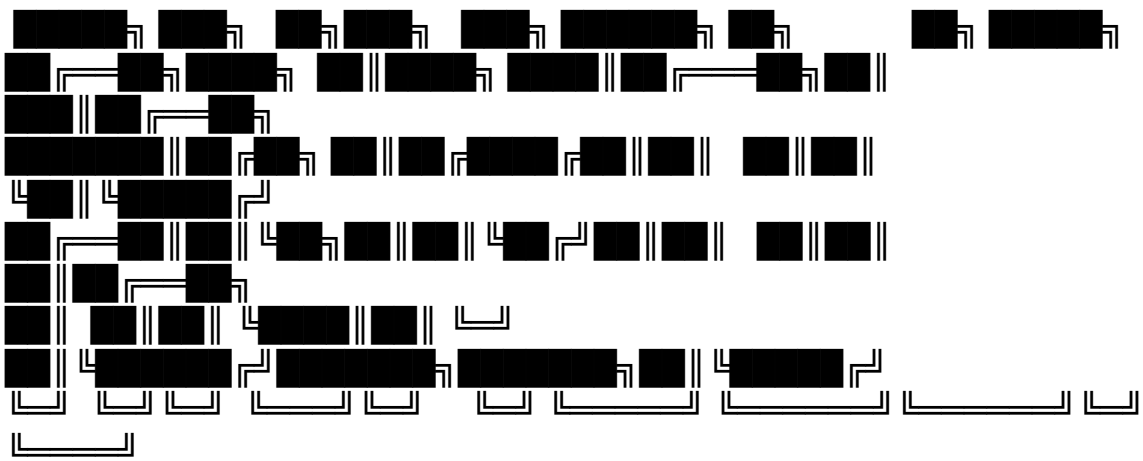
Information Theory and Coding (ITC)

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

Classwork

Q.1 Extended 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>;

class Node {
public:
    string symbol;
    double probability;
    Node* left;
    Node* right;

    Node(string sym = "", double prob = 0.0) {
        symbol = sym;
        probability = prob;
        left = right = nullptr;
    }
};

// Custom comparator for priority queue
struct CompareNode {
    bool operator()(const Node* a, const Node* b) {
        return a->probability > b->probability;
    }
};

// Generate all possible n-length combinations
```

```
mp<string, double> generate_key_expansion(const mp<string, double>&
symbols_probs, int n) {
    mp<string, double> combinations;
    vector<string> symbols;
    vector<double> probs;

    // Split symbols and probabilities
    for (const auto& pair : symbols_probs) {
        symbols.pb(pair.f);
        probs.pb(pair.s);
    }

    // Generate combinations using recursive function
    function<void(string, double, int)> generate = [&](string current, double current_prob,
int depth) {
        if (depth == n) {
            combinations[current] = current_prob;
            return;
        }

        for (ll i = 0; i < (ll)symbols.size(); i++) {
            generate(current + symbols[i], current_prob * probs[i], depth + 1);
        }
    };

    generate("", 1.0, 0);
    return combinations;
}

// Build Huffman tree
Node* build_huffman_tree(const mp<string, double>& symbols_probs) {
    priority_queue<Node*, vector<Node*>, CompareNode> tree;

    // Create nodes and add to priority queue
    for (const auto& pair : symbols_probs) {
        tree.push(new Node(pair.f, pair.s));
    }

    // Build tree
    while (tree.size() > 1) {
```

```
        Node* left = tree.top(); tree.pop();
        Node* right = tree.top(); tree.pop();

        Node* merged = new Node();
        merged->probability = round((left->probability + right->probability) * 10000) /
10000;
        merged->left = left;
        merged->right = right;

        tree.push(merged);
    }

    return tree.empty() ? nullptr : tree.top();
}

// Generate Huffman codes
void generate_huffman_codes(Node* node, string code, mp<string, string>&
huffman_codes) {
    if (node != nullptr) {
        if (!node->symbol.empty()) {
            huffman_codes[node->symbol] = code;
        }
        generate_huffman_codes(node->left, code + "0", huffman_codes);
        generate_huffman_codes(node->right, code + "1", huffman_codes);
    }
}

int main() {
    // Example
    mp<string, double> symbols_probs = {
        {"A", 0.3},
        {"B", 0.2},
        {"C", 0.5}
    };
    int n = 2;

    // Generate combinations
    auto combinations = generate_key_expansion(symbols_probs, n);

    // Build Huffman tree
```

```
    Node* root = build_huffman_tree(combinations);

    // Generate and print Huffman codes
    mp<string, string> huffman_codes;
    generate_huffman_codes(root, "", huffman_codes);

    cout << "\nHuffman codes:" << endl;
    for (const auto& pair : huffman_codes) {
        cout << "Character: " << pair.f << ", Code: " << pair.s << endl;
    }

    return 0;
}
```

Output

```
Huffman codes:
Character: AA, Code: 1101
Character: CA, Code: 111
Character: AB, Code: 1100
Character: CC, Code: 10
Character: AC, Code: 011
Character: BA, Code: 0101
Character: BB, Code: 0100
Character: BC, Code: 001
Character: CB, Code: 000


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

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