# CHAPTER V
# IMPLEMENTATION AND TESTING

## 5.1. Implementation

### 5.1.1. Compression

Compression process begin after send button clicked. Compression algorithm used by the algorithm chosen in preferences.

#### 5.1.1.1. LZW

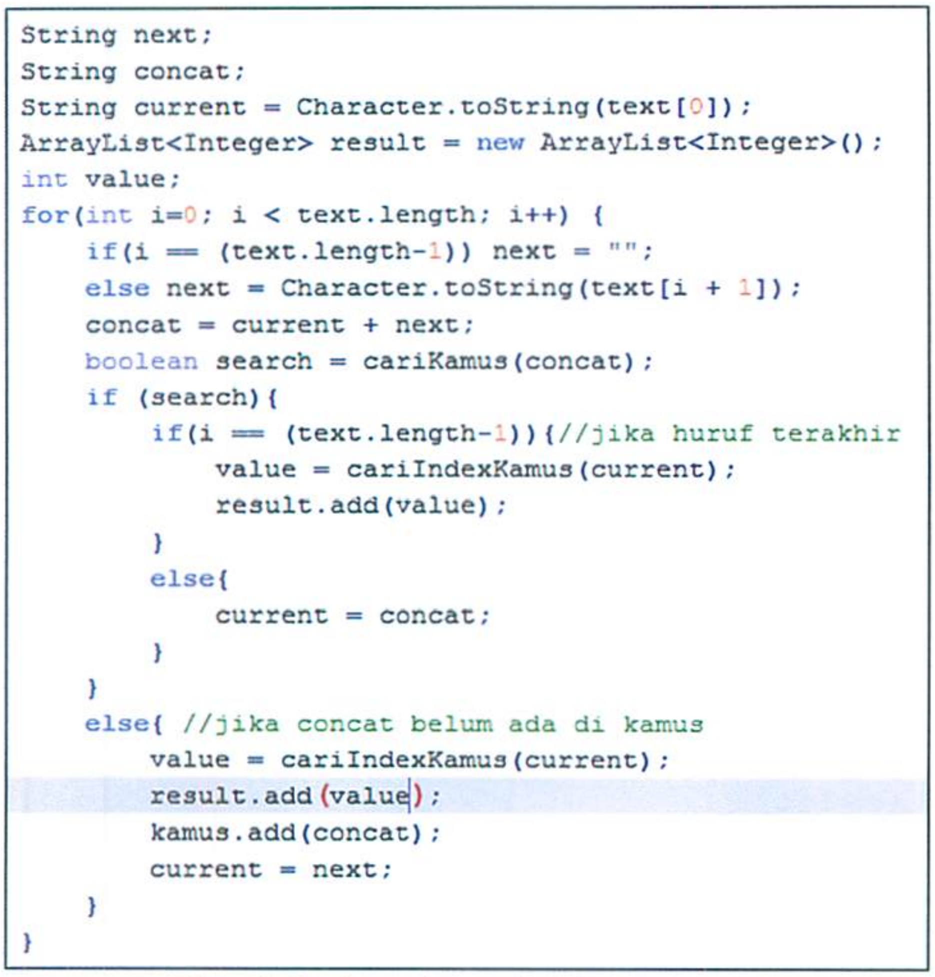
The input of this process is array of character. First process is building the dictionary, the dictionary consist of 0-255 decimal in ASCII table.

```
for(int i = 0; i < 256; i++) {
    kamus.add(Character.toString((char) i));
}
```

Figure 10. Building dictionary code

Then create 3 string variable ('next', 'current', 'concat'), 1 integer variable ('value'), and 1 arraylist of integer ('result').String 'next' is character of the next index in the loop. String 'current' is character of the current index in the loop. String 'concat' is combination of 'current' and 'next'. Integer 'value' is index number in the dictionary. Arraylist 'result' is temporary result before processed into byte array.

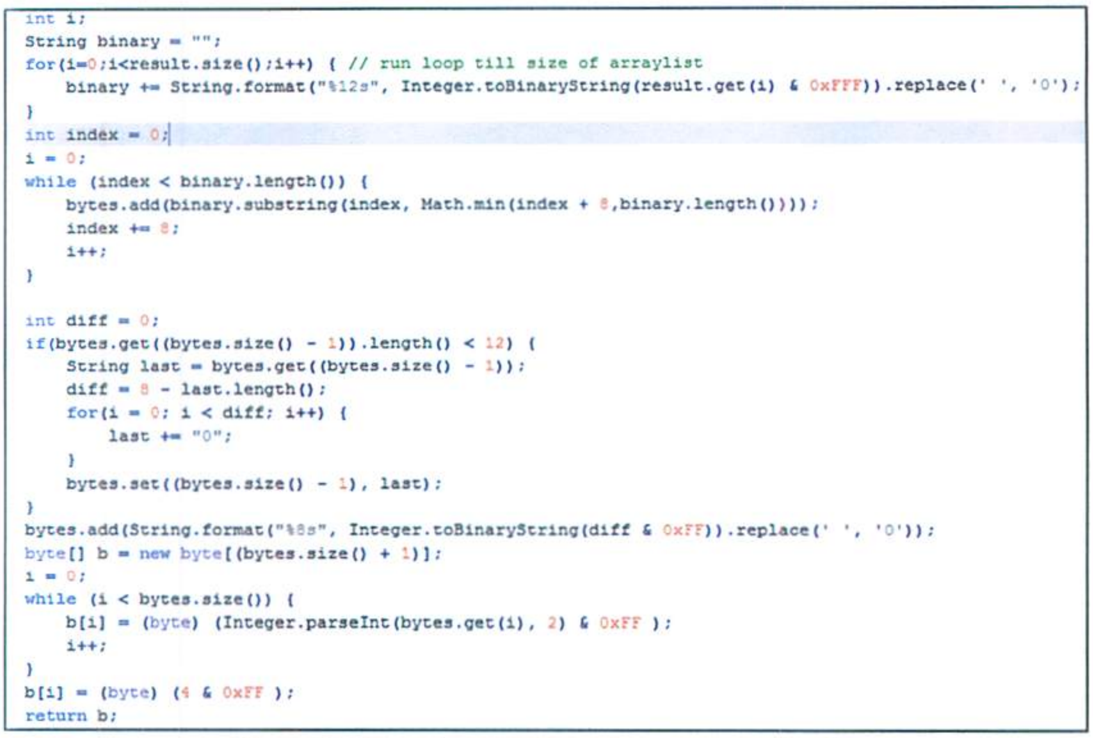
In the looping process, check the 'concat' String available or not in the dictionary. If string 'concat' exist in the dictionary, then check again if the loop index is last character in the message then add value to temporary result, if the character is not last character in the message then change string 'current' same as string 'concat'. If string 'concat' not exist, then add index of current character in the dictionary into temporary result and add the string concat into dictionary. Loop until the end of message.



```java
String next;
String concat;
String current = Character.toString(text[0]);
ArrayList<Integer> result = new ArrayList<Integer>();
int value;
for(int i=0; i < text.length; i++) {
    if(i == (text.length-1)) next = "";
    else next = Character.toString(text[i + 1]);
    concat = current + next;
    boolean search = cariKamus(concat);
    if (search){
        if(i == (text.length-1)){//jika huruf terakhir
            value = cariIndexKamus(current);
            result.add(value);
        }
        else{
            current = concat;
        }
    }
    else{ //jika concat belum ada di kamus
        value = cariIndexKamus(current);
        result.add(value);
        kamus.add(concat);
        current = next;
    }
}
```

Figure 11. Declare variable and looping process code

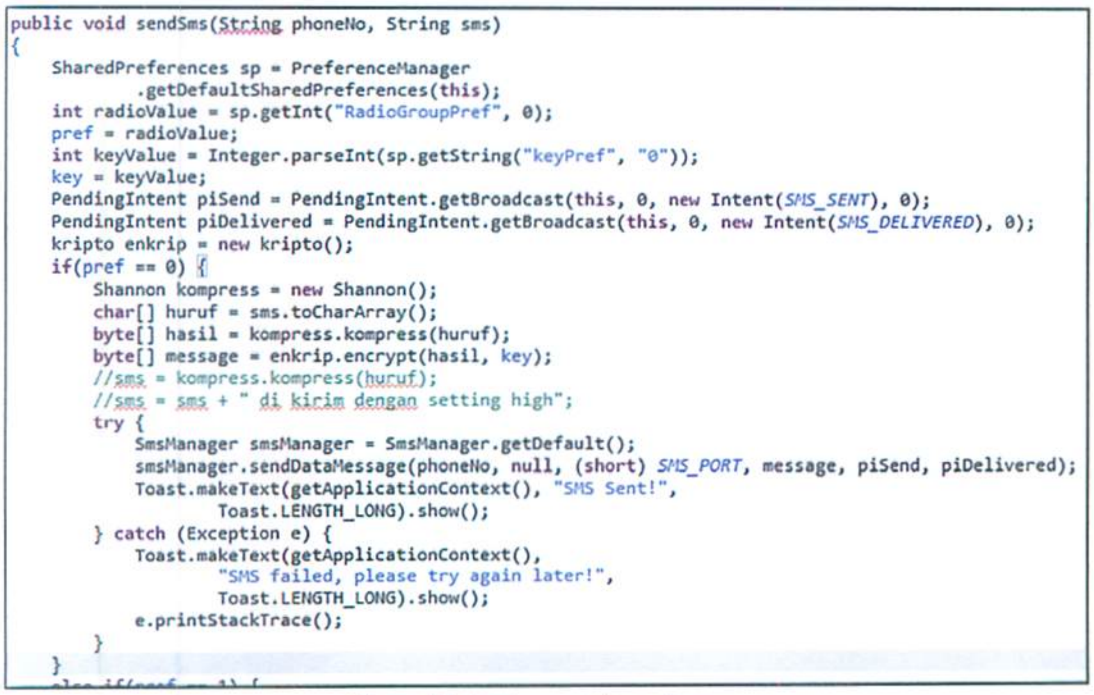
The maximum number that can be accommodated reach 4096 number of entries or 12 bit, but the output require byte array or 8 bit. To solve this, convert each data of temporary result to 12 bit binary string and insert into one string, and then divide it each 8 bits.



```
int i;
String binary = "";
for(i=0;i<result.size();i++) { // run loop till size of arraylist
    binary += String.format("%12s", Integer.toBinaryString(result.get(i) & 0xFFF)).replace(' ', '0');
}
int index = 0;
i = 0;
while (index < binary.length()) {
    bytes.add(binary.substring(index, Math.min(index + 8,binary.length())));
    index += 8;
    i++;
}

int diff = 0;
if(bytes.get((bytes.size() - 1)).length() < 12) {
    String last = bytes.get((bytes.size() - 1));
    diff = 8 - last.length();
    for(i = 0; i < diff; i++) {
        last += "0";
    }
    bytes.set((bytes.size() - 1), last);
}
bytes.add(String.format("%8s", Integer.toBinaryString(diff & 0xFF)).replace(' ', '0'));
byte[] b = new byte[(bytes.size() + 1)];
i = 0;
while (i < bytes.size()) {
    b[i] = (byte) (Integer.parseInt(bytes.get(i), 2) & 0xFF );
    i++;
}
b[i] = (byte) (4 & 0xFF );
return b;
```

Figure 12. Processing output from temporary result code

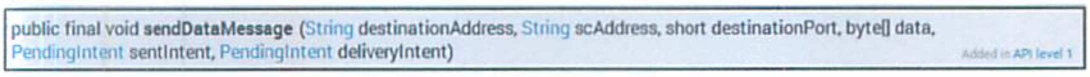
In this process, array and arraylist is used as data structure. Array of character as the input, array of byte as the output. Array of character is used for processing the message each character.



```
public void sendSms(String phoneNo, String sms)
{
    SharedPreferences sp = PreferenceManager
            .getDefaultSharedPreferences(this);
    int radioValue = sp.getInt("RadioGroupPref", 0);
    pref = radioValue;
    int keyValue = Integer.parseInt(sp.getString("keyPref", "0"));
    key = keyValue;
    PendingIntent piSend = PendingIntent.getBroadcast(this, 0, new Intent(SMS_SENT), 0);
    PendingIntent piDelivered = PendingIntent.getBroadcast(this, 0, new Intent(SMS_DELIVERED), 0);
    kripto enkrip = new kripto();
    if(pref == 0) {
        Shannon kompress = new Shannon();
        char[] huruf = sms.toCharArray();
        byte[] hasil = kompress.kompress(huruf);
        byte[] message = enkrip.encrypt(hasil, key);
        //sms = kompress.kompress(huruf);
        //sms = sms + " di kirim dengan setting high";
        try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendDataMessage(phoneNo, null, (short) SMS_PORT, message, piSend, piDelivered);
            Toast.makeText(getApplicationContext(), "SMS Sent!",
                    Toast.LENGTH_LONG).show();
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(),
                    "SMS failed, please try again later!",
                    Toast.LENGTH_LONG).show();
            e.printStackTrace();
        }
    }
    else if(pref == 1) {
```

Figure 13. Implementation of array

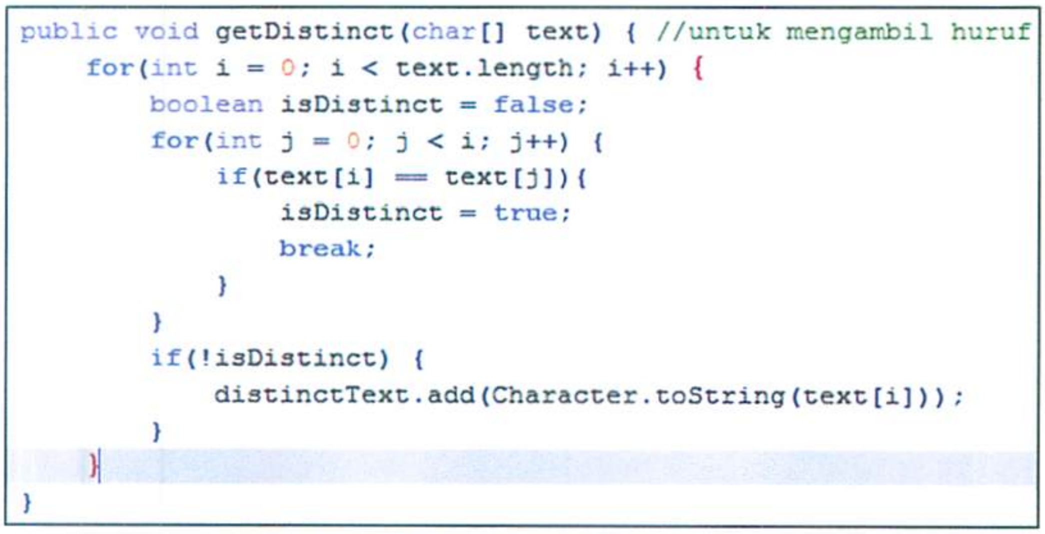
Array of byte is used for sending sms to the receiver by using class sendDataMessage in android. This class require array of byte to send the message.



```
public final void sendDataMessage (String destinationAddress, String scAddress, short destinationPort, byte[] data,
PendingIntent sentIntent, PendingIntent deliveryIntent)                                        Added in API level 1
```

Figure 14. sendDataMessage Class

Arraylist is used because the number of  required arrays is not fixed while processing the message. For example, when processing the message contains how many letters, symbols or numbers.

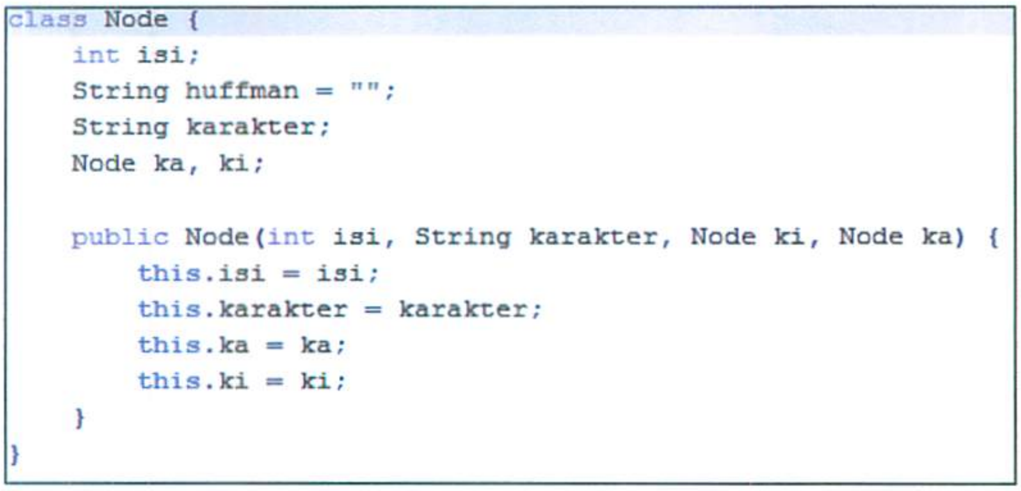


```
public void getDistinct(char[] text) { //untuk mengambil huruf
    for(int i = 0; i < text.length; i++) {
        boolean isDistinct = false;
        for(int j = 0; j < i; j++) {
            if(text[i] == text[j]){
                isDistinct = true;
                break;
            }
        }
        if(!isDistinct) {
            distinctText.add(Character.toString(text[i]));
        }
    }
}
```

Figure 15. Implementation of arraylist

#### 5.1.1.2. Huffman

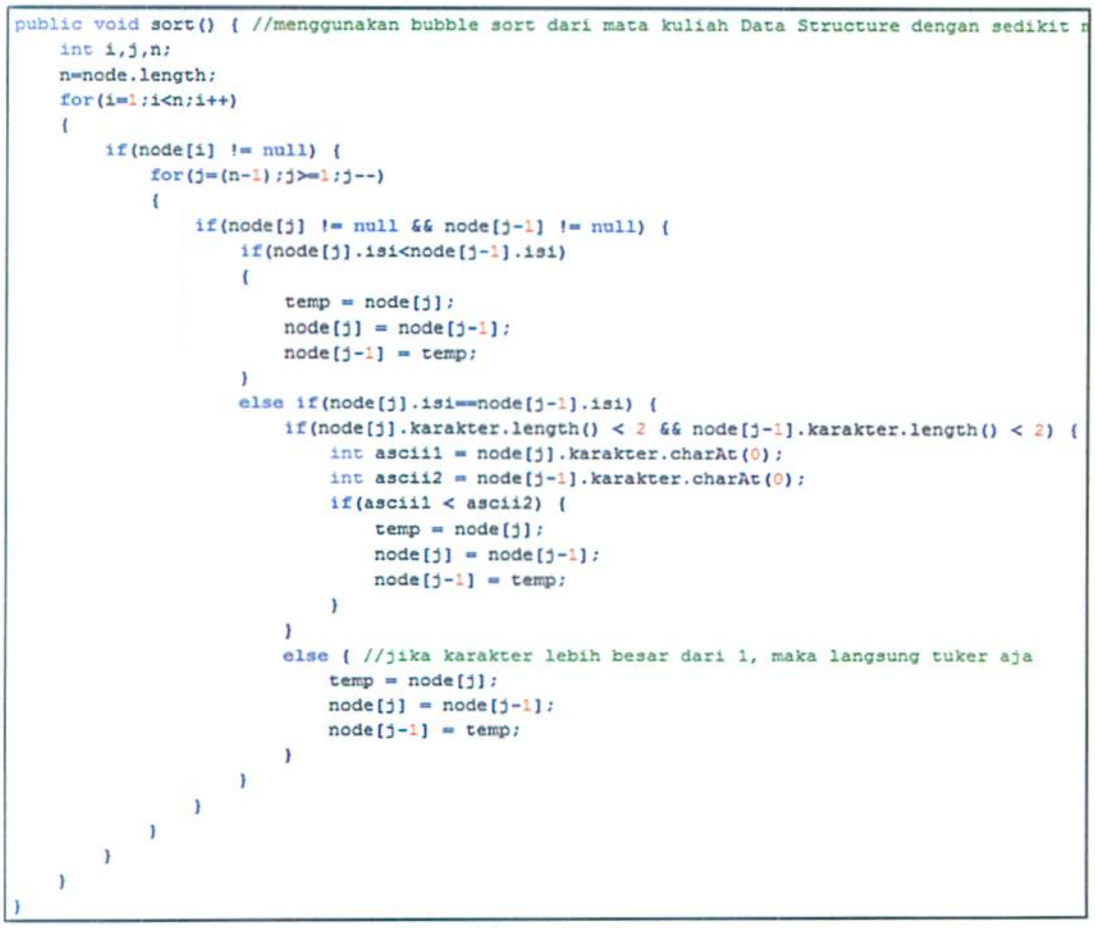
The input of this process is array of character. make array of node from each character of this input, the node consist of character, frequent of character, and Huffman code.



```
class Node {
    int isi;
    String huffman = "";
    String karakter;
    Node ka, ki;

    public Node(int isi, String karakter, Node ki, Node ka) {
        this.isi = isi;
        this.karakter = karakter;
        this.ka = ka;
        this.ki = ki;
    }
}
```

Figure 16. Node class code

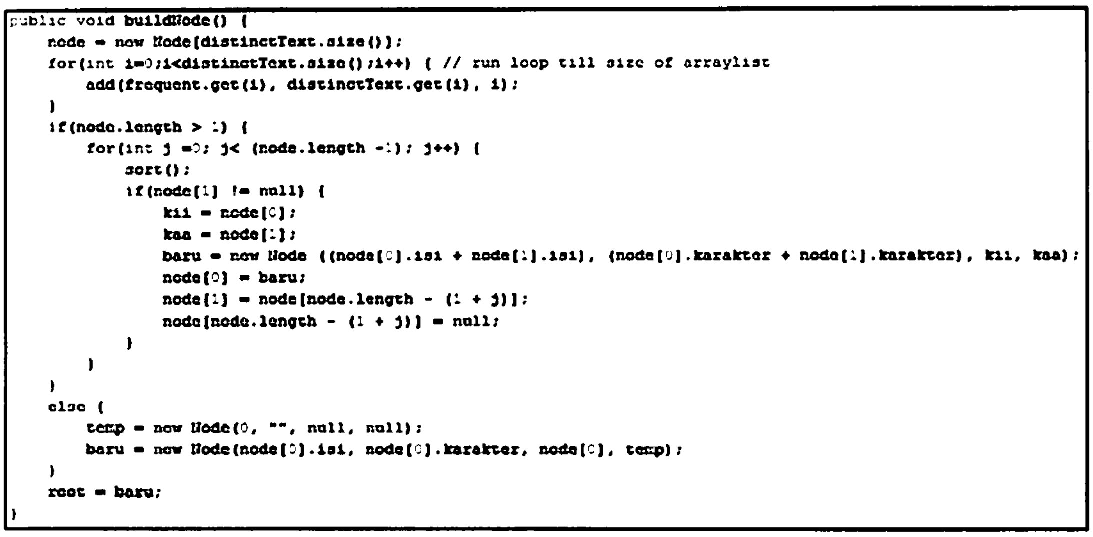
The first step to build tree is sort ascending the node by frequent of each character, if node frequent same as another node then sort by ascii value.



```
public void sort() { //menggunakan bubble sort dari mata kuliah Data Structure dengan sedikit
    int i,j,n;
    n=node.length;
    for(i=1;i<n;i++)
    {
        if(node[i] != null) {
            for(j=(n-1);j>=1;j--)
            {
                if(node[j] != null && node[j-1] != null) {
                    if(node[j].isi<node[j-1].isi)
                    {
                        temp = node[j];
                        node[j] = node[j-1];
                        node[j-1] = temp;
                    }
                    else if(node[j].isi==node[j-1].isi) {
                        if(node[j].karakter.length() < 2 && node[j-1].karakter.length() < 2) {
                            int ascii1 = node[j].karakter.charAt(0);
                            int ascii2 = node[j-1].karakter.charAt(0);
                            if(ascii1 < ascii2) {
                                temp = node[j];
                                node[j] = node[j-1];
                                node[j-1] = temp;
                            }
                        }
                        else { //jika karakter lebih besar dari 1, maka langsung tuker aja
                            temp = node[j];
                            node[j] = node[j-1];
                            node[j-1] = temp;
                        }
                    }
                }
            }
        }
    }
}
```

