ASSIGNMENT

(Pre-Mini Project)



HUFFMAN ENCODING

Submitted by-

Name: Tanveer Ahmed

Scholar id:1815010

Section: A

1. Implement the Huffman code using the following steps:

a. Implement the min-priority queue, Q for storing and manipulating the input using a binary min-heap.

->

Using custom compare class to compare two Huffman tree node.

class Compare

{

public:

bool operator()(HuffmanTreeNode\* a, HuffmanTreeNode\* b)

{

return a->freq > b->freq;

}

};

priority\_queue<HuffmanTreeNode\*, vector<HuffmanTreeNode\*>, compare> Q

\*above we declare the min priority queue(Q)

b. Implement the function for merging two nodes for the full binary tree, T that is used to output the codeword. (The tree is to be built in a bottom-up fashion as the main algorithm proceeds. Hence pointers have to be added wherever and whenever needed.)

->

HuffmanTreeNode\* mergeNode(priority\_queue<HuffmanTreeNode\*, vector<HuffmanTreeNode\*>, Compare> Q)

{

while (Q.size() != 1) {

HuffmanTreeNode\* left = Q.top();

Q.pop();

HuffmanTreeNode\* right = Q.top();

Q.pop();

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

newnode->left = left;

newnode->right = right;

Q.push(newnode);

}

return Q.top();

}

c.Now combine your codes in (a) and (b) to implement the Huffman code

void HuffmanCodes(char data[], int freq[], int size)

{

priority\_queue<HuffmanTreeNode\*, vector<HuffmanTreeNode\*>, Compare> Q; for (int i = 0; i < size; i++) {

HuffmanTreeNode\* newNode = new HuffmanTreeNode(data[i], freq[i]); Q.push(newNode);

}

* Generation of Huffman Encoding Tree and get the root node HuffmanTreeNode\* root = mergeNode(Q);

int arr[MAX\_SIZE], top = 0; printCodes(root, arr, top);

}

d.Finally,write a recursive function to generate the codeword for each character in the alphabet C which is the input to your algorithm.

->

void printCodes(HuffmanTreeNode\* root, int arr[], int top)

{

if (root->left) {

arr[top] = 0;

printCodes(root->left, arr, top + 1);

}

if (root->right) {

arr[top] = 1;

printCodes(root->right, arr, top + 1);

}

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

cout << root->data << " ";

for (int i = 0; i < top; i++) {

cout << arr[i];

}

cout << endl;

}

}

e.Also give the structure of the nodes of Q and T

class HuffmanTreeNode {

public:

char data;

int freq;

HuffmanTreeNode\* left;

HuffmanTreeNode\* right;

HuffmanTreeNode(char data, int freq){

this->data = data;

this->freq = freq;

left = right = NULL;

}

};

