PRACTICAL : 3

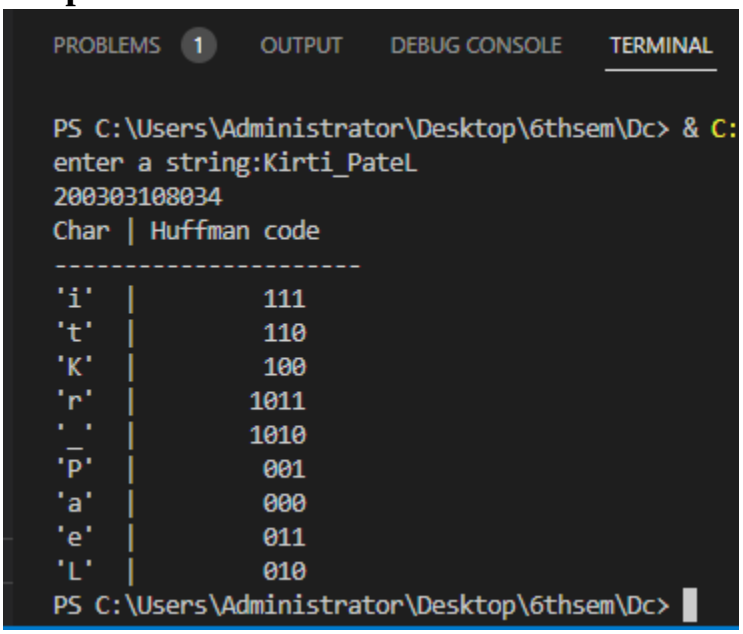
Aim: Implement Huffman Code(HC) to generate binary code when symbol and probabilities are given.

Code:

```
string = input("enter a string:")
class NodeTree(object):
    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right
    def children(self):
        return (self.left, self.right)
    def nodes(self):
        return (self.left, self.right)
    def __str__(self):
        return '%s_%s' % (self.left, self.right)
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        1
        freq[c] = 1
freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)
nodes = freq
while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))
    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)
huffmanCode = huffman_code_tree(nodes[0][0])
print('Char | Huffman code ')
print('----------------------')
```

```
for (char,frequency) in freq:
    print('%-4r |%12s' % (char, huffmanCode[char]))
```

Output:



PRACTICAL : 4

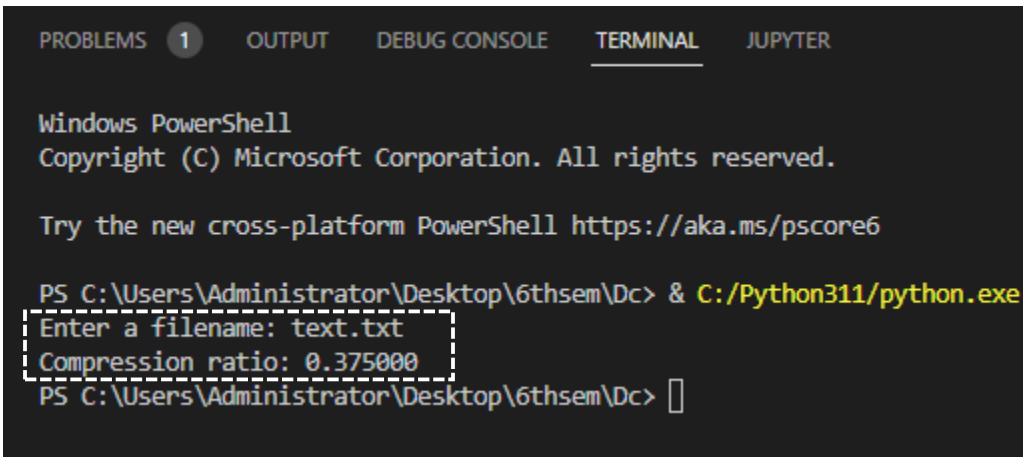
Aim: Implement Huffman code which can compress given file and decompress compressed file.

