Indian Institute of Technology Roorkee

Department of Computer Science and Engineering

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

Lab Assignment-7 (L7) **Date: September 26, 2019** **Duration: 1 Week**

General Instructions:

1. Every Lab Assignment will be performed by the students individually**.** No group formation is required, and the evaluations will be done every week for the students individually**.**

Submission and Evaluation Instructions:

1. **Submit your** zipped folder (**<filename>.zip** or **<filename>.tar.gz**) through your account in Moodle through the submission link for this Lab Assignment in Moodle course site: https://moodle.iitr.ac.in/course/view.php?id=47
2. **Hard deadline for Final submission in Moodle: October 3, 2019 (9:00 am Indian Time).** For any submission after Final Deadline, 20% marks will be deducted (irrespective of it is delayed by a few seconds