Project Milestone #02

Algorithm Pseudo code

Group ID# CS311-G28

→ Rehman Ali (2018-CS-17)

→ Muhamad Nauman (2018-CS-98)

⇒ Huffman's Coding Compression Technique:

The technique works by creating a binary tree of Nodes. These can stored in a regular array, the size of which depends upon the number of symbols, n. A node can either be a leaf Node or an internal Node. Initially all Nodes are leaf Nodes, which contains the symbol itself, its frequency and optionally a link to its child Nodes. As a convention bit '1' represent right child and bit '0' represent left child. Priority Queue is used to store the nodes, which provide the Nodes with lowest frequency when poped.

Pseudo Code :

```
Huffman_Compression (C)   // C is set of characters.
n = C.size
Q = priority_queue()
for i=1 to n
    n = node(C[i])
    Q.push(n)

While Q.size() is Not equal to 1
    Z = new node()
    Z.left = x = Q.pop()
    Z.right = y = Q.pop()
    Z.frequency = x.frequency + y.frequency
    Q.push(Z)

return Q
```

Decompression Technique :

The process of decompression is simply a matter of translating the stream of prefix code of individual byte value, usually by traversing a leaf node by node as each bit is read from input stream. Reaching the leaf Node necessarily Terminates the search for that particular byte value.

The leaf value represent the desired character. Usually the Huffman Tree is constructed using statistically adjusted data on each compression cycle thus the reconstruction is fairly simple.

Pseudo Code :

```
Huffman_Decompression(root, s)   // root is Tree root
n = s.length                     // s is bit stream decompose
for i = 1 to n
    current = root
    while current.left != NULL and current.right
                                        != NULL
        if s[i] is equal to '0'
            current = current.left
        else
            current = current.right
        i = i + 1
    Print current.Symbol
```

Our idea is to assign variable length codes to input characters, length of the assigned code are based on the frequency of corresponding character. We create a binary Tree are as far to operate in bottom-up manner so that least character gets the minimum code.