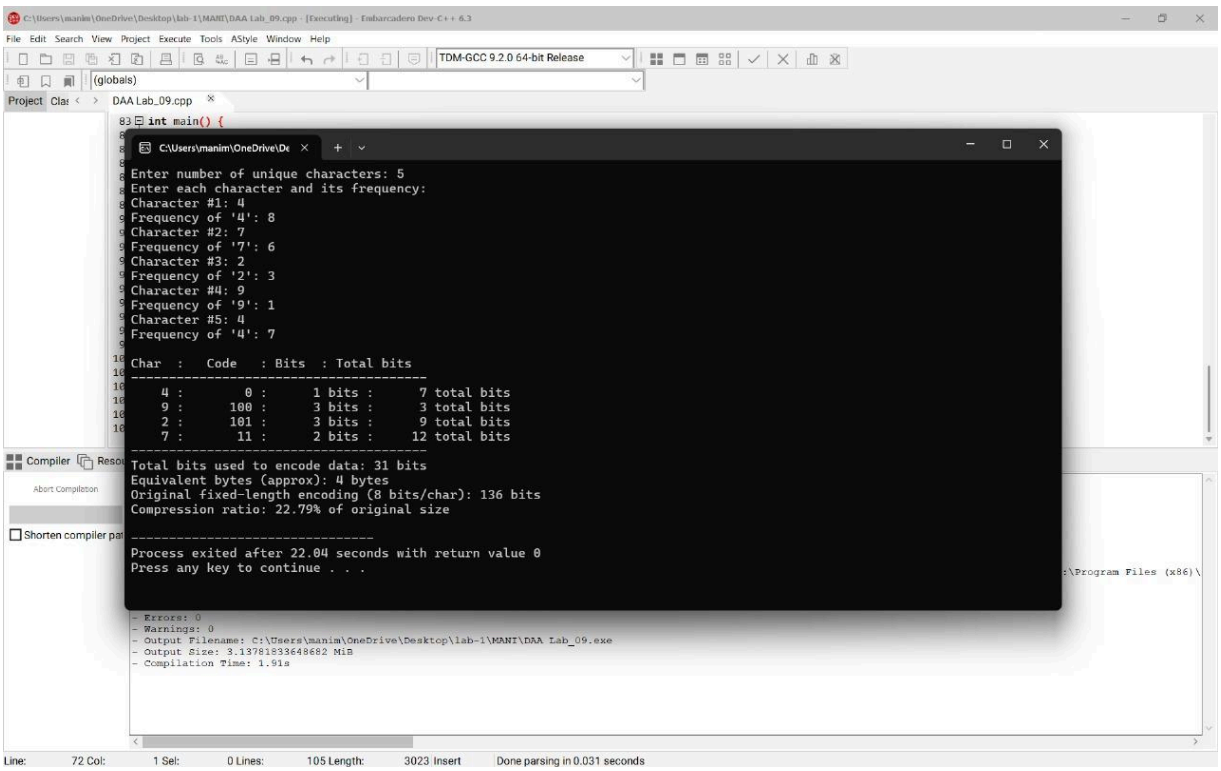
1.**HUFFMAN CODING**

```
#include <iostream>
#include <queue>
#include <unordered_map>
#include <iomanip>
using namespace std;

struct HuffmanNode {
    char ch;
    int freq;
    HuffmanNode *left, *right;

    HuffmanNode(char character, int frequency) {
        ch = character;
        freq = frequency;
        left = right = nullptr;
    }
};

struct Compare {
    bool operator()(HuffmanNode* l, HuffmanNode* r) {
        return l->freq > r->freq;
    }
};

void printHuffmanCodes(HuffmanNode* root, string code, const unordered_map<char, int>&
freqMap, int& totalBits) {
    if (!root) return;

    if (!root->left && !root->right) {
        int bits = code.length();
        int freq = freqMap.at(root->ch);
        int total = freq * bits;
        totalBits += total;

        cout << setw(5) << root->ch << " : "
             << setw(8) << code << " : "
             << setw(6) << bits << " bits : "
             << setw(6) << total << " total bits\n";
    }

    printHuffmanCodes(root->left, code + "0", freqMap, totalBits);
    printHuffmanCodes(root->right, code + "1", freqMap, totalBits);
```

```
}
void huffmanCoding(const unordered_map<char, int>& freqMap) {
    priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare> minHeap;

    for (auto pair : freqMap) {
        minHeap.push(new HuffmanNode(pair.first, pair.second));
    }

    while (minHeap.size() > 1) {
        HuffmanNode* left = minHeap.top(); minHeap.pop();
        HuffmanNode* right = minHeap.top(); minHeap.pop();

        HuffmanNode* merged = new HuffmanNode('\0', left->freq + right->freq);
        merged->left = left;
        merged->right = right;
        minHeap.push(merged);
    }

    HuffmanNode* root = minHeap.top();

    cout << "\nChar  :   Code   : Bits  : Total bits\n";
    cout << "---------------------------------------\n";

    int totalBits = 0;
    printHuffmanCodes(root, "", freqMap, totalBits);

    cout << "---------------------------------------\n";
    cout << "Total bits used to encode data: " << totalBits << " bits\n";
    cout << "Equivalent bytes (approx): " << (totalBits + 7) / 8 << " bytes\n";
    int totalChars = 0;
    for (auto& p : freqMap)
        totalChars += p.second;

    int originalBits = totalChars * 8;
    cout << "Original fixed-length encoding (8 bits/char): " << originalBits << " bits\n";
    cout << "Compression ratio: " << fixed << setprecision(2)
         << (100.0 * totalBits / originalBits) << "% of original size\n";
}

int main() {
    int n;
    cout << "Enter number of unique characters: ";
    cin >> n;
```

```
    unordered_map<char, int> freqMap;

    cout << "Enter each character and its frequency:\n";
    for (int i = 0; i < n; ++i) {
        char ch;
        int freq;
        cout << "Character #" << i + 1 << ": ";
        cin >> ch;
        cout << "Frequency of '" << ch << "': ";
        cin >> freq;
        freqMap[ch] = freq;
    }

    huffmanCoding(freqMap);

    return 0;
}
```