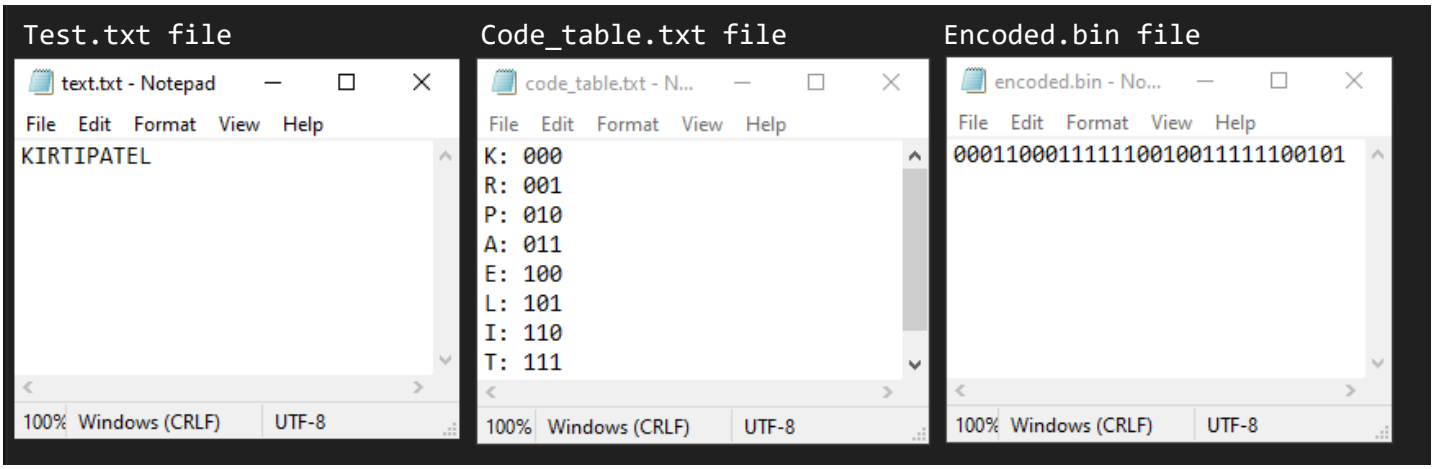
Code:

```
import heapq
from collections import Counter, namedtuple
class HuffmanNode(namedtuple("HuffmanNode", ["left", "right"])):
    def walk(self, code, acc):
        self.left.walk(code, acc + "0")
        self.right.walk(code, acc + "1")
class LeafNode(namedtuple("LeafNode", ["char"])):
    def walk(self, code, acc):
        code[self.char] = acc or "0"
def huffman_encode(s):
    h = []
    for ch, freq in Counter(s).items():
        h.append((freq, len(h), LeafNode(ch)))
# Build heap
    heapq.heapify(h)
    count = len(h)
    while len(h) > 1:
        freq1, _count1, left = heapq.heappop(h)
        freq2, _count2, right = heapq.heappop(h)
        heapq.heappush(h, (freq1 + freq2, count, HuffmanNode(left, right)))
        count += 1
    code = {}
    if h:
        [(_freq, _count, root)] = h
        root.walk(code, "")
    return code
def compress(text):
    huff = huffman_encode(text)
    encoded = "".join(huff[ch] for ch in text)
    ratio = len(encoded) / ( 8.0 * len(text))
    return encoded, huff, ratio
def decompress(encoded, huff):
    reverse_huff = {huff[ch]: ch for ch in huff}
    current_code = ""
    decoded = ""
    for bit in encoded:
        current_code += bit
        if current_code in reverse_huff:
```

```
            character = reverse_huff[current_code]
            decoded += character
            current_code = ""
        return decoded
def main():
    file_name = input("Enter a filename: ")
    with open(file_name) as f:
        text = f.read()
    encoded, huff, ratio = compress(text)
    print("Compression ratio: %f" % ratio)
    with open("encoded.bin", "wb") as f:
        f.write(bytes(encoded, "utf-8"))
    with open("code_table.txt", "w") as f:
        for char in huff:
            f.write("%s: %s\n" % (char, huff[char]))
    decoded = decompress(encoded, huff)
    with open("decoded.txt", "w") as f:
        f.write(decoded)
main()
```

Output:





PRACTICAL : 2

Aim: Write a program to generate binary code in case of arithmetic coding.

Code:

ar.py

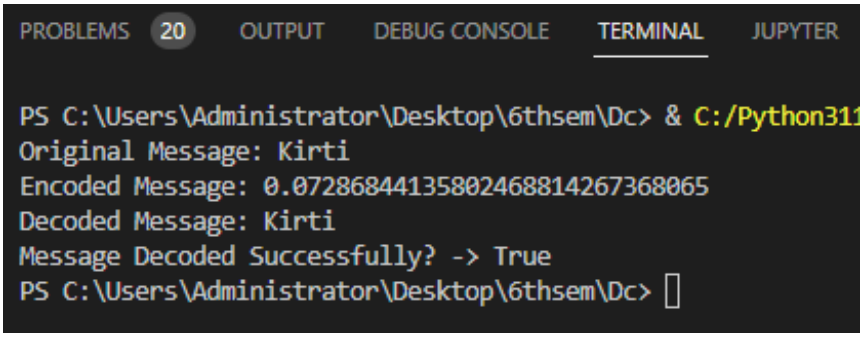
```
from decimal import Decimal
class ArithmeticEncoding:
    def __init__(self, frequency_table):
        self.probability_table = self.get_probability_table(frequency_table)
    def get_probability_table(self, frequency_table):
        total_frequency = sum(list(frequency_table.values()))
        probability_table = {}
        for key, value in frequency_table.items(): probability_table[key] = value/total_frequency
        return probability_table
    def get_encoded_value(self, encoder):
        last_stage = list(encoder[-1].values())
        last_stage_values = []
        for sublist in last_stage:
            for element in sublist: last_stage_values.append(element)
        last_stage_min = min(last_stage_values)
        last_stage_max = max(last_stage_values)
        return (last_stage_min + last_stage_max)/2
    def process_stage(self, probability_table, stage_min, stage_max):
        stage_probs = {}
        stage_domain = stage_max - stage_min
        for term_idx in range(len(probability_table.items())):
            term = list(probability_table.keys())[term_idx]
            term_prob = Decimal(probability_table[term])
            cum_prob = term_prob * stage_domain + stage_min
            stage_probs[term] = [stage_min, cum_prob]
            stage_min = cum_prob
        return stage_probs
    def encode(self, msg, probability_table):
        encoder = []
        stage_min = Decimal(0.0)
        stage_max = Decimal(1.0)
        for msg_term_idx in range(len(msg)):
            stage_probs = self.process_stage(probability_table, stage_min, stage_max)
            msg_term = msg[msg_term_idx]
            stage_min = stage_probs[msg_term][0]
            stage_max = stage_probs[msg_term][1]
            encoder.append(stage_probs)
        stage_probs = self.process_stage(probability_table, stage_min, stage_max)
```

```
        encoder.append(stage_probs)
        encoded_msg = self.get_encoded_value(encoder)
        return encoder, encoded_msg
    def decode(self, encoded_msg, msg_length, probability_table):
        decoder = []
        decoded_msg = ""
        stage_min = Decimal(0.0)
        stage_max = Decimal(1.0)
        for idx in range(msg_length):
            stage_probs = self.process_stage(probability_table, stage_min, stage_max)
            for msg_term, value in stage_probs.items():
                if encoded_msg >= value[0] and encoded_msg <= value[1]: break
            decoded_msg = decoded_msg + msg_term
            stage_min = stage_probs[msg_term][0]
            stage_max = stage_probs[msg_term][1]
            decoder.append(stage_probs)
        stage_probs = self.process_stage(probability_table, stage_min, stage_max)
        decoder.append(stage_probs)
        return decoder, decoded_msg
```

ar1.py

```
import ar
frequency_table = {"K": 2,"i": 3,"r": 1,"t": 4, "i": 5}
AE = ar.ArithmeticEncoding(frequency_table)
original_msg = "Kirti"
print("Original Message: {msg}".format(msg=original_msg))
encoder, encoded_msg = AE.encode(msg=original_msg,
                probability_table=AE.probability_table)
print("Encoded Message: {msg}".format(msg=encoded_msg))
decoder, decoded_msg = AE.decode(encoded_msg=encoded_msg,
msg_length=len(original_msg),probability_table=AE.probability_table)
print("Decoded Message: {msg}".format(msg=decoded_msg))
print("Message Decoded Successfully? -> {result}".format(result=original_msg == decoded_msg))
```

Output:



PRACTICAL : 6

Aim: Write a program to Implement LZ77 algorithm.

