**DAA Assignment 2**

**Huffman Encoding using Greedy Algorithm**

**Neeti Kurulkar**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**#include <chrono>**

**#include <map>**

**#include <cmath>**

**#include <iomanip>**

**#include <algorithm>**

**using namespace std;**

**class Node {**

**public:**

**int freq;**

**char symbol;**

**Node\* left;**

**Node\* right;**

**char huff;**

**Node(int freq, char symbol, Node\* left = nullptr, Node\* right = nullptr)**

**: freq(freq), symbol(symbol), left(left), right(right), huff(0) {}**

**bool operator<(const Node& other) const {**

**return freq > other.freq; // For min-heap behavior in priority\_queue**

**}**

**};**

**void calculateHuffmanCodes(const Node\* node, const string& code, map<char, string>& huffmanCodes) {**

**if (node) {**

**if (!node->left && !node->right) {**

**huffmanCodes[node->symbol] = code;**

**}**

**calculateHuffmanCodes(node->left, code + "0", huffmanCodes);**

**calculateHuffmanCodes(node->right, code + "1", huffmanCodes);**

**}**

**}**

**int main() {**

**int n;**

**cout << "Enter the number of characters: ";**

**cin >> n;**

**vector<char> chars(n);**

**vector<int> freq(n);**

**cout << "\nEnter characters and their frequencies:\n";**

**for (int i = 0; i < n; ++i) {**

**cout << "Character " << i + 1 << ": ";**

**cin >> chars[i];**

**cout << "Frequency of " << chars[i] << ": ";**

**cin >> freq[i];**

**}**

**priority\_queue<Node> nodes;**

**for (int i = 0; i < n; ++i) {**

**nodes.push(Node(freq[i], chars[i]));**

**}**

**cout << "\n------------------------------------------\n";**

**cout << "[ Huffman Tree Construction ]\n";**

**auto start\_time = chrono::high\_resolution\_clock::now();**

**while (nodes.size() > 1) {**

**Node\* left = new Node(nodes.top());**

**nodes.pop();**

**Node\* right = new Node(nodes.top());**

**nodes.pop();**

**left->huff = '0';**

**right->huff = '1';**

**Node\* merged = new Node(left->freq + right->freq, left->symbol + right->symbol, left, right);**

**nodes.push(\*merged);**

**}**

**auto end\_time = chrono::high\_resolution\_clock::now();**

**auto duration = chrono::duration\_cast<chrono::nanoseconds>(end\_time - start\_time);**

**cout << "Time required : " << duration.count() << " nanoseconds\n";**

**cout << "\n[ Huffman Code Generation ]\n";**

**auto code\_start = chrono::high\_resolution\_clock::now();**

**map<char, string> huffmanCodes;**

**calculateHuffmanCodes(&nodes.top(), "", huffmanCodes);**

**auto code\_end = chrono::high\_resolution\_clock::now();**

**auto code\_duration = chrono::duration\_cast<chrono::nanoseconds>(code\_end - code\_start);**

**cout << "Time required : " << code\_duration.count() << " nanoseconds\n";**

**cout << "\n[ Space Estimation ]\n";**

**auto space\_start = chrono::high\_resolution\_clock::now();**

**double spaceUsed = 0;**

**for (const auto& kv : huffmanCodes) {**

**spaceUsed += kv.second.length() \* freq[find(chars.begin(), chars.end(), kv.first) - chars.begin()];**

**}**

**spaceUsed = ceil(spaceUsed / 8); // Convert bits to bytes**

**auto space\_end = chrono::high\_resolution\_clock::now();**

**auto space\_duration = chrono::duration\_cast<chrono::nanoseconds>(space\_end - space\_start);**

**cout << "Estimated space used : " << spaceUsed << " bytes\n";**

**cout << "Space calculation time : " << space\_duration.count() << " nanoseconds\n";**

**cout << "\n[ Huffman Codes ]\n";**

**cout << left << setw(10) << "Char" << setw(10) << "Code" << endl;**

**cout << "---------------------------\n";**

**for (const auto& kv : huffmanCodes) {**

**cout << left << setw(10) << kv.first << setw(10) << kv.second << endl;**

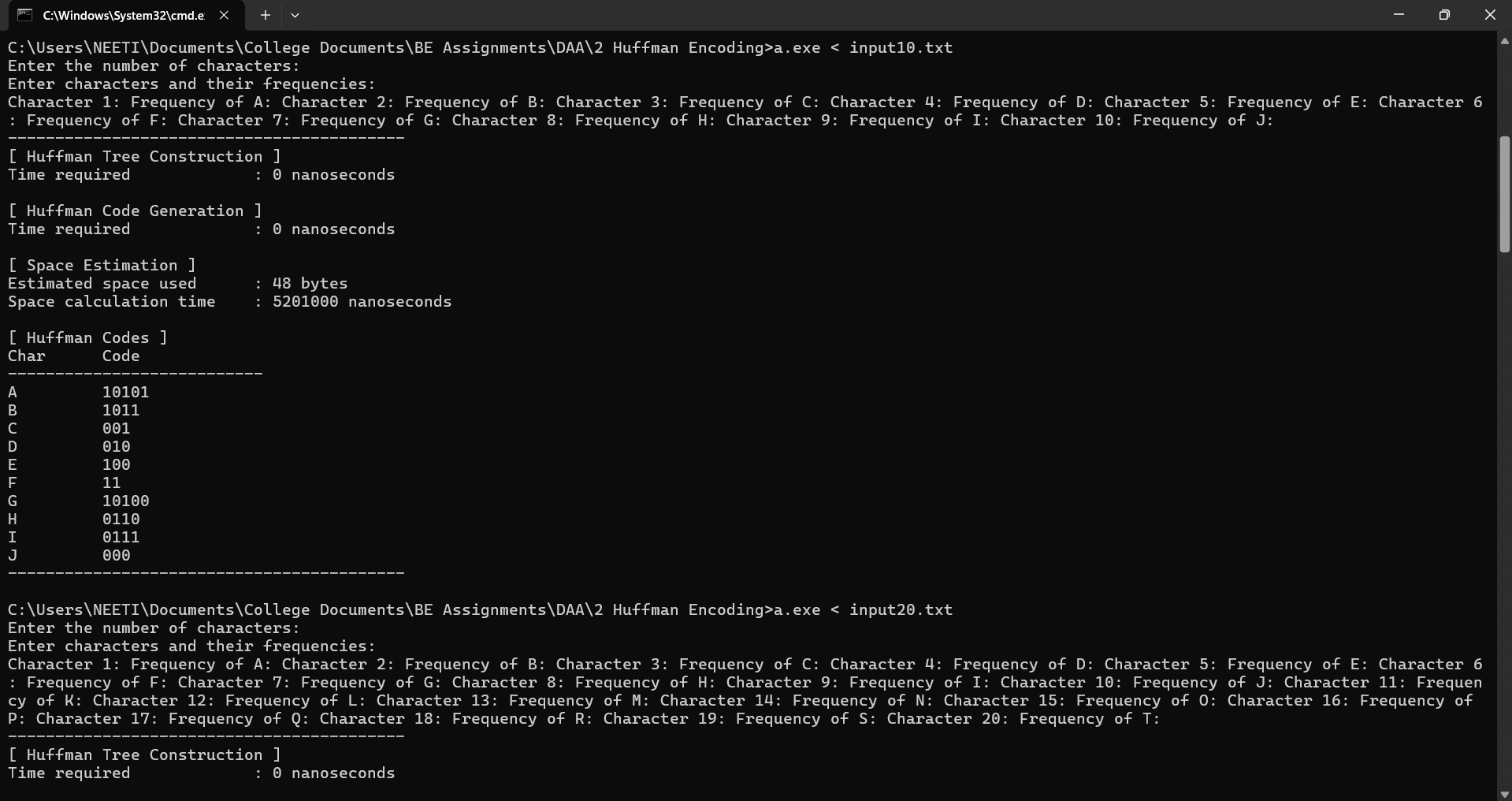
**}**

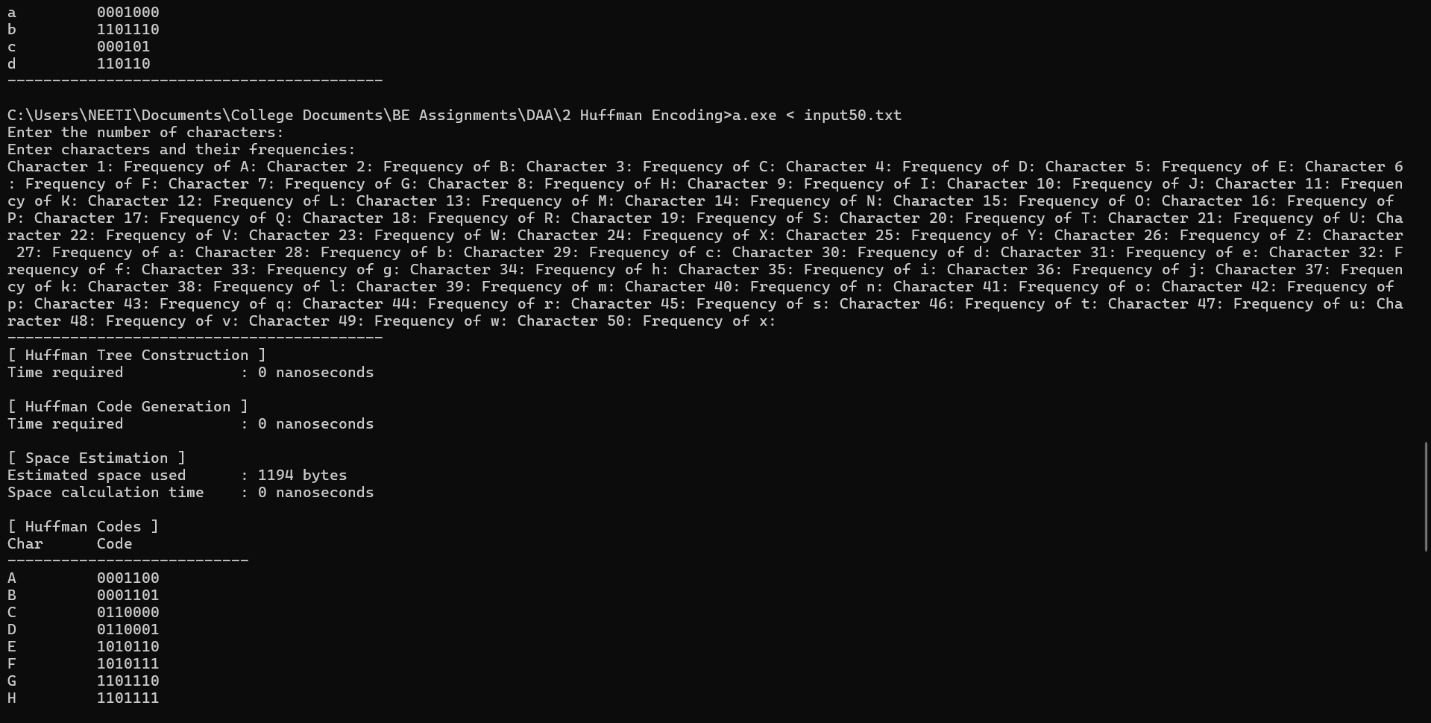
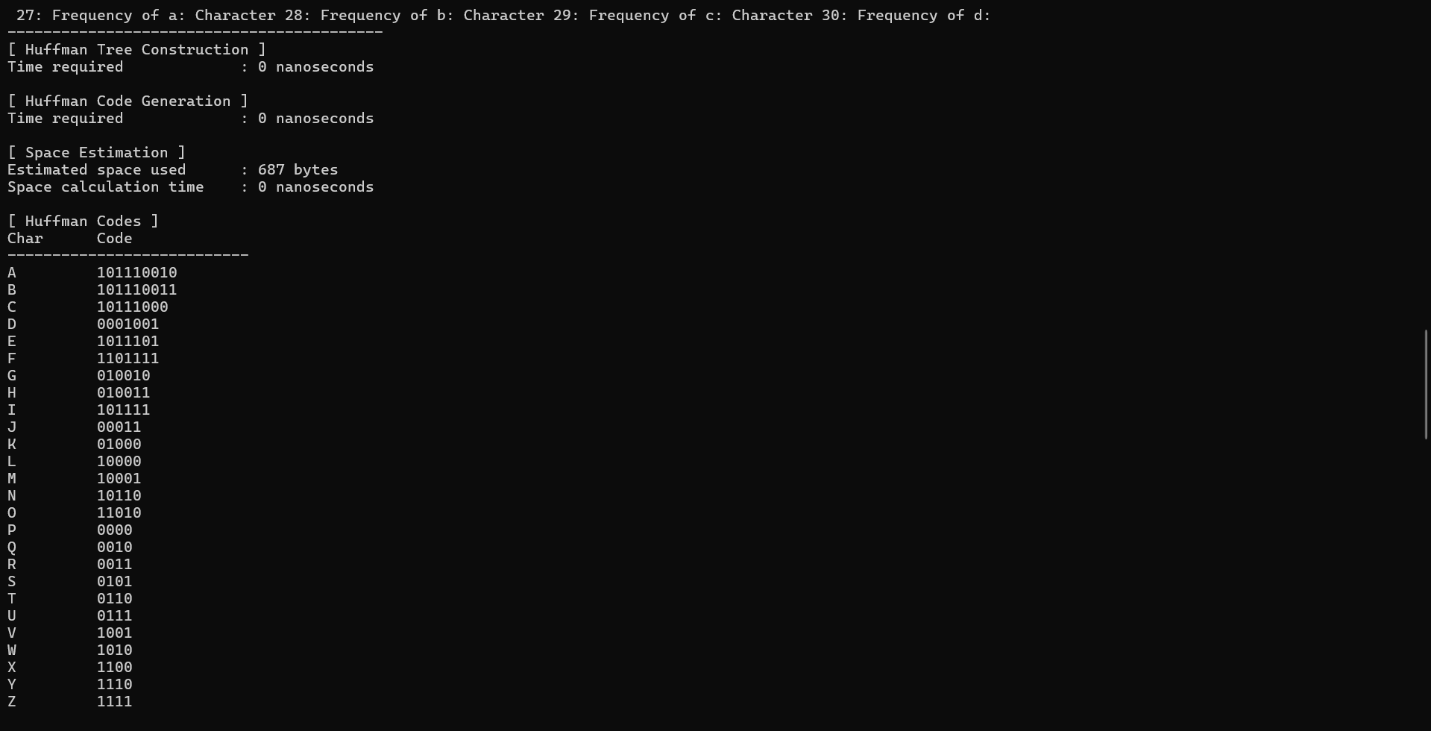
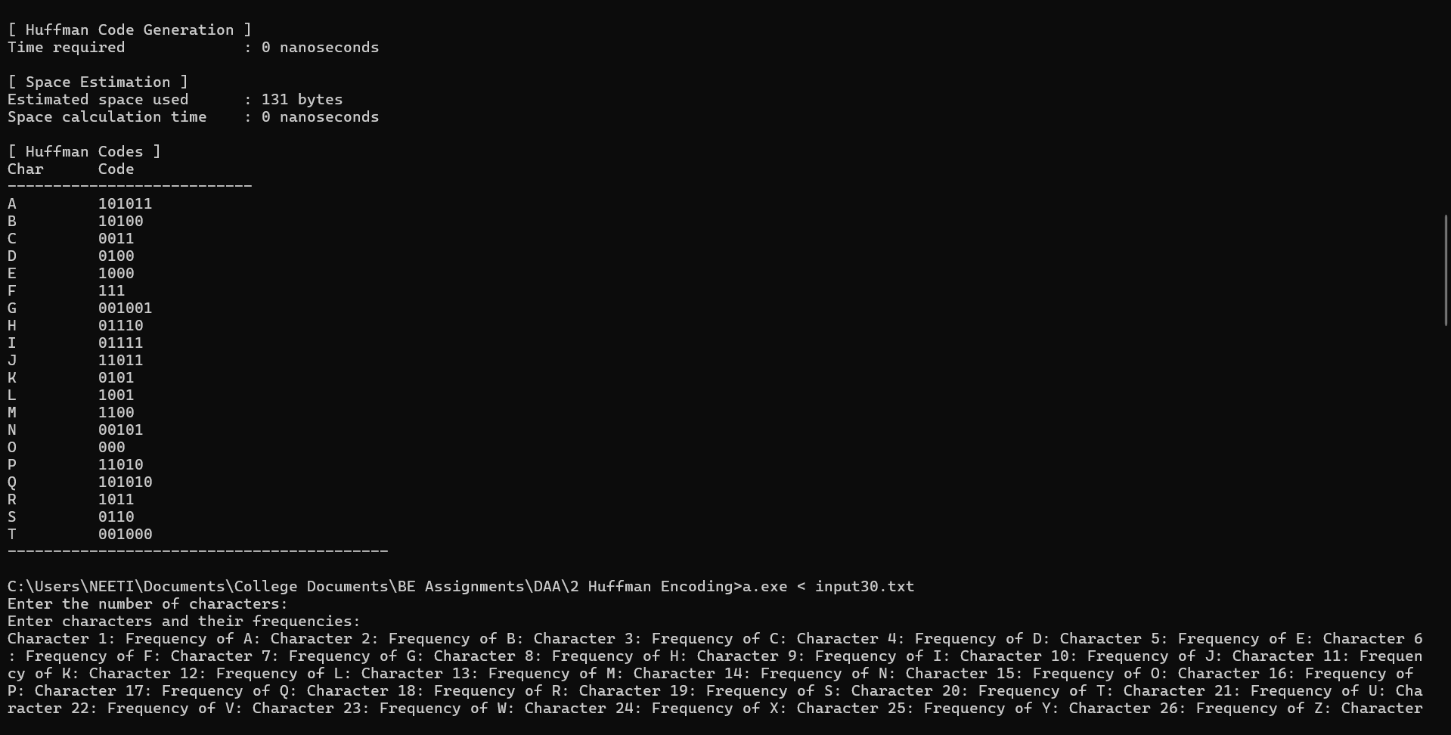
**cout << "------------------------------------------\n";**

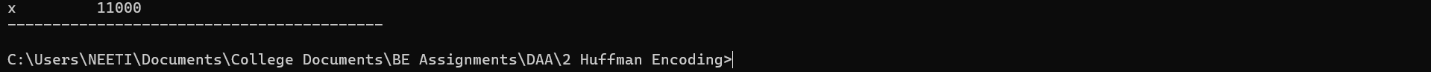
**return 0;**

**}**

**Output:**

****

**A black screen with a black background

AI-generated content may be incorrect.**