

|  |
| --- |
| Pre mini-Project 1  COMPUTER ALGORITHMS  Huffman Encoding |
|  |
| May 23  SWAGAT SHUBHAM BHUYAN  18-1-5-059  Sec: A SEM: IV |



SUBMITTED TO:

Dr. Shyamosree Pal

Q.:

Assignment (Pre mini project)

1. Implement the Huffman code using the following steps:

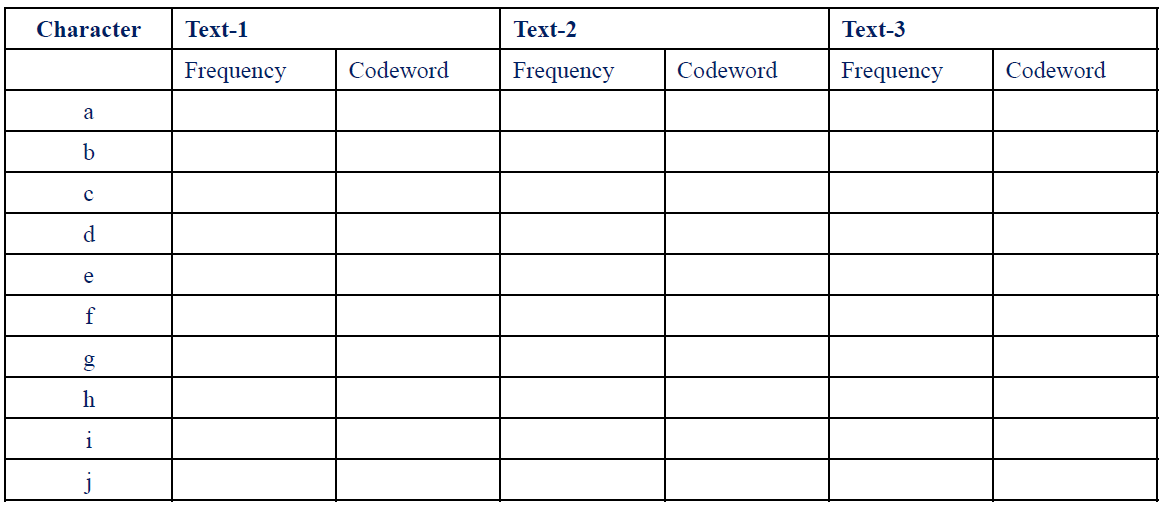
1. Implement the min priority queue, Q for storing and manipulating the input using a binary min heap. (See sections 6.1, 6.2, 6.3, and 6.5 of Textbook, Cormen)
2. Implement the function for merging two nodes for the full binary tree, T that is used to output the codeword. (The tree is to be built in a bottom up fashion as the main algorithm proceeds. Hence pointers have to be added wherever and whenever needed.)
3. Now combine your codes in (a) and (b) to implement the Huffman code.
4. Finally, write a recursive function to generate the codeword for each character in the alphabet C which is the input to your algorithm.
5. Also give the structure of the nodes of Q and T
6. For the demonstration of your algorithm
7. Take an input alphabet of 10 characters, C = {a, b, c, d, e, f, g, h, i, j }
8. 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.
9. Report the frequencies in tabular form for each text (Table format given in the next slide).
10. Create the optimal tree for each of the three texts using your code. Report the codewords generated for C , in case of eachtext in the above table and also give the snapshots from running your code to generate the 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).

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

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



Instructions for submission:

1. Everything must be compiled in one file (preferably pdf format). If you put in separate files, I will check only the first file that you submit.

2. Mail your assignment to shyamosree.pal@gmail.com and in the subject part of the mail write: Assignment (Pre mini project) B. Tech, 4th Sem, CSE

1. Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Roll Number:

3. Please add your Mobile number and Email id in this mail.

4. Date of Submission: Within 23.05.2020 (11.59pm).

5. Please submit as soon as possible, I might consider giving bonus marks to the first 25 submissions. Also as soon as you mail your assignment you will get to know your midsem marks.

