**PRACTICAL-3**

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

**Algorithm :**

Huffman (C)

n=|C|

Q=C

for i=1 to n-1

do

z=allocate\_Node()

Node()

x=left[z]=Extract\_Min(Q)

y=right[z]=Extract\_Min(Q)

f[z]=f[x]+f[y]

Insert(Q,z)

return Extract\_Min(Q)

**Code:**

import java.util.PriorityQueue;

import java.util.Scanner;

import java.util.Comparator;

class Huffman {

public static void printCode(HuffmanNode root, String s) {

if (root.left == null && root.right == null && Character.isLetter(root.c)) {

System.out.println(root.c + ":" + s);

return; }

printCode(root.left, s + "0");

printCode(root.right, s + "1"); }

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter size of string:");

int n = sc.nextInt();

char charArray[] = new char[n];

System.out.println("Enter Characters of String:");

for (int k = 0; k < n; k++) {

charArray[k] = sc.next().charAt(0);

}

int charfreq[] = new int[n];

System.out.println("Enter Frequency of characters:");

for (int k = 0; k < n; k++) { charfreq[k] = sc.nextInt(); }

PriorityQueue < HuffmanNode > q = new PriorityQueue < HuffmanNode > (n, new MyComparator());

for (int i = 0; i < n; i++) {

HuffmanNode hn = new HuffmanNode();

hn.c = charArray[i];

hn.data = charfreq[i];

hn.left = null;

hn.right = null;

q.add(hn); }

HuffmanNode root = null;

while (q.size() > 1) {

HuffmanNode x = q.peek();

q.poll();

HuffmanNode y = q.peek();

q.poll();

HuffmanNode f = new HuffmanNode();

f.data = x.data + y.data;

f.c = '-';

f.left = x;

f.right = y;

root = f;

q.add(f); }

System.out.println("Generated Binary code : ");

printCode(root, ""); }}

class HuffmanNode {

int data;

char c;

HuffmanNode left;

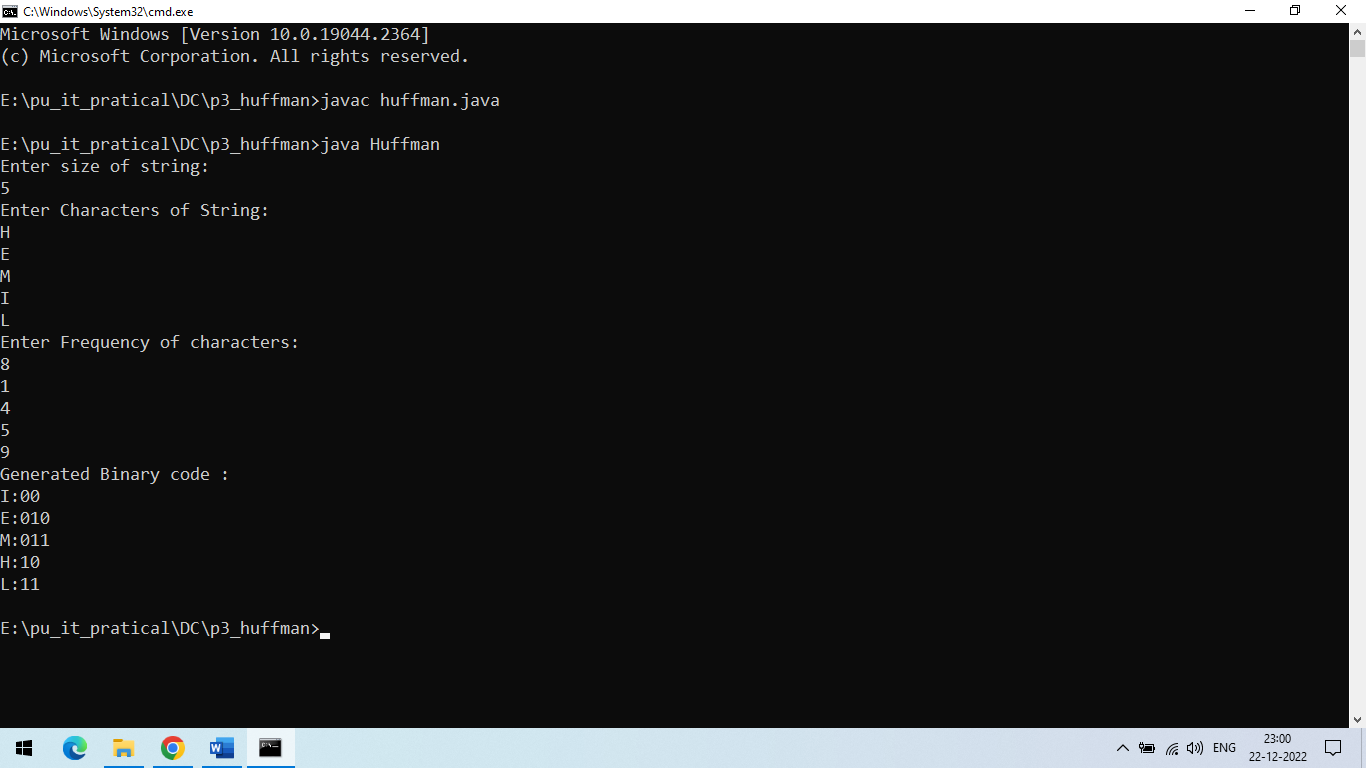
HuffmanNode right; }

class MyComparator implements Comparator < HuffmanNode > {

public int compare(HuffmanNode x, HuffmanNode y) {

return x.data - y.data; } }

**Output:**



**PRACTICAL-4**

**AIM: Implement Huffman code which can compress given file and decompress compressed file:**

**Algorithm :**

Steps of Huffman Encoding are:

* Create And Initialize A Priorityqueue Queue Consisting Of Each Unique Character.
* Sort In Ascending Order Of Their Frequencies.
* For All The Unique Characters:
* Create A New\_Node
* Get Minimum\_Value From Queue And Set It To Left Child Of New\_Node
* Get Minimum\_Value From Queue And Set It To Right Child Of New\_Node
* Calculate The Sum Of These Two Minimum Values As Sum\_Of\_Two\_Minimum
* Assign Sum\_Of\_Two\_Minimum To The Value Of New\_Node
* Insert New\_Node Into The Tree
* Return Root\_Node

**Code:**

import java.util.\*;

class Node {

Character ch;

Integer freq;

Node left = null;

Node right = null;

Node(Character ch, Integer freq) {

this.ch = ch;

this.freq = freq;

}

public Node(Character ch, Integer freq, Node left, Node right) {

this.ch = ch;

this.freq = freq;

this.left = left;

this.right = right;

}

}

public class HuffmanCode {

public static void createHuffmanTree(String text) {

if (text == null || text.length() == 0) {

return;

}

Map < Character, Integer > freq = new HashMap < > ();

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

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

}

PriorityQueue < Node > pq = new PriorityQueue < > (Comparator.comparingInt(l - > l.freq));

for (var entry: freq.entrySet()) {

pq.add(new Node(entry.getKey(), entry.getValue()));

}

while (pq.size() != 1) {

Node left = pq.poll();

Node right = pq.poll();

int sum = left.freq + right.freq;

pq.add(new Node(null, sum, left, right));

}

Node root = pq.peek();

Map < Character, String > huffmanCode = new HashMap < > ();

encodeData(root, "", huffmanCode);

System.out.println("Huffman Codes of the characters are: " + huffmanCode);

System.out.println("The initial string is: " + text);

StringBuilder sb = new StringBuilder();

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

sb.append(huffmanCode.get(c));

}

System.out.println("The encoded string is: " + sb);

System.out.print("The decoded string is: ");

if (isLeaf(root)) {

while (root.freq-- > 0) {

System.out.print(root.ch);

}

} else {

int index = -1;

while (index < sb.length() - 1) {

index = decodeData(root, index, sb);

}

}

}

public static void encodeData(Node root, String str, Map < Character, String > huffmanCode) {

if (root == null) {

return;

}

if (isLeaf(root)) {

huffmanCode.put(root.ch, str.length() > 0 ? str : "1");

}

encodeData(root.left, str + '0', huffmanCode);

encodeData(root.right, str + '1', huffmanCode);

}

public static int decodeData(Node root, int index, StringBuilder sb) {

if (root == null) {

return index;

}

if (isLeaf(root)) {

System.out.print(root.ch);

return index;

}

index++;

root = (sb.charAt(index) == '0') ? root.left : root.right;

index = decodeData(root, index, sb);

return index;

}

public static boolean isLeaf(Node root) {

return root.left == null && root.right == null;

}

public static void main(String args[]) {

Scanner scyy = new Scanner(System.in);

System.out.println("Enter String : ");

