequency = frequencyMap.get(c);

String binaryCode = Integer.toBinaryString(frequency);

encoded.append(binaryCode);

}

return encoded.toString();

}

public static String decode(String binary) {

StringBuilder decoded = new StringBuilder();

int i = 0;

while (i < binary.length()) {

StringBuilder binaryCode = new StringBuilder();

while (i < binary.length() && binary.charAt(i) == '1') {

binaryCode.append(binary.charAt(i));

i++;

}

int frequency = binaryCode.length();
```

```
decoded.append((char) frequency);
i++;
}
return decoded.toString();
}
}
```

Interface Utilisateur

L'interface utilisateur est réalisée en utilisant la bibliothèque Swing de Java, offrant une disposition claire et intuitive des composants pour une expérience utilisateur agréable.



Code pour l'interface graphique main:

```
package codage;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class MainWindow extends JFrame {
private JComboBox<String> algorithmComboBox;
```

```
private JButton encodeButton, decodeButton, fileButton, fileOperationButton;

private JTextArea inputTextArea, resultTextArea, statsTextArea;

public MainWindow() {

setTitle("Text Encoder/Decoder");

setSize(600, 400);

setDefaultCloseOperation(EXIT_ON_CLOSE);

setLocationRelativeTo(null);

// Create a gradient background

JPanel mainPanel = new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g.create();

int w = getWidth();

int h = getHeight();

Color color1 = new Color(247, 204, 204); // Light pink

Color color2 = new Color(230, 184, 190); // Light rose

GradientPaint gp = new GradientPaint(0, 0, color1, 0, h, color2);

g2d.setPaint(gp);

g2d.fillRect(0, 0, w, h);

g2d.dispose();

}

};

mainPanel.setLayout(new BorderLayout());

JPanel topPanel = new JPanel(new FlowLayout());

topPanel.setOpaque(false); // Make the panel transparent

JLabel algorithmLabel = new JLabel("Algorithm:");

algorithmComboBox = new JComboBox<>(new String[]{"Huffman", "RunLengthEncoding",
"FanoShanon"});