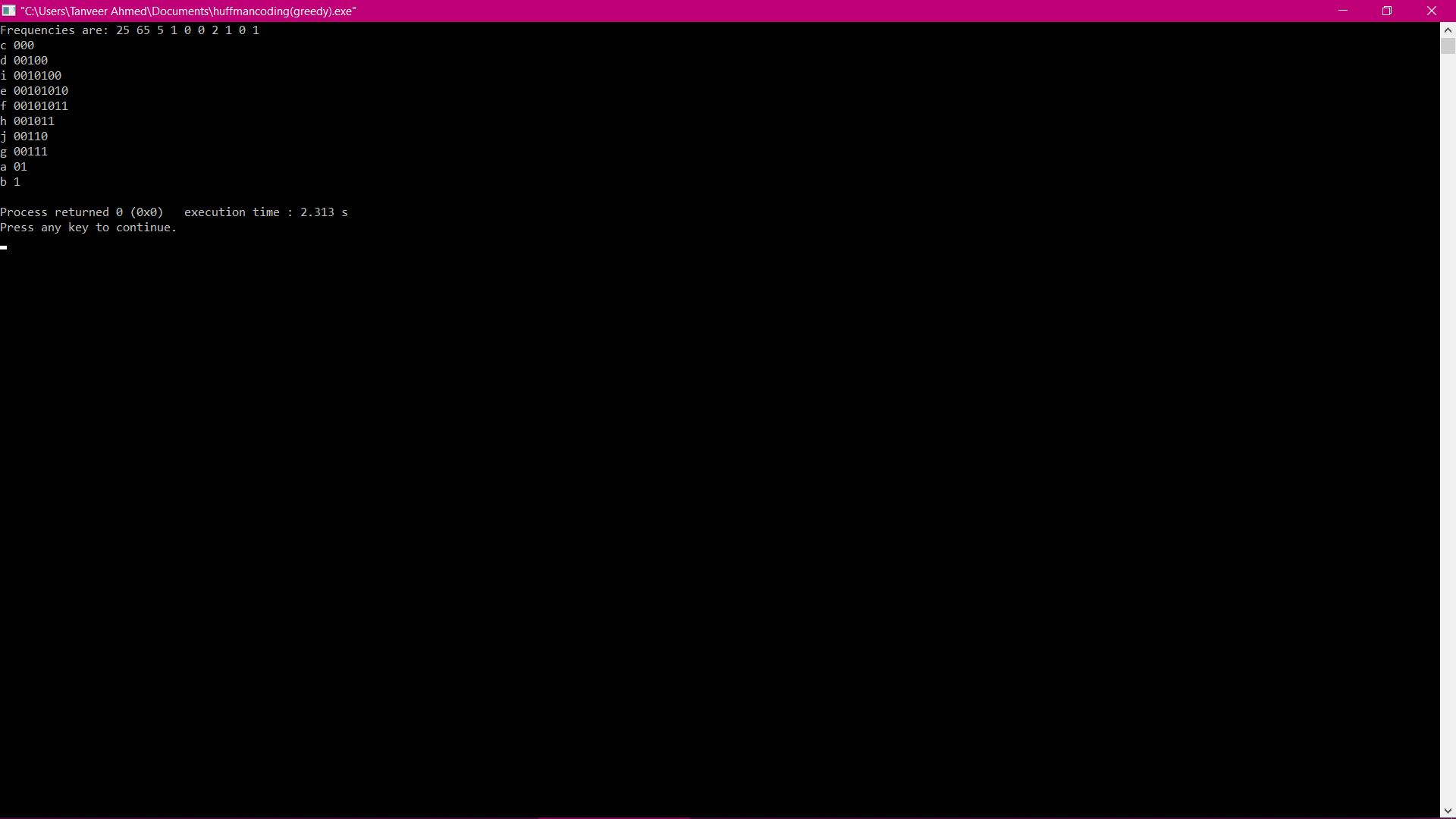
f. For the demonstration of your algorithm

i. Take an input alpahabet of 10 characters, C = { a, b, c, d, e, f, g, h, i, j }