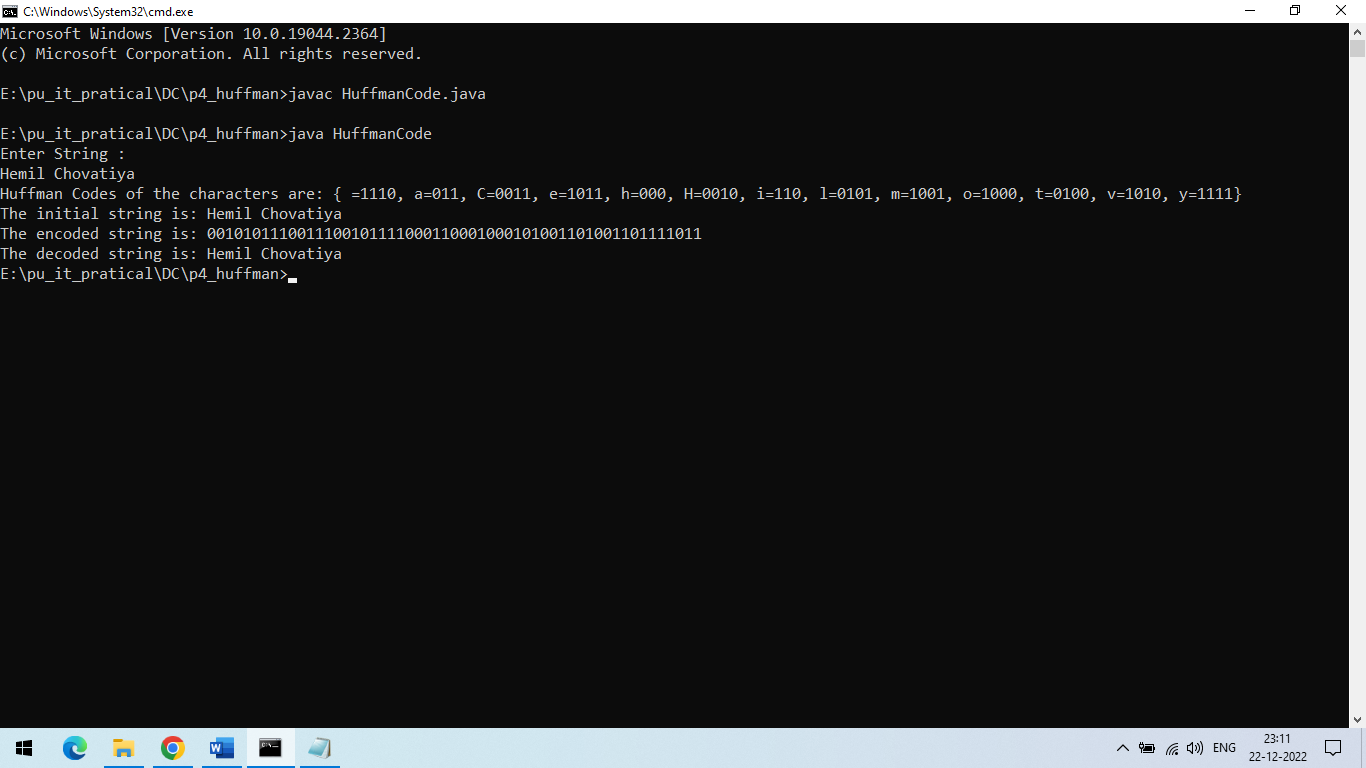
String text = scyy.nextLine();

createHuffmanTree(text);

}

}

**Output:**



**PRACTICAL-5**

**AIM: Implement Huffman code which can compress given file and decompress compressed file:**

**Code:**

**AdaptiveHuffmanCoding.java**

public class AdaptiveHuffmanCoding {

public static void main(String[] args) {

System.out.println(Compress("Hemil\_Chovatiya"));

System.out.println(Decompress(Compress("Hemil\_Chovatiya"))); }

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; } }

**Tree.java**

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 ' '; } }

**Node.java**

public class Node {

public Node parent = null; // for parent

public Node left = null; // for left

public Node right = null; // for right

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; }

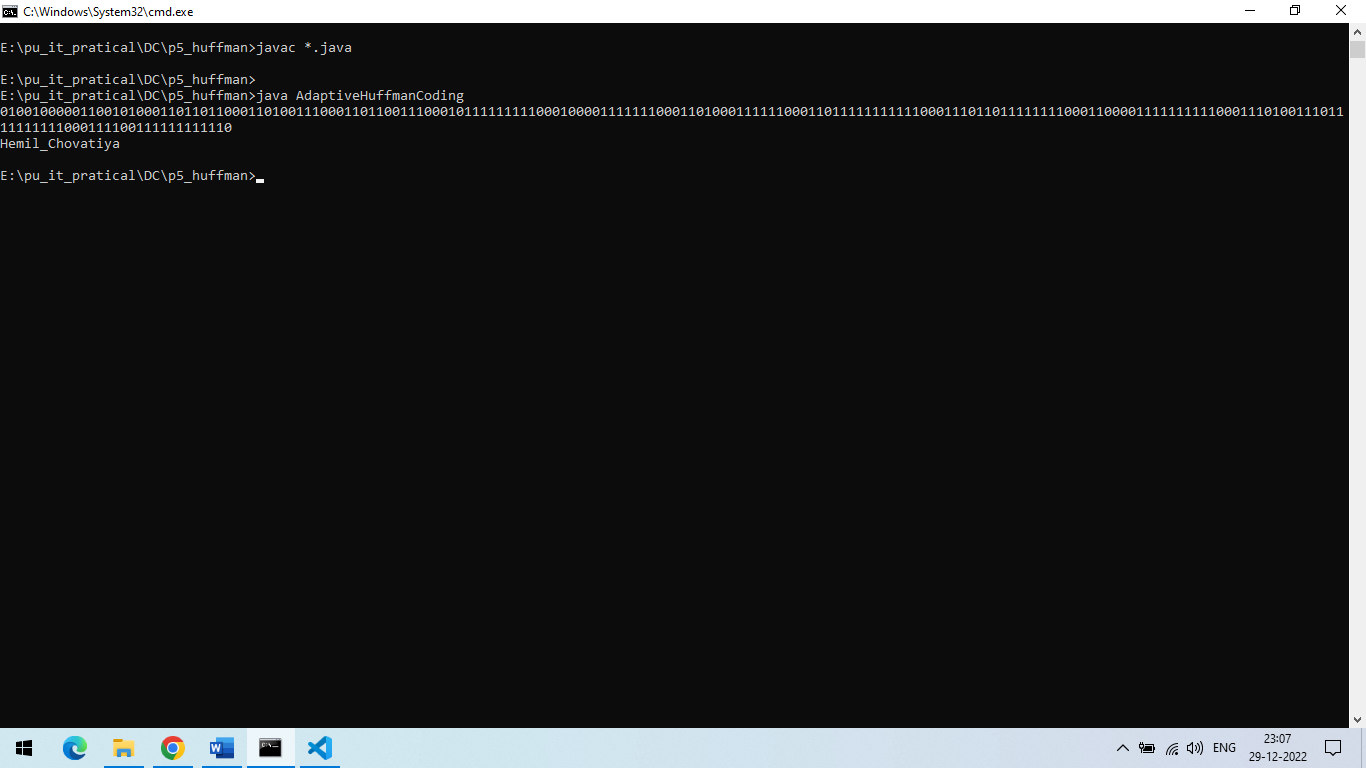
public void setSymbol(char symbol) { this.symbol = symbol; }

public void setCode(String code) { this.code = code; }

public void setNumber(int number) { this.number = number; }

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

**Output:**



**PRACTICAL-2**

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

**Code:**

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

struct node

{    double prob, range\_from, range\_to; };

double encoding(unordered\_map<char, node> arr, string s)

{

    cout << "\nEncoding\n";

    double low\_v = 0.0, high\_v = 1.0, diff = 1.0;

    cout << "Symbol\tLow\_v\tHigh\_v\tdiff\n";

    for (int i = 0; i < s.size(); i++)

    {   high\_v = low\_v + diff \* arr[s[i]].range\_to;

        low\_v = low\_v + diff \* arr[s[i]].range\_from;

        diff = high\_v - low\_v;

        cout << s[i] << "\t" << low\_v << "\t" << high\_v << "\t" << diff << endl;    }

    return low\_v; }

string decoding(unordered\_map<char, node> arr, double code\_word, int len)

{  cout << "\nDecoding: \n";

    char ch;

    string text = "";

    int j = 0;

    unordered\_map<char, node>::iterator it;

    cout << "Code\tOutput\tRange\_from\tRange\_to\n";

    while (j < len)

    {  cout << code\_word << "\t";

        for (it = arr.begin(); it != arr.end(); it++)

        {  char i = (\*it).first;

            if (arr[i].range\_from <= code\_word && code\_word < arr[i].range\_to)

            {  ch = i;

        code\_word = (code\_word - arr[i].range\_from) / (arr[i].range\_to - arr[i].range\_from);

                break;            }        }

        cout << ch << "\t" << arr[ch].range\_from << "\t\t" << arr[ch].range\_to << endl;

        text += ch;

        j++;     }

    return text; }

int main()

