Information Theory and Coding (ITC)

Roll No.-122CS0300
Name- Anmol Agrawal   Date:12/02/2025

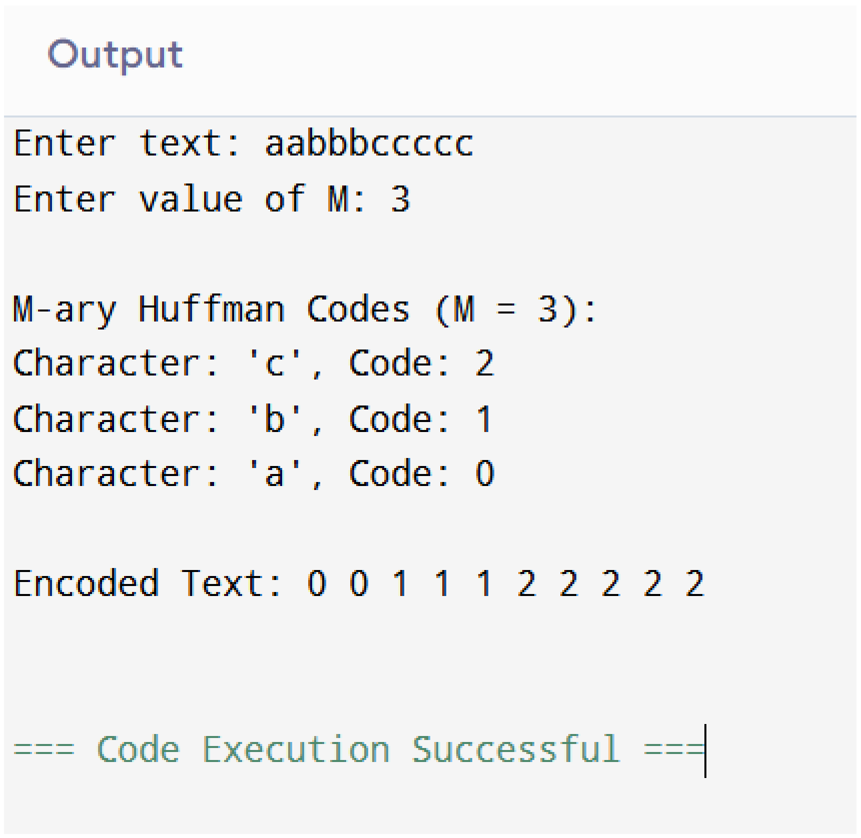
Classwork

Q.2 m-ary Huffman Coding (in cpp)

```
/*

 █████╗ ███╗   ██╗███╗   ███╗ ██████╗ ██╗        ██╗ █████╗
██╔══██╗████╗  ██║████╗ ████║██╔═══██╗██║
███║██╔══██╗
███████║██╔██╗ ██║██╔████╔██║██║   ██║██║
╚██║ ╚█████╔╝
██╔══██║██║╚██╗██║██║╚██╔╝██║██║   ██║██║
██║██╔══██╗
██║  ██║██║ ╚████║██║ ╚═╝
██║ ╚██████╔╝███████╗███████╗██║ ╚█████╔╝
╚═╝  ╚═╝╚═╝  ╚═══╝╚═╝   ╚═╝ ╚══════╝ ╚══════╝╚══════╝╚═╝
╚════╝

*/
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
#define f first
#define s second
vector<vector<int>>dir {{-1,0},{0,-1},{1,0},{0,1}};
#define MOD 1000000007
#define YES cout << "YES" << endl;
#define NO cout << "NO" << endl;
#define takenk(n,k) cin >> n >> k;
#define takenxy(n,x,y)  cin >> n >> x >> y;
#define sort(nums) sort(begin(nums), end(nums))
#define rsort(nums) sort(rbegin(nums), rend(nums))
```