Figure 17. Sort Node code

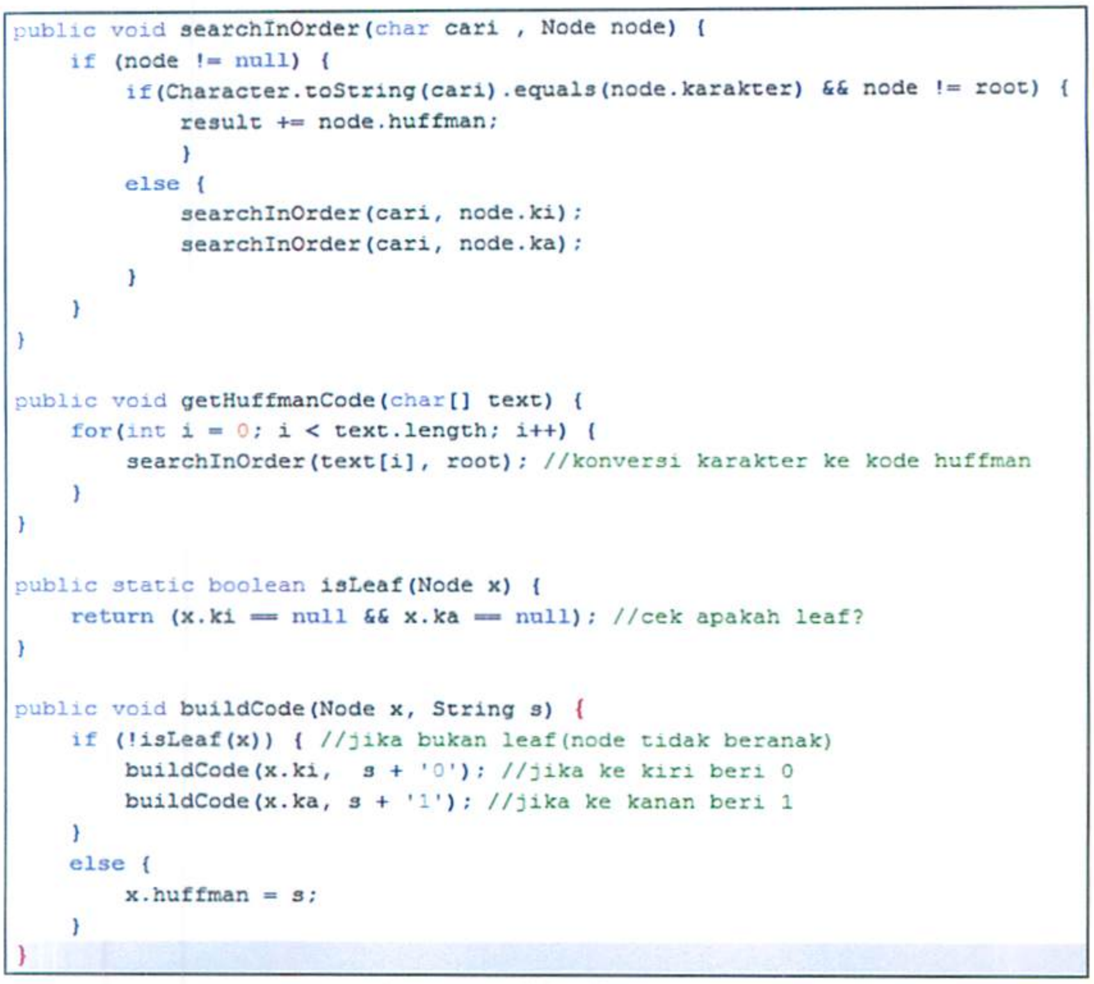
Combine two smallest frequent into one node, then sort again and repeat this process until there is only one node.



```
public void buildNode() {
    node = new Node[distinctText.size()];
    for(int i=0;i<distinctText.size();i++) { // run loop till size of arraylist
        add(frequent.get(i), distinctText.get(i), i);
    }
    if(node.length > 1) {
        for(int j =0; j< (node.length -1); j++) {
            sort();
            if(node[1] != null) {
                kii = node[0];
                kaa = node[1];
                baru = new Node ((node[0].isi + node[1].isi), (node[0].karakter + node[1].karakter), kii, kaa);
                node[0] = baru;
                node[1] = node[node.length - (1 + j)];
                node[node.length - (1 + j)] = null;
            }
        }
    }
    else {
        temp = new Node(0, "", null, null);
        baru = new Node(node[0].isi, node[0].karakter, node[0], temp);
    }
    root = baru;
}
```

Figure 18. Implementation of binary tree

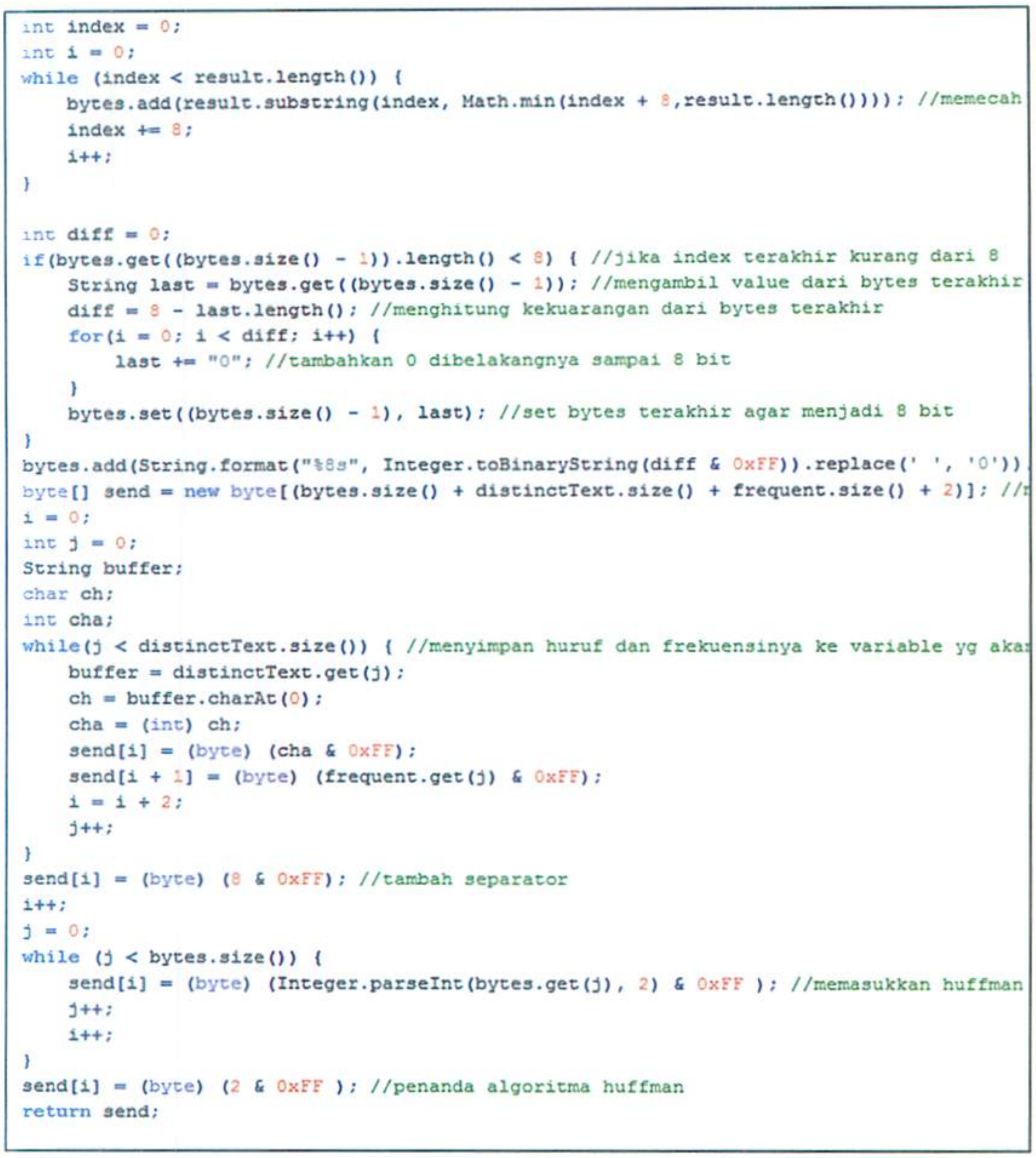
Next step is build the Huffman code in every character in the binary tree using recursive method. Left child encode with '0' and right child encode with '1'. Leaf means that node is contains 1 ascii character. the Huffman code of each character in the message stored in string variable.



```
public void searchInOrder(char cari , Node node) {
    if (node != null) {
        if(Character.toString(cari).equals(node.karakter) && node != root) {
            result += node.huffman;
            }
        else {
            searchInOrder(cari, node.ki);
            searchInOrder(cari, node.ka);
        }
    }
}

public void getHuffmanCode(char[] text) {
    for(int i = 0; i < text.length; i++) {
        searchInOrder(text[i], root); //konversi karakter ke kode huffman
    }
}

public static boolean isLeaf(Node x) {
    return (x.ki == null && x.ka == null); //cek apakah leaf?
}

public void buildCode(Node x, String s) {
    if (!isLeaf(x)) { //jika bukan leaf(node tidak beranak)
        buildCode(x.ki,  s + '0'); //jika ke kiri beri 0
        buildCode(x.ka, s + '1'); //jika ke kanan beri 1
    }
    else {
        x.huffman = s;
    }
}
```

Figure 19. Build Huffman code

Send the output as a array of byte, the output contains character of message, the frequent, Huffman code and algorithm sign in the last byte. Processing the output by split the Huffman code string of each character per 8 bit then add to arraylist. First data in the output is character and the frequent, second data is the Huffman code which already in 8 bit.

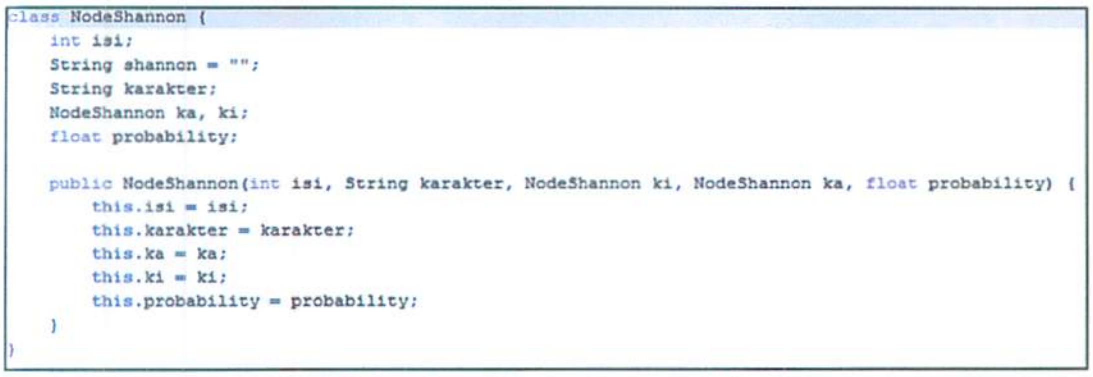


```
int index = 0;
int i = 0;
while (index < result.length()) {
    bytes.add(result.substring(index, Math.min(index + 8,result.length()))); //memecah
    index += 8;
    i++;
}

int diff = 0;
if(bytes.get((bytes.size() - 1)).length() < 8) { //jika index terakhir kurang dari 8
    String last = bytes.get((bytes.size() - 1)); //mengambil value dari bytes terakhir
    diff = 8 - last.length(); //menghitung kekuarangan dari bytes terakhir
    for(i = 0; i < diff; i++) {
        last += "0"; //tambahkan 0 dibelakangnya sampai 8 bit
    }
    bytes.set((bytes.size() - 1), last); //set bytes terakhir agar menjadi 8 bit
}
bytes.add(String.format("%8s", Integer.toBinaryString(diff & 0xFF)).replace(' ', '0'))
byte[] send = new byte[(bytes.size() + distinctText.size() + frequent.size() + 2)]; //
i = 0;
int j = 0;
String buffer;
char ch;
int cha;
while(j < distinctText.size()) { //menyimpan huruf dan frekuensinya ke variable yg aka
    buffer = distinctText.get(j);
    ch = buffer.charAt(0);
    cha = (int) ch;
    send[i] = (byte) (cha & 0xFF);
    send[i + 1] = (byte) (frequent.get(j) & 0xFF);
    i = i + 2;
    j++;
}
send[i] = (byte) (8 & 0xFF); //tambah separator
i++;
j = 0;
while (j < bytes.size()) {
    send[i] = (byte) (Integer.parseInt(bytes.get(j), 2) & 0xFF ); //memasukkan huffman
    j++;
    i++;
}
send[i] = (byte) (2 & 0xFF ); //penanda algoritma huffman
return send;
```

Figure 20. Processing the output

### 5.1.1.3. Shannon Fano

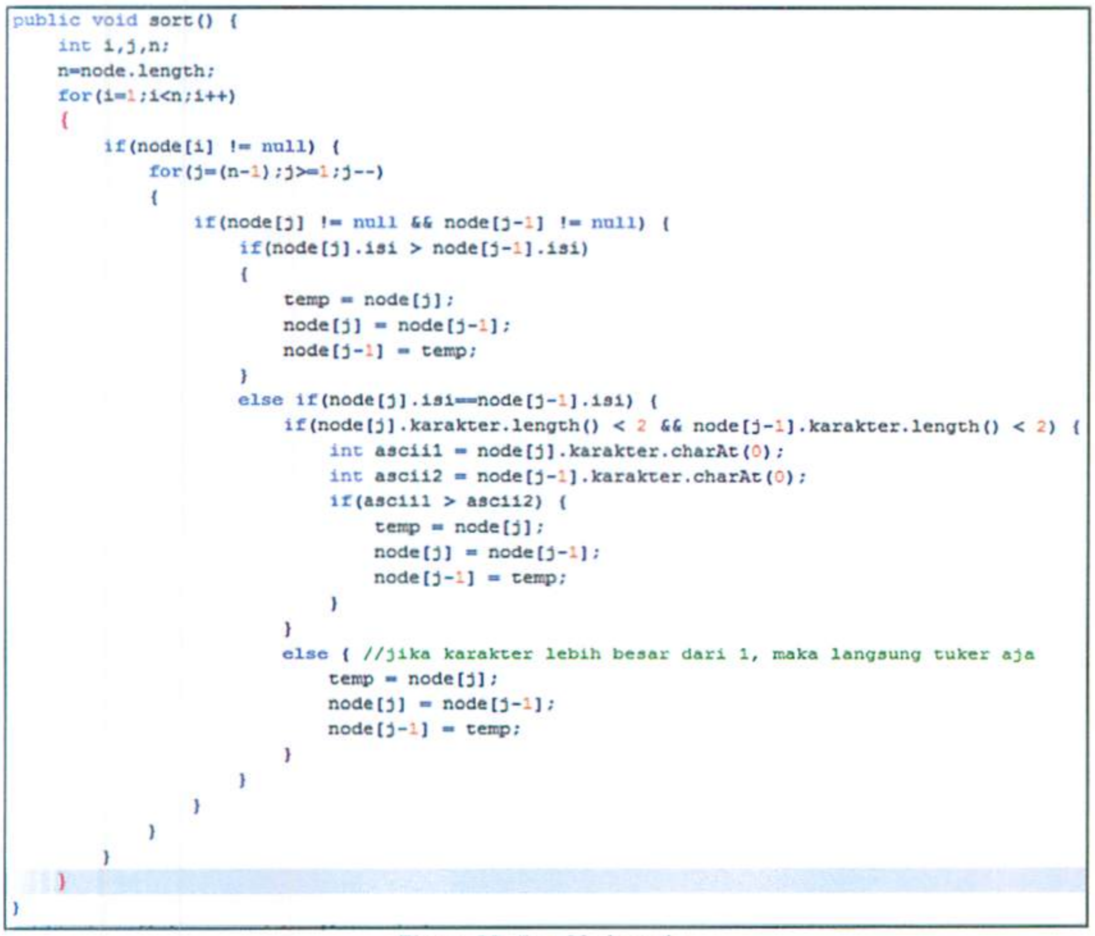
The input of this process is array of character. make array of node from each character of this input, the node consist of character, frequent of character, probabilities and Shannon code.



```
class NodeShannon {
    int isi;
    String shannon = "";
    String karakter;
    NodeShannon ka, ki;
    float probability;

    public NodeShannon(int isi, String karakter, NodeShannon ki, NodeShannon ka, float probability) {
        this.isi = isi;
        this.karakter = karakter;
        this.ka = ka;
        this.ki = ki;
        this.probability = probability;
    }
}
```

Figure 21. NodeShannon class code

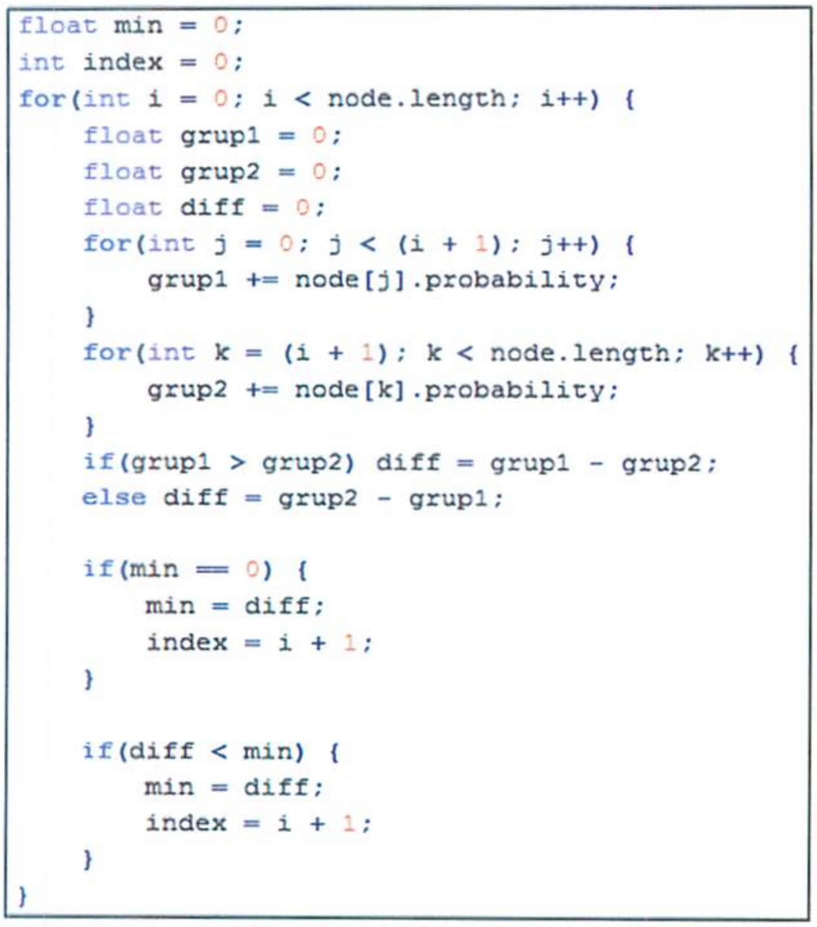
The first step to build tree is sort ascending the node by frequent of each character, if node frequent same as another node then sort by ascii value.



```
public void sort() {
    int i,j,n;
    n=node.length;
    for(i=1;i<n;i++)
    {
        if(node[i] != null) {
            for(j=(n-1);j>=1;j--)
            {
                if(node[j] != null && node[j-1] != null) {
                    if(node[j].isi > node[j-1].isi)
                    {
                        temp = node[j];
                        node[j] = node[j-1];
                        node[j-1] = temp;
                    }
                    else if(node[j].isi==node[j-1].isi) {
                        if(node[j].karakter.length() < 2 && node[j-1].karakter.length() < 2) {
                            int ascii1 = node[j].karakter.charAt(0);
                            int ascii2 = node[j-1].karakter.charAt(0);
                            if(ascii1 > ascii2) {
                                temp = node[j];
                                node[j] = node[j-1];
                                node[j-1] = temp;
                            }
                        }
                        else { //jika karakter lebih besar dari 1, maka langsung tuker aja
                            temp = node[j];
                            node[j] = node[j-1];
                            node[j-1] = temp;
                        }
                    }
                }
            }
        }
    }
}
```

Figure 22. Sort Node code

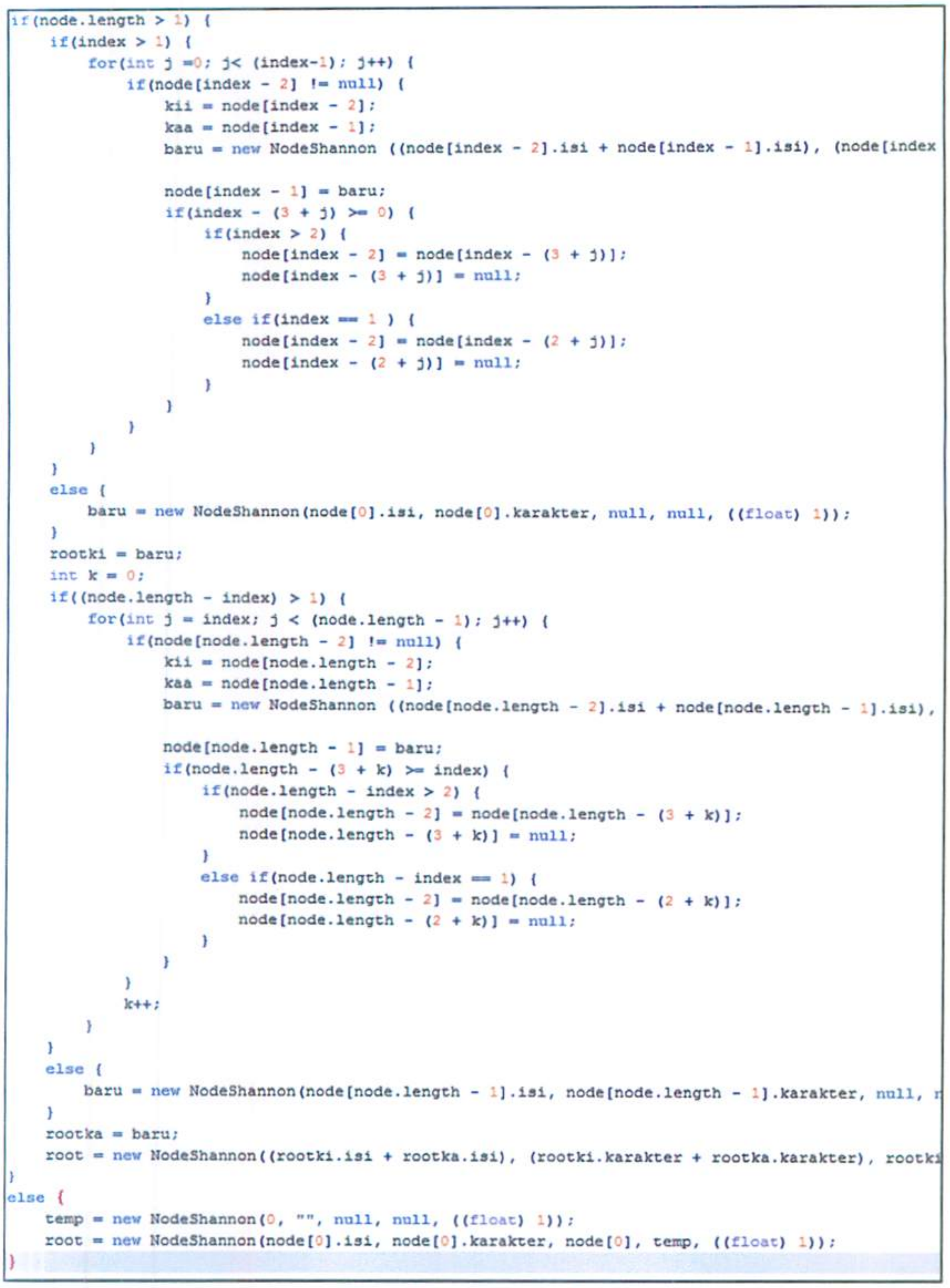
Divided into two based on the difference in the probability of at least two groups of characters of the value of the sorted earlier.



```
float min = 0;
int index = 0;
for(int i = 0; i < node.length; i++) {
    float grup1 = 0;
    float grup2 = 0;
    float diff = 0;
    for(int j = 0; j < (i + 1); j++) {
        grup1 += node[j].probability;
    }
    for(int k = (i + 1); k < node.length; k++) {
        grup2 += node[k].probability;
    }
    if(grup1 > grup2) diff = grup1 - grup2;
    else diff = grup2 - grup1;

    if(min == 0) {
        min = diff;
        index = i + 1;
    }

    if(diff < min) {
        min = diff;
        index = i + 1;
    }
}
```

Figure 23. Divide group code

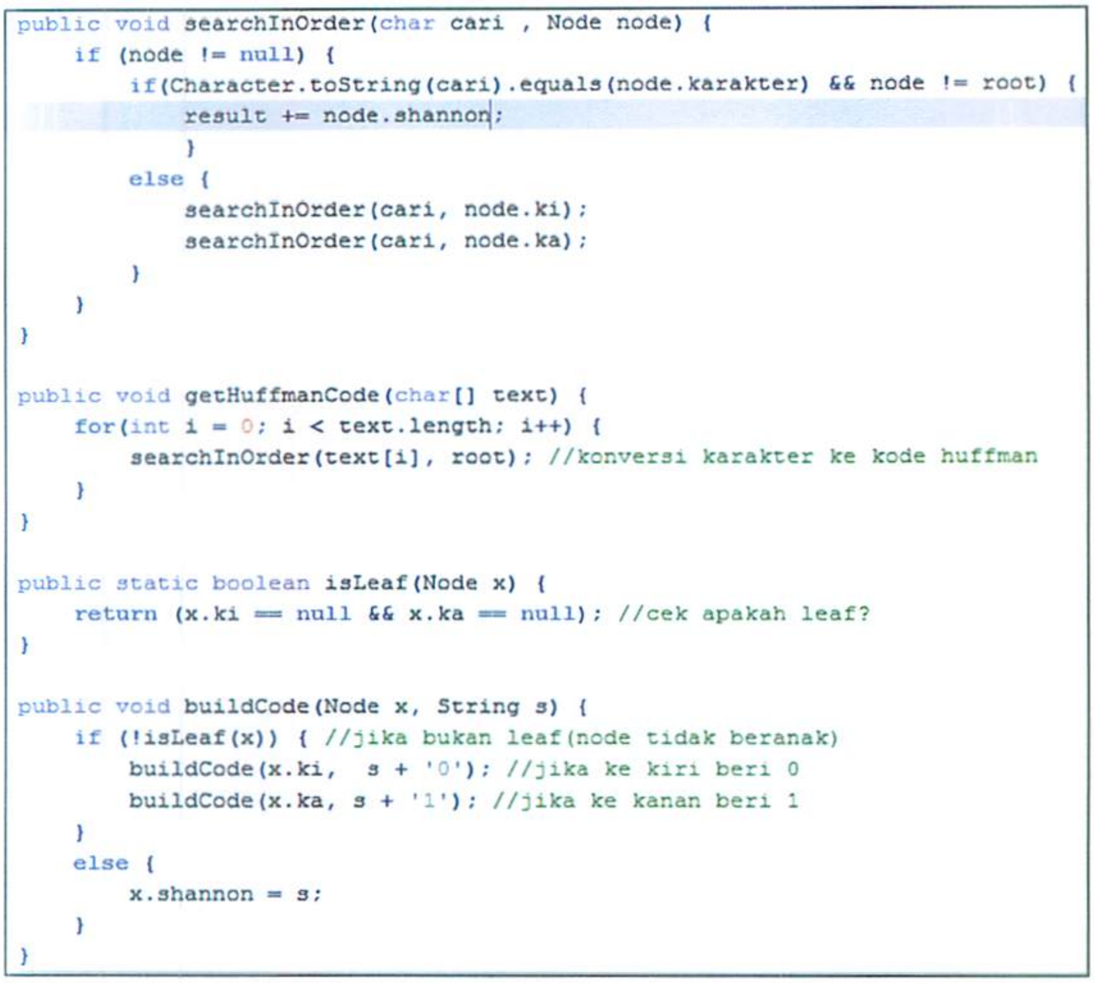
Build the binary tree based on two group, group 1 are considered left side of binary tree and group two are considered right side of binary tree. Combine the last node and 1 node before last into one on every group until there is one node on every group. Then combine left side and right side of binary tree into one.



```
if(node.length > 1) {
    if(index > 1) {
        for(int j =0; j< (index-1); j++) {
            if(node[index - 2] != null) {
                kii = node[index - 2];
                kaa = node[index - 1];
                baru = new NodeShannon ((node[index - 2].isi + node[index - 1].isi), (node[index

                node[index - 1] = baru;
                if(index - (3 + j) >= 0) {
                    if(index > 2) {
                        node[index - 2] = node[index - (3 + j)];
                        node[index - (3 + j)] = null;
                    }
                    else if(index == 1 ) {
                        node[index - 2] = node[index - (2 + j)];
                        node[index - (2 + j)] = null;
                    }
                }
            }
        }
    }
    else {
        baru = new NodeShannon(node[0].isi, node[0].karakter, null, null, ((float) 1));
    }
    rootki = baru;
    int k = 0;
    if((node.length - index) > 1) {
        for(int j = index; j < (node.length - 1); j++) {
            if(node[node.length - 2] != null) {
                kii = node[node.length - 2];
                kaa = node[node.length - 1];
                baru = new NodeShannon ((node[node.length - 2].isi + node[node.length - 1].isi),

                node[node.length - 1] = baru;
                if(node.length - (3 + k) >= index) {
                    if(node.length - index > 2) {
                        node[node.length - 2] = node[node.length - (3 + k)];
                        node[node.length - (3 + k)] = null;
                    }
                    else if(node.length - index == 1) {
                        node[node.length - 2] = node[node.length - (2 + k)];
                        node[node.length - (2 + k)] = null;
                    }
                }
            }
            k++;
        }
    }
    else {
        baru = new NodeShannon(node[node.length - 1].isi, node[node.length - 1].karakter, null, r
    }
    rootka = baru;
    root = new NodeShannon((rootki.isi + rootka.isi), (rootki.karakter + rootka.karakter), rootki
}
else {
    temp = new NodeShannon(0, "", null, null, ((float) 1));
    root = new NodeShannon(node[0].isi, node[0].karakter, node[0], temp, ((float) 1));
}
```

Figure 24. Build tree code

Next step is build the Shannon code in every character in the binary tree using recursive method. Left child encode with '0' and right child encode with '1'. Leaf means that node is contains 1 ascii character. the Shannon code of each character in the message stored in string variable.



```
public void searchInOrder(char cari , Node node) {
    if (node != null) {
        if(Character.toString(cari).equals(node.karakter) && node != root) {
            result += node.shannon;
            }
        else {
            searchInOrder(cari, node.ki);
            searchInOrder(cari, node.ka);
        }
    }
}

public void getHuffmanCode(char[] text) {
    for(int i = 0; i < text.length; i++) {
        searchInOrder(text[i], root); //konversi karakter ke kode huffman
    }
}

public static boolean isLeaf(Node x) {
    return (x.ki == null && x.ka == null); //cek apakah leaf?
}

public void buildCode(Node x, String s) {
    if (!isLeaf(x)) { //jika bukan leaf(node tidak beranak)
        buildCode(x.ki,  s + '0'); //jika ke kiri beri 0
        buildCode(x.ka, s + '1'); //jika ke kanan beri 1
    }
    else {
        x.shannon = s;
    }
}
```

Figure 25. Build Shannon Code

Send the output as a array of byte, the output contains character of message, the frequent, Shannon code and algorithm sign in the last byte. Processing the output by split the Huffman code string of each character per 8 bit then add to arraylist. First data in the output is character and the frequent, second data is the Shannon code which already in 8 bit.

```
int index = 0;
int i = 0;
while (index < result.length()) {
    bytes.add(result.substring(index, Math.min(index + 8,result.length()))); //memecah
    index += 8;
    i++;
}

int diff = 0;
if(bytes.get((bytes.size() - 1)).length() < 8) { //jika index terakhir kurang dari 8
    String last = bytes.get((bytes.size() - 1)); //mengambil value dari bytes terakhir
    diff = 8 - last.length(); //menghitung kekuarangan dari bytes terakhir
    for(i = 0; i < diff; i++) {
        last += "0"; //tambahkan 0 dibelakangnya sampai 8 bit
    }
    bytes.set((bytes.size() - 1), last); //set bytes terakhir agar menjadi 8 bit
}
bytes.add(String.format("%8s", Integer.toBinaryString(diff & 0xFF)).replace(' ', '0'))
byte[] send = new byte[(bytes.size() + distinctText.size() + frequent.size() + 2)]; //
i = 0;
int j = 0;
String buffer;
char ch;
int cha;
while(j < distinctText.size()) { //menyimpan huruf dan frekuensinya ke variable yg aka
    buffer = distinctText.get(j);
    ch = buffer.charAt(0);
    cha = (int) ch;
    send[i] = (byte) (cha & 0xFF);
    send[i + 1] = (byte) (frequent.get(j) & 0xFF);
    i = i + 2;
    j++;
}
send[i] = (byte) (8 & 0xFF); //tambah separator
i++;
j = 0;
while (j < bytes.size()) {
    send[i] = (byte) (Integer.parseInt(bytes.get(j), 2) & 0xFF );
    j++;
    i++;
}
send[i] = (byte) (2 & 0xFF );
return send;
```

Figure 26. Processing the output

## 5.1.2. Decompression

### 5.1.2.1. LZW

The input of this process is array of byte. First process is building the dictionary, the dictionary consist of 0-255 decimal in ASCII table.

```
for(int i = 0; i < 256; i++) {
    kamus.add(Character.toString((char) i));
}
```

Figure 27. Building dictionary code

The input need to convert to 12 bits because the maximum number that can be accommodated reach 4096 number of entries or 12 bit and stored in arraylist.

```
String binary = "";
int j;
for(j = 0; j < (message.length - 2); j++) {
    binary += String.format("%8s", Integer.toBinaryString(message[j] & 0xFF)).replace(' ', '0');
}
binary = binary.substring(0, (binary.length() - (int) message[j]));
int index = 0;
ArrayList<String> code = new ArrayList<String>();
while (index < binary.length()) {
    code.add(binary.substring(index, Math.min(index + 12,binary.length())));
    index += 12;
}
```

Figure 28. Convert 8 bit data into 12 bit code

Implement the pseudocode decompression of LZW algorithm.

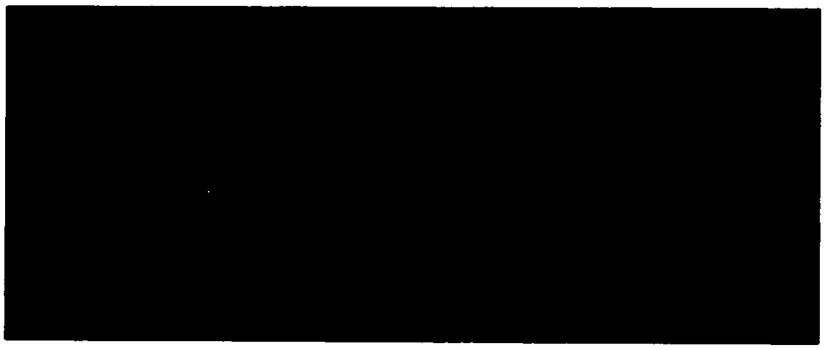


Figure 29. Psudocode decompression of LZW algorithm

```
int k = Integer.parseInt(code.get(0), 2);
String s = kamusawal.get(k);
result = s;
String entry = "";
char[] masuk;
for(int i = 1; i < code.size(); i++) {
    k = Integer.parseInt(code.get(i), 2);
    if(k < 0) k = 256 + k;
    entry = kamusawal.get(k);
    result += entry;
    masuk = entry.toCharArray();
    kamusawal.add(s + masuk[0]);
    s = entry;
}
```

Figure 30. Decompression code of LZW algorithm

#### 5.1.2.2. Huffman

The input of this process is array of byte. First process is building tree like the compression from the character and frequent that contains in the message.

Read all of Huffman code, if 1 code go to the right child, if 0 code go to the left child until found leaf. The leaf character is decode character of the Huffman code. Loop until the end of Huffman code.

```
while(j < huruf.length) {
    Node x = root;
    while (!isLeaf(x)) {
        if (huruf[j] == '1') x = x.ka;
        else x = x.ki;
        j++;
    }
    result += x.karakter;
}
```

Figure 31. Huffman decode process code

#### 5.1.2.3. Shannon Fano

The input of this process is array of byte. First process is building tree like the compression from the character and frequent that contains in the message.

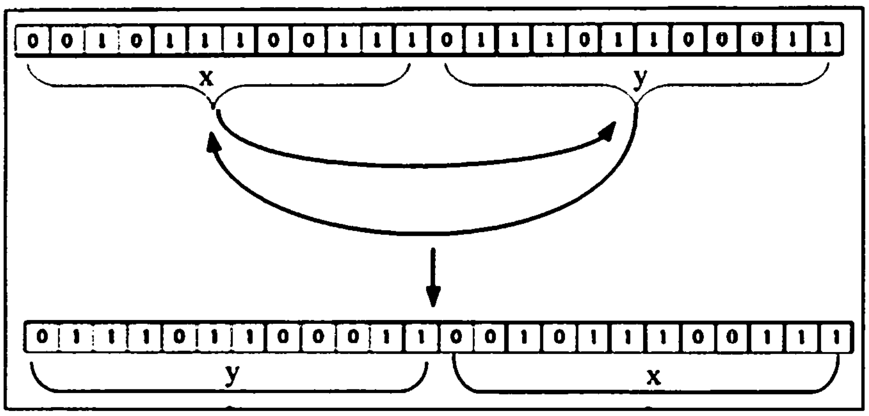
Read all of Shannon code, if 1 code go to the right child, if 0 code go to the left child until found leaf. The leaf character is decode character of the Shannon code. Loop until the end of Shannon code.

```
while(j < huruf.length) {
    NodeShannon x = root;
    while (!isLeaf(x)) {
        if (huruf[j] == '1') x = x.ka;
        else x = x.ki;
        j++;
    }
    result += x.karakter;
}
```

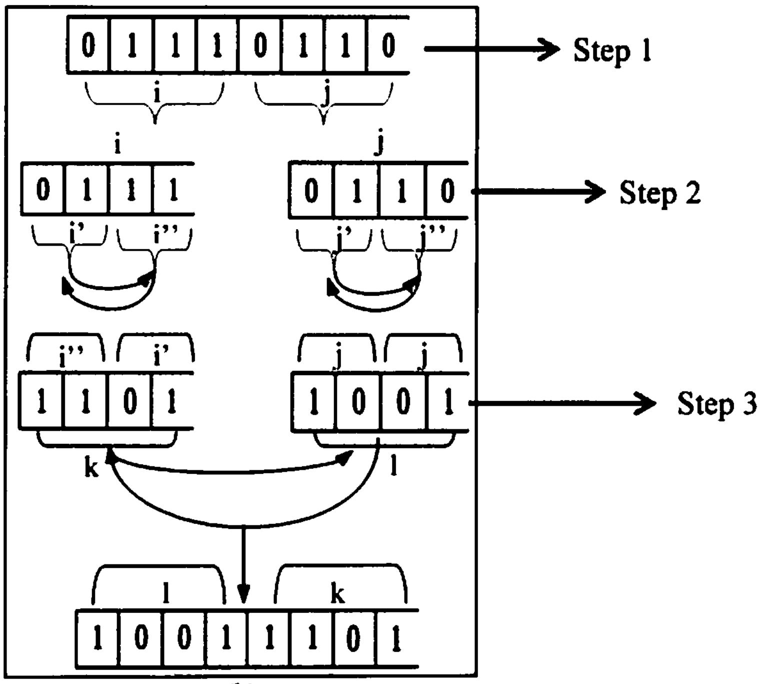
Figure 32. Shannon Fano decode process code

### 5.1.2. Encryption

The input of this process is compressed message with data type array of byte. Encryption using Caesar algorithm combined with custom byte manipulation. First process is plus the decimal value of each character until the end of the message with the public key in preferences. Then combine with custom byte manipulation. Here is the concept of custom byte manipulation.



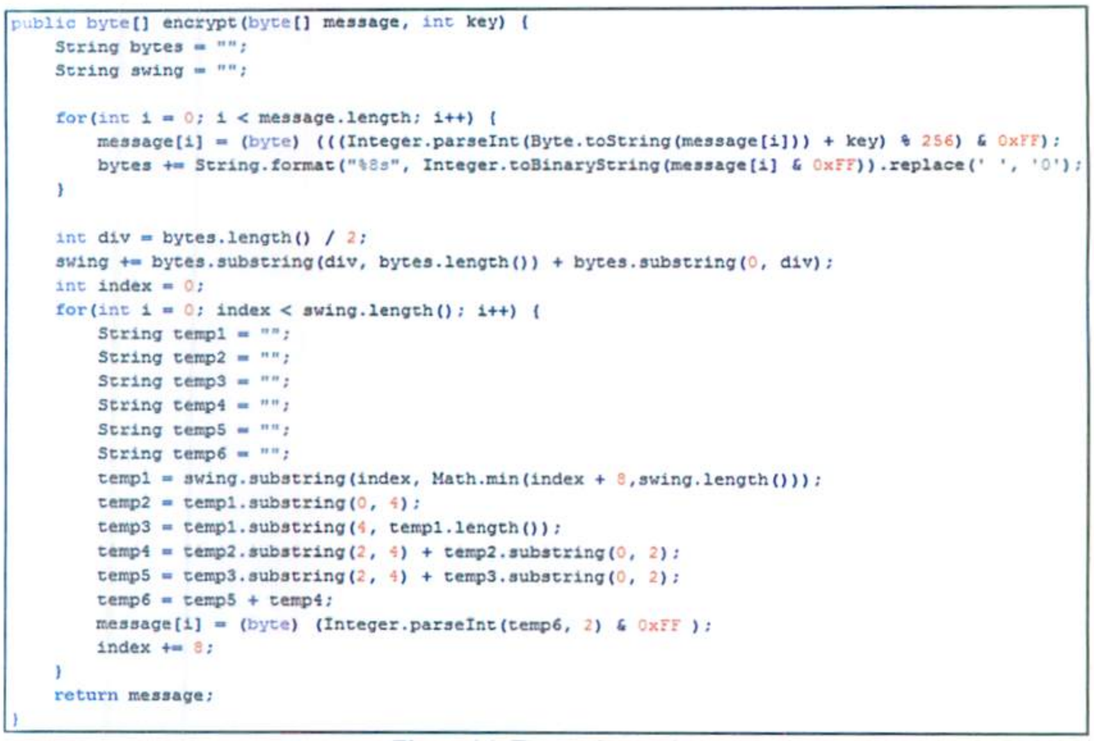
(a)



(b)

Figure 33. Custom byte manipulation concept

Figure 33 (a) shows that all of binary string messages are divided into 2 part and then swap. Figure 33 (b) Step 1 - every byte in the message are divided into 2 part and then swap. Step 2 - each 4 bit are divided by 2 again then swap. Step 3 - after that swap again the 4 bit which swapped in Step 2. This is the code:



```
public byte[] encrypt(byte[] message, int key) {
    String bytes = "";
    String swing = "";

    for(int i = 0; i < message.length; i++) {
        message[i] = (byte) (((Integer.parseInt(Byte.toString(message[i])) + key) % 256) & 0xFF);
        bytes += String.format("%8s", Integer.toBinaryString(message[i] & 0xFF)).replace(' ', '0');
    }

    int div = bytes.length() / 2;
    swing += bytes.substring(div, bytes.length()) + bytes.substring(0, div);
    int index = 0;
    for(int i = 0; index < swing.length(); i++) {
        String temp1 = "";
        String temp2 = "";
        String temp3 = "";
        String temp4 = "";
        String temp5 = "";
        String temp6 = "";
        temp1 = swing.substring(index, Math.min(index + 8,swing.length()));
        temp2 = temp1.substring(0, 4);
        temp3 = temp1.substring(4, temp1.length());
        temp4 = temp2.substring(2, 4) + temp2.substring(0, 2);
        temp5 = temp3.substring(2, 4) + temp3.substring(0, 2);
        temp6 = temp5 + temp4;
        message[i] = (byte) (Integer.parseInt(temp6, 2) & 0xFF );
        index += 8;
    }
    return message;
}
```

Figure 34. Encryption code

### 5.1.3. Decryption

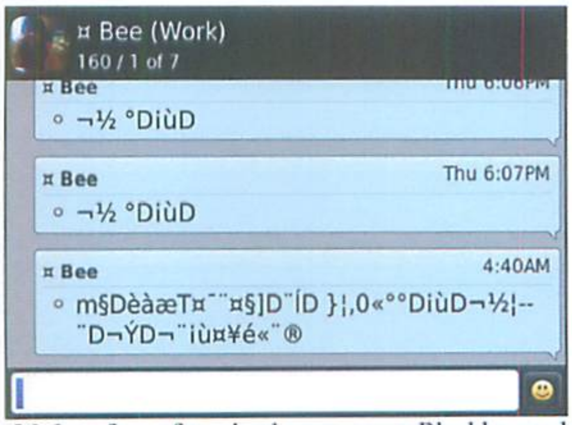
The input of this process is encrypted & compressed message in the array of byte. First process is reverse of custom byte manipulation process and then minus the decimal value of each character until the end of the message with the public key in preferences.

```
public byte[] decrypt(byte[] message, int key) {
    String bytes = "";
    String swing = "";
    for(int i = 0; i < message.length; i++) {
        bytes += String.format("%8s", Integer.toBinaryString(message[i] & 0xFF)).replace(' ', '0');
    }
    int index = 0;
    for(int i = 0; index < bytes.length(); i++) {
        String temp1 = "";
        String temp2 = "";
        String temp3 = "";
        String temp4 = "";
        String temp5 = "";
        String temp6 = "";
        temp1 = bytes.substring(index, Math.min(index + 8,bytes.length()));
        temp2 = temp1.substring(0, 4);
        temp3 = temp1.substring(4, temp1.length());
        temp4 = temp2.substring(2, 4) + temp2.substring(0, 2);
        temp5 = temp3.substring(2, 4) + temp3.substring(0, 2);
        temp6 = temp5 + temp4;
        swing += temp6;
        index += 8;
    }
    int div = swing.length() / 2;
    swing = swing.substring(div, swing.length()) + swing.substring(0, div);
    index = 0;
    for(int i = 0; index < swing.length(); i++) {
        message[i] = (byte) (Integer.parseInt(swing.substring(index, Math.min(index + 8,swing.length())), 2) & 0xFF );
        index += 8;
    }
    for(int i = 0; i < message.length; i++) {
            message[i] = (byte) (((Integer.parseInt(Byte.toString(message[i])) - key) % 256) & 0xFF);
    }
    return message;
}
```

Figure 35. Decryption Code

## 5.2. Testing

To know the result of this application, it need some test to be done. First test is sending message to another platform, in symbian doesn't receive any message from the sender. It could be happen because this application is sending message from specific port, therefore to receive that message the device must be has specific port too. In BlackBerry platform message could be received, but it show encrypted and compressed message (Figure 15) because in BlackBerry device there is no decryptor and decompressor application.



¤ Bee (Work)
160 / 1 of 7

¤ Bee Thu 6:06PM
¬½ °DiùD

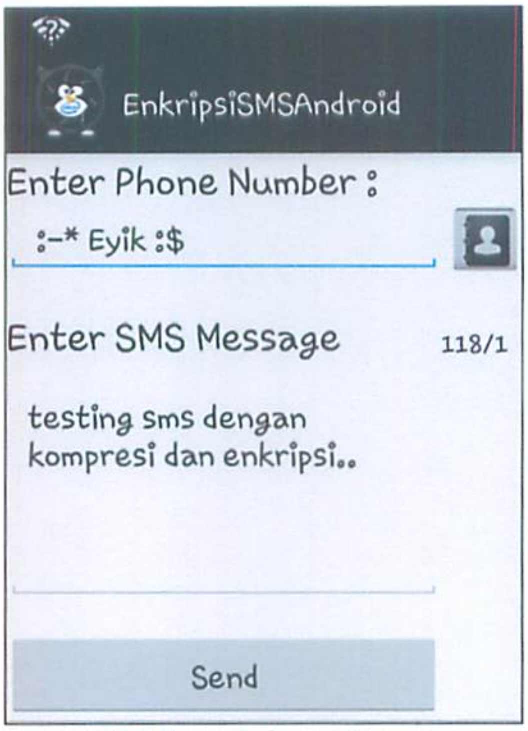
¤ Bee Thu 6:07PM
¬½ °DiùD

¤ Bee 4:40AM
m§DèàæT¤¯¨¤§]D¨ÍD }¦,0«°°DiùD¬½¦--
¨D¬ÝD¬¨iù¤¥é«¨®

Figure 36. Interface of received message on Blackberry platform

Second test is sending message to Android platform without this application. In this test, message is not received.

Third test is sending message to Android platform with this application.

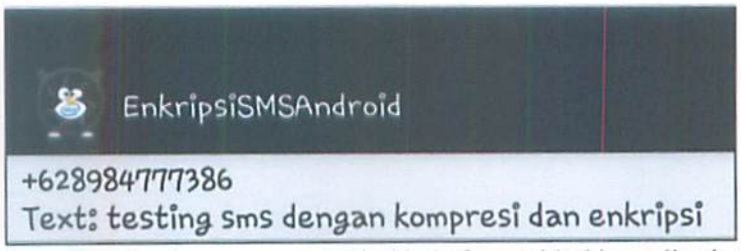


:-* Eyik :$

testing sms dengan kompresi dan enkripsi..

Figure 37. Example for sending message

If the public key receiver is same with public key sender, the message can show original message from sender.

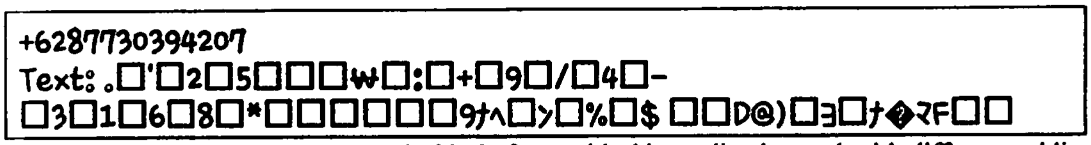


+6289847773 86

Text: testing sms dengan kompresi dan enkripsi

Figure 38. Interface of received message on Android platform with this application and with same public key

But if the public key receiver is different with public key sender, the message show compressed message with wrong decryption.



```
+6287730394207
Text: .□'□2□5□□□w□:□+□9□/□4□-
□3□1□6□8□*□□□□□□□9ナ^□ゝ□%□$ □□D@)□ョ□ナ�マF□□
```

Figure 39. Received message on Android platform with this application and with different public key

This is the performance result of each compression algorithm. Data in this test is different word and length.

Table 2. Compression Result

| Data | Message Length | Huffman (Char) | Shannon Fano (Char) | LZW (Char) |
|---|---|---|---|---|
| Data 1 | 144 | 126 | 147 | 166 |
| Data 2 | 272 | 199 | 240 | 286 |
| Data 3 | 544 | 346 | 432 | 508 |
| Data 4 | 667 | 413 | 524 | 602 |
| Data 5 | 756 | 464 | 593 | 667 |
| Data 6 | 836 | 508 | 650 | 727 |
| Data 7 | 922 | 556 | 713 | 788 |
| Data 8 | 1080 | 641 | 824 | 901 |
| Data 9 | 1306 | 764 | 969 | 1045 |
| Data 10 | 1435 | 831 | 1056 | 1120 |
| Data 11 | 1705 | 971 | 1261 | 1273 |
| Data 12 | 1850 | 1047 | 1362 | 1351 |
| Data 13 | 1978 | 1114 | 1451 | 1415 |
| Data 14 | 2249 | 1253 | 1631 | 1544 |
| Data 15 | 2373 | 1319 | 1721 | 1613 |

From Table 2 it can bee seen that by using Huffman compression gives the best result which give the smallest compression ratio. Shannon fano compression is the second best after Huffman compression and the last is LZW compression.

## 5.3. Interface

### 5.3.1. Main Menu Window

The main menu window contains Create Message, Inbox, Send Item, Settings and Help.

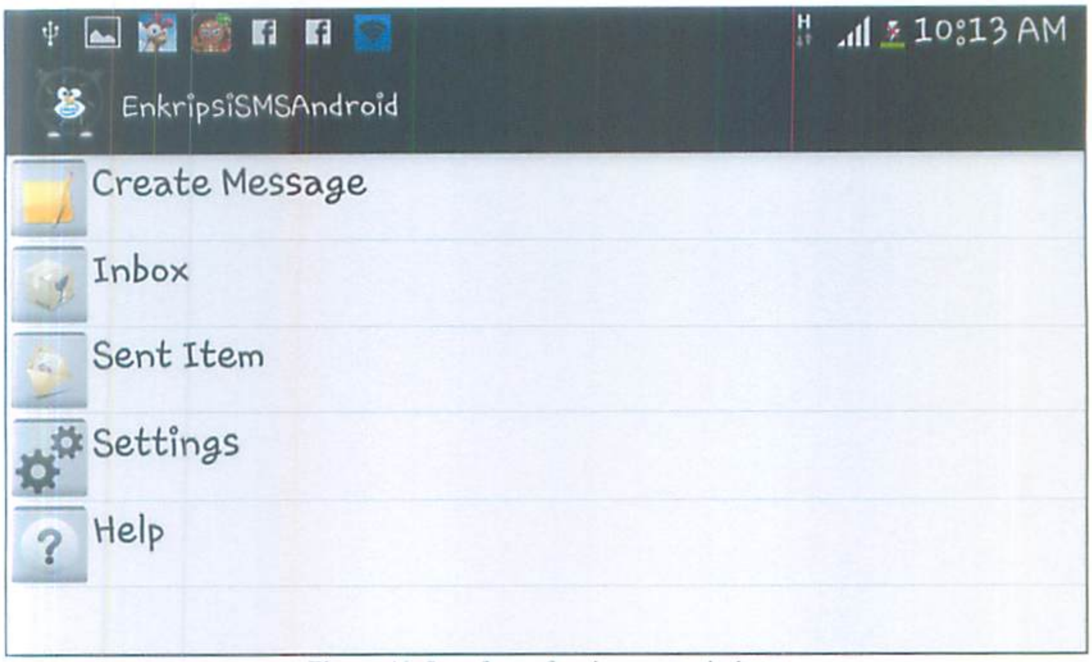


Figure 40. Interface of main menu window

### 5.3.2. Create Message

This menu for create message, user just input the recipient and the message.