{  int n;

    cout << "Enter number of characters: ";

    cin >> n;

    unordered\_map<char, node> arr;

    vector<char> ar;

    double range\_from = 0;

    cout << "Enter probability of each character:\n";

    for (int i = 0; i < n; i++)

    {  char ch;

        cin >> ch;

        ar.push\_back(ch);

        cin >> arr[ch].prob;

        arr[ch].range\_from = range\_from;

        arr[ch].range\_to = range\_from + arr[ch].prob;

        range\_from = arr[ch].range\_to;     }

    cout << "Symbol\tProbability\tRange\_from\tRange\_to\n";

    cout << "   \n";

    for (int i = 0; i < ar.size(); i++)

    {   char ch = ar[i];

cout << ch << "\t" << arr[ch].prob << "\t\t" << arr[ch].range\_from << "\t\t" << arr[ch].range\_to << endl;     }

    cout << endl;

    string s;

    cout << "Enter text: ";

    cin >> s;

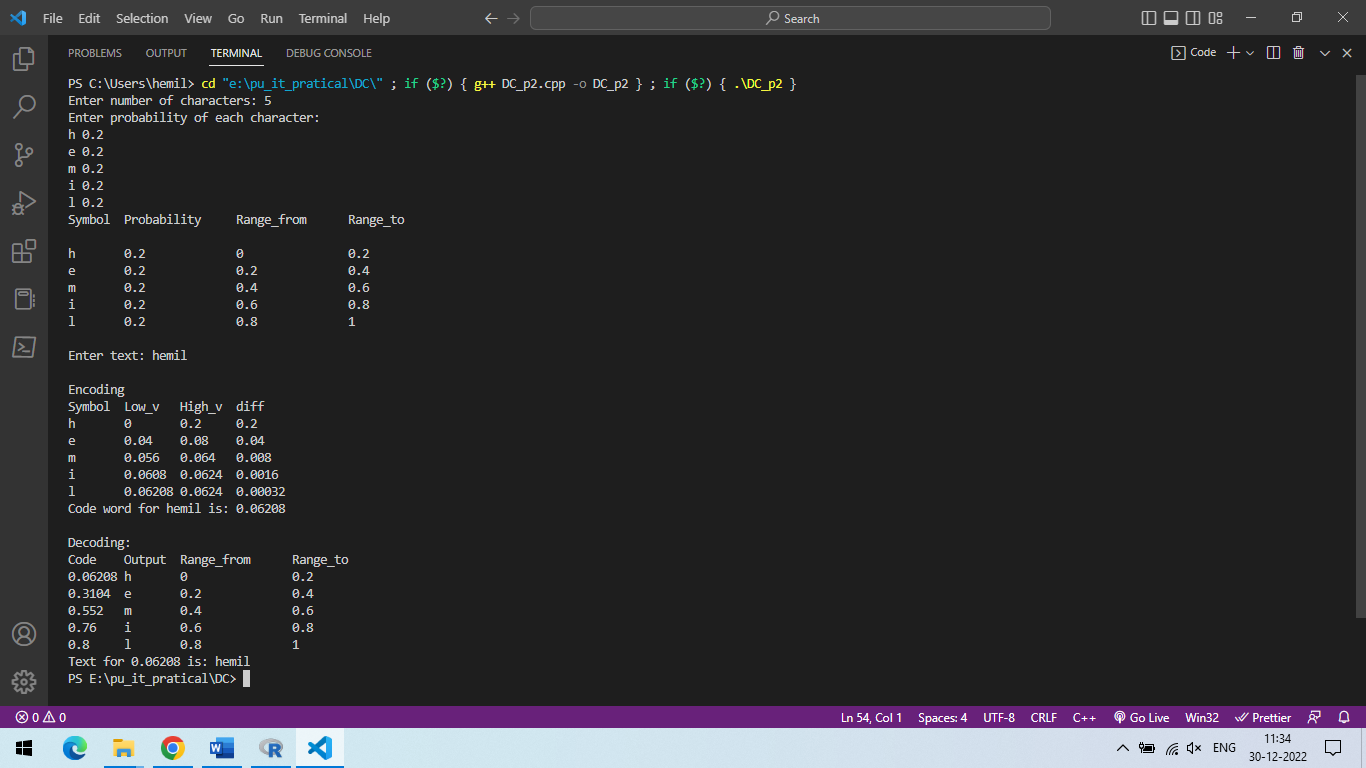
    double code\_word = encoding(arr, s);

    cout << "Code word for " << s << " is: " << code\_word << endl;

    string text = decoding(arr, code\_word, s.size());

    cout << "Text for " << code\_word << " is: " << text << endl; }

**Output:**



**PRACTICAL-6**

**AIM:** **Write a program to Implement LZ77 algorithm.**

**Code:**

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.util.ArrayList;

import java.util.List;

public class LZW {

public static void main(String[] args) {

try

{ if (args.length == 2) {

File file = new File(args[1]);

if (file.exists()) {

if (args[0].toLowerCase().contains("c")) { // Compress

String input = Files.readString(file.toPath());

byte[] compressedTags = compress(input);

for (byte tag : compressedTags) {

System.out.println("Tag<" + Byte.toUnsignedInt(tag) + ">");

}

String newPath = file.getPath() + ".lzw";

File compressedFile = new File(newPath);

compressedFile.createNewFile();

Files.write(compressedFile.toPath(), compressedTags);

} else if (args[0].toLowerCase().contains("d")) { // Decompress

byte[] input = Files.readAllBytes(file.toPath());

String decompressedTxt = decompress(input);

System.out.println(decompressedTxt);

String newPath = file.getPath() + ".txt";

File decompressedFile = new File(newPath);

decompressedFile.createNewFile();

List<String> lines = new ArrayList<>();

lines.add(decompressedTxt);

Files.write(decompressedFile.toPath(), lines);

} else { System.out.println(args[0] + " is invalid argument"); }

} else { System.out.println(args[1] + " is not an existing file"); }

} else { System.out.println("No arguments were supplied"); } }

catch (IOException e) { System.out.println(e.getMessage()); } }

private static byte[] compress(String str) {

ArrayList<String> dictionary = new ArrayList<>();

List<Byte> tags = new ArrayList<>();

for (int i = 0; i < str.length();){

if (dictionary.size() == 128) {

dictionary.clear(); }

int dictionaryIndex = -1;

StringBuilder temp\_str = new StringBuilder();

temp\_str.append(str.charAt(i));

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

temp\_str.append(str.charAt(j));

if (dictionary.contains(temp\_str.toString())) {

dictionaryIndex = dictionary.indexOf(temp\_str.toString());

if (j == str.length() - 1) {

byte tag = (byte) (dictionaryIndex + 128);

tags.add(tag);

i += temp\_str.length(); } }

else { dictionary.add(temp\_str.toString());

byte tag = (byte) (dictionaryIndex + 128);

if (dictionaryIndex == -1) { tag = (byte) str.charAt(i); }

tags.add(tag);

i += temp\_str.length() - 1;

break; } }

if(str.length() - i == 1) {

tags.add((byte) str.charAt(i));

break; } }

byte[] bytes = new byte[tags.size()];

for (int i = 0; i < tags.size(); i++) {

bytes[i] = tags.get(i); }

return bytes; }

private static String decompress(byte[] input) {

StringBuilder decompressed = new StringBuilder();

ArrayList<String> dictionary = new ArrayList<>();

StringBuilder nextDic = new StringBuilder();

nextDic.append((char) input[0]);

decompressed.append((char) input[0]);

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

if (dictionary.size() == 128) { dictionary.clear(); }

int currentTag = Byte.toUnsignedInt(input[i]) - 128;

if(currentTag < 0) {

nextDic.append((char) input[i]);

decompressed.append((char) input[i]);

dictionary.add(nextDic.toString());

nextDic = new StringBuilder();

nextDic.append((char) input[i]); }

else {

if(currentTag >= dictionary.size()) {

nextDic.append(nextDic.substring(0, 1));

decompressed.append(nextDic); }

else { nextDic.append(dictionary.get(currentTag).charAt(0));

