Huffman code

#include <bits/stdc++.h>

using namespace std;

// A Huffman tree node

struct Node {

char ch; // character

int freq; // frequency

Node \*left, \*right; // left and right child

Node(char c, int f) {

ch = c;

freq = f;

left = right = NULL;

}

};

// Custom comparator for min-heap

struct compare {

bool operator()(Node\* l, Node\* r) {

return l->freq > r->freq; // smaller freq has higher priority

}

};

// Print Huffman codes using DFS

void printCodes(Node\* root, string str) {

if (!root) return;

// If leaf node → print the character and its code

if (!root->left && !root->right) {

cout << root->ch << " : " << str << "\n";

}

printCodes(root->left, str + "0");

printCodes(root->right, str + "1");

}

// Build Huffman Tree and print codes

void HuffmanCodes(vector<char>& chars, vector<int>& freq) {

priority\_queue<Node\*, vector<Node\*>, compare> minHeap;

// Step 1: Create a leaf node for each character

for (int i = 0; i < chars.size(); i++) {

minHeap.push(new Node(chars[i], freq[i]));

}

// Step 2: Iterate until heap has only one node (root)

while (minHeap.size() > 1) {

Node\* left = minHeap.top(); minHeap.pop();

Node\* right = minHeap.top(); minHeap.pop();

// Create new internal node with freq = sum of two smallest

Node\* newNode = new Node('$', left->freq + right->freq);

newNode->left = left;

newNode->right = right;

minHeap.push(newNode);

}

// Step 3: Print codes from root

Node\* root = minHeap.top();

printCodes(root, "");

}

int main() {

int n;

cout << "Enter number of characters: ";

cin >> n;

vector<char> chars(n);

vector<int> freq(n);

cout << "Enter characters: ";

for (int i = 0; i < n; i++) cin >> chars[i];

cout << "Enter frequencies: ";

for (int i = 0; i < n; i++) cin >> freq[i];

cout << "\nHuffman Codes:\n";

HuffmanCodes(chars, freq);

return 0;

}

2.

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

1. combine d(2) + u(2) → 4

2. combine 4 + m(8) → 12

3. combine e(10) + s(10) → 20

4. combine c(11) + 12 → 23

5. combine o(15) + a(20) → 35

6. combine 20 + t(22) → 42

7. combine 23 + 35 → 58

8.combine 42 + 58 → 100 (root)

a : 111

c : 100

d : 10100

e : 000

m : 1011

o : 110

s : 001

t : 01

u : 10101

Weighted sum (using the given counts) = 294 bits

Total bits to encode the whole file with the above codes = 294 \* 100 = 29,400 bits

Step 1

(d:2 + u:2) = 4

a:20 c:11 e:10 o:15 m:8 s:10 t:22 (du):4

Step 2

(du:4 + m:8) = 12

a:20 c:11 e:10 o:15 s:10 t:22 (dum):12

Step 3

(e:10 + s:10) = 20

a:20 c:11 o:15 t:22 (dum):12 (es):20

Step 4

(c:11 + dum:12) = 23

a:20 o:15 t:22 (es):20 (cdum):23

Step 5

(o:15 + a:20) = 35

t:22 (es):20 (cdum):23 (oa):35

Step 6

(es:20 + t:22) = 42

(cdum):23 (oa):35 (est):42

Step 7

(cdum:23 + oa:35) = 58

(est):42 (cdumoa):58

Step 8

(est:42 + cdumoa:58) = 100

Final Huffman Tree

(100)

/ \

(42) (58)

/ \ / \

(es:20) (t:22) (23) (oa:35)

/ \ / \ / \

(e:10)(s:10) (c:11)(dum:12)(o:15)(a:20)

/ \

(du:4)(m:8)

/ \

(d:2)(u:2)

2 table

|  |  |  |
| --- | --- | --- |
| letter | frequency | Huffman code |
| A | 20 | 111 |
| C | 11 | 100 |
| D | 2 | 10100 |
| E | 10 | 000 |
| O | 15 | 110 |
| M | 8 | 1011 |
| S | 10 | 001 |
| T | 22 | 01 |
| U | 2 | 10101 |

3. Step 1 – Frequencies

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

total file size =

(20+11+2+10+15+8+10+22+2)×100=100×100=10,000 characters(

a : 111 → length = 3

c : 100 → length = 3

d : 10100 → length = 5

e : 000 → length = 3

o : 110 → length = 3

m : 1011 → length = 4

s : 001 → length = 3

t : 01 → length = 2

u : 10101 → length = 5

**– Weighted sum of bits**

Multiply each frequency by its code length:

* a: 20 × 3 = 60
* c: 11 × 3 = 33
* d: 2 × 5 = 10
* e: 10 × 3 = 30
* o: 15 × 3 = 45
* m: 8 × 4 = 32
* s: 10 × 3 = 30
* t: 22 × 2 = 44
* u: 2 × 5 = 10

**Total = 294 bits**

Since the file has **10,000 characters**

294×100=29,400 bits

The file will require:

29,400 bits=29,4008=3,675 bytes≈3.6 kbs