topPanel.add(algorithmLabel);
```

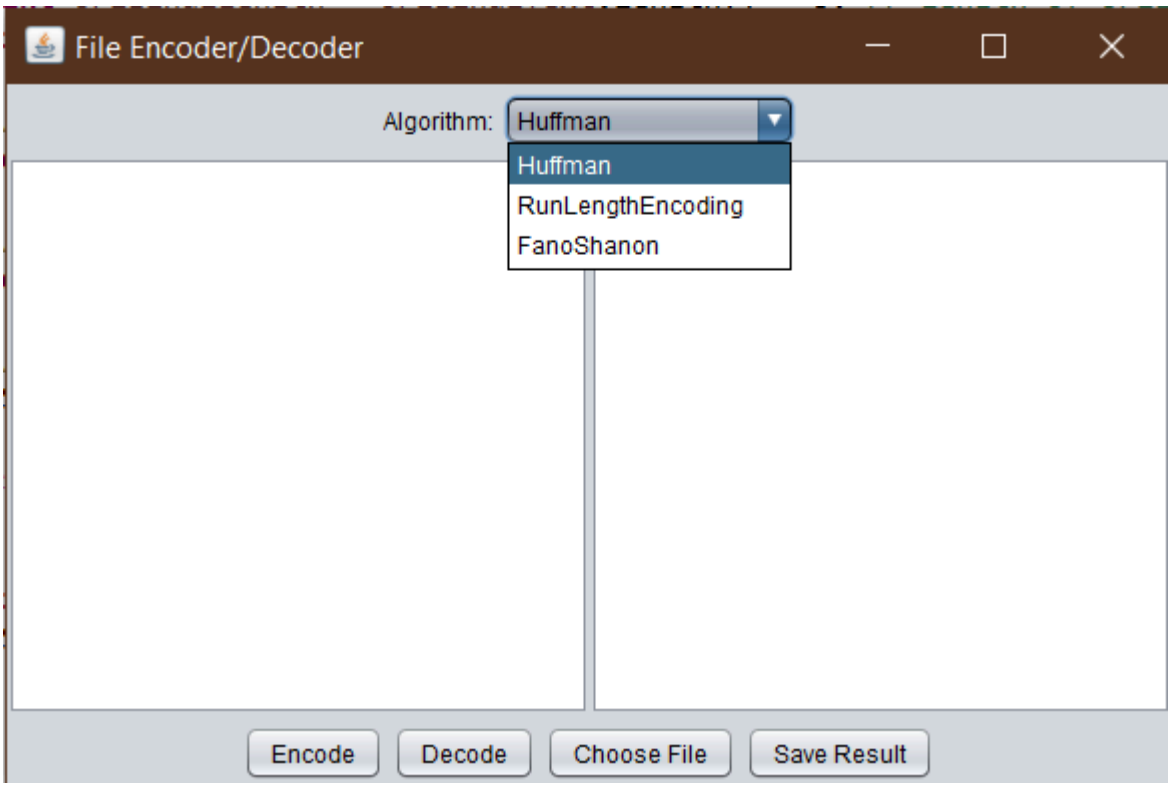
```
topPanel.add(algorithmComboBox);
mainPanel.add(topPanel, BorderLayout.NORTH);
JPanel buttonPanel = new JPanel(new FlowLayout());
buttonPanel.setOpaque(false);
encodeButton = new JButton("Encode");
decodeButton = new JButton("Decode");
fileButton = new JButton("Choose File");
fileOperationButton = new JButton("Encode/Decode Files");
buttonPanel.add(encodeButton);
buttonPanel.add(decodeButton);
buttonPanel.add(fileButton);
buttonPanel.add(fileOperationButton);
mainPanel.add(buttonPanel, BorderLayout.SOUTH);
JPanel textPanel = new JPanel(new GridLayout(1, 2));
textPanel.setOpaque(false);
inputTextArea = new JTextArea();
JScrollPane inputScrollPane = new JScrollPane(inputTextArea);
textPanel.add(inputScrollPane);
resultTextArea = new JTextArea();
JScrollPane resultScrollPane = new JScrollPane(resultTextArea);
textPanel.add(resultScrollPane);
mainPanel.add(textPanel, BorderLayout.CENTER);
statsTextArea = new JTextArea();
statsTextArea.setEditable(false);
JScrollPane statsScrollPane = new JScrollPane(statsTextArea);
mainPanel.add(statsScrollPane, BorderLayout.EAST);
add(mainPanel);
encodeButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
```

```
String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();
String s = inputTextArea.getText();
if (s.isEmpty()) {
JOptionPane.showMessageDialog(null, "InputTextField is empty");
} else {
String c;
switch (selectedAlgorithm) {
case "Huffman":
c = Huffman.encode(s);
break;
case "RunLengthEncoding":
c = RunLengthEncoding.encode(s);
break;
default:
c = FanoShanon.encode(s);
break;
}
resultTextArea.setText(c);
computeStatistics(s, c);
}
}
});
decodeButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();
String s = inputTextArea.getText();
if (s.isEmpty()) {
JOptionPane.showMessageDialog(null, "InputTextField is empty");
} else {
```

```
String c;
try {
switch (selectedAlgorithm) {
case "Huffman":
c = Huffman.decode(s);
break;
case "RunLengthEncoding":
c = RunLengthEncoding.decode(s);
break;
default:
c = FanoShanon.decode(s);
break;
}
resultTextArea.setText(c);
computeStatistics(s, c);
} catch (Exception ex) {
JOptionPane.showMessageDialog(null, ex);
}
}
}
});
fileButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
JFileChooser fileChooser = new JFileChooser();
int result = fileChooser.showOpenDialog(null);
if (result == JFileChooser.APPROVE_OPTION) {
File selectedFile = fileChooser.getSelectedFile();
try {
BufferedReader reader = new BufferedReader(new FileReader(selectedFile));
```