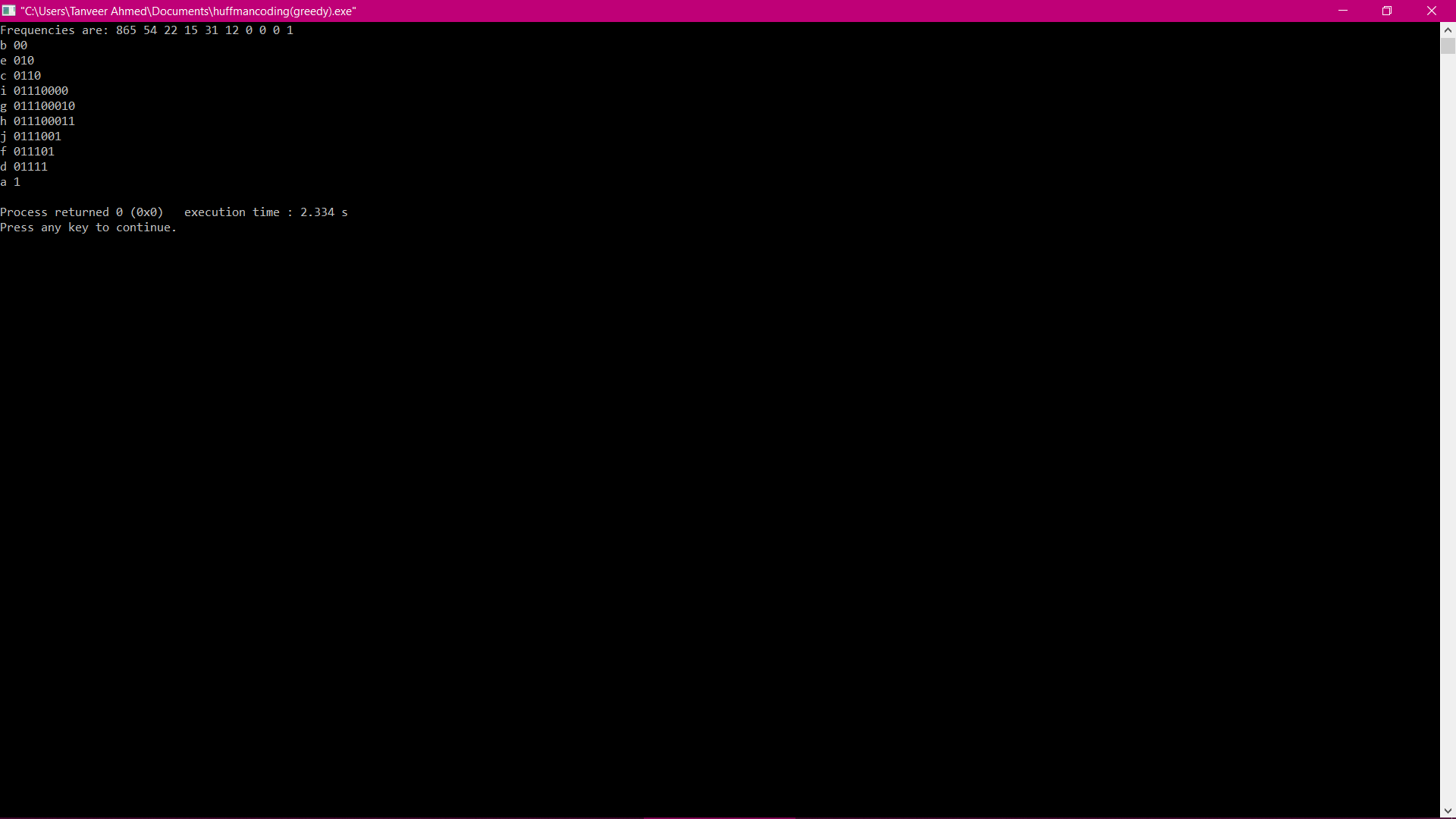
1. Create arbitrary texts of length 100 (Text-1), 1000 (Text-2), and 10000 (Text-3) characters each by randomly selecting each character from the alphabet C.
2. Report the frequencies in tabular form for each text (Table format given in the next slide).
3. Create the optimal tree for each of the three texts using your code. Report the codewords generated for C, in case of each text in the above table and also give the snapshots from running your code to generate the output.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Character | Text-1 | | Text-2 | | Text-3 | |
|  |  |  |  |  |  |  |
|  | Frequency | Codeword | Frequency | Codeword | Frequency | Codeword |
|  |  |  |  |  |  |  |
| a | 25 | 01 | 865 | 1 | 5143 | 1 |
|  |  |  |  |  |  |  |
| b | 65 | 1 | 54 | 00 | 898 | 001 |
|  |  |  |  |  |  |  |
| c | 5 | 000 | 22 | 0110 | 3757 | 01 |
|  |  |  |  |  |  |  |
| d | 1 | 00100 | 15 | 01111 | 66 | 00011 |
|  |  |  |  |  |  |  |
| e | 0 | 0010100 | 31 | 010 | 73 | 0000 |
|  |  |  |  |  |  |  |
| f | 0 | 00101011 | 12 | 011101 | 4 | 000100001 |
|  |  |  |  |  |  |  |
| g | 2 | 00111 | 0 | 011100010 | 37 | 000101 |
|  |  |  |  |  |  |  |
| h | 1 | 001011 | 0 | 011100011 | 2 | 000100001 |
|  |  |  |  |  |  |  |
| i | 0 | 0010100 | 0 | 01110000 | 0 | 000100000 |
|  |  |  |  |  |  |  |
| j | 1 | 001100 | 1 | 011101 | 20 | 0001001 |
|  |  |  |  |  |  |  |