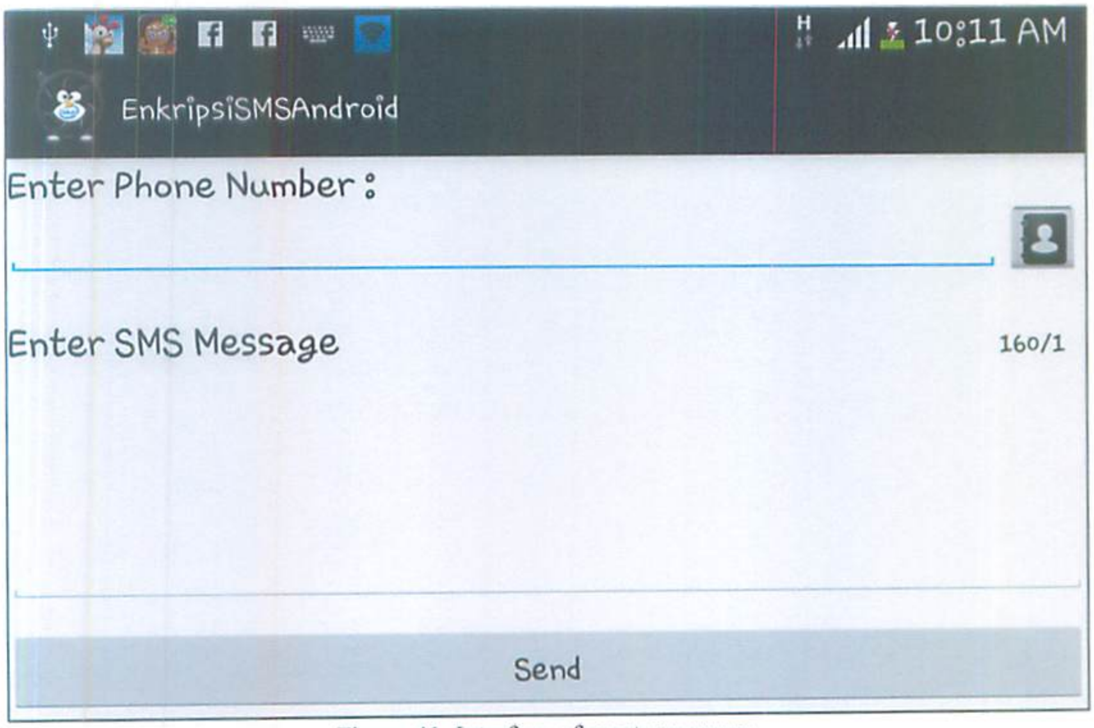
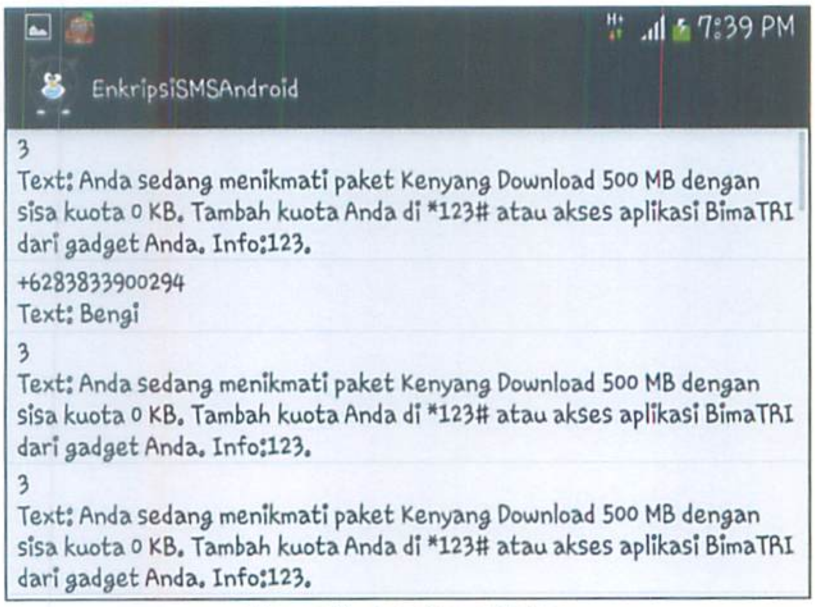


Figure 41. Interface of create message

### 5.3.3. Inbox

Inbox menu show message which is received from the sender.



3
Text: Anda sedang menikmati paket Kenyang Download 500 MB dengan sisa kuota 0 KB. Tambah kuota Anda di *123# atau akses aplikasi BimaTRI dari gadget Anda. Info:123.

+6283833900294
Text: Bengi

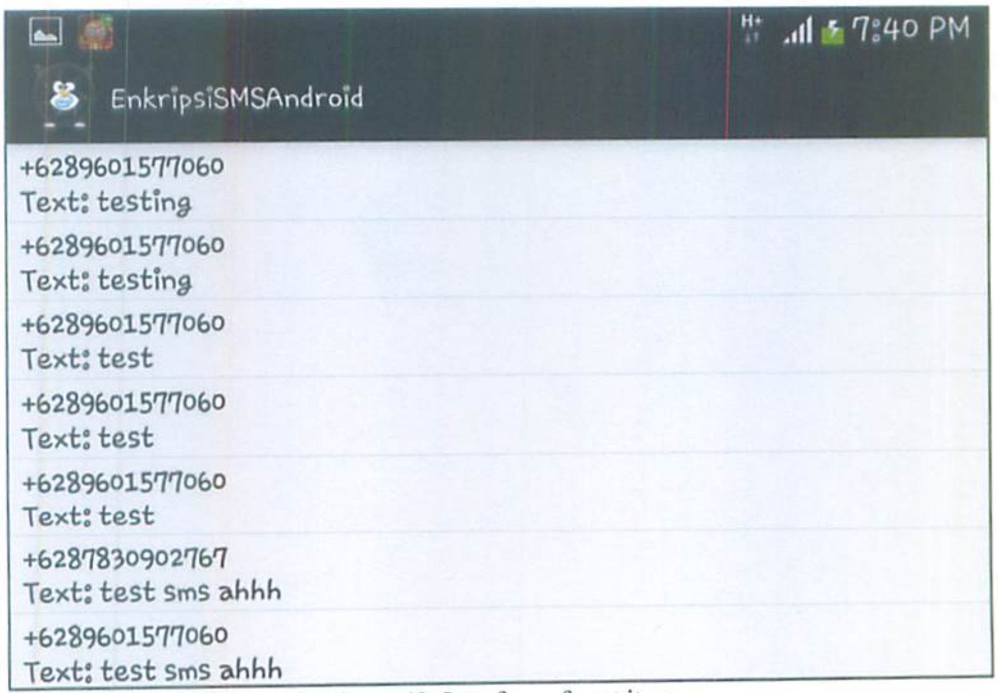
3
Text: Anda sedang menikmati paket Kenyang Download 500 MB dengan sisa kuota 0 KB. Tambah kuota Anda di *123# atau akses aplikasi BimaTRI dari gadget Anda. Info:123.

3
Text: Anda sedang menikmati paket Kenyang Download 500 MB dengan sisa kuota 0 KB. Tambah kuota Anda di *123# atau akses aplikasi BimaTRI dari gadget Anda. Info:123.

Figure 42. Interface of Inbox

### 5.3.4. Sent Item

Sent item show message which is sent to receiver.



+6289601577060
Text: testing

+6289601577060
Text: testing

+6289601577060
Text: test

+6289601577060
Text: test

+6289601577060
Text: test

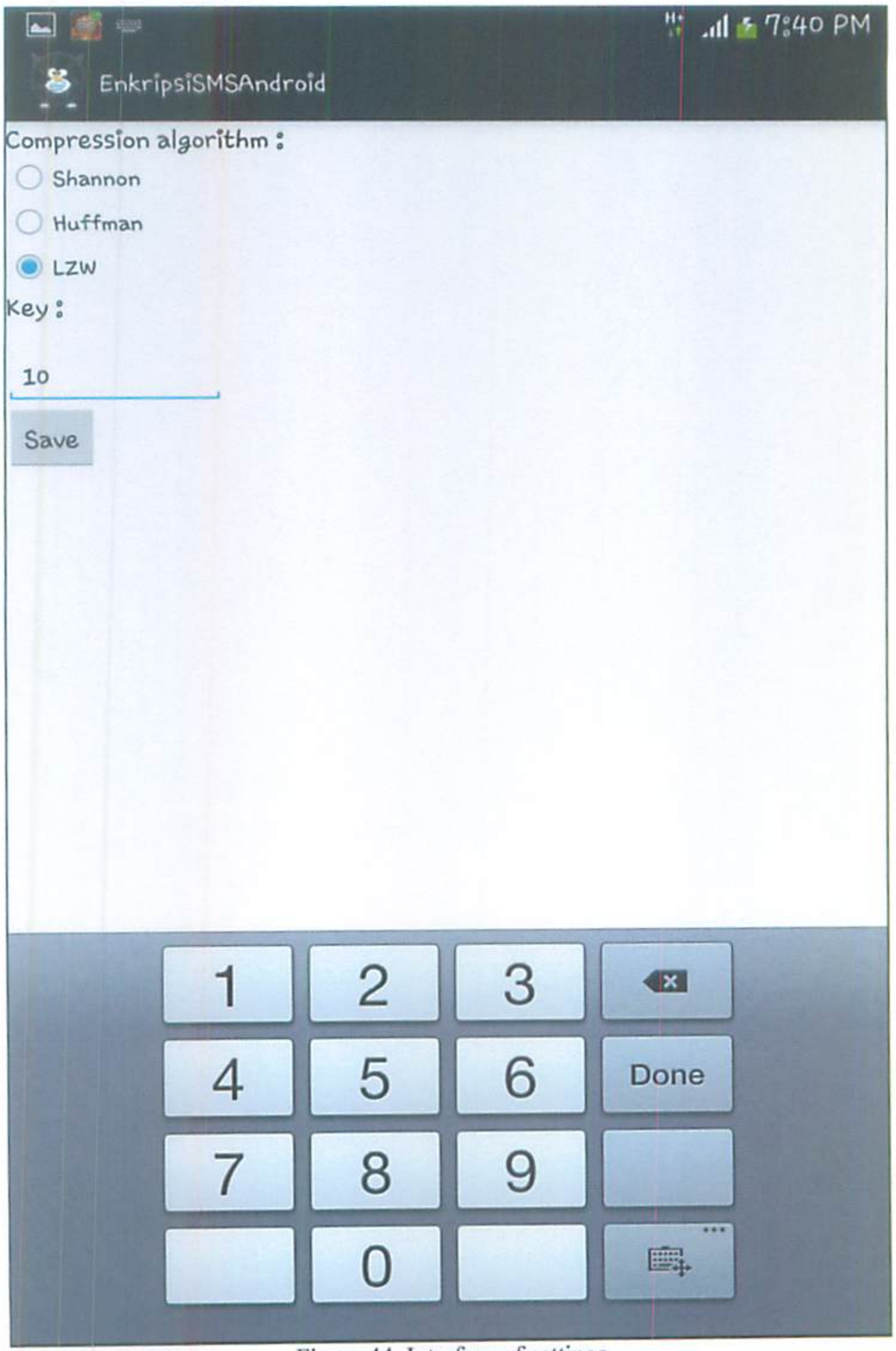
+6287830902767
Text: test sms ahhh

+6289601577060
Text: test sms ahhh

Figure 43. Interface of sent item

### 5.3.5. Settings

This menu to choose compression algorithm and public key for encryption.



Compression algorithm :

Shannon [✓ LZW] Huffman

Key :

10

Save

Figure 44. Interface of settings

### 5.3.6. Help

This menu for instruction how to use this application and a little bit explanation about algorithm.

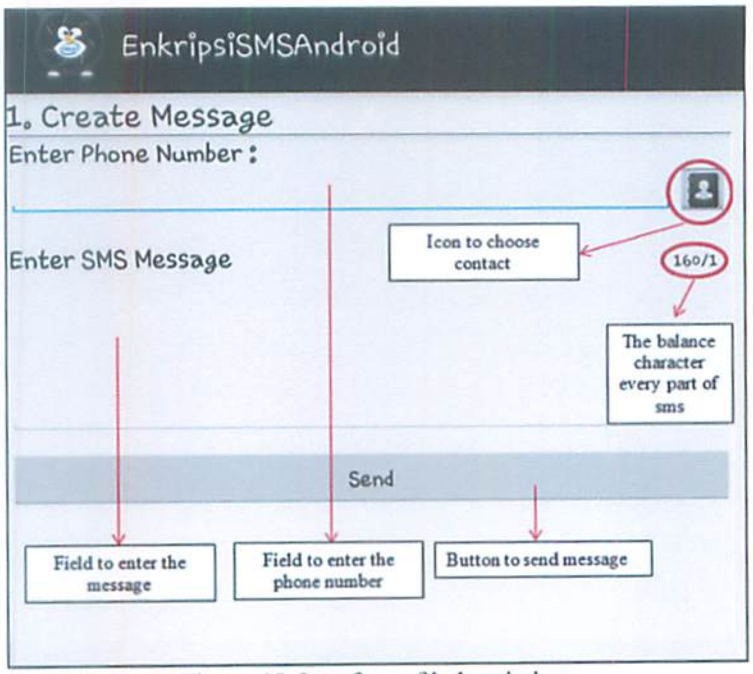


Figure 45. Interface of help window