

Lab Assignment-7

Indian Institute of Technology Roorkee Department of Computer Science and Engineering

CSN-361: Computer Networks Laboratory (Autumn 2019-2020)

Aman jaiswal
Enrollment No:-17114008
B.tech CSE 3rd Year.

Problem Statement 1:

Transmit a binary message (from a sender to a receiver) using socket programming in C and report whether the received msg is correct or not; using the following error detection algorithms:

1.Single Parity Check

2.Two-dimensional Parity Check

3.Checksum

4.Cyclic Redundancy Check (CRC)

Sol-

Data structure and Algorithm used->

1. Simple Parity check

Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

- 1 is added to the block if it contains odd number of 1's, and
- 0 is added if it contains even number of 1's

2. Two-dimensional Parity check

Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns, then both are sent along with the data. At the receiving end these are compared with the parity bits calculated on the received data.

3. Checksum

- In checksum error detection scheme, the data is divided into k segments each of m bits.
- In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.
- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded.

4. Cyclic redundancy check (CRC)

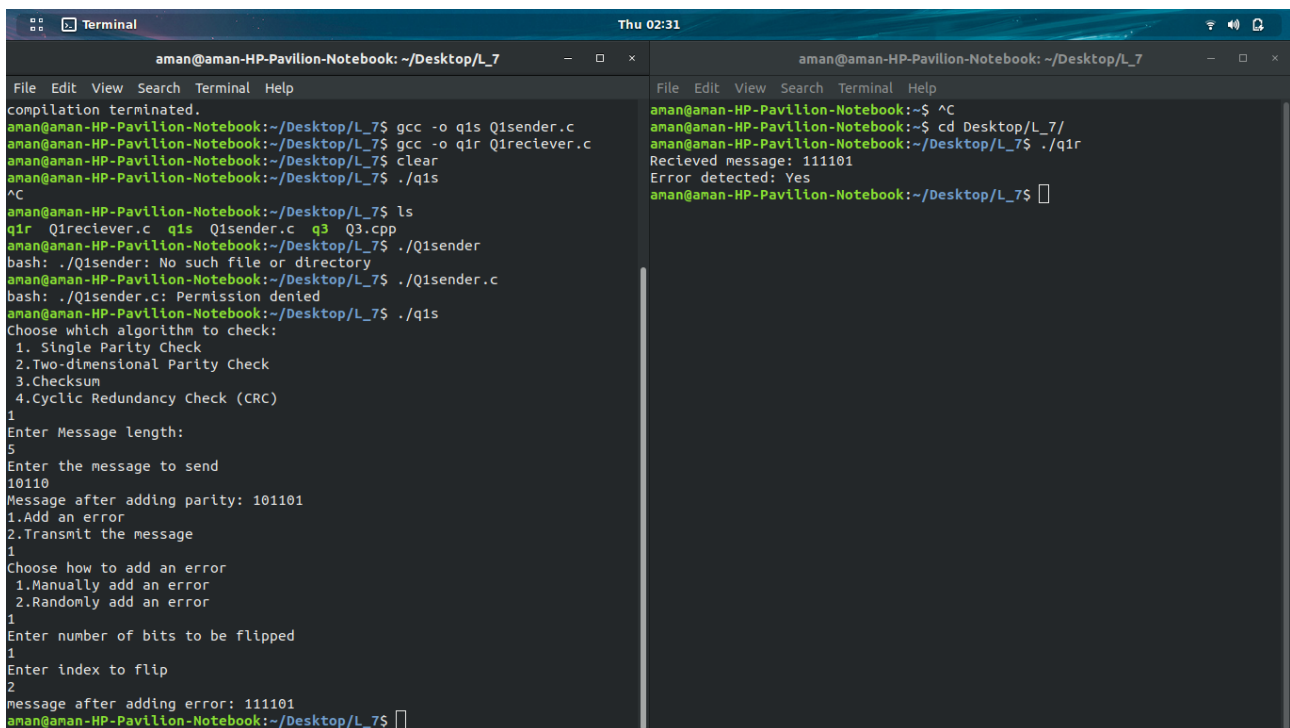
- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

Data Structure used->

- Socket creation:
sockfd: socket descriptor, an integer
struct sockaddr_in : structure to store internet addresses like IP address, port.
- Bind:
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
bind function binds the socket to the address and port number specified in addr.
- Listen:
int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow.

- Accept:
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.
- Socket connection: same as that of server's socket creation
- Connect: int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
- Two-Dimensional Arrays
- Character Arrays



```
aman@aman-HP-Pavilion-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
compilation terminated.
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ gcc -o q1s Q1sender.c
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ gcc -o q1r Q1receiver.c
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ clear
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1s
^C
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ls
q1r Q1receiver.c q1s Q1sender.c q3 Q3.cpp
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./Q1sender
bash: ./Q1sender: No such file or directory
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./Q1sender.c
bash: ./Q1sender.c: Permission denied
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1s
Choose which algorithm to check:
1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)
1
Enter Message length:
5
Enter the message to send
10110
Message after adding parity: 101101
1. Add an error
2. Transmit the message
1
Choose how to add an error
1. Manually add an error
2. Randomly add an error
1
Enter number of bits to be flipped
1
Enter index to flip
2
message after adding error: 111101
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$

aman@aman-HP-Pavilion-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
aman@aman-HP-Pavilion-Notebook:~$ ^C
aman@aman-HP-Pavilion-Notebook:~$ cd Desktop/L_7/
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1r
Recieved message: 111101
Error detected: Yes
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$
```

```

aman@aman-HP-Pavilion-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)
2
Enter Message length:
12
Enter Number of segments of message
3
Enter 3 segments of 12 bits message
1010
1101
0101
Two dimensional parity matrix
1 0 1 0 0
1 1 0 1 1
0 1 0 1 0
0 0 1 0 1
1. Add an error
2. Transmit the message
1
Choose how to add an error
1. Manually add an error
2. Randomly add an error
1
Enter number of bits to be flipped
1
Enter the row and column of bit to flip
2
3
Two dimensional parity matrix after adding error
1 0 1 0 0
1 1 1 1 1
0 1 0 1 0
0 0 1 0 1
Transmitted message: 10100111110101000101
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$

aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1r
Number of segments recieved: 3
Recieved message: 10100111110101000101
Found Error at Row: 2
Found Error at Column: 3
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$
```

```

aman@aman-HP-Pavilion-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1s
Choose which algorithm to check:
1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)
3
Enter Message length:
12
Enter Number of segments of message
3
Enter 3 segments of 12 bits message
1010
1110
0001
Checksum: 0101
Message after adding checksum: 1010111000010101
1. Add an error
2. Transmit the message
1
Choose how to add an error
1. Manually add an error
2. Randomly add an error
2
Enter probability of induced error
0.9
Transmitted message: 1010011000010110
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$

aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1r
Number of segments recieved: 3
Recieved message: 1010011000010110
Checksum: 0111
Error found
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$
```

```
Thu 02:45
aman@aman-HP-Pavilion-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1s
Choose which algorithm to check:
1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)
4
Enter length of Divisor:
4
Enter Divisor:
1011
Enter Message length:
10
Enter the message to send
1010111011
Remainder: 111
Message after CRC: 101011101111
1. Add an error
2. Transmit the message
1
Choose how to add an error
1. Manually add an error
2. Randomly add an error
1
Enter number of bits to be flipped
1
Enter index to flip
5
Transmitted message: 101001101111
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$

aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$ ./q1r
Divisor length recieved: 4
Divisor recieved: 1011
Recieved message: 101001101111
Remainder: 010
Error found
aman@aman-HP-Pavilion-Notebook:~/Desktop/L_7$
```

Problem Statement 2:

Transmit a binary message (from a sender to a receiver) using socket programming in C. Using Hamming code detect and correct errors in the transmitted message, if any.

Algorithm used :

- First we get the length of the dataword to be sent to the server from the client.
- After taking the data word as the input, we calculate the number of redundancy bits we need to add, to form the necessary code word, by using the formula,
$$2^m - 1 - m \geq k$$
where m is the number of redundant bits, k is the size of the data word.
- We add the redundant bits in the places whose numerical values are in powers of 2.
- We calculate the values of the redundant bits, done as follows,
For 1st checkbit (1st position), we add the values of 1, 3, 5, 7 th bits and so on, mod 2.
For 2nd checkbit (2nd position), we add the values of 2, 3, 6, 7 th bits and so on, mod 2.
For 3rd checkbit (4th position), we add the values of 4-7, 12-15 th bits and so on, mod 2.
- We can get some errors by flipping some bits, according to the user's needs.
- Since the dmin is constant for all sizes of data words, and is 3, the maximum error which we can detect is 2, and the maximum error which we can correct is 1.
- The server receives the codeword, and calculates the values of all the check bits again.
- The numerical value of the checkbit, gives the position of the error.
- If it is 0, then no error has been detected.
- For non zero values, we can flip the bit at that position and get the correct codeword.
- We then extract the data word from the codeword by neglecting the characters at positions which are powers of 2.

Data structures used :

- Socket creation:
sockfd: socket descriptor, an integer
struct sockaddr_in : structure to store internet addresses like IP address, port.
- Bind:
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
bind function binds the socket to the address and port number specified in addr.
- Accept:
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.
- Socket connection:
same as that of server's socket creation
- Connect:
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

```
Thu 03:22
aman@aman-HP-Pavillon-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
aman@aman-HP-Pavillon-Notebook:~/Desktop/L_7$ ./q2s
The message from the client is 01011101010111
14
Number of redundancy bits 4
0 1 0 1 1 1 0 1 0 1 0 1 1 1
Redundancy bits: 1 1 1 0
Assuming single error, the error location is 7
The data word is: 0101110011
aman@aman-HP-Pavillon-Notebook:~/Desktop/L_7$

aman@aman-HP-Pavillon-Notebook: ~/Desktop/L_7
File Edit View Search Terminal Help
aman@aman-HP-Pavillon-Notebook:~/Desktop/L_7$ ./q2c
Enter the size of the data word: 10
0 1 0 1 1 1 0 0 1 1
The number of redundancy bits is: 4
The encoded code word is: 0 1 0 1 1 1 0 0 1 0 1 1 1
Enter the number of errors ( we can correct only one error ): 1
Enter the location of the erroneous bit: 7
Flipping the 7 bit
0 1 0 1 1 1 0 1 0 1 0 1 1 1
Client : Sending the code word
Message from server : Read your message
aman@aman-HP-Pavillon-Notebook:~/Desktop/L_7$
```

Problem Statement 3:

Write a C++ program to compress a message (non-binary, can be anything like a text message or a code like hexadecimal, etc.) using the following data compression algorithm:

1. Huffman 2. Shannon-Fano

Huffman Coding

There are mainly two major parts in Huffman Coding

1) Build a Huffman Tree from input characters.

Steps to build Huffman Tree

- Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.
-
- 1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
-
- 2. Extract two nodes with the minimum frequency from the min heap.
-
- 3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
-
- 4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

2) Traverse the Huffman Tree and assign codes to characters.

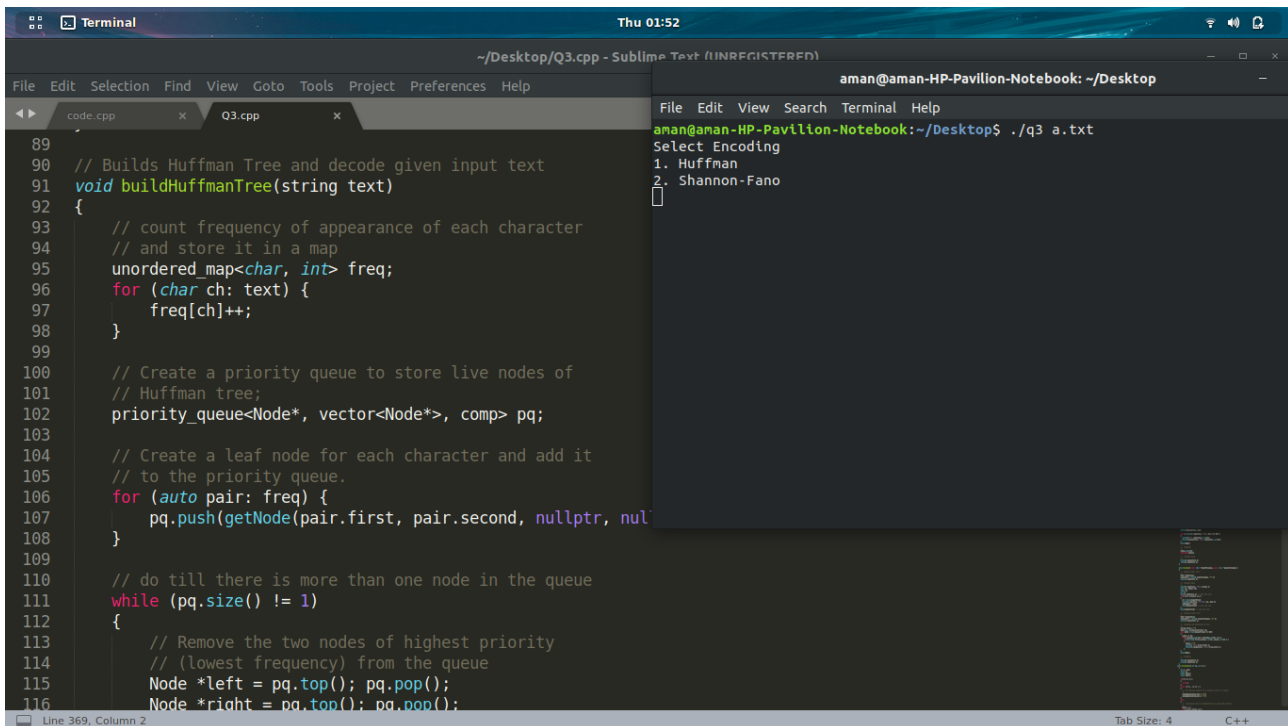
SHANNON FANO CODING

The steps of the algorithm are as follows:

- Create a list of probabilities or frequency counts for the given set of symbols so that the relative frequency of occurrence of each symbol is known.
- Sort the list of symbols in decreasing order of probability, the most probable ones to the left and least probable to the right.
- Split the list into two parts, with the total probability of both the parts being as close to each other as possible.
- Assign the value 0 to the left part and 1 to the right part.
- Repeat the steps 3 and 4 for each part, until all the symbols are split into individual subgroups.

Data Structure and Functions used:-

- ofstream: Stream class to write on files
- ifstream: Stream class to read from files
- fstream: Stream class to both read and write from/to files.
- Node- a struct tree node
- getNode- Function to allocate a new tree node
- comp-Comparison object to be used to order the heap
- encode-traverse the Huffman Tree and store Huffman Codes in a map.
- Qsort -sort ptable according to probabilities
- decode-traverse the Huffman Tree and decode the encoded string
- buildHuffmanTree - Builds Huffman Tree and decode given input text
- ptable- table of probability
- EncShannon-Calculating sum of probabilities at specified interval



The screenshot shows a Sublime Text editor window with two tabs: 'code.cpp' and 'Q3.cpp'. The 'code.cpp' tab is active, displaying C++ code for Huffman coding. The code includes comments and uses data structures like `unordered_map` and `priority_queue`. The 'Q3.cpp' tab is also visible, showing a `buildHuffmanTree` function. To the right, a terminal window is open, showing the command `aman@aman-HP-Pavilion-Notebook: ~/Desktop$./q3 a.txt` and its output: 'Select Encoding' followed by a list: '1. Huffman' and '2. Shannon-Fano'.

```
89
90 // Builds Huffman Tree and decode given input text
91 void buildHuffmanTree(string text)
92 {
93     // count frequency of appearance of each character
94     // and store it in a map
95     unordered_map<char, int> freq;
96     for (char ch: text) {
97         freq[ch]++;
98     }
99
100     // Create a priority queue to store live nodes of
101     // Huffman tree;
102     priority_queue<Node*, vector<Node*>, comp> pq;
103
104     // Create a leaf node for each character and add it
105     // to the priority queue.
106     for (auto pair: freq) {
107         pq.push(getNode(pair.first, pair.second, nullptr, nul
108     }
109
110     // do till there is more than one node in the queue
111     while (pq.size() != 1)
112     {
113         // Remove the two nodes of highest priority
114         // (lowest frequency) from the queue
115         Node *left = pq.top(); pq.pop();
116         Node *right = pq.top(); pq.pop();
```

```
Terminal
Thu 01:52
aman@aman-HP-Pavillon-Notebook: ~/Desktop

File Edit View Search Terminal Help

1
Huffman Codes are :

r:1111
t:1110
e:110
o:1011
a:01000
n:01001
c:0011
s:0110
p:00101

:011000
b:00100
s:01010
g:01011
d:011001
:100
u:10101
k:01101
w:101000
l:101001

Original string was :
computer networks lab computer science and engineering iit roorkee

Encoded string is :
0111101101001001011010100111101111000001100011101000101111101101010100101001010000010010001111011010010010110100111101111000101001111101100000
11111010001000000001100110011000001011111000011011011111100000101110011101110001110011111011111011011101101100011000
Encoded message length:270

Decoded string is:
computer networks lab computer science and engineering iit roorkee
aman@aman-HP-Pavillon-Notebook:~/Desktop$
```

```
Terminal
Thu 01:53
aman@aman-HP-Pavillon-Notebook: ~/Desktop

File Edit View Search Terminal Help

aman@aman-HP-Pavillon-Notebook:~/Desktop$ ./q3 a.txt
Select Encoding
1. Huffman
2. Shannon-Fano
2

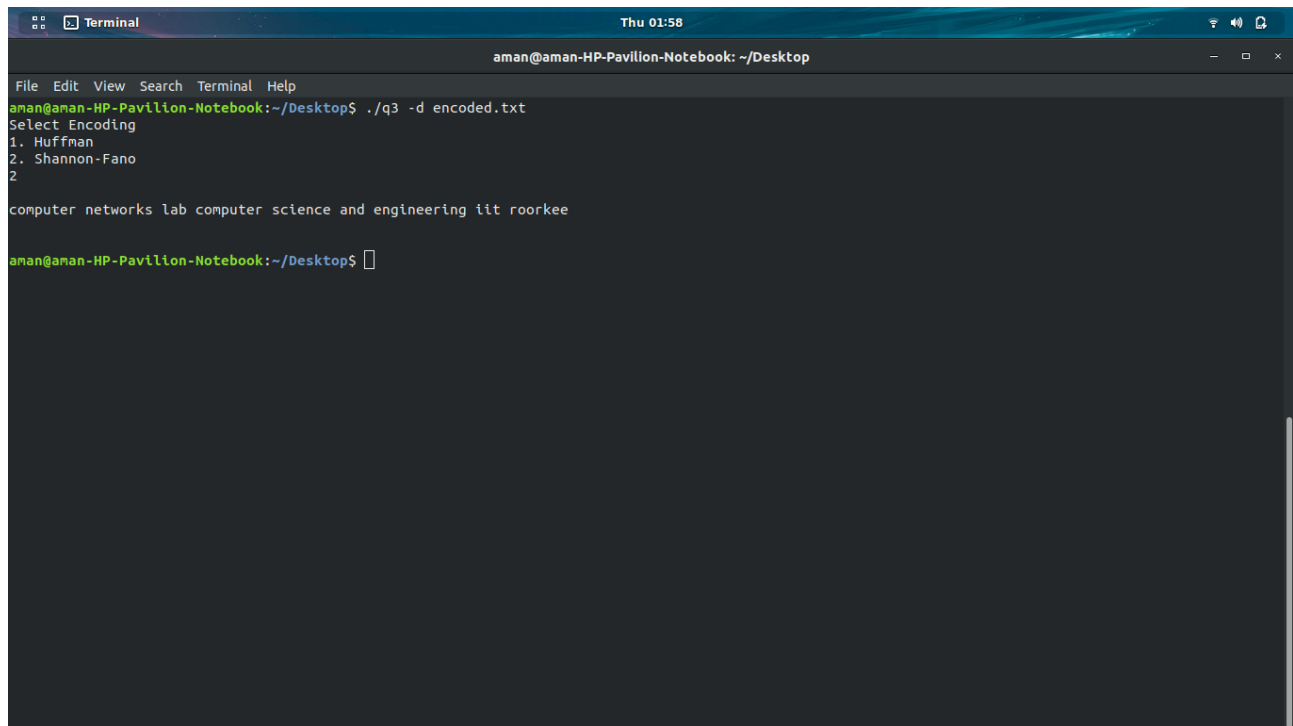
20
e      0.147059      00
      0.132353      010
n      0.088235      0110
r      0.088235      0111
t      0.073529      100
o      0.073529      1010
c      0.058824      10110
t      0.058824      10111
a      0.029412      11000
g      0.029412      11001
k      0.029412      11010
m      0.029412      11011
p      0.029412      11100
s      0.029412      11101
u      0.029412      111100

      0.014706      111101
b      0.014706      1111100
d      0.014706      1111101
l      0.014706      1111110
w      0.014706      1111111

1011010101101111001111001011100011101001111000111100010101101011110110001111000101011010110111100111100101110001110011101101101101
000001010110000101100001101111101010000110110011000110000001111000110110010101001001011010011110101010011110100000010111101

Length of encoded message : 277

aman@aman-HP-Pavillon-Notebook:~/Desktop$ clear
```

A screenshot of a terminal window titled "Terminal" with a blue header bar. The window shows a user named "aman" at a machine named "aman-HP-Pavilion-Notebook" in the directory "~/Desktop". The user has run the command `./q3 -d encoded.txt`. The program prompts "Select Encoding" and lists two options: "1. Huffman" and "2. Shannon-Fano". The user has entered "2". The program then outputs the decoded text: "computer networks lab computer science and engineering iit roorkee". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help".

```
aman@aman-HP-Pavilion-Notebook: ~/Desktop
File Edit View Search Terminal Help
aman@aman-HP-Pavilion-Notebook:~/Desktop$ ./q3 -d encoded.txt
Select Encoding
1. Huffman
2. Shannon-Fano
2
computer networks lab computer science and engineering iit roorkee
aman@aman-HP-Pavilion-Notebook:~/Desktop$
```

github:-<https://github.com/jaiswalaman/Assignment-7-CSN-361->