PRACTICAL-3

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

Algorithm :

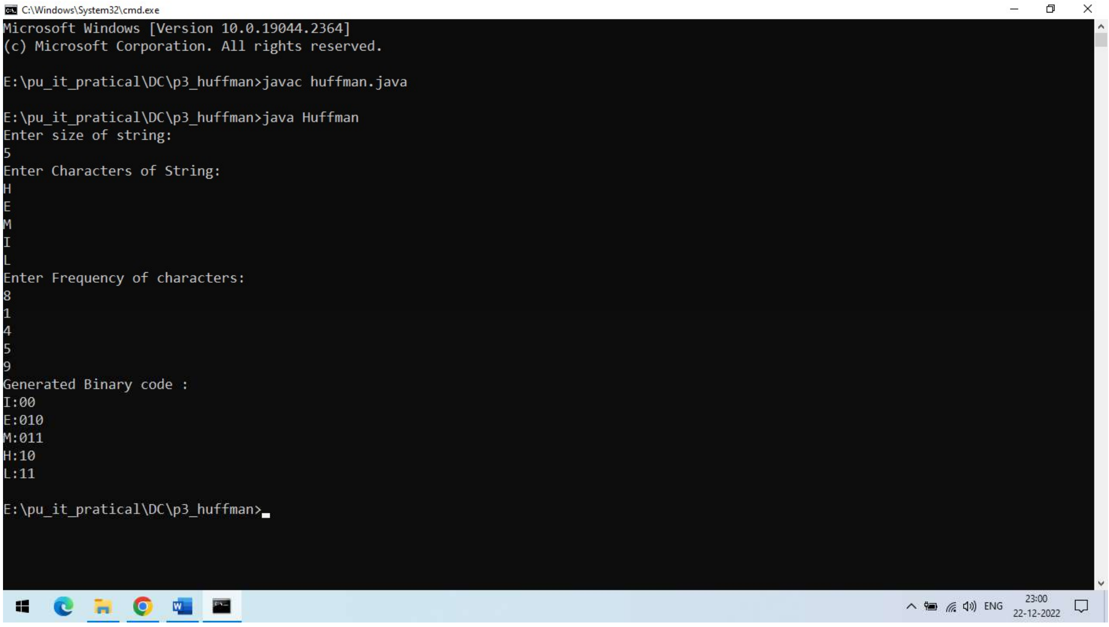
```
Huffman (C)
n=|C|
Q=C
for i=1 to n-1
   do
    z=allocate_Node()
Node()
x=left[z]=Extract_Min(Q)
y=right[z]=Extract_Min(Q)
f[z]=f[x]+f[y]
Insert(Q,z)
return Extract_Min(Q)
```

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:



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

E:\pu_it_pratical\DC\p3_huffman>javac huffman.java

E:\pu_it_pratical\DC\p3_huffman>java Huffman
Enter size of string:
5
Enter Characters of String:
H
E
M
I
L
Enter Frequency of characters:
8
1
4
5
9
Generated Binary code :
I:00
E:010
M:011
H:10
L:11

E:\pu_it_pratical\DC\p3_huffman>
```

PRACTICAL-4

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

Algorithm :

Steps of Huffman Encoding are:

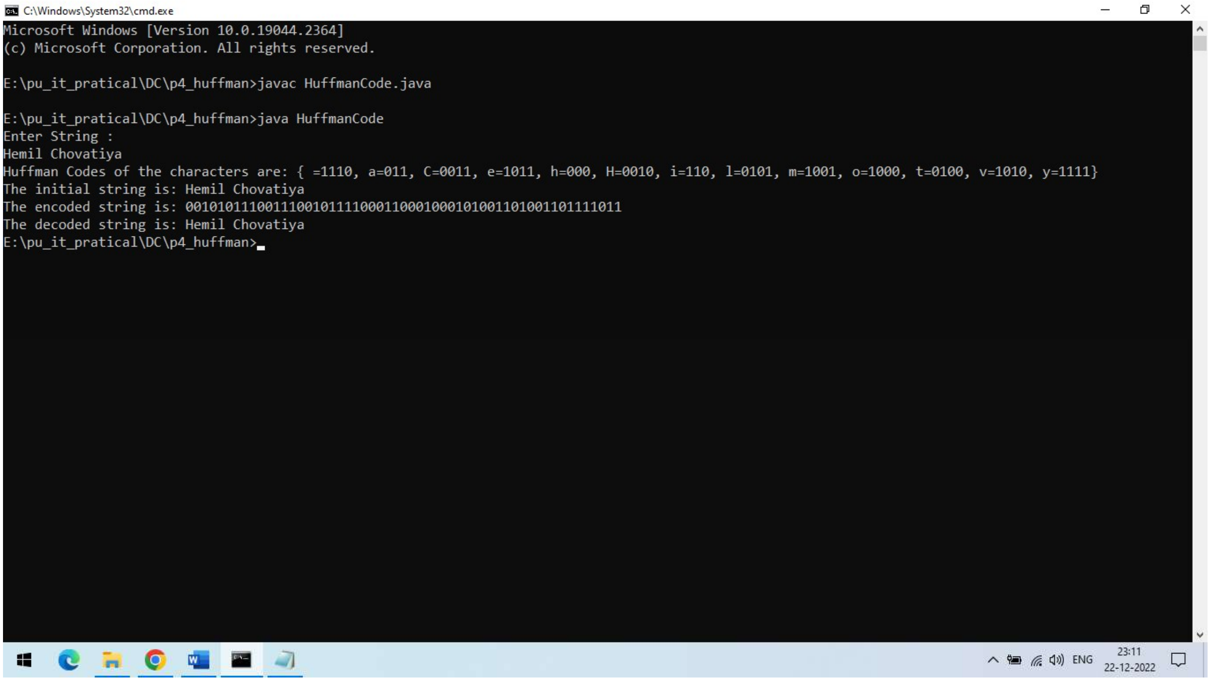
- Create And Initialize A Priorityqueue Queue Consisting Of Each Unique Character.
- Sort In Ascending Order Of Their Frequencies.
- For All The Unique Characters:
- Create A New_Node
- Get Minimum_Value From Queue And Set It To Left Child Of New_Node
- Get Minimum_Value From Queue And Set It To Right 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:



```
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

E:\pu_it_pratical\DC\p4_huffman>javac HuffmanCode.java

E:\pu_it_pratical\DC\p4_huffman>java HuffmanCode
Enter String :
Hemil Chovatiya
Huffman Codes of the characters are: { =1110, a=011, C=0011, e=1011, h=000, H=0010, i=110, l=0101, m=1001, o=1000, t=0100, v=1010, y=1111}
The initial string is: Hemil Chovatiya
The encoded string is: 0010101110011100101111000110001000101001101001101111011
The decoded string is: Hemil Chovatiya
E:\pu_it_pratical\DC\p4_huffman>
```