1.a) We use a custom comparator (CmpCharcNodes) class to set condition of priority for the required Priority Queue:

PSEUDO-CODE:

MIN\_HEAPIFY(A , i)

1. l = 2\*i +1

2. r = 2\*i +2

3. largest = i

4. if l< heap.size() && A[l]< A[largest]

5. largest = l

6. If r< heap.size() && A[r]< A[largest]

7. largest = r

8. if i!= largest

9. swap(A[i], A[largest])

10. MIN\_HEAPIFY(A, largest)

CODE:

class CmpCharcNodes

{

    public:

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

        { return a->freq > b->freq; }

};

priority\_queue<CharcTreeNode\*, vector<CharcTreeNode\*>,

CmpCharcNodes> PriorityHuffmanQ;

1.b)

PSEUDO-CODE:

HUFFMAN(C)

1. n= |C|

2. Q = C

3. for i=0 to n

4. allocate a new node z

5. z.left = x = EXTRACT\_MIN(Q)

6. z.right = y= EXTRACT\_MIN(Q)

7. z.freq = x.freq + y.freq

8. INSERT(Q, z)

9. return EXTRACT\_MIN(Q)

CODE:

while (PriorityHuffmanQ.size() != 1)

    {

        //EXTRACT-MIN(Q)

        CharcTreeNode\* left = PriorityHuffmanQ.top();

        PriorityHuffmanQ.pop();

        //EXTRACT-MIN(Q)

        CharcTreeNode\* right = PriorityHuffmanQ.top();

        PriorityHuffmanQ.pop();

        //Creating new Charc node 'z' and inserting to PriorityHuffmanQueue

        CharcTreeNode\* z = new CharcTreeNode('.', left->freq + right->freq);

        z->left = left;

        z->right = right;

        PriorityHuffmanQ.push(z);

    }

    //Returning Last node left in PriorityQueue, root of HuffmanTree

    return PriorityHuffmanQ.top();

1.c) From (a) and (b), we combine the two codes where (b) calls (a) to implement accordingly-

PSEUDO-CODE:

MIN\_HEAPIFY(A , i)

1. l = 2\*i +1

2. r = 2\*i +2

3. largest = i

4. if l< heap.size() && A[l]< A[largest]

5. largest = l

6. If r< heap.size() && A[r]< A[largest]

7. largest = r

8. if i!= largest

9. swap(A[i], A[largest])

10. MIN\_HEAPIFY(A, largest)

HUFFMAN(C)

11. n= |C|

12. Q = C

13. for i = n/2 down to 0

14. MIN\_HEAPIFY(Q,i)

15. for i=0 to n

16. allocate a new node z

17. z.left = x = EXTRACT\_MIN(Q)

18. z.right = y= EXTRACT\_MIN(Q)

19. z.freq = x.freq + y.freq

20. INSERT(Q, z)

21. return EXTRACT\_MIN(Q)

1.d)

PSEUDO-CODE:

PRINT\_CODE\_WORD(C, top)

1. if A.left {

2. arr[top] = 0

3. PRINT\_CODE\_WORD(A.left, top+1)

4. if A.right{

5. arr[top] = 1

6. PRINT\_CODE\_WORD(A.right, top+1)

7. if !A.left && !A.right {

8. for i = 0 to top

9. print arr[i]

CODE:

    // Assigning 0 to the left node

    if (root->left)

    {

        arr[top] = 0;

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

    }

    // Assigning 1 to the right node

    if (root->right)

    {

        arr[top] = 1;

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

    }

    //Leaf Node Printing

    if (!root->left && !root->right)

    {

        cout << root->ch << " ";

        for (int i = 0; i < top; i++) cout << arr[i];

        cout << "\n";

    }

1.e)

Structure of Tree T:

class CharcTreeNode

{

    public:

        char ch;

        unsigned freq;

        CharcTreeNode \*left, \*right;

        CharcTreeNode(char ch, unsigned freq)

        {

            left = right = NULL;

            this->ch = ch;

            this->freq = freq;

        }

};

