|  |  |  |
| --- | --- | --- |
| Image result for latest marwadi university logo | **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;

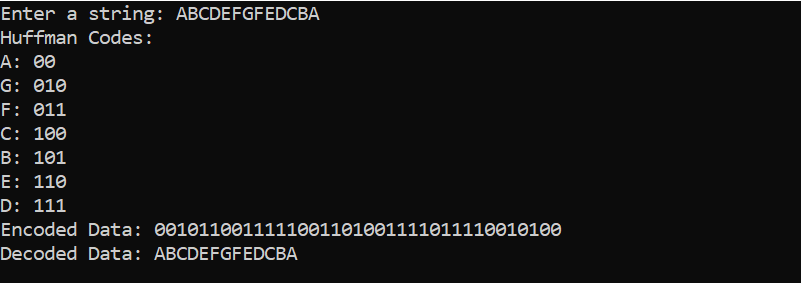
string decodedData = huffman.decodeData(encodedData);

cout << "Decoded Data: " << decodedData << endl;

return 0;

}

**Output:**

****