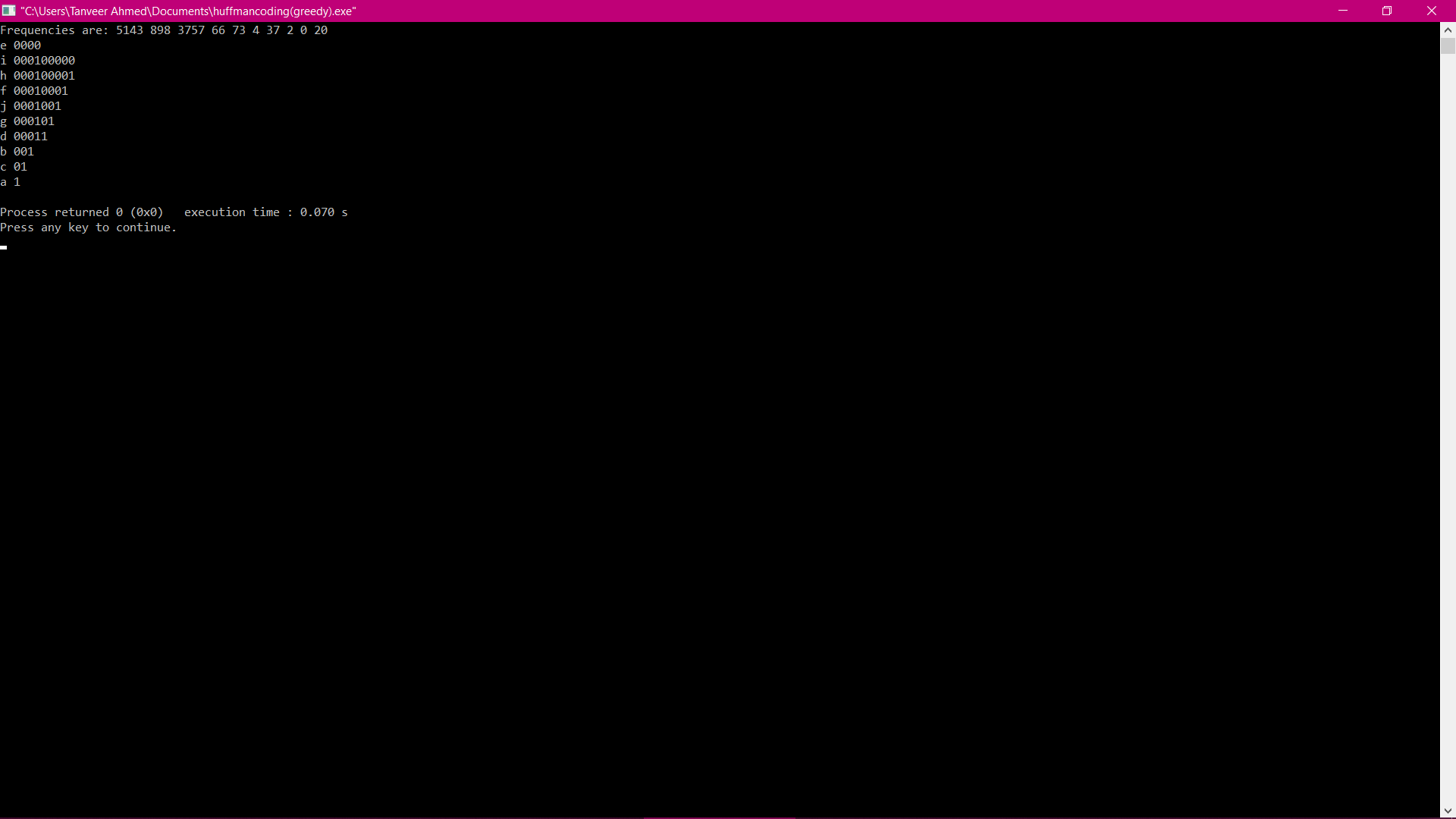
Text 1 output:



Test 2 output:



Test 3 output:



2. You must give the algorithms for questions 1.(a), 1(b), 1(c), and 1(d) separately as pseudocodes. Give the structure of nodes only for question 1(e).