decompressed.append(dictionary.get(currentTag)); }

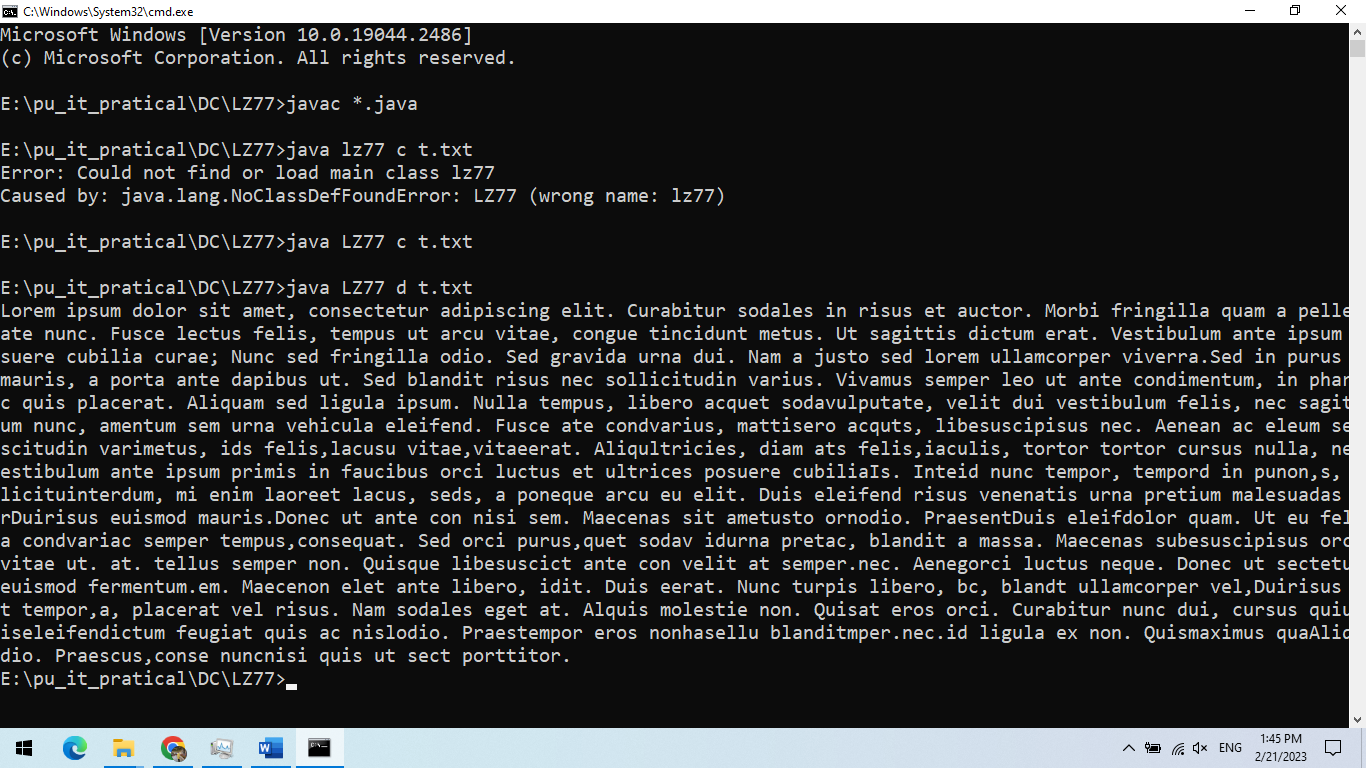
dictionary.add(nextDic.toString());

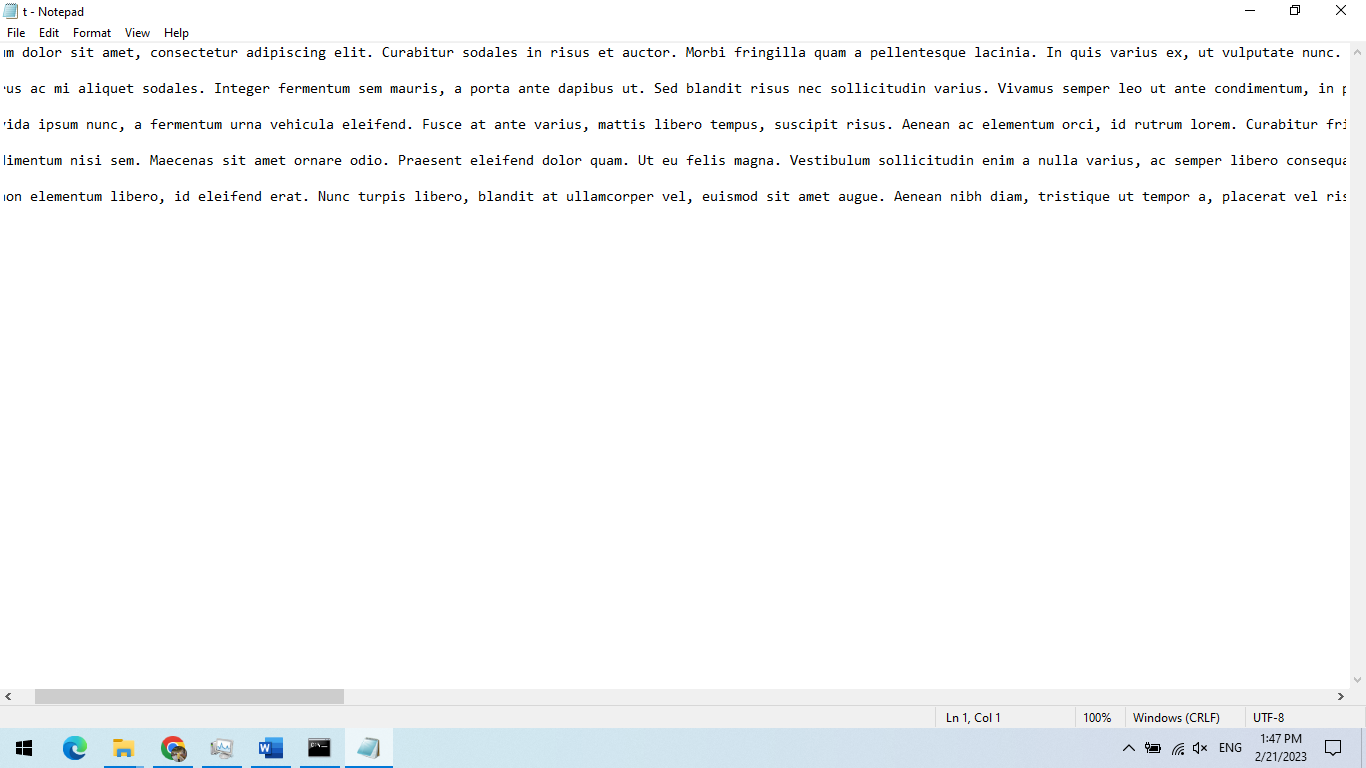
nextDic = new StringBuilder();

nextDic.append(dictionary.get(currentTag)); } }

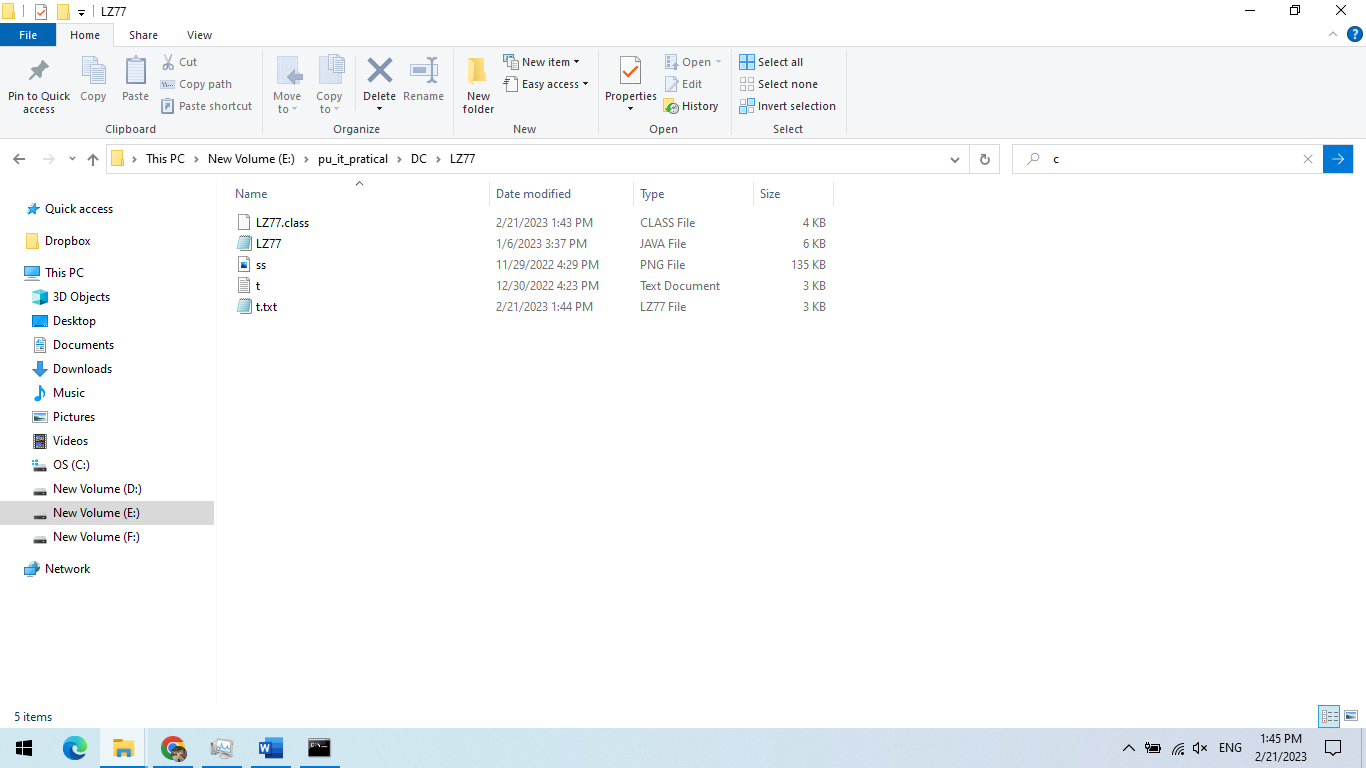
return decompressed.toString(); } }

**Output:**





t.txt file for compressing



t.txt.LZ77 file generated while running

**PRACTICAL-7**

**AIM:** **Write a program to Implement LZW algorithm.**

**Code:**

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.util.ArrayList;

import java.util.List;

public class LZW {

public static void main(String[] args) {

try {

if (args.length == 2) {

File file = new File(args[1]);

if (file.exists()) {

if (args[0].toLowerCase().contains("c")) { // Compress

String input = Files.readString(file.toPath());

byte[] compressedTags = compress(input);

for (byte tag : compressedTags)