```
StringBuilder text = new StringBuilder();
String line;
while ((line = reader.readLine()) != null) {
text.append(line).append("\n");
}
inputTextArea.setText(text.toString());
reader.close();
} catch (IOException ex) {
ex.printStackTrace();
}
}
}
});
fileOperationButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
FileOperationWindow fileOperationWindow = new FileOperationWindow();
fileOperationWindow.setVisible(true);
}
});
}
private void computeStatistics(String originalText, String encodedText) {
int originalLength = originalText.length() * 8; // Length of original text in bits
int encodedLength = encodedText.length(); // Length of encoded text in characters
// Calculate compression ratio
double compressionRatio = (double) originalLength / encodedLength;
// Calculate average bits per character
double bitsPerCharacter = (double) encodedLength / originalText.length();
// Display statistics
String stats = String.format("Original Text Length: %d bits\nEncoded Text Length: %d
bits\nCompression Ratio: %.2f\nAverage Bits per Character: %.2f",
```

```
originalLength, encodedLength, compressionRatio, bitsPerCharacter);
statsTextArea.setText(stats);
}
public static void main(String[] args) {
SwingUtilities.invokeLater(new Runnable() {
public void run() {
// Set Nimbus look and feel for modern UI
try {
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
} catch (Exception e) {
e.printStackTrace();
}
new MainWindow().setVisible(true);
}
});
}
}
```



Le code pour l'importation du fichier.txt d'encodage ainsi que l'enregistrement de fichier résultat.

```
package codage;

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.*;

class FileOperationWindow extends JFrame {

private JComboBox<String> algorithmComboBox;

private JButton encodeButton, decodeButton, chooseFileButton, saveButton;

private JTextArea inputTextArea, resultTextArea;

private File selectedFile;

public FileOperationWindow() {

setTitle("File Encoder/Decoder");

setSize(600, 400);

setDefaultCloseOperation(DISPOSE_ON_CLOSE);
```

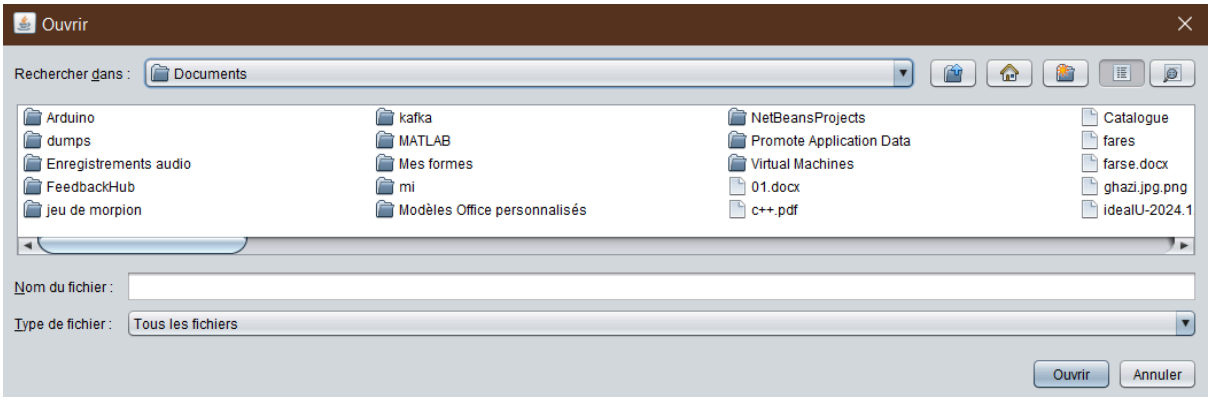
```
setLocationRelativeTo(null);

JPanel mainPanel = new JPanel(new BorderLayout());