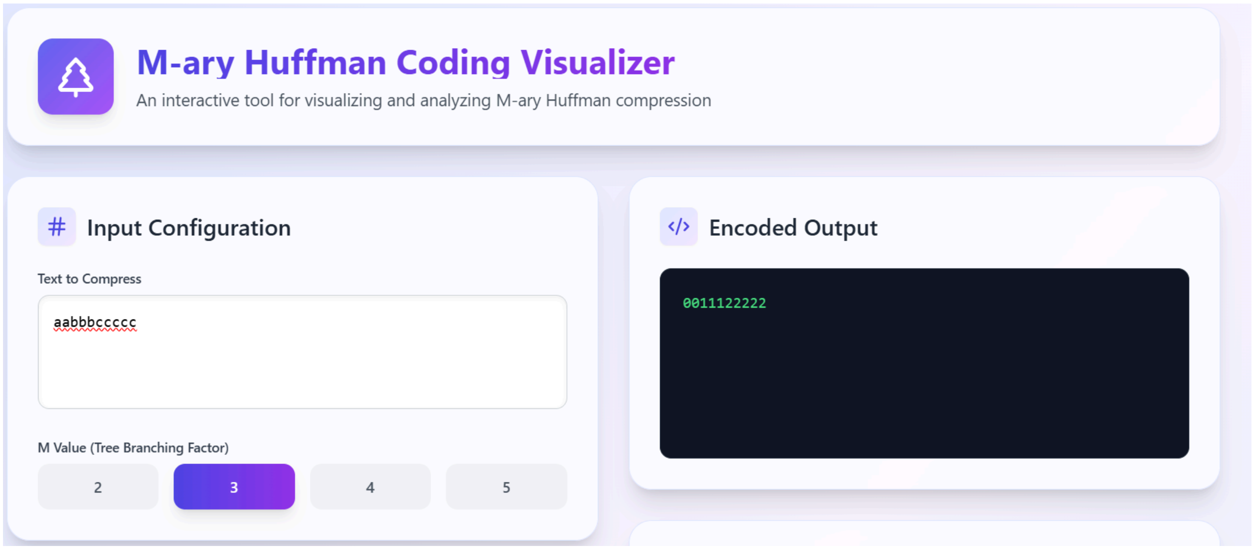
```
#define tot(nums) accumulate(begin(nums), end(nums), 0)
#define minv(nums) *min_element(begin(nums), end(nums))
#define maxv(nums) *max_element(begin(nums), end(nums))
#define rev(nums) reverse(begin(nums),end(nums))
typedef vector<int> vi;
typedef vector<vector<int>> vvi;
typedef vector<ll> vll;
typedef priority_queue<int> pq;
typedef priority_queue<int, vector<int>, greater<int>> pqq;
typedef pair<int, int> pi;
typedef set<int> gf;
typedef stack<int> st;
template<typename K, typename V>
using mpp = map<K, V>;
template<typename K, typename V>
using mp = unordered_map<K, V>;

// Node structure for M-ary Huffman Tree
class Node {
public:
    char character;
    int frequency;
    vector<Node*> children;
    Node(char ch, int freq) : character(ch), frequency(freq) {}
};

// Comparator for priority queue (min-heap)
struct Compare {
    bool operator()(Node* a, Node* b) {
        return a->frequency > b->frequency;
    }
};

// Huffman Coding for M-ary trees
class MAryHuffman {
public:
    // Function to calculate character frequencies
    mp<char, int> calculateFrequencies(const string& text) {
        mp<char, int> freqTable;
        for (char ch : text) {
```

```
        freqTable[ch]++;
    }
    return freqTable;
}

// Build the M-ary Huffman Tree
Node* buildHuffmanTree(mp<char, int>& freqTable, int M) {
    priority_queue<Node*, vector<Node*>, Compare> minHeap;

    // Insert all characters into priority queue
    for (auto& p : freqTable) {
        minHeap.push(new Node(p.f, p.s));
    }

    // Calculate dummy nodes required for balancing
    int totalSymbols = freqTable.size();
    int extraNodes = (M - 1 - (totalSymbols - 1) % (M - 1)) % (M - 1);

    // Add dummy nodes (zero frequency) to balance the tree
    for (int i = 0; i < extraNodes; i++) {
        minHeap.push(new Node('\0', 0));
    }

    // Merge M least frequent nodes iteratively
    while (minHeap.size() > 1) {
        vector<Node*> selectedNodes;
        int mergedFrequency = 0;

        // Extract M least frequent nodes
        for (int i = 0; i < M && !minHeap.empty(); i++) {
            Node* minNode = minHeap.top();
            minHeap.pop();
            selectedNodes.pb(minNode);
            mergedFrequency += minNode->frequency;
        }

        // Create new parent node
        Node* parent = new Node('\0', mergedFrequency);
        parent->children = selectedNodes;
        minHeap.push(parent);
        }

        return minHeap.top();
    }

    // Generate Huffman Codes using DFS traversal
    void generateHuffmanCodes(Node* node, string code, mp<char, string>&
huffman_codes) {
        if (!node) return;

        // If it's a valid character (not an internal node)
        if (node->character != '\0') {
            huffman_codes[node->character] = code;
        }

        // Traverse children and assign codes
        for (int i = 0; i < (int)node->children.size(); i++) {
            generateHuffmanCodes(node->children[i], code + to_string(i), huffman_codes);
        }
    }
};

int main() {
    string text;
    int M;

    cout << "Enter text: ";
    getline(cin, text);
    cout << "Enter value of M: ";
    cin >> M;

    MAryHuffman huffman;
    mp<char, int> freqTable = huffman.calculateFrequencies(text);
    Node* root = huffman.buildHuffmanTree(freqTable, M);
    mp<char, string> huffman_codes;
    huffman.generateHuffmanCodes(root, "", huffman_codes);

    cout << "\nM-ary Huffman Codes (M = " << M << "):\n";
    for (auto& p : huffman_codes) {
        cout << "Character: '" << p.f << "', Code: " << p.s << endl;
    }

    string encodedText = "";
    for (char ch : text) {
        encodedText += huffman_codes[ch] + " ";
    }
    cout << "\nEncoded Text: " << encodedText << endl;

    return 0;
}
```



Output

```
Enter text: aabbbccccc
Enter value of M: 3

M-ary Huffman Codes (M = 3):
Character: 'c', Code: 2
Character: 'b', Code: 1
Character: 'a', Code: 0

Encoded Text: 0 0 1 1 1 2 2 2 2 2



=== Code Execution Successful ===
```



M-ary Huffman Coding Visualizer

An interactive tool for visualizing and analyzing M-ary Huffman compression

Input Configuration

Text to Compress

```
aabbbccccc
```

M Value (Tree Branching Factor)

2 [✓ 3] 4 5

Encoded Output

```
0011122222
```

[Note: Code was run on programmiz (online compiler)]