{System.out.println("Tag<" + Byte.toUnsignedInt(tag) + ">"); }

String newPath = file.getPath() + ".lzw";

File compressedFile = new File(newPath);

compressedFile.createNewFile();

Files.write(compressedFile.toPath(), compressedTags);

} else if (args[0].toLowerCase().contains("d")) { // Decompress

byte[] input = Files.readAllBytes(file.toPath());

String decompressedTxt = decompress(input);

System.out.println(decompressedTxt);

String newPath = file.getPath() + ".txt";

File decompressedFile = new File(newPath);

decompressedFile.createNewFile();

List<String> lines = new ArrayList<>();

lines.add(decompressedTxt);

Files.write(decompressedFile.toPath(), lines);

} else {

System.out.println(args[0] + " is invalid argument"); }

} else {

System.out.println(args[1] + " is not an existing file");

} } else { System.out.println("No arguments were supplied"); } }

catch (IOException e) { System.out.println(e.getMessage()); } }

private static byte[] compress(String str) {

ArrayList<String> dictionary = new ArrayList<>();

List<Byte> tags = new ArrayList<>();

for (int i = 0; i < str.length();){

if (dictionary.size() == 128) {

dictionary.clear(); }

int dictionaryIndex = -1;

StringBuilder temp\_str = new StringBuilder();

temp\_str.append(str.charAt(i));

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

temp\_str.append(str.charAt(j));

if (dictionary.contains(temp\_str.toString())) {

dictionaryIndex = dictionary.indexOf(temp\_str.toString());

if (j == str.length() - 1) {

byte tag = (byte) (dictionaryIndex + 128);

tags.add(tag);

i += temp\_str.length(); } }

else { dictionary.add(temp\_str.toString());

byte tag = (byte) (dictionaryIndex + 128);

if (dictionaryIndex == -1) { tag = (byte) str.charAt(i); }

tags.add(tag);

i += temp\_str.length() - 1;

break; } }

if(str.length() - i == 1) {

tags.add((byte) str.charAt(i));

break; } }

byte[] bytes = new byte[tags.size()];

for (int i = 0; i < tags.size(); i++) {

bytes[i] = tags.get(i); }

return bytes; }

private static String decompress(byte[] input) {

StringBuilder decompressed = new StringBuilder();

ArrayList<String> dictionary = new ArrayList<>();

StringBuilder nextDic = new StringBuilder();

nextDic.append((char) input[0]);

decompressed.append((char) input[0]);

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

// Reset the dictionary

if (dictionary.size() == 128) {

dictionary.clear(); }

int currentTag = Byte.toUnsignedInt(input[i]) - 128;

if(currentTag < 0) {

nextDic.append((char) input[i]);

decompressed.append((char) input[i]);

dictionary.add(nextDic.toString());

nextDic = new StringBuilder();

nextDic.append((char) input[i]); }

else { if(currentTag >= dictionary.size()) {

nextDic.append(nextDic.substring(0, 1));

decompressed.append(nextDic); }

else { nextDic.append(dictionary.get(currentTag).charAt(0));

decompressed.append(dictionary.get(currentTag)); }

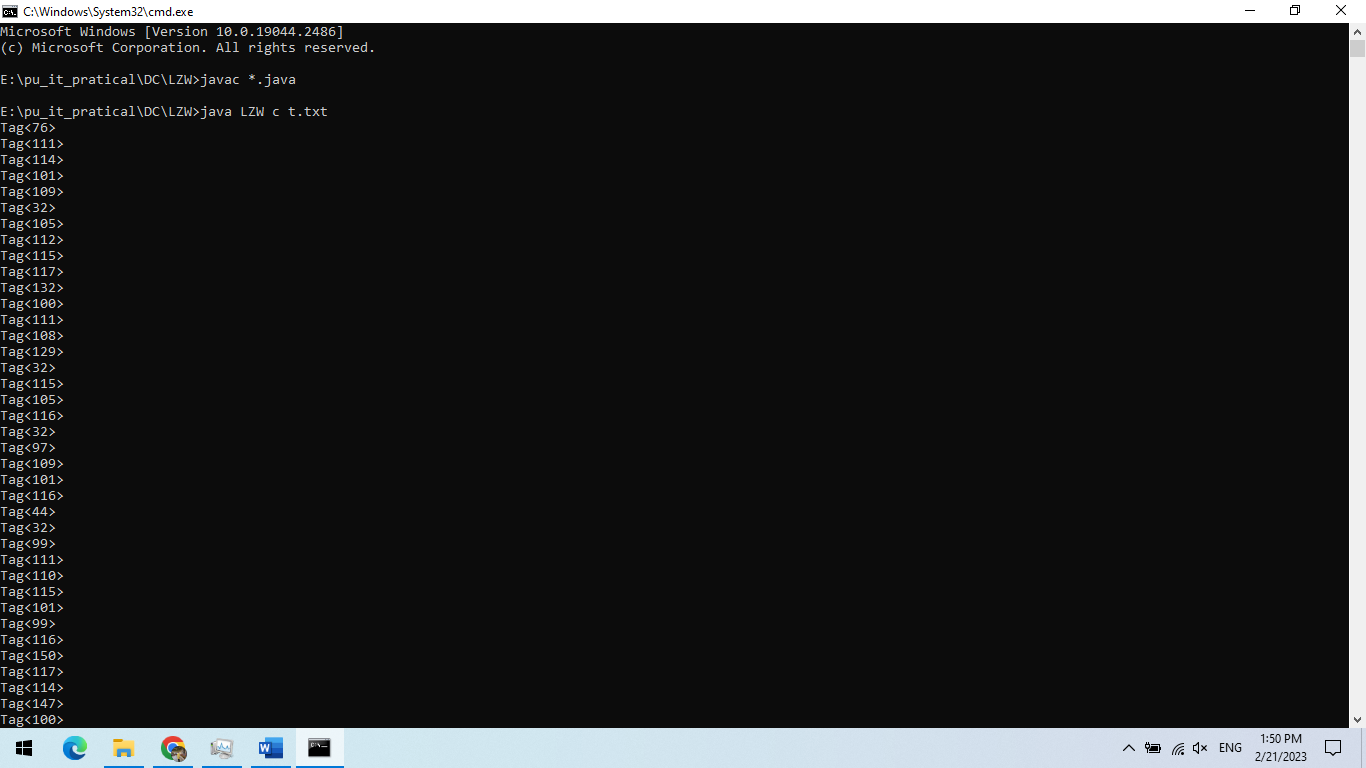
dictionary.add(nextDic.toString());

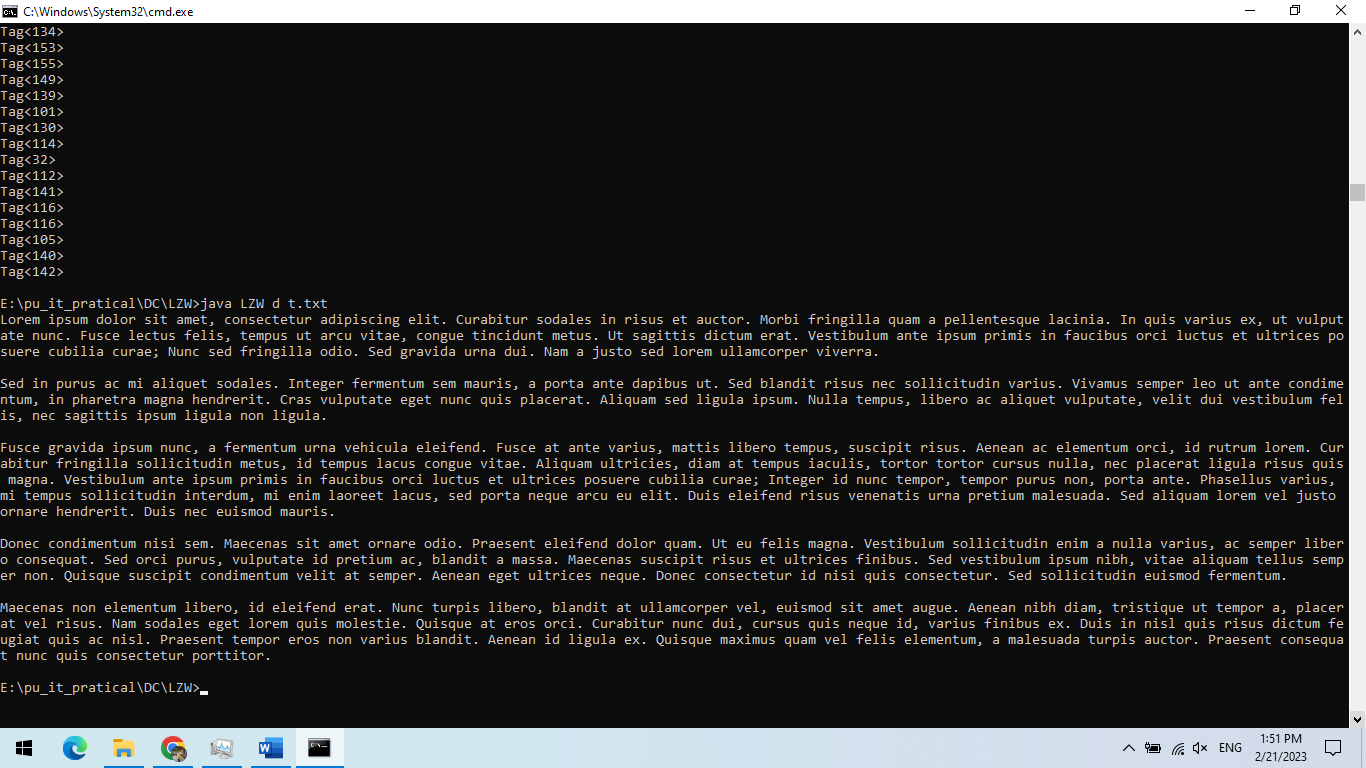
nextDic = new StringBuilder();

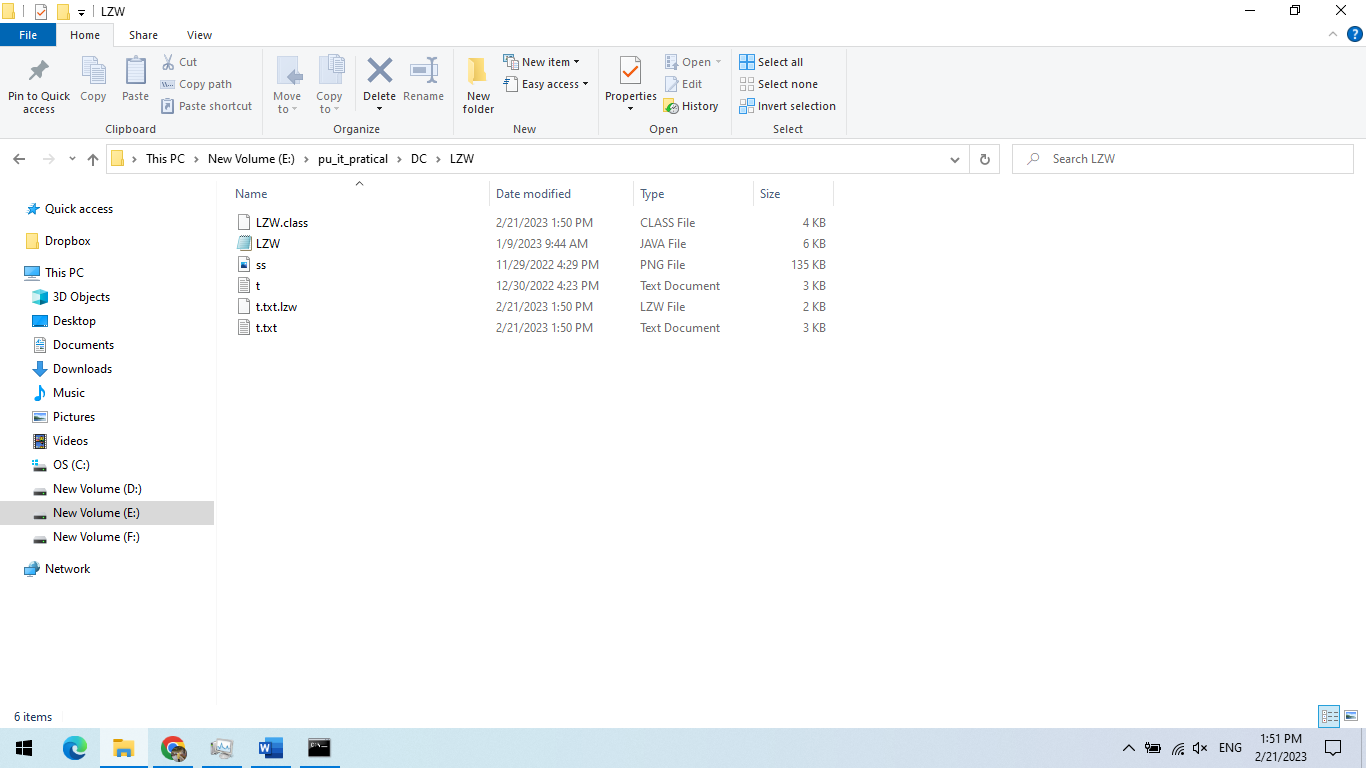
nextDic.append(dictionary.get(currentTag)); } }

return decompressed.toString(); }}

**Output:**







**PRACTICAL-8**

**AIM:** **Write a program to Implement LZ78 algorithm.**

**Code: LZW.java file**

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.util.ArrayList;

import java.util.List;

public class LZ78 {

public static void main(String[] args) {

try { if (args.length == 2) {

File file = new File(args[1]);

if (file.exists()) {

if (args[0].toLowerCase().contains("c")) { // Compress

String input = Files.readString(file.toPath());

List<Tag> compressedTags = compress(input);

for (Tag tag : compressedTags) {

System.out.println("Tag<" + tag.getPosition() + ", " + tag.getNextChar() + ">"); }

byte[] compressedBytes = Tag.tagsToByteArray(compressedTags);

String newPath = file.getPath() + ".lz78";

File compressedFile = new File(newPath);

compressedFile.createNewFile();

Files.write(compressedFile.toPath(), compressedBytes);

} else if (args[0].toLowerCase().contains("d")) { // Decompress

byte[] input = Files.readAllBytes(file.toPath());

String decompressedTxt = decompress(Tag.bytesToTags(input));

System.out.println(decompressedTxt);

String newPath = file.getPath() + ".txt";

File decompressedFile = new File(newPath);

decompressedFile.createNewFile();

List<String> lines = new ArrayList<>();

lines.add(decompressedTxt);

Files.write(decompressedFile.toPath(), lines);

} else { System.out.println(args[0] + " is invalid argument"); }

} else { System.out.println(args[1] + " is not an existing file"); }

} else { System.out.println("No arguments were supplied"); } }

catch (IOException e) {

System.out.println(e.getMessage()); } }

public static List<Tag> compress(String str){

ArrayList<String> dictionary = new ArrayList<>();

List<Tag> tags = new ArrayList<>();

dictionary.add("\0");

for (int i = 0; i < str.length();){ // Reset the dictionary

if (dictionary.size() == 256) {

dictionary.clear();

dictionary.add("\0"); }

int dictionaryIndex = 0;

StringBuilder temp\_str = new StringBuilder();

for (int j = i; j < str.length(); j++){

temp\_str.append(str.charAt(j));

if (dictionary.contains(temp\_str.toString())) {

dictionaryIndex = dictionary.indexOf(temp\_str.toString());

if (j == str.length() - 1) {

Tag tag = new Tag((byte) dictionaryIndex, '\0');

tags.add(tag);

i += temp\_str.length(); } }

else { char next = temp\_str.charAt(temp\_str.length() - 1);

dictionary.add(temp\_str.toString());

Tag tag = new Tag((byte) dictionaryIndex, next);

tags.add(tag);

i += temp\_str.length();

break; } } }

return tags; }

private static String decompress(List<Tag> tags) {

ArrayList<String> dictionary = new ArrayList<>();

StringBuilder decompressed = new StringBuilder();

for (Tag tag : tags) {

// Reset the dictionary

if (dictionary.size() == 255) {

dictionary.clear(); }

if(tag.getPosition() == 0) {

dictionary.add(String.valueOf(tag.getNextChar()));

if(tag.getNextChar() != 0)

decompressed.append(tag.getNextChar()); }

else { StringBuilder tagBuilder = new StringBuilder();

tagBuilder.append(dictionary.get(tag.getPosition() - 1));

if(tag.getNextChar() != 0)

tagBuilder.append(tag.getNextChar());

dictionary.add(tagBuilder.toString());

decompressed.append(tagBuilder); } }

return decompressed.toString(); }}

**Tag.java file**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Tag {

private byte position;

private char nextChar;

public Tag(byte position, char nextChar) {

this.position = position;

this.nextChar = nextChar; }

public Tag(byte[] tagBytes) {

position = tagBytes[0];

nextChar = (char) tagBytes[1]; }

public byte[] toByteArray() {

byte[] bytes = new byte[2];

bytes[0] = position;

bytes[1] = (byte) nextChar;

return bytes; }

public static byte[] tagsToByteArray(List<Tag> tags){

byte[] result = new byte[tags.size() \* 2];

int tagsCounter = 0, resultCounter = 0;

while (tagsCounter < tags.size()){

byte[] tagByte = tags.get(tagsCounter).toByteArray();

result[resultCounter] = tagByte[0];

result[resultCounter + 1] = tagByte[1];

resultCounter += 2;

tagsCounter++; }

return result; }

public static List<Tag> bytesToTags(byte[] bytes) {

List<Tag> tags = new ArrayList<>(bytes.length / 2);