->

Done above in pages 2-4

3. Submit your complete code along with the required snapshots and output table.

#include <iostream>

#include <queue>

#include <stdlib.h>

#include<time.h>

using namespace std;

// Maximum Height of Huffman Tree.

#define MAX\_SIZE 100

class HuffmanTreeNode {

public:

char data;

int freq;

HuffmanTreeNode\* left;

HuffmanTreeNode\* right;

// Initializing the current node

HuffmanTreeNode(char character, int frequency)

{

data = character;

freq = frequency;

left = right = NULL;

}

};

// Custom comparator class

class Compare {

public:

bool operator()(HuffmanTreeNode\* a, HuffmanTreeNode\* b)

{

return a->freq > b->freq;

}

};

HuffmanTreeNode\* mergeNode(priority\_queue<HuffmanTreeNode\*,

vector<HuffmanTreeNode\*>, Compare> Q)

{

while (Q.size() != 1) {

HuffmanTreeNode\* left = Q.top();

Q.pop();

HuffmanTreeNode\* right = Q.top();

Q.pop();

HuffmanTreeNode\* node = new HuffmanTreeNode('$', left->freq + right->

freq);

node->left = left;

node->right = right;

Q.push(node);

}

return Q.top();

}

void printCodes(HuffmanTreeNode\* root, int arr[], int top)

{

if (root->left) {

arr[top] = 0;

printCodes(root->left, arr, top + 1);

}

if (root->right) {

arr[top] = 1;

printCodes(root->right, arr, top + 1);

}

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

cout << root->data << " ";

for (int i = 0; i < top; i++) {

cout << arr[i];

}

cout << endl;

}

}

void HuffmanCodes(char data[], int freq[], int size)

{

priority\_queue<HuffmanTreeNode\*, vector<HuffmanTreeNode\*>, Compare> Q;

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

HuffmanTreeNode\* newNode = new HuffmanTreeNode(data[i], freq[i]);

Q.push(newNode);

}

HuffmanTreeNode\* root = mergeNode(Q);

int arr[MAX\_SIZE], top = 0;

printCodes(root, arr, top);

}

// Main function

int main()

{

char data[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j' };

int size = sizeof(data) / sizeof(char);

int freq[10];

int sum = 100;

int subSum = sum;

srand(time(0));

for(int i=0; i<size-1; i++){

freq[i] = rand()%subSum;

subSum -= freq[i];

}

freq[size-1] = subSum;

cout << "Frequencies are: ";

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

cout << freq[i] <<" ";

}

cout <<endl;

HuffmanCodes(data, freq, size);

return 0;