2. **BUILDING TREE BY HUFFMAN CODING**

2) Assume that the numbers given below represent counts of letters in the hundreds from a file (similar to the CLRS example). For example, in the file there will be exactly 20 * 100 occurrences of a letter 'a', 11 * 100 occurrence of the letter 'c', etc.

a:20, c:11, d:2, e:10, o:15, m:8, s:10, t:22, u:2

(a) What is an optimal Huffman code based on following set of frequencies.

1) Draw the tree. Show your work at every step.
2) Fill in the table on the right the Huffman encoding for each letter.
3) We encode the file using the Huffman codes produced above. How much memory will the file require with this encoding?

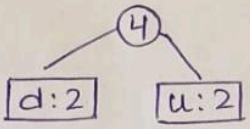
Given frequencies

a: 20, c:11, d:2, e:10, o:15, m:8, s:10, t:22, u:2

arranging in ascending order

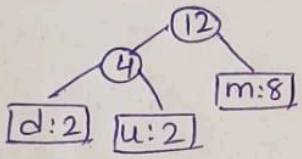
d:2, u:2, m:8, e:10, s:10, c:11, o:15, a:20, t:22

First two minimum frequencies (d,u) = (2,2)



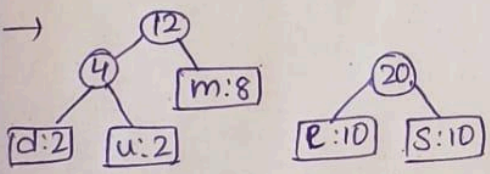
(4), 8, 10, 10, 11, 15, 20, 22

→



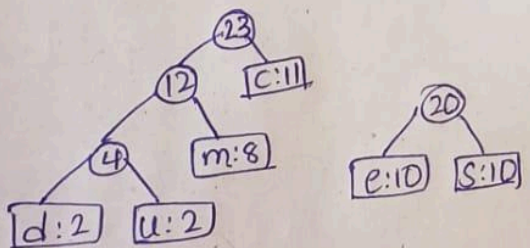
10, 10, 11, (12), 15, 20, 22

→



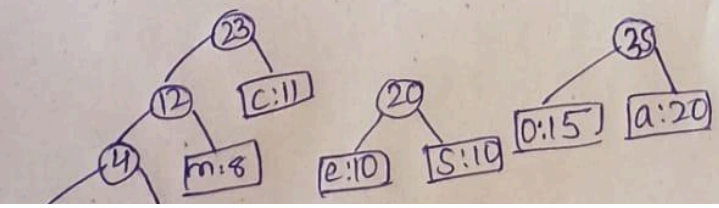
11, (12), 15, 20, (20), 22

→



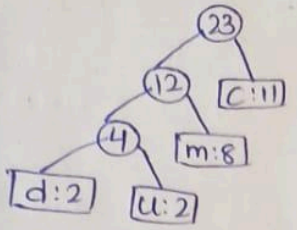
15, 20, (20), 22, (23)

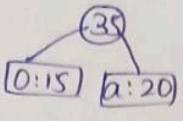
→

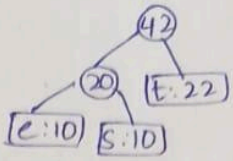


(20), 22, (23), (35)

→

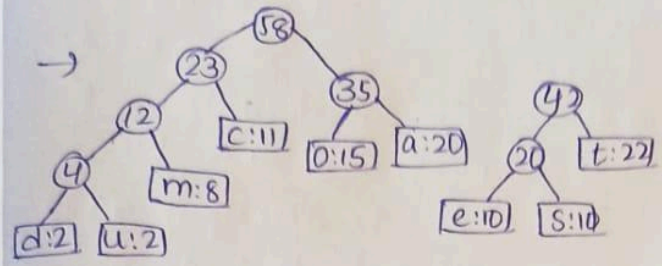






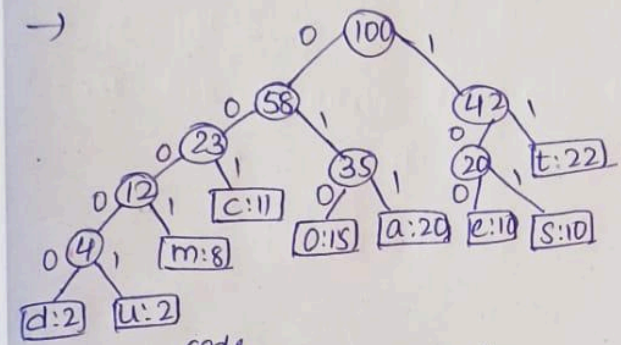
23, 35, 42

→



42, 58

→



	code	code length
d	00000	5
u	00001	5
m	0001	4
c	001	3
o	010	3
a	011	3
e	100	3
s	101	3
t	11	2

Total no. of bits = $\sum$(freq * length of codeword)

$= 2\times5 + 2\times5 + 8\times4 + 11\times3 + 15\times3 + 20\times3 + 10\times3 + 10\times3 + 22\times2$

$= 10 + 10 + 32 + 33 + 45 + 60 + 30 + 30 + 44$

$= 294$ bits