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

tags.add(new Tag(Arrays.copyOfRange(bytes, i\*2, (i\*2 + 2)))); }

return tags; }

public int getPosition() { return Byte.toUnsignedInt(position); }

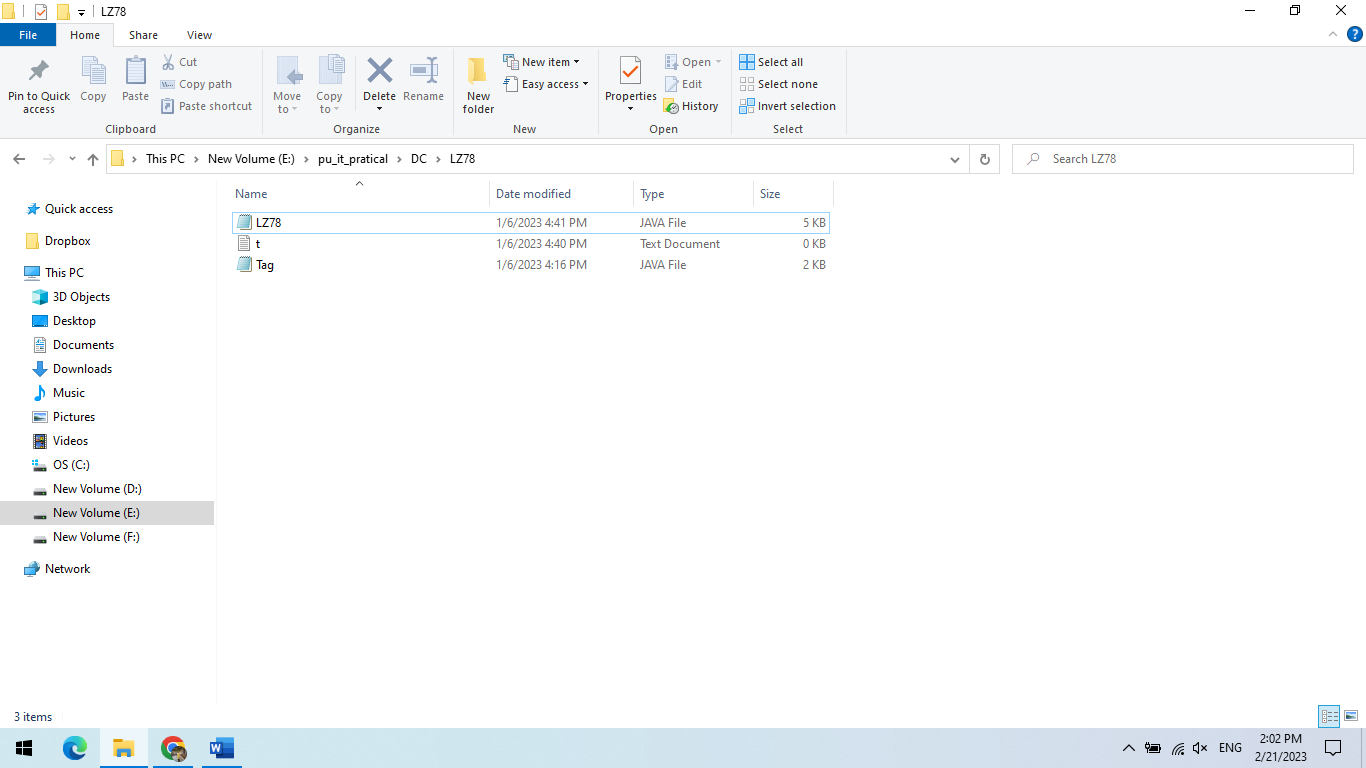
public void setPosition(int position) { this.position = (byte) position; }

public char getNextChar() { return nextChar; }

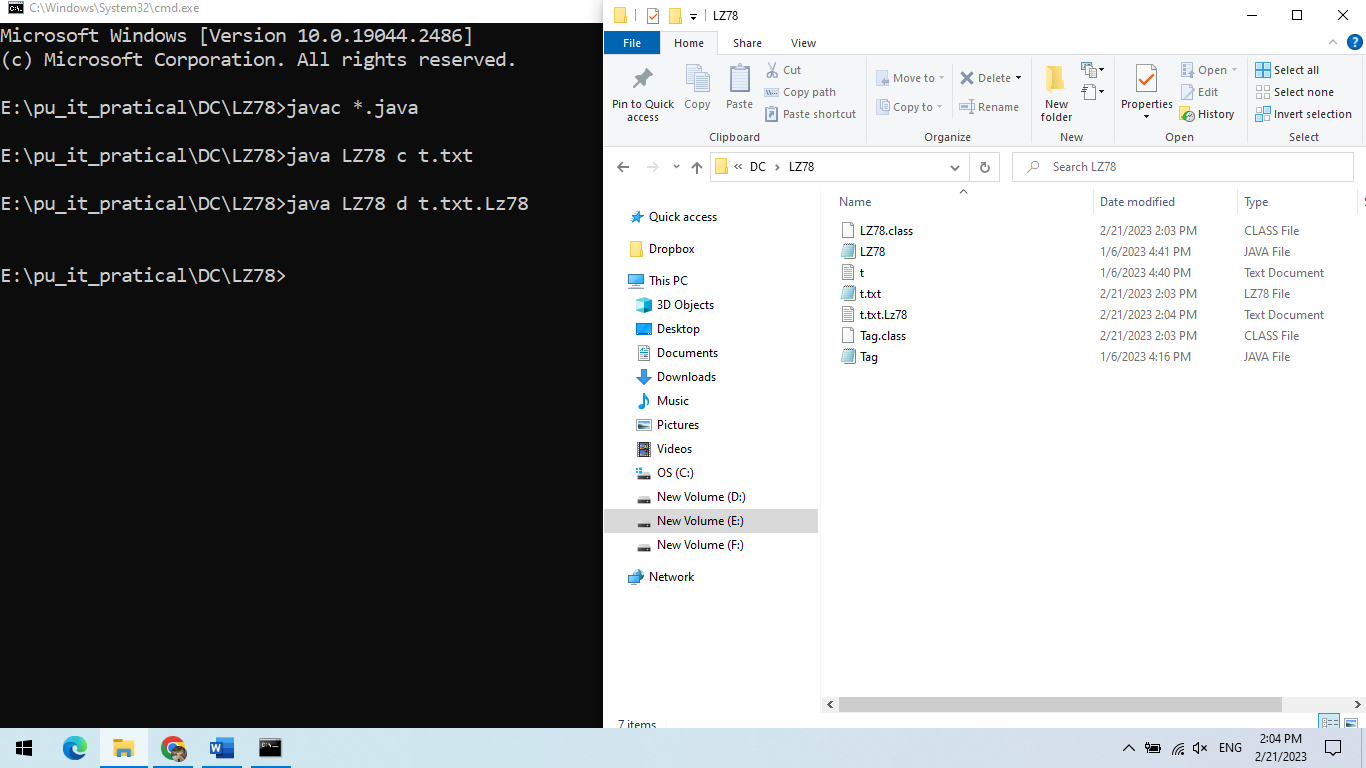
public void setNextChar(char nextChar) { this.nextChar = nextChar; }}

**Output:**

**Before**



**After**



**PRACTICAL-9**

**AIM:** **Write a program which performs JPEG compression, process step by step for given 8x8 block and decompress.**

**Code:**

import numpy as np

quantization\_matrix = 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],

]

)

def jpeg\_compress(block):

shifted\_block = block - 128

dct\_block = shifted\_block

quantized\_block = np.round(dct\_block / quantization\_matrix)

quantized\_block = quantized\_block.flatten()

return quantized\_block

def jpeg\_decompress(quantized\_block):

quantized\_block = quantized\_block.reshape((8, 8))

dequantized\_block = quantized\_block \* quantization\_matrix

idct\_block = dequantized\_block

reconstructed\_block = np.clip(idct\_block + 128, 0, 255).astype(np.uint8)

return reconstructed\_block

def random\_8x8():

return np.random.randint(0, 256, size=(8, 8)).astype(np.uint8)

matrix = random\_8x8()

print("Original Matrix:")

print(matrix)

compressed\_matrix = jpeg\_compress(matrix)

print("\nCompressed Matrix:")

