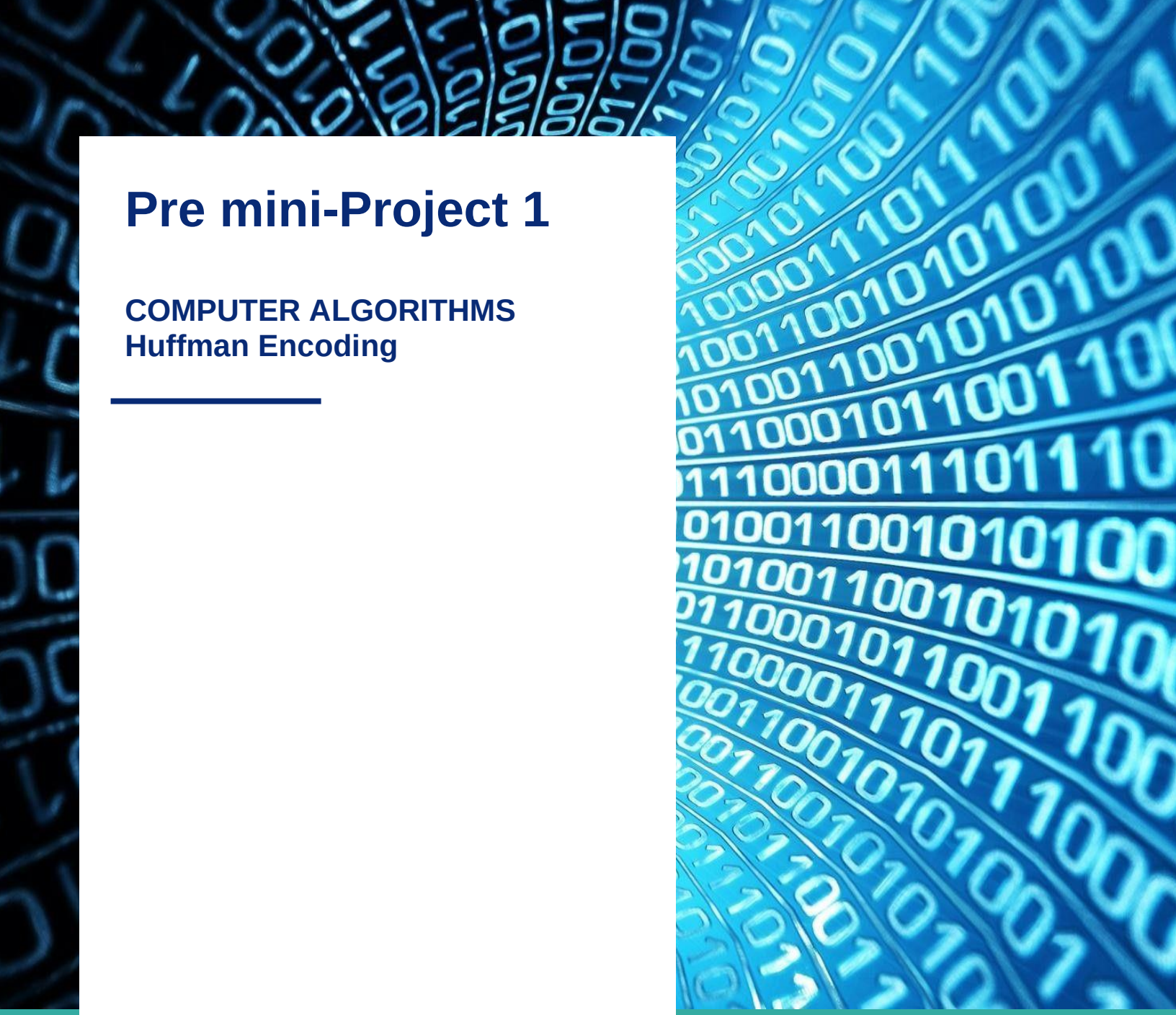


Pre mini-Project 1

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:

a) **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)**
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.)**
c) **Now combine your codes in (a) and (b) to implement the Huffman code.**
d) **Finally, write a recursive function to generate the codeword for each character in the alphabet C which is the input to your algorithm.**
e) **Also give the structure of the nodes of Q and T**
f) **For the demonstration of your algorithm**
    i. **Take an input alphabet of 10 characters, C = {a, b, c, d, e, f, g, h, i, j }**
    ii. **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.**
    iii. **Report the frequencies in tabular form for each text (Table format given in the next slide).**
    iv. **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).

Character	Text-1		Text-2		Text-3	
	Frequency	Codeword	Frequency	Codeword	Frequency	Codeword
a						
b						
c						
d						
e						
f						
g						
h						
i						
j						

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 PriorityH
uffmanQueue
```

```
        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 Huffman
Tree
    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

```cpp
#include <bits/stdc++.h>

using namespace std;

#define MAX 10
#define TRIALS 3

class CharcTreeNode
{
    public:
        char ch;
        unsigned freq;
        CharcTreeNode *left, *right;
        CharcTreeNode(char ch, unsigned freq)
        {
            left = right = NULL;
            this->ch = ch;
            this->freq = freq;
        }
};

class CmpCharcNodes
{
    public:
        bool operator()(CharcTreeNode* a, CharcTreeNode* b)
        { return a->freq > b->freq; }
};

```

```cpp

//Given Algorithm to Generate Huffman Encoding Tree
CharcTreeNode* HuffmanTree(priority_queue<CharcTreeNode*, vector<CharcTreeNode*>, CmpCharcNodes> PriorityHuffmanQ)
{
    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();
}


```

```
54
55  void DisplayHuffmanCodes(CharcTreeNode* root, int freq[], int arr[], int top)
56  {
57      // Assigning 0 to the left node
58      if (root->left)
59      {
60          arr[top] = 0;
61          DisplayHuffmanCodes(root->left, freq, arr, top + 1);
62      }
63
64      // Assigning 1 to the right node
65      if (root->right)
66      {
67          arr[top] = 1;
68          DisplayHuffmanCodes(root->right, freq, arr, top + 1);
69      }
70
71      //Leaf Node Printing
72      if (!root->left && !root->right)
73      {
74          cout << root->ch << " ";
75          for (int i = 0; i < top; i++)
76              cout << arr[i];
77          cout << "\n";
78      }
79  }
80
81
```

```
82  void Huffman(char ch[], int freq[], int size)
83  {
84      //Initializing Huffman Min-Priority Queue with CharcTreeNodes using STL
85      priority_queue<CharcTreeNode*, vector<CharcTreeNode*>, CmpCharcNodes> PriorityHuffmanQ;
86      for (int i = 0; i < size; i++)
87      {
88          CharcTreeNode* newNode = new CharcTreeNode(ch[i], freq[i]);
89          PriorityHuffmanQ.push(newNode);
90      }
91
92      //Generating HuffmanTree
93      CharcTreeNode* root = HuffmanTree(PriorityHuffmanQ);
94
95      //Displaying Huffman Codes
96      int arr[100], top = 0;
97      cout << "\nRequired Huffman Codes:\n";
98      DisplayHuffmanCodes(root, freq, arr, top);
99  }
100
101 string GetRandomString(char ch[], int n, int size)
102 {
103     string res = "";
104     for (int i = 0; i < n; i++)
105         res = res + ch[rand() % size];
106
107     return res;
108 }
```

```
110 void FrequencyFinder(int freq[], string txt)
111 {
112     int n = txt.size();
113     for (int i = 0; i < n; i++) freq[txt[i] - 'a']++;
114     for (int i = 0; i < n; i++) {
115         if (freq[txt[i] - 'a'] != 0) {
116             cout << txt[i] << ": " << freq[txt[i] - 'a'] << "\n"; freq[txt[i] - 'a'] = 0; }
117     }
118 }
119
120 int main()
121 {
122     //To randomize seed
123     srand(time(NULL));
124     char ch[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
125     int size = sizeof(ch) / sizeof(ch[0]), mulx = 100;
126     int freq[10] = {0};
127     int T = TRIALS;
```

```
129     while (T--)
130     {
131         cout << "\n\nTEXT-" << 3-T;
132         string txt = GetRandomString(ch, mulx, size);
133         cout << "\nRandomly Generated String: \n" << txt;
134         cout << "\nFrequencies of characters in generated text: \n";
135         int a = 0, b;
136         while (txt[a]!='\0')
137         {
138             if (txt[a] >= 'a' && txt[a] <= 'z')
139             {
140                 b = txt[a] - 'a';
141                 ++freq[b];
142             }
143             ++a;
144         }
145         for (int i = 0; i < 10; i++)
146         {
147             cout << ch[i] << ": " << freq[i] << "\n";
148         }
149
150         Huffman(ch, freq, 10);
151         mulx *= 10;
152     }
153     return 0;
154 }
```

CODE TERMINAL OUTPUT

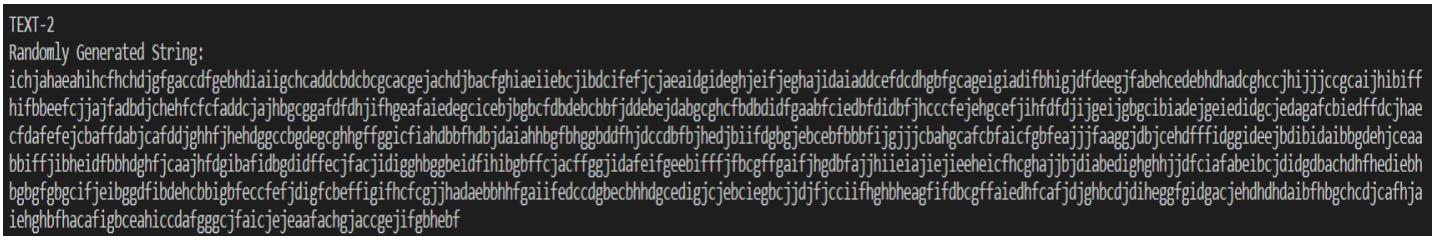
```
PS E:\gHeek\NITS\SEM 4\ASSIGNMENTs\ALGORITHMs\code> g++ HuffmanCode.cpp
PS E:\gHeek\NITS\SEM 4\ASSIGNMENTs\ALGORITHMs\code> .\a.exe


TEXT-1
Randomly Generated String:
hchhhhaggaghjicifieiafbhighcgfbbeejgadgciieieieceaghdcabfhhcfjcdejfgiafacjbfdbfehhijbgjeafcfjcgbbdai
Frequencies of characters in generated text:
a: 10
b: 9
c: 11
d: 5
e: 10
f: 11
g: 11
h: 13
i: 12
j: 8

Required Huffman Codes:
a 000
c 001
g 010
f 011
i 100
d 1010
j 1011
h 110
b 1110
e 1111
```

FIG1: TEXT-1, Frequencies and Huffman Codes

TEXT-2
Randomly Generated String:




```
Frequencies of characters in generated text:
a: 93
b: 111
c: 118
d: 112
e: 97
f: 133
g: 117
h: 103
i: 112
j: 104

Required Huffman Codes:
b 000
i 001
d 010
g 011
c 100
f 101
a 1100
e 1101
h 1110
j 1111
```

FIG2: TEXT-2, Frequencies and Huffman Codes

TEXT-3
Randomly Generated String:

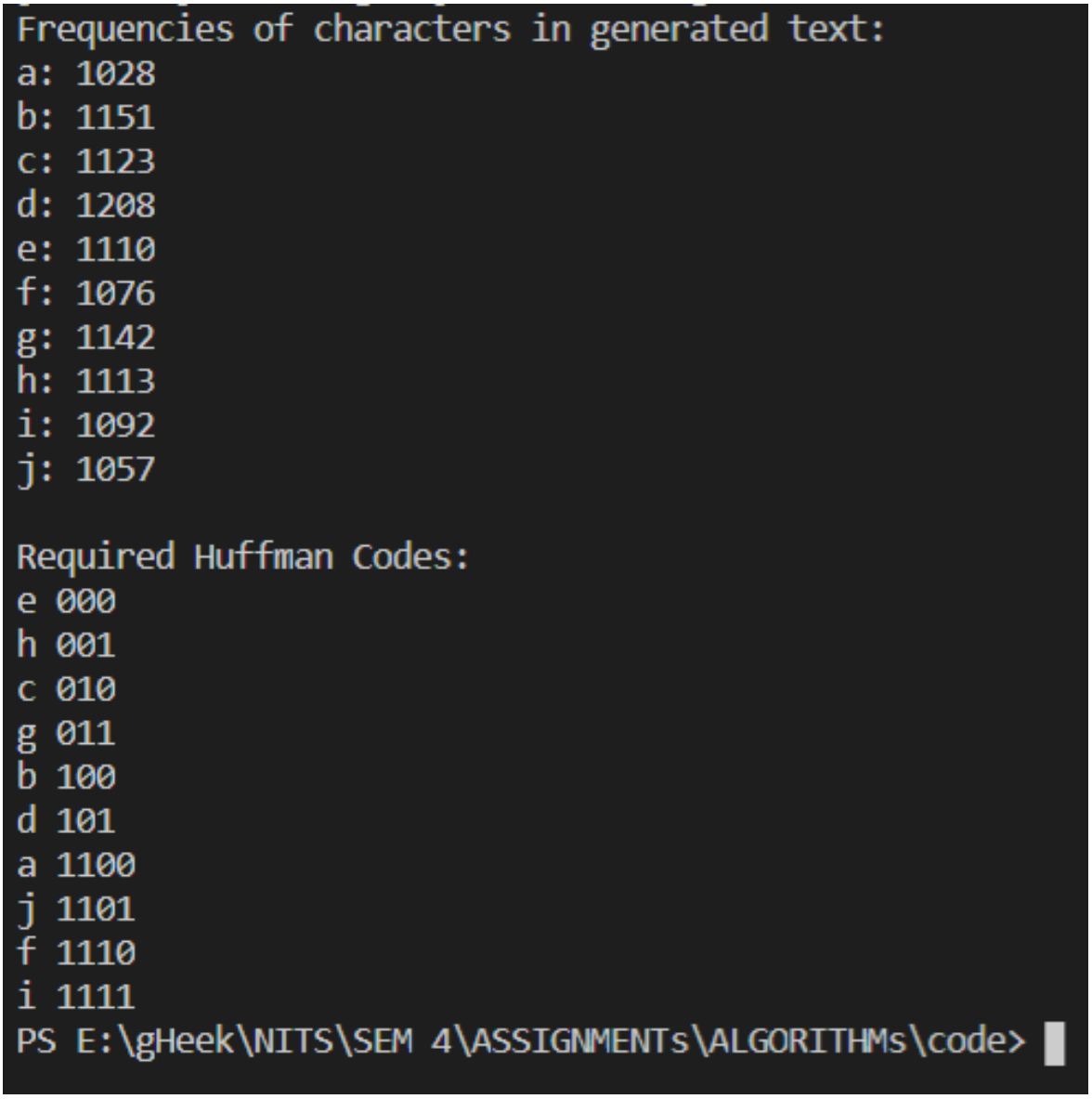


FIG3: TEXT-3, Frequencies and Hoffman Codes

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

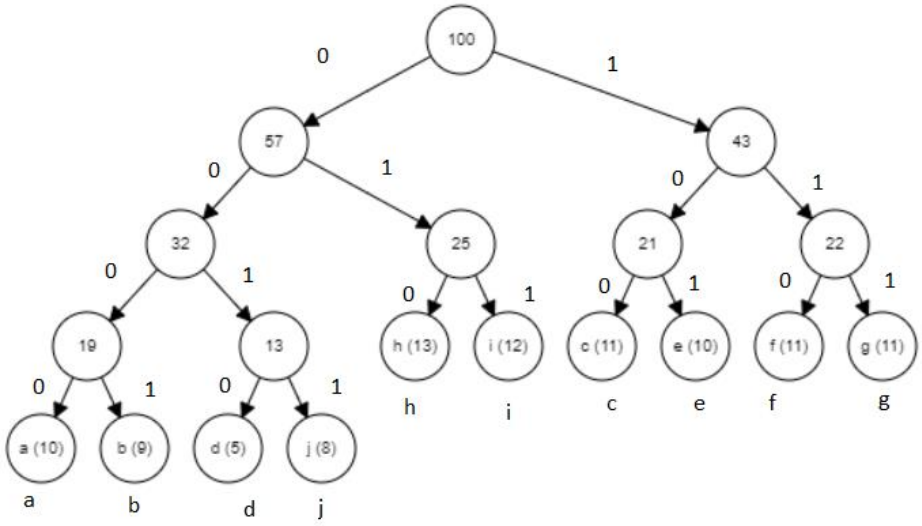


FIG4: Tree T1 for TEXT-1

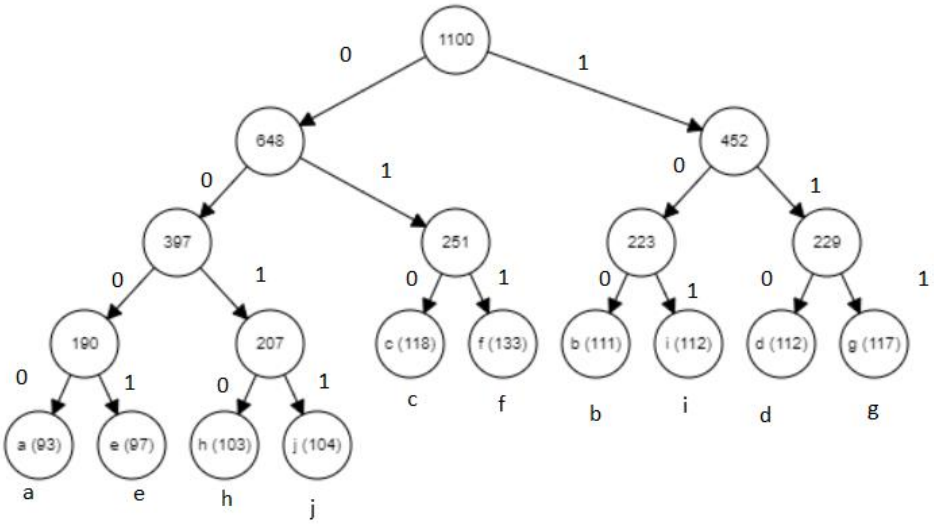


FIG4: Tree T2 for TEXT-2

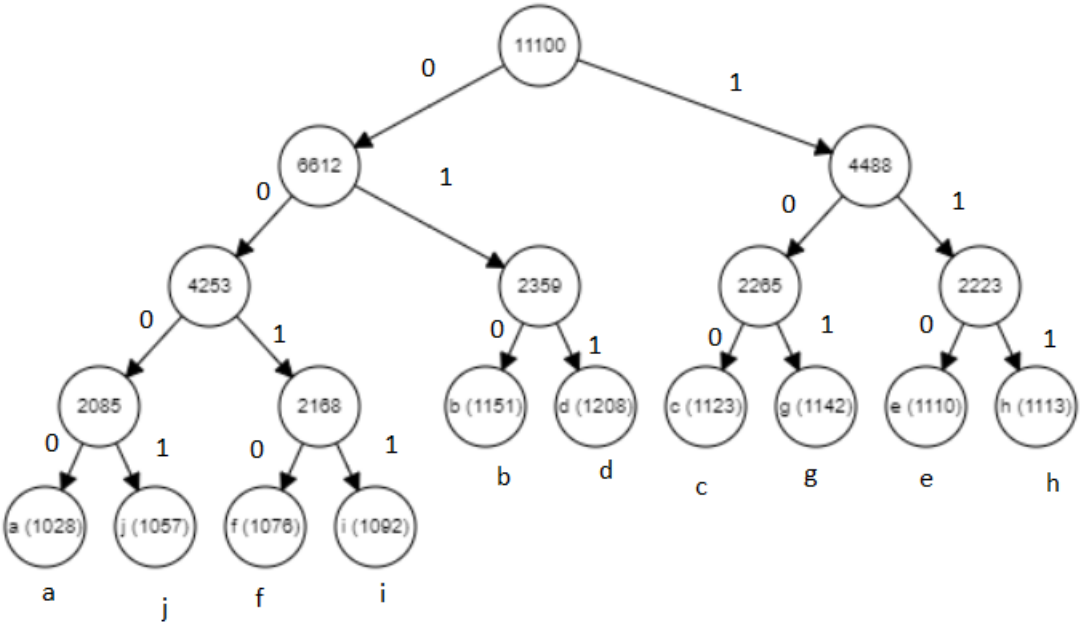


FIG4: Tree T3 for TEXT-3