Structure of PriorityQueue

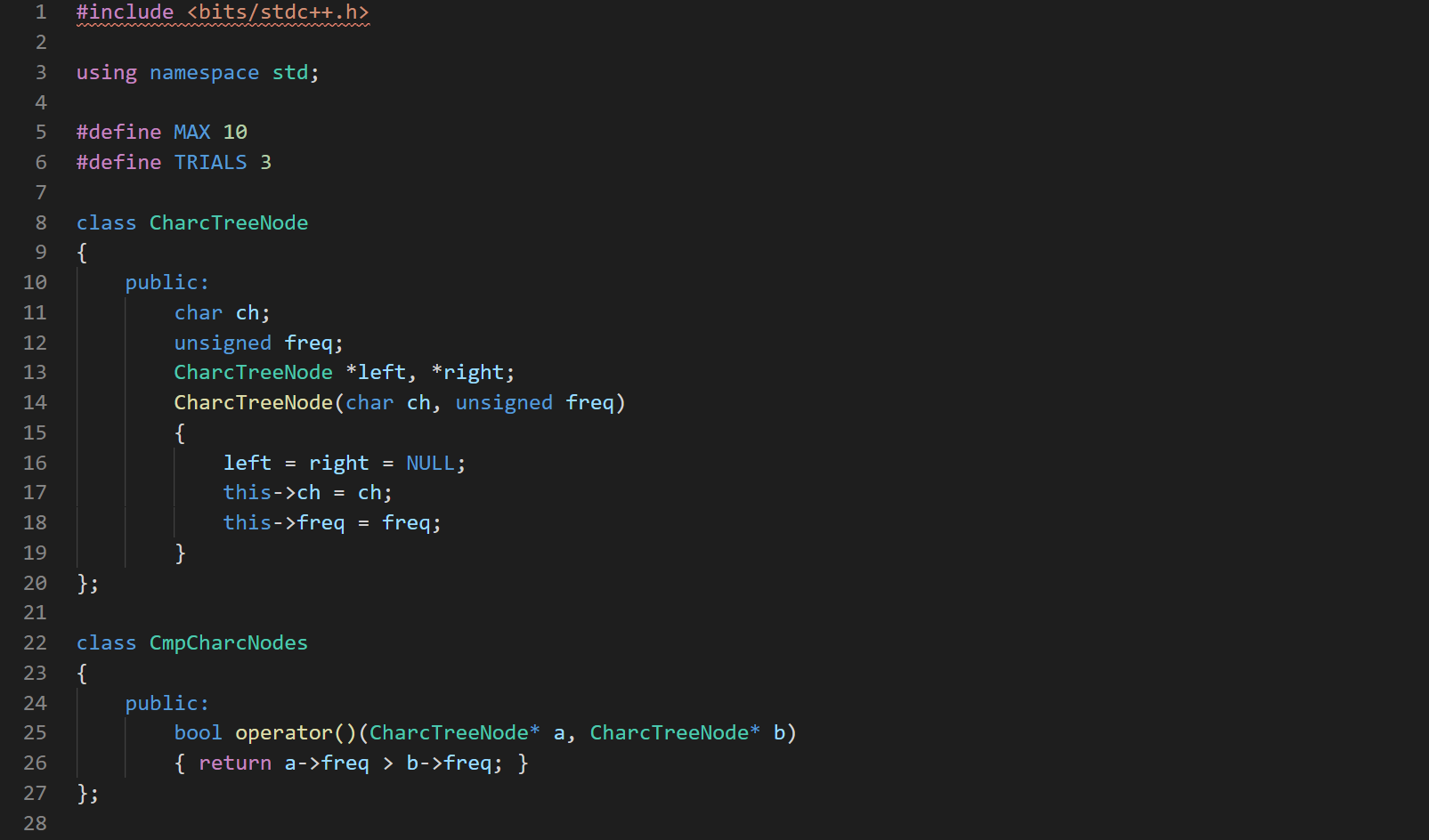