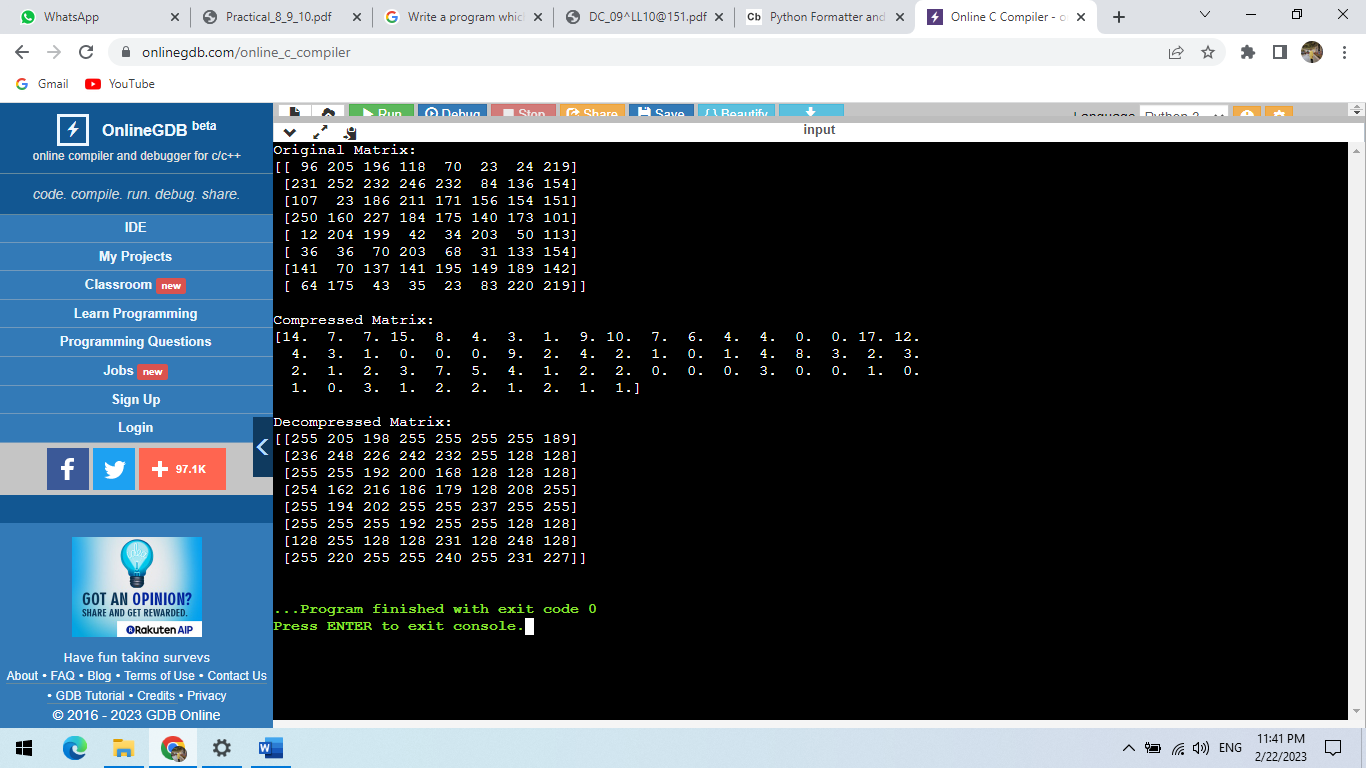
print(compressed\_matrix)

decompressed\_matrix = jpeg\_decompress(compressed\_matrix)

print("\nDecompressed Matrix:")

print(decompressed\_matrix)

**Output:**



**PRACTICAL-10**

**AIM:** **Write A Program To Implement Move-To-Front Algorithm.**

**Code:**

import java.util.\*;

public class MoveToFront {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the text to encode: ");

String text = scanner.nextLine();

String encodedText = encode(text);

System.out.println("Encoded text: " + encodedText);

String decodedText = decode(encodedText);

System.out.println("Decoded text: " + text);

}

public static String encode(String text) {

List < Character > alphabet = new ArrayList < > ();

for (int i = 0; i < 256; i++) {

alphabet.add((char) i);

}

StringBuilder sb = new StringBuilder();

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

char c = text.charAt(i);

int index = alphabet.indexOf(c);

sb.append(index);

alphabet.remove(index);

alphabet.add(0, c);

}

return sb.toString();

}

public static String decode(String encodedText) {

List < Character > alphabet = new ArrayList < > ();

for (int i = 0; i < 256; i++) {

alphabet.add((char) i);

}

StringBuilder sb = new StringBuilder();

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

int index = Integer.parseInt(Character.toString(encodedText.charAt(i)));

char c = alphabet.get(index);

sb.append(c);

alphabet.remove(index);

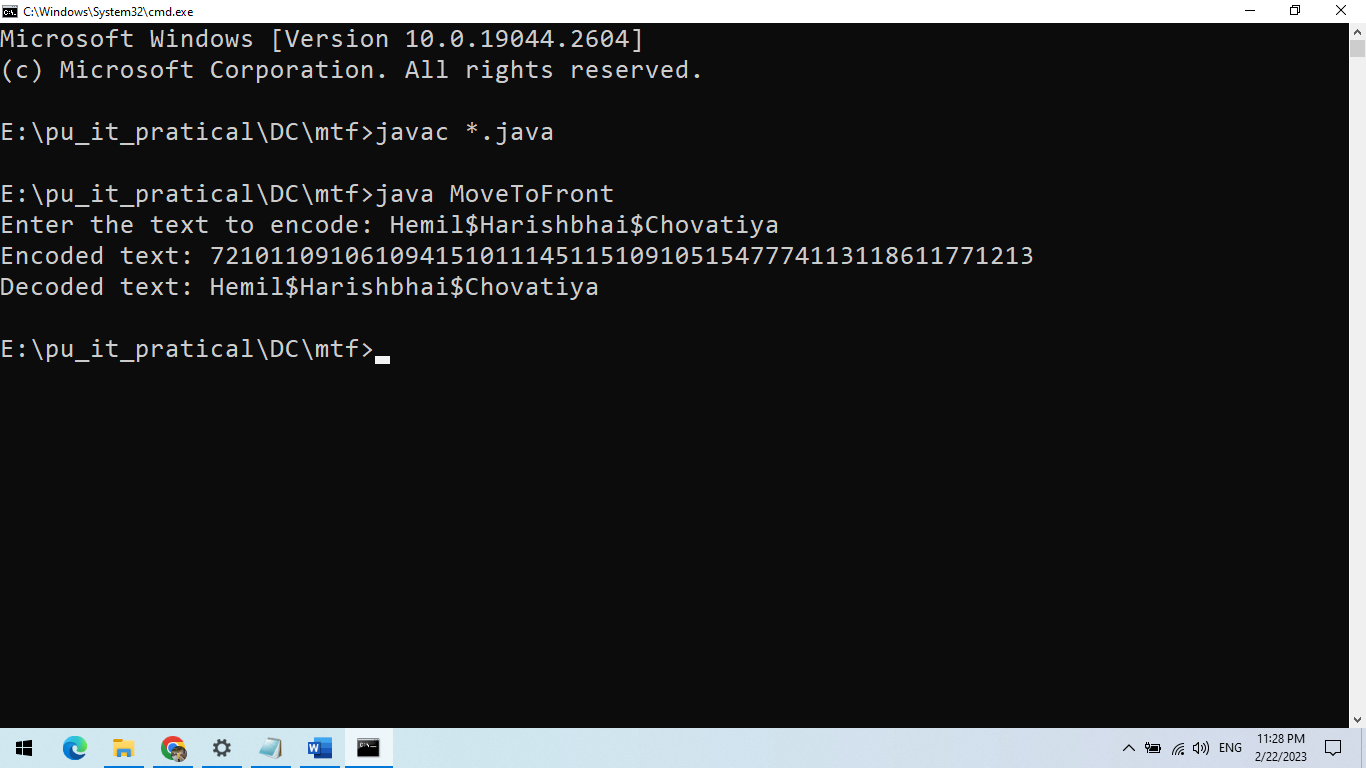
alphabet.add(0, c);

}

return sb.toString();

} }

**Output:**



**PRACTICAL-1**

**AIM:** **Write A Program To Implement Burrows Wheeler Transform Algorithm.**

**Code:**

import java.util.\*;

public class BurrowsWheelerTransform {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the text to transform: ");

String text = scanner.nextLine();

String transformedText = transform(text);

System.out.println("Transformed text: " + transformedText);

System.out.println("Decode Message:" + text); }

public static String transform(String text) {

int length = text.length();

String[] rotations = new String[length];

for (int i = 0; i < length; i++) { rotations[i] = text.substring(i) + text.substring(0, i); }

Arrays.sort(rotations);

StringBuilder sb = new StringBuilder();

for (int i = 0; i < length; i++) { sb.append(rotations[i].charAt(length - 1)); }

return sb.toString();

} }

**Output:**