Code:

```
def longest_common_substring(s1, s2):
    maxLongest = 0
    offset = 0
    for i in range(0, len(s1)):
        longest = 0
        if ((i == len(s1) - len(s2) - 2)):
            break
        for j in range(0, len(s2)):
            if (i+j < len(s1)):
                if s1[i+j] == s2[j]:
                    longest = longest + 1
                    if (maxLongest < longest):
                        maxLongest = longest
                        offset = i
                else:
                    break
            else:
                break
    return maxLongest, offset
def encode_lz77(text, searchWindowSize, previewWindowSize):
    encodedNumbers = []
    encodedSizes = []
    encodedLetters = []
    i = 0
    while i < len(text):
        if i < previewWindowSize:
            encodedNumbers.append(0)
            encodedSizes.append(0)
            encodedLetters.append(text[i])
            i = i + 1
        else:
            previewString = text[i:i+previewWindowSize]
            searchWindowOffset = 0
            if (i < searchWindowSize):
                searchWindowOffset = i
            else:
                searchWindowOffset = searchWindowSize
            searchString = text[i - searchWindowOffset:i]
            result = longest_common_substring(searchString, previewString)
            nextLetter = ''
            if (result[0] == len(previewString)):
                if (i + result[0] == len(text)):
                    nextLetter = ''
```

```
            else:
                nextLetter = text[i+previewWindowSize]
        else:
            nextLetter = previewString[result[0]]
        if (result[0] == 0):
            encodedNumbers.append(0)
            encodedSizes.append(0)
            encodedLetters.append(nextLetter)
        else:
            encodedNumbers.append(searchWindowOffset - result[1])
            encodedSizes.append(result[0])
            encodedLetters.append(nextLetter)
        i = i + result[0] + 1
    return encodedNumbers, encodedSizes, encodedLetters

def decode_lz77(encodedNumbers, encodedSizes, encodedLetters):
    i = 0
    decodedString = []
    while i < len(encodedNumbers):
        if (encodedNumbers[i] == 0):
            decodedString.append(encodedLetters[i])
        else:
            currentSize = len(decodedString)
            for j in range(0, encodedSizes[i]):
                decodedString.append(decodedString[currentSize-encodedNumbers[i]+j])
            decodedString.append(encodedLetters[i])
        i = i+1
    return decodedString

print("LZ77 Compression and Decompression")
print("200303108034 | Kirti Patel")
stringToEncode = input("Enter the string you want to compress:")
searchWindowSize = int(input("Enter the Search Window Size:"))
previewWindowSize = int(input("Enter the Preview Window Size:"))

[encodedNumbers, encodedSizes, encodedLetters] = encode_lz77(stringToEncode, searchWindowSize,
previewWindowSize)

print("Encoded string: ", end="")
i = 0
while i < len(encodedNumbers):
    print("{", encodedNumbers[i], ":", encodedSizes[i], ":", encodedLetters[i], "}", end=" ")
    i = i + 1
print("\n")

decodedString = decode_lz77(encodedNumbers, encodedSizes, encodedLetters)
print("Decoded string:", "".join(decodedString))
```

Output:

```
PS C:\Users\Administrator\Desktop\6thsem\Dc> & C:/Python311/python.exe c:/Users/Administrator/Desktop/6thsem/Dc/lz771.py
LZ77 Compression and Decompression
200303108034 | Kirti Patel
Enter the string you want to compress:KirtiPatel
Enter the Search Window Size:3
Enter the Preview Window Size:3
Encoded string: {0:0:K}{0:0:i}{0:0:r}{0:0:t}{3:1:P}{0:0:a}{0:0:t}{0:0:e}{0:0:l}

Decoded string: KirtiPatel
PS C:\Users\Administrator\Desktop\6thsem\Dc>
```

PRACTICAL : 8

Aim: Write a program to Implement LZ78 algorithm.

Code:

```
def encode_lz78(text):
    dictionary = {}
    encoded = []
    currentWord = ""
    for c in text:
        newWord = currentWord + c
        if newWord in dictionary:
            currentWord = newWord
        else:
            encoded.append((dictionary.get(currentWord, 0), c))
            dictionary[newWord] = len(dictionary) + 1
            currentWord = ""
    if currentWord:
        encoded.append((dictionary.get(currentWord, 0), None))
    return encoded

def decode_lz78(encoded):
    dictionary = {}
    decoded = []
    currentWord = ""
    for (i, c) in encoded:
        if i == 0:
            decoded.append(currentWord + c)
            dictionary[len(dictionary) + 1] = currentWord + c
            currentWord = ""
        else:
            currentWord = dictionary[i]
            if c is not None:
                decoded.append(currentWord + c)
                dictionary[len(dictionary) + 1] = currentWord + c
                currentWord = ""
    return "".join(decoded)

print("LZ78 Compression and Decompression")
print("200303108034 | Kirti Patel")
stringToEncode = input("Enter the string you want to compress:")

encoded = encode_lz78(stringToEncode)

print("Encoded string: ", encoded)

decoded = decode_lz78(encoded)
print("Decoded string:", decoded)
```

Output:

```
PS C:\Users\Administrator\Desktop\6thsem\Dc> & C:/Python311/python.exe c:/Users/Administrator/Desktop/6thsem/Dc/lz781.py
LZ78 Compression and Decompression
200303108034 | Kirti Patel
Enter the string you want to compress:kirtipatel
Encoded string:  [(0, 'k'), (0, 'i'), (0, 'r'), (0, 't'), (2, 'p'), (0, 'a'), (4, 'e'), (0, 'l')]
Decoded string: kirtipatel
```

PRACTICAL : 5

Aim: Implement adaptive Huffman program to compress decompressed file.