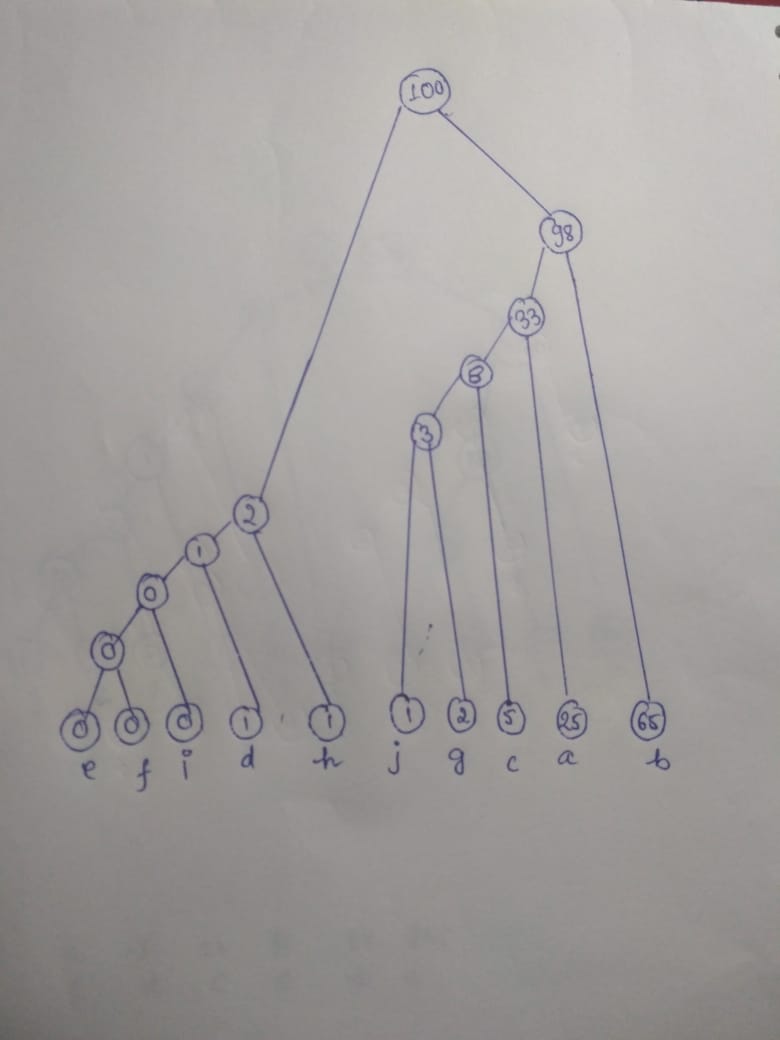
}

**Output table**

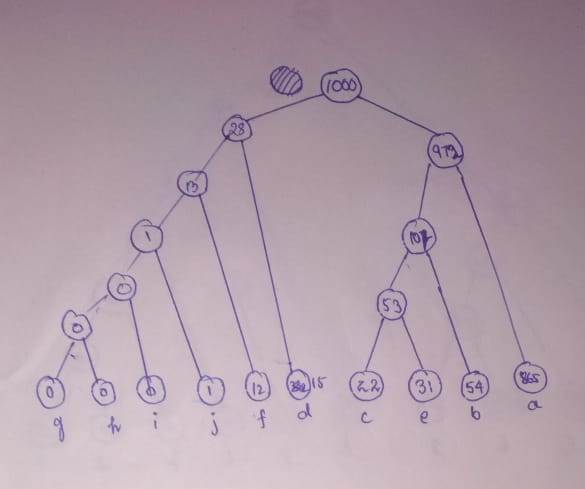
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Character | Text-1 | | Text-2 | | Text-3 | |
|  |  |  |  |  |  |  |
|  | Frequency | Codeword | Frequency | Codeword | Frequency | Codeword |
|  |  |  |  |  |  |  |
| a | 25 | 01 | 865 | 1 | 5143 | 1 |
|  |  |  |  |  |  |  |
| b | 65 | 1 | 54 | 00 | 898 | 001 |
|  |  |  |  |  |  |  |
| c | 5 | 000 | 22 | 0110 | 3757 | 01 |
|  |  |  |  |  |  |  |
| d | 1 | 00100 | 15 | 01111 | 66 | 00011 |
|  |  |  |  |  |  |  |
| e | 0 | 0010100 | 31 | 010 | 73 | 0000 |
|  |  |  |  |  |  |  |
| f | 0 | 00101011 | 12 | 011101 | 4 | 000100001 |
|  |  |  |  |  |  |  |
| g | 2 | 00111 | 0 | 011100010 | 37 | 000101 |
|  |  |  |  |  |  |  |
| h | 1 | 001011 | 0 | 011100011 | 2 | 000100001 |
|  |  |  |  |  |  |  |
| i | 0 | 0010100 | 0 | 01110000 | 0 | 000100000 |
|  |  |  |  |  |  |  |
| j | 1 | 001100 | 1 | 011101 | 20 | 0001001 |
|  |  |  |  |  |  |  |

4. Draw the output tree T-1 (Text-1), T-2 (Text-2), and T-3 (Text-3).

Test case 1:



Test case 2:



Test case 3:

