Marwadi University	**Marwadi University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: DSC (01CT0308)	**Aim:** Implementations of Huffman code construction.	
Experiment No: 7	**Date: 26- 10 - 2023**	**Enrolment No:-** 92200133030

Experiment – 7

Objective: Implementations of Huffman code construction.

Code :-

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
class HuffmanNode {
public:
    char symbol;
    int frequency;
    HuffmanNode* left;
    HuffmanNode* right;
    HuffmanNode(char sym, int freq) : symbol(sym), frequency(freq), left(nullptr),
right(nullptr) {}
};
class CompareNodes {
public:
    bool operator()(HuffmanNode* a, HuffmanNode* b) {
        return a->frequency > b->frequency;
    }
};
class HuffmanCoding {
public:
    HuffmanCoding(const string& input) : input_(input) {}
    void buildHuffmanTree() {
        vector<pair<char, int>> frequencies;
        for (char c : input_) {
            bool found = false;
            for (auto& pair : frequencies) {
                if (pair.first == c) {
```

```
                pair.second++;
                found = true;
                break;
            }
        }
        if (!found) {
            frequencies.push_back({c, 1});
        }
    }
    nodes_.clear();
    for (const auto& pair : frequencies) {
        HuffmanNode* node = new HuffmanNode(pair.first, pair.second);
        nodes_.push_back(node);
    }
    while (nodes_.size() > 1) {
        sort(nodes_.begin(), nodes_.end(), CompareNodes());
        HuffmanNode* left = nodes_.back();
        nodes_.pop_back();
        HuffmanNode* right = nodes_.back();
        nodes_.pop_back();
        HuffmanNode* newNode = new HuffmanNode('\0', left->frequency + right-
>frequency);
        newNode->left = left;
        newNode->right = right;
        nodes_.push_back(newNode);
    }
    root_ = nodes_[0];
}
void generateHuffmanCodes() {
    huffmanCodes_.clear();
    generateHuffmanCodes(root_, "", huffmanCodes_);
}
string encodeData() {
    string encodedData;
    for (char c : input_) {
        for (const auto& pair : huffmanCodes_) {
            if (pair.first == c) {
                encodedData += pair.second;
                break;
            }
        }
    }
    return encodedData;
}
string decodeData(const string& encodedData) {
    string decodedData;
    HuffmanNode* currentNode = root_;
```

```
        for (char bit : encodedData) {
            if (bit == '0') {
                currentNode = currentNode->left;
            } else {
                currentNode = currentNode->right;
            }
            if (currentNode->symbol != '\0') {
                decodedData += currentNode->symbol;
                currentNode = root_;
            }
        }
        return decodedData;
    }
    void printHuffmanCodes() const {
        cout << "Huffman Codes:" << endl;
        for (const auto& pair : huffmanCodes_) {
            cout << pair.first << ": " << pair.second << endl;
        }
    }
private:
    string input_;
    HuffmanNode* root_;
    vector<HuffmanNode*> nodes_;
    vector<pair<char, string>> huffmanCodes_;
    void generateHuffmanCodes(HuffmanNode* root, string currentCode, vector<pair<char,
string>>& huffmanCodes) {
        if (!root) {
            return;
        }
        if (root->symbol != '\0') {
            huffmanCodes.push_back({root->symbol, currentCode});
        }
        generateHuffmanCodes(root->left, currentCode + '0', huffmanCodes);
        generateHuffmanCodes(root->right, currentCode + '1', huffmanCodes);
    }
};
int main() {
    cout << "Enter a string: ";
    string input;
    getline(cin, input);
    HuffmanCoding huffman(input);
    huffman.buildHuffmanTree();
    huffman.generateHuffmanCodes();
    huffman.printHuffmanCodes();
    string encodedData = huffman.encodeData();
    cout << "Encoded Data: " << encodedData << endl;
```

```
    string decodedData = huffman.decodeData(encodedData);
    cout << "Decoded Data: " << decodedData << endl;
    return 0;
}
```

Output:

```
Enter a string: ABCDEFGFEDCBA
Huffman Codes:
A: 00
G: 010
F: 011
C: 100
B: 101
E: 110
D: 111
Encoded Data: 00101100111110011010011110111110010100
Decoded Data: ABCDEFGFEDCBA
```