Struct Q {

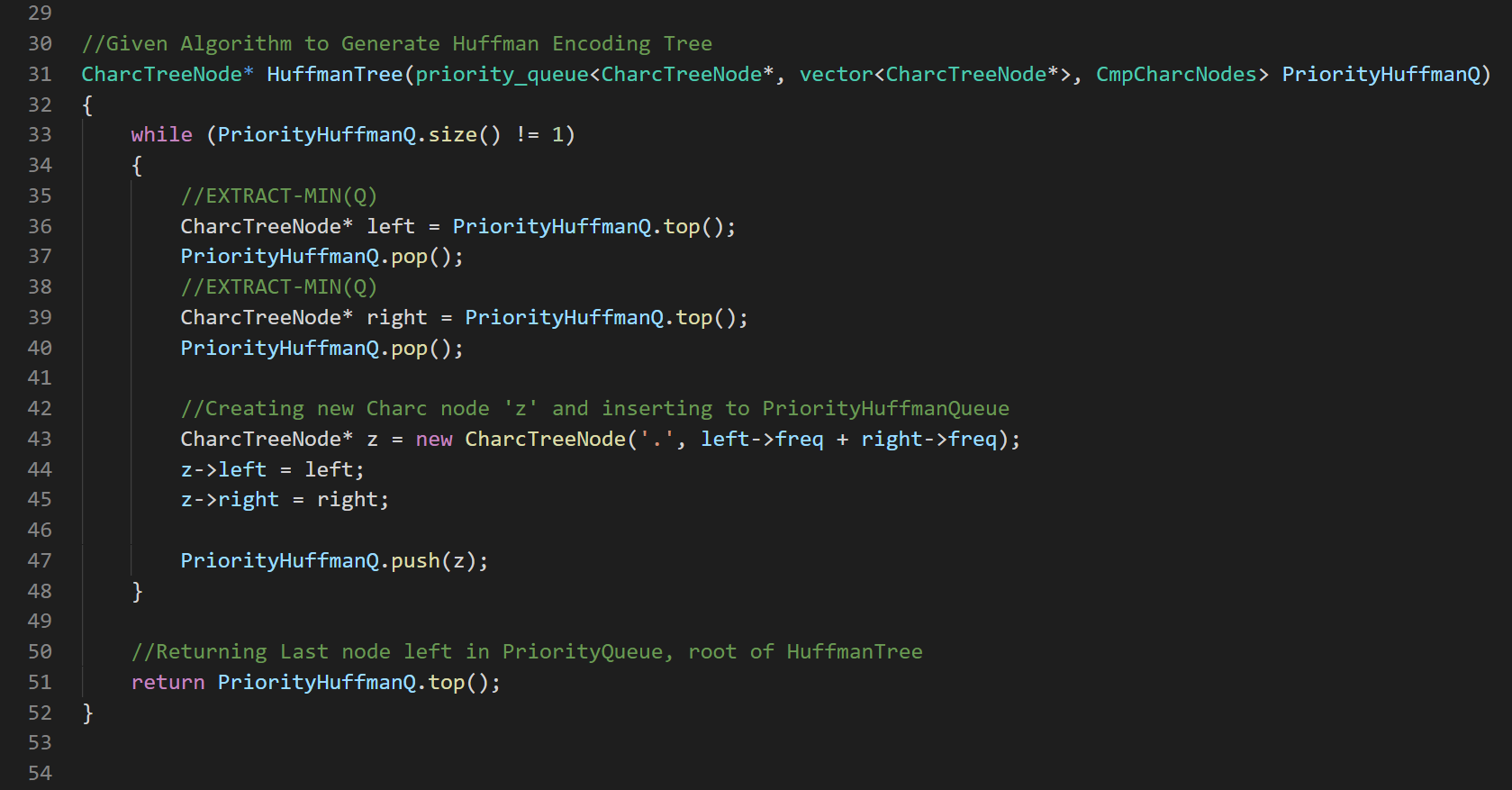
int freq;

char ch;

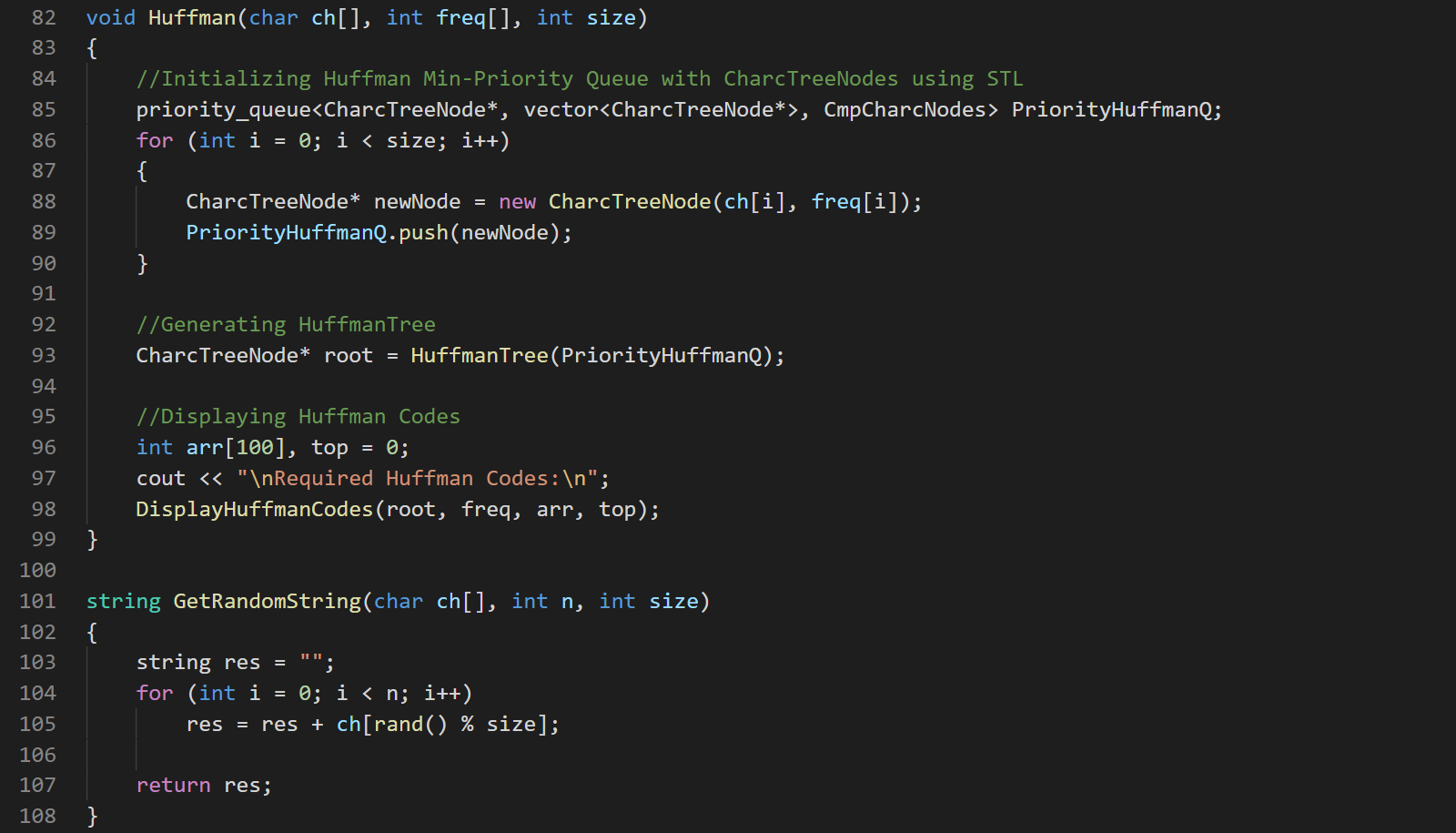
};

Q3. CODE and OUTPUT TABLE

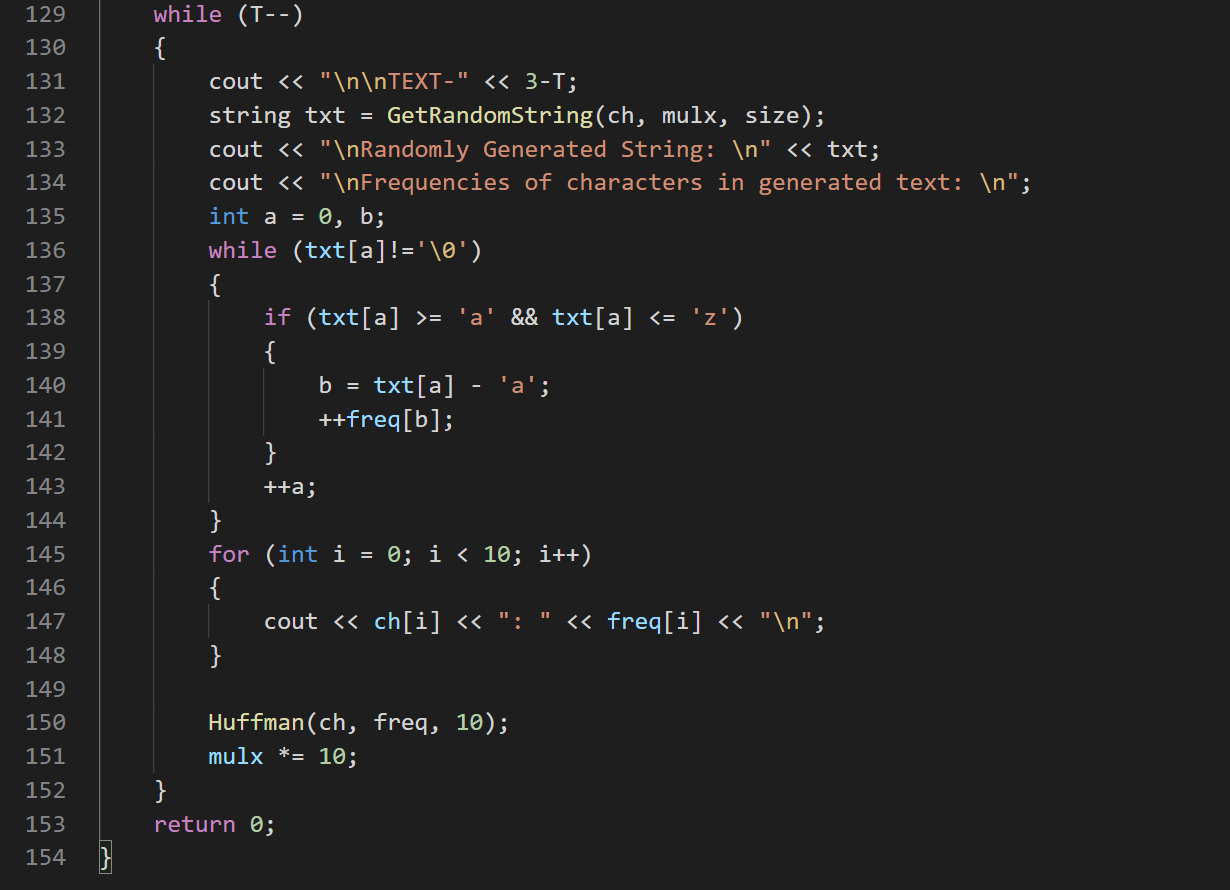












CODE TERMINAL OUTPUT

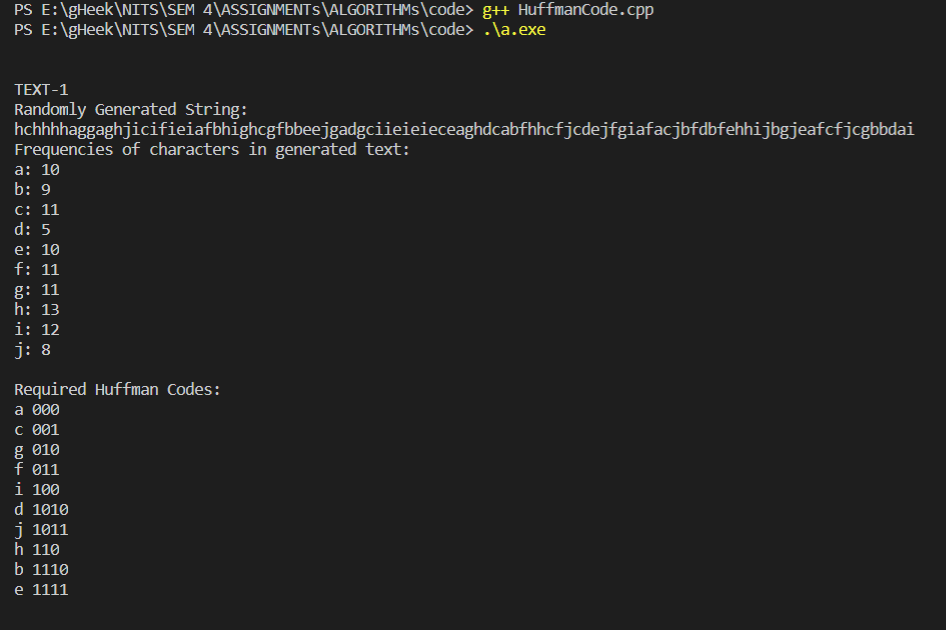
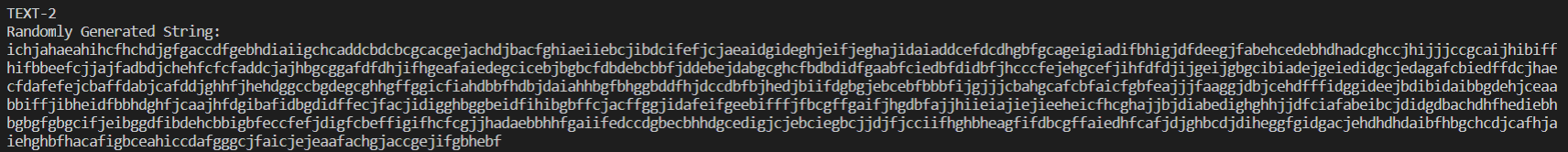


FIG1: TEXT-1, Frequencies and Huffman Codes



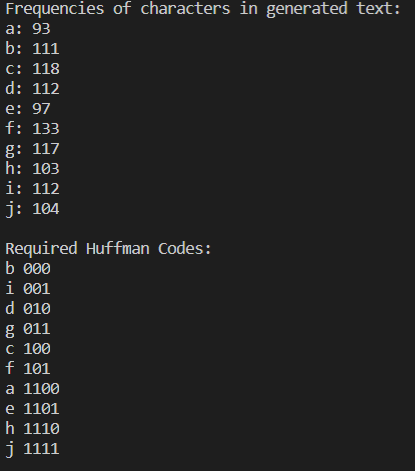
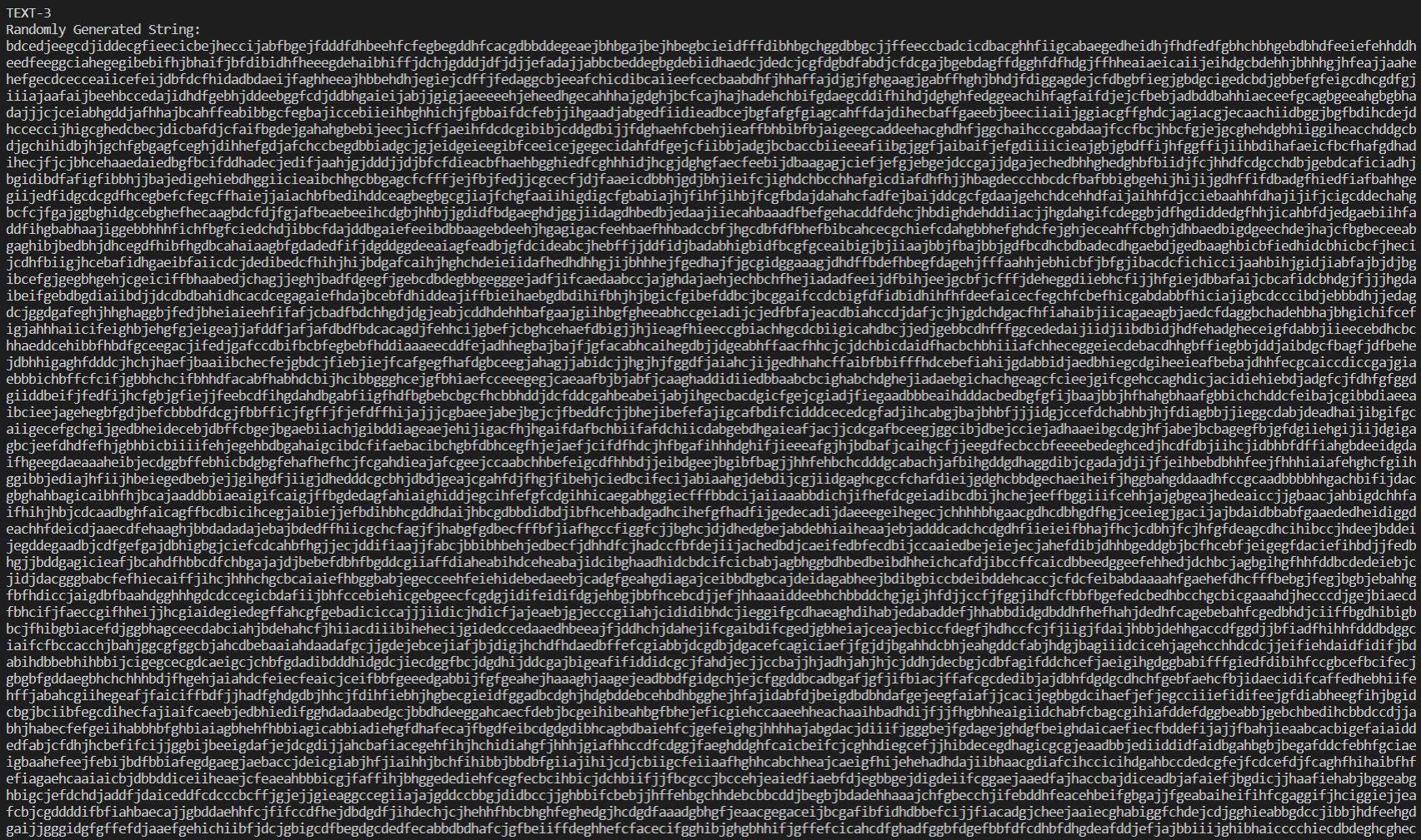


FIG2: TEXT-2, Frequencies and Huffman Codes



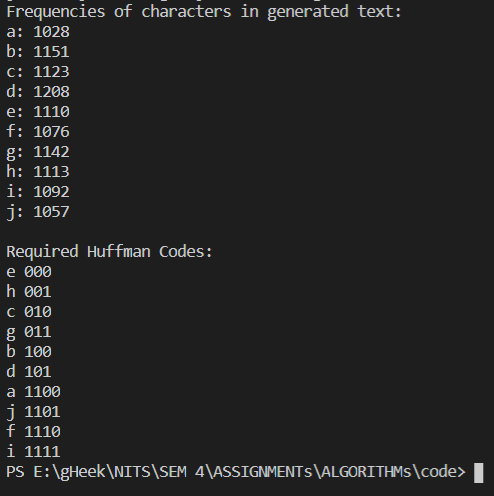


FIG3: TEXT-3, Frequencies and Hoffman Codes

FREQUENCY-CODEWORD TABLE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Characters | TEXT - 1 | | TEXT – 2 | | TEXT - 3 | |
| Frequency | CodeWord | Frequency | CodeWord | Frequency | CodeWord |
| a | 10 | 000 | 93 | 1100 | 1028 | 1100 |
| b | 9 | 1110 | 111 | 000 | 1151 | 100 |
| c | 11 | 001 | 118 | 100 | 1123 | 010 |
| d | 5 | 1010 | 112 | 010 | 1208 | 101 |
| e | 10 | 1111 | 97 | 1101 | 1110 | 000 |
| f | 11 | 011 | 133 | 101 | 1076 | 1110 |
| g | 11 | 010 | 117 | 011 | 1142 | 011 |
| h | 13 | 110 | 103 | 1110 | 1113 | 001 |
| i | 12 | 100 | 112 | 001 | 1092 | 1111 |
| j | 8 | 1011 | 104 | 1111 | 1057 | 1101 |

Huffman Coding Binary Trees for

TEXT-1, TEXT-2, TEXT-3

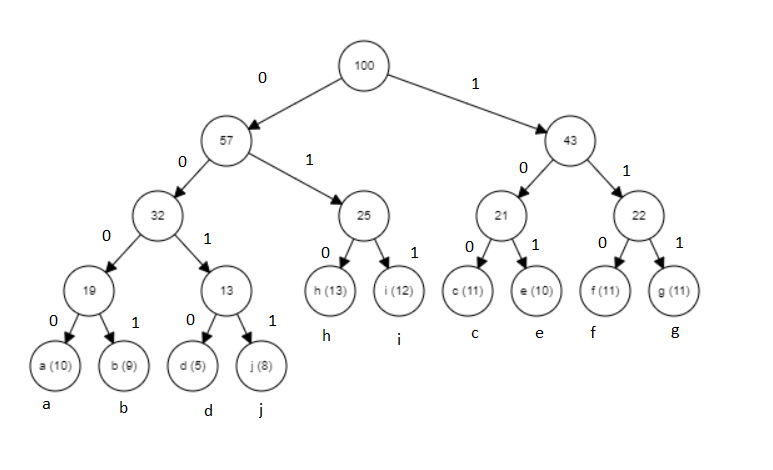


FIG4: Tree T1 for TEXT-1

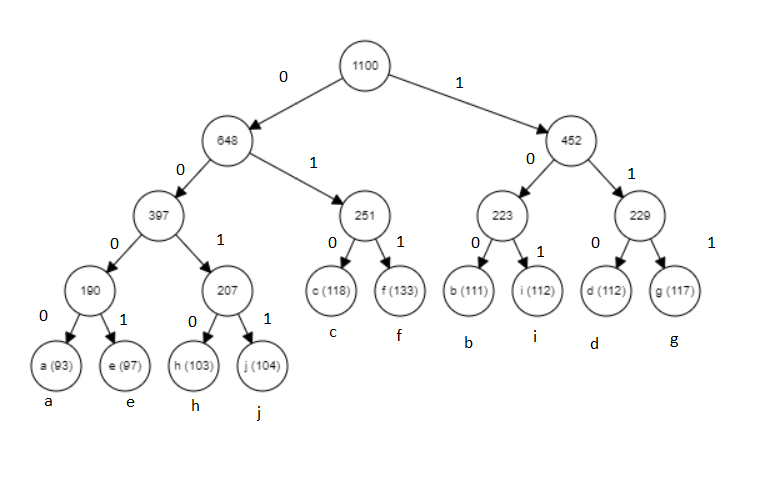


FIG4: Tree T2 for TEXT-2

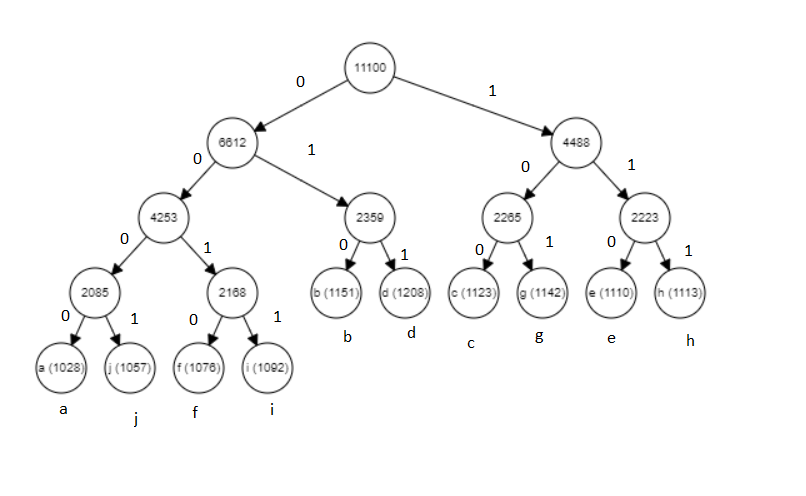


FIG4: Tree T3 for TEXT-3