JPanel topPanel = new JPanel(new FlowLayout());

JLabel algorithmLabel = new JLabel("Algorithm:");

algorithmComboBox = new JComboBox<>(new String[]{"Huffman", "RunLengthEncoding",
"FanoShanon"});

topPanel.add(algorithmLabel);

topPanel.add(algorithmComboBox);

mainPanel.add(topPanel, BorderLayout.NORTH);

JPanel buttonPanel = new JPanel(new FlowLayout());

encodeButton = new JButton("Encode");

decodeButton = new JButton("Decode");

chooseFileButton = new JButton("Choose File");

saveButton = new JButton("Save Result");

buttonPanel.add(encodeButton);

buttonPanel.add(decodeButton);

buttonPanel.add(chooseFileButton);

buttonPanel.add(saveButton);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

JPanel textPanel = new JPanel(new GridLayout(1, 2));

inputTextArea = new JTextArea();

JScrollPane inputScrollPane = new JScrollPane(inputTextArea);

textPanel.add(inputScrollPane);

resultTextArea = new JTextArea();

JScrollPane resultScrollPane = new JScrollPane(resultTextArea);

textPanel.add(resultScrollPane);

mainPanel.add(textPanel, BorderLayout.CENTER);

add(mainPanel);

encodeButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {
```

```
String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();
String s = inputTextArea.getText();
if (s.isEmpty()) {
JOptionPane.showMessageDialog(null, "InputTextField is empty");
} else {
String c;
switch (selectedAlgorithm) {
case "Huffman":
c = Huffman.encode(s);
break;
case "RunLengthEncoding":
c = RunLengthEncoding.encode(s);
break;
default:
c = FanoShanon.encode(s);
break;
}
resultTextArea.setText(c);
computeStatistics(s, c);
}
}
});
decodeButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();
String s = inputTextArea.getText();
if (s.isEmpty()) {
JOptionPane.showMessageDialog(null, "InputTextField is empty");
} else {
```

```
String c;
try {
switch (selectedAlgorithm) {
case "Huffman":
c = Huffman.decode(s);
break;
case "RunLengthEncoding":
c = RunLengthEncoding.decode(s);
break;
default:
c = FanoShanon.decode(s);
break;
}
resultTextArea.setText(c);
computeStatistics(s, c);
} catch (Exception ex) {
JOptionPane.showMessageDialog(null, ex);
}
}
}
});
chooseFileButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
JFileChooser fileChooser = new JFileChooser();
int result = fileChooser.showOpenDialog(null);
if (result == JFileChooser.APPROVE_OPTION) {
selectedFile = fileChooser.getSelectedFile();
try {
BufferedReader reader = new BufferedReader(new FileReader(selectedFile));
```

```
StringBuilder text = new StringBuilder();
String line;
while ((line = reader.readLine()) != null) {
text.append(line).append("\n");
}
inputTextArea.setText(text.toString());
reader.close();
} catch (IOException ex) {
ex.printStackTrace();
}
}
}
});
saveButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
if (selectedFile == null) {
JOptionPane.showMessageDialog(null, "No file selected.");
return;
}
String result = resultTextArea.getText();
if (result.isEmpty()) {
JOptionPane.showMessageDialog(null, "No result to save.");
return;
}
JFileChooser fileChooser = new JFileChooser();
fileChooser.setSelectedFile(new File(selectedFile.getParent(), "encoded_decoded_" +
selectedFile.getName()));
int resultDialog = fileChooser.showSaveDialog(null);
if (resultDialog == JFileChooser.APPROVE_OPTION) {
File outputFile = fileChooser.getSelectedFile();
```

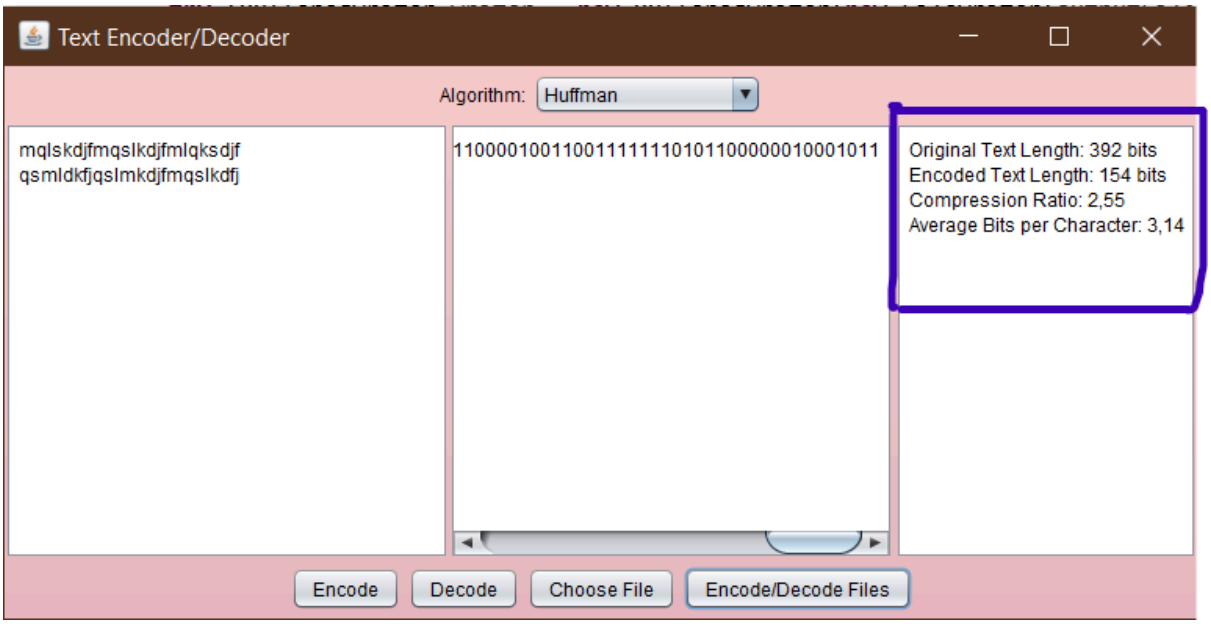
```
try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
writer.write(result);
JOptionPane.showMessageDialog(null, "File saved successfully.");
} catch (IOException ex) {
ex.printStackTrace();
JOptionPane.showMessageDialog(null, "Error saving file: " + ex.getMessage());
}
}
}
});
}
private void computeStatistics(String originalText, String encodedText) {
int originalLength = originalText.length() * 8; // Length of original text in bits
int encodedLength = encodedText.length(); // Length of encoded text in characters
// Calculate compression ratio
double compressionRatio = (double) originalLength / encodedLength;
// Calculate average bits per character
double bitsPerCharacter = (double) encodedLength / originalText.length();
// Display statistics
String stats = String.format("Original Text Length: %d bits\nEncoded Text Length: %d
bits\nCompression Ratio: %.2f\nAverage Bits per Character: %.2f",
originalLength, encodedLength, compressionRatio, bitsPerCharacter);
JOptionPane.showMessageDialog(null, stats, "Statistics", JOptionPane.INFORMATION_MESSAGE);
}
}
```



Analyse des Performances

l'application calcule les statistiques suivantes :

- Longueur binaire du texte encodé.
- Taux de compression : rapport entre la longueur du texte original et celle du texte encodé.
- Nombre de bits utilisés par caractère en moyenne.



Conclusion

Le projet de codage et de décodage de texte a été réalisé avec succès, fournissant une application fonctionnelle et conviviale pour l'encodage et le décodage de texte avec différents algorithmes. L'analyse des performances permet de comparer les algorithmes en termes d'efficacité de compression et de décompression. Ce projet offre une base solide pour explorer davantage les techniques de compression de données et leur application dans divers domaines.