AdaptiveHuffmancoding.java :-

```
package myproject;
import java.util.*;

public class AdaptiveHuffmanCoding
{
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter your string: ");
String firststr = sc.next();
System.out.println(Compress(firststr));
System.out.println(Decompress(Compres(firststr)));
}
public static String Compress(String S)
{
Tree comp = new Tree();
String out = "";
for (int i = 0; i < S.length(); ++i)
{
out =  comp.InsertSymbol(S.charAt(i), out);
} return out;
}
public static String Decompress(String S)
{String out = "";
        Tree decom = new Tree();
        String x = "";
char symbol = decom.ShortCodeKey(S.substring(0, 8));
x = decom.InsertSymbol(symbol, x);
out += symbol;
int i = 8;
Boolean flag = false;
while (!flag) {
if (S.length() - i >=decom.NYT.code.length()) {
x = S.substring(i, i + decom.NYT.code.length());
i += decom.NYT.code.length();
}
else { x = S.substring(i); i += x.length();
}
if (decom.NYT.code.matches(x)) {
String Shortcode = S.substring(i, i + 8);
i += 8;
```

```
symbol = decom.ShortCodeKey(Shortcode);
out += symbol;
x = decom.InsertSymbol(symbol, x);
} else {
while (true) {
symbol = decom.getcharfromcode(x);
if (symbol != ' ') {
        x = decom.InsertSymbol(symbol, x);
        out += symbol;
        break;
} else { i--; x = x.substring(0, x.length() - 1); } }
}
if (i == S.length()) {flag = true;} }
return out;
}
}
```

Node.java:-

```
package myproject;

public class Node {
        public Node parent = null;
        public Node left = null;
        public Node right = null;
        public char symbol;
        public String code;
        public int number;
        public int count;
        public Node(Node parent, Node left, Node right, char symbol, String code, int Number, int count) {
        this.parent = parent;
        this.left = left;
        this.right = right;
        this.symbol = symbol;
        this.code = code;
        this.number = Number;
        this.count = count;
        }
        public Node(Node parent) {
        this.parent = parent;
        }
        public void setSymbol(char symbol) {
        this.symbol = symbol;
        }
        public void setCode(String code) {
        this.code = code;
        }
         public void setNumber(int number) {
```

```
this.number = number;
 }
  public void setCount(int count) {
 this.count = count;
}
 }
```

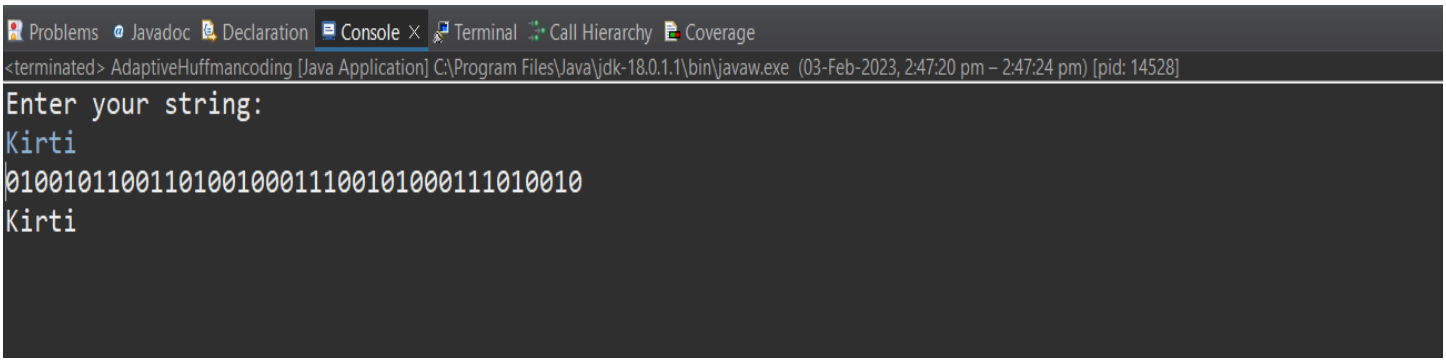
Tree.java:-

```
package myproject;

import java.util.*;
public class Tree {
public HashMap<Character,String> dic = new HashMap<Character, String>();
public HashMap<Character,Node> content = new HashMap<Character,Node>();
public Node root,NYT;
public Tree()
{
        root = new Node(null,null,null,' ',"-1",100,0);
        NYT=root;
for ( int i = 0 ; i < 128 ; i++)
        { String code=Integer.toBinaryString(i);
while(code.length()<8)
code='0'+code;
dic.put( ((char)i ) ,code);
} }
public String InsertSymbol(char symbol ,String out) {
if (content.isEmpty())
{ out += dic.get(symbol);
root.right = new Node(root ,null,null,symbol,"1",NYT.number - 1,1);
content.put(symbol,NYT.right);
root.left = new Node(root ,null,null,' ',"0",NYT.number - 2,0);
root.count = 1; NYT = root.left;
 }
else if ( content.containsKey(symbol) ) {
Node SNode = content.get(symbol);
out += SNode.code;
SNode.setCount(SNode.count + 1);
UpdateInsert(SNode);
}
else { out = out + NYT.code + dic.get(symbol);
NYT.right = new Node(NYT ,null ,null ,symbol ,NYT.code+"1" ,NYT.number - 1 ,1);
content.put(symbol,NYT.right);
NYT.left = new Node(NYT ,null ,null ,' ' ,NYT.code+"0" ,NYT.number - 2 ,0);
NYT = NYT.left; UpdateInsert(NYT.parent.right);
}
return out; }
        public void UpdateInsert(Node cur)
{ Node r = root;Boolean alg1 = false; Increment();
while( !(r.right == cur || r.left == cur) ) {
```

```
if(r.right.symbol != ' '){
if(r.right.number > cur.number && cur.count > r.right.count){
alg1=true;
char temp0 = cur.symbol; cur.symbol = r.right.symbol;
r.right.symbol = temp0;
Integer temp2 = cur.count; cur.count = r.right.count;
r.right.count = temp2;
content.put(r.right.symbol,r.right);
content.put(cur.symbol,cur);
Increment();
break; }
 r = r.left;
}
else if(r.left.symbol != ' '){
if(r.left.number > cur.number && cur.count > r.left.count){
alg1=true;
char temp0 = cur.symbol;
cur.symbol = r.left.symbol;
r.left.symbol = temp0;
Integer temp2 = cur.count;
cur.count = r.left.count;
r.left.count = temp2;
content.put(r.left.symbol,r.left);
content.put(cur.symbol,cur);
Increment();
break;
}
 r = r.right;
 } }
if (alg1) return;
while(cur != root){
if(cur.parent.left.count > cur.parent.right.count ) {
Node temp = cur.parent.left;
cur.parent.left = cur.parent.right; cur.parent.right = temp;
Integer temp1 = cur.parent.left.number;
cur.parent.left.number = cur.parent.right.number; cur.parent.right.number = temp1;
String temp3 = cur.parent.left.code;
cur.parent.left.code = cur.parent.right.code; cur.parent.right.code = temp3;
FixCode(cur);
return;
}
cur = cur.parent;
}
}
public void Increment(){
 Node r = NYT;
while ( !(r == root ) ) {
r.parent.count = r.parent.left.count + r.parent.right.count;
r = r.parent;
 } }
```

```
public void FixCode(Node cur){
if(cur.parent.right.symbol == ' ')
cur = cur.parent.right;
else cur = cur.parent.left;
while ( cur != NYT ) {
cur.right.code = cur.code + cur.right.code.substring(cur.right.code.length()-1);
cur.left.code = cur.code + cur.left.code.substring(cur.left.code.length()-1);
cur = cur.left;
if(cur.parent.right.symbol == ' ')
 cur = cur.parent.right;
else cur = cur.parent.left; } }
public void PrintNode(Node P){
System.out.println(P.symbol); System.out.println(P.code);
System.out.println(P.number); System.out.println(P.count);
}
public void PrintTree(){
Node P = root;
PrintNode(P); System.out.println();
while(P != NYT){
PrintNode(P.right); System.out.println();
PrintNode(P.left); System.out.println();
if(P.right.symbol == ' ') P = P.right;
else P = P.left;
} }
public char ShortCodeKey (String shortcode){
for (char i : dic.keySet())
{if
(dic.get(i).matches(shortcode)) return i;} return ' '; }
public char getcharfromcode (String code){
for (char i : content.keySet()) {if (content.get(i).code.matches(code)) return i;}
return ' ';
 }
}
```

Output:



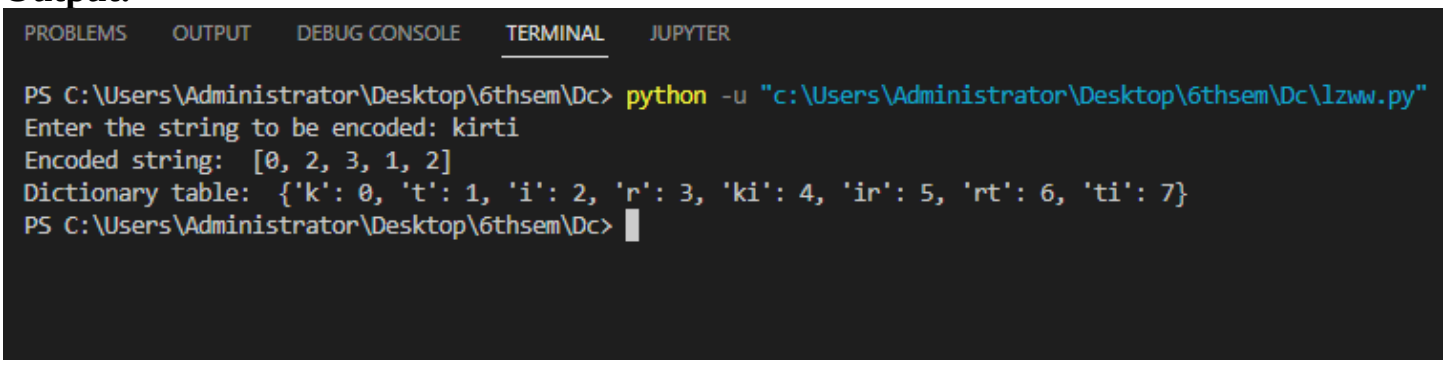
PRACTICAL : 7

Aim: Write a program to Implement LZW algorithm.

Code:

```
def LZW_encode(string):
    dictionary = {char: i for i, char in enumerate(set(string))}
    current_sequence = ""
    encoded_chars = []

    for char in string:
        new_sequence = current_sequence + char
        if new_sequence in dictionary:
            current_sequence = new_sequence
        else:
            dictionary[new_sequence] = len(dictionary)
            encoded_chars.append(dictionary[current_sequence])
            current_sequence = char

    if current_sequence:
        encoded_chars.append(dictionary[current_sequence])

    print("Encoded string: ", encoded_chars)
    print("Dictionary table: ", dictionary)

string = input("Enter the string to be encoded: ")
LZW_encode(string)
```

Output:



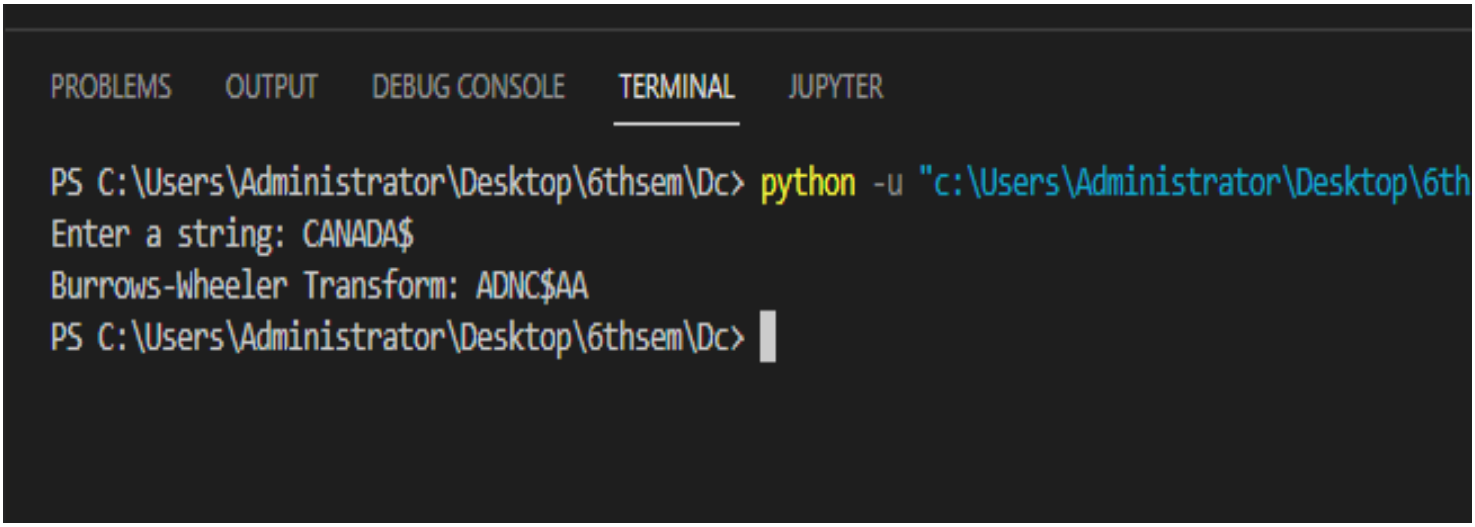
PRACTICAL : 1

Aim: Write a program to Implement Burrow Wheeler Transform algorithm.

Code:

```
def bwt(text):
    n = len(text)
    rotations = [text[i:] + text[:i] for i in range(n)]
    rotations.sort()
    return ''.join(r[-1] for r in rotations)

text = input("Enter a string: ")
bwt_text = bwt(text)
print("Burrows-Wheeler Transform:", bwt_text)
```

Output:



PRACTICAL : 9

Aim: Write a program which performs JPEG compression, process step by step for given 8x8 block and decompression also.

Code:

```
from scipy.fftpack import dct
import heapq
from PIL import Image
import numpy as np

img = Image.open("034.jpg").convert("L")
img = np.array(img)
def zigzag(input):
    return np.concatenate([np.diagonal(input[::-1,:], i)[::(2*(i % 2)-1)] for i in range(1-
    input.shape[0], input.shape[0])])
def rle_encode(arr):
    rle = []
    count = 0
    for i in range(len(arr)):
        if arr[i] == 0:
            count += 1
        else:
            rle.append(count)
            rle.append(arr[i])
            count = 0
    rle.append(count)
    return rle
def huffman_encoding(arr):
    freq_dict = {}
    for i in range(len(arr)):
        if arr[i] not in freq_dict:
            freq_dict[arr[i]] = 1
        else:
            freq_dict[arr[i]] += 1
    heap = [[freq, [val, ""]] for val, freq in freq_dict.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        for pair in left[1:]:
            pair[1] = '0' + pair[1]
        for pair in right[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [left[0] + right[0]] + left[1:] + right[1:])
    huff_dict = dict(heapq.heappop(heap)[1:])
    huff_encoded = ""
    for i in range(len(arr)):
        huff_encoded += huff_dict[arr[i]]
```

```
    return huff_encoded, huff_dict
height, width = img.shape
if height % 8 != 0:
    pad_height = 8 - (height % 8)
else:
    pad_height = 0
if width % 8 != 0:
    pad_width = 8 - (width % 8)
else:
    pad_width = 0
img = np.pad(img, ((0, pad_height), (0, pad_width)), mode='constant')
dct_blocks = np.zeros(img.shape)
for i in range(0, img.shape[0], 8):
    for j in range(0, img.shape[1], 8):
        dct_blocks[i:i+8, j:j+8] = dct(dct(img[i:i+8, j:j+8].T).T)
quant = np.array([[16, 11, 10, 16, 24, 40, 51, 61],
    [12, 12, 14, 19, 26, 58, 60, 55],
    [14, 13, 16, 24, 40, 57, 69, 56],
    [14, 17, 22, 29, 51, 87, 80, 62],
    [18, 22, 37, 56, 68, 109, 103, 77],
    [24, 35, 55, 64, 81, 104, 113, 92],
    [49, 64, 78, 87, 103, 121, 120, 101],[72, 92, 95, 98, 112, 100, 103, 99]])
quant_blocks = np.zeros(img.shape)
for i in range(0, img.shape[0], 8):
    for j in range(0, img.shape[1], 8):
        quant_blocks[i:i+8, j:j+8] = np.round(dct_blocks[i:i+8, j:j+8] / quant)
rle_compressed = []
for i in range(0, img.shape[0], 8):
    for j in range(0, img.shape[1], 8):
        rle_block = rle_encode(zigzag(quant_blocks[i:i+8, j:j+8]))
rle_compressed.extend(rle_block)
huff_encoded, huff_dict = huffman_encoding(rle_compressed)
with open("034_compress.txt", "w") as f:
    for key, value in huff_dict.items():
        f.write(str(key) + ":" + str(value) + "\n")
binary_encoded = ""
for char in huff_encoded:
    binary_encoded += huff_dict[int(char)]
while len(binary_encoded) % 8 != 0:
    binary_encoded += "0"
with open("output.bin", "wb") as f:
    for i in range(0, len(binary_encoded), 8):
        f.write(bytes([int(binary_encoded[i:i+8], 2)]))
```

Output:

034_compress.txt

```
0:0
5.0:10000
9.0:10001
13.0:100100
16:100101
17.0:10011
20.0:101000
35.0:101001
42.0:101010
60.0:101011
-1.0:1011
1:110
62.0:111000
-11.0:1110010
-10.0:1110011
-8.0:1110100
-5.0:1110101
-3.0:1110110
-2.0:1110111
2.0:11110
3.0:111110
4.0:1111110
6.0:1111111
```

PRACTICAL : 10

Aim: Write a program to Implement Move to Front algorithm.

Code:

```
def encode(text):
    alphabet = [chr(i) for i in range(256)]
    encoded_text = []
    for char in text:
        index = alphabet.index(char)
        encoded_text.append(index)
        alphabet.pop(index)
        alphabet.insert(0, char)
    return encoded_text

def decode(encoded_text):
    alphabet = [chr(i) for i in range(256)]
    decoded_text = ""
    for index in encoded_text:
        char = alphabet[index]
        decoded_text += char
        alphabet.pop(index)
        alphabet.insert(0, char)
    return decoded_text

text = input("Enter the text to encode: ")
encoded_text = encode(text)
print("Encoded text:", encoded_text)
decoded_text = decode(encoded_text)
print("Decoded text:", decoded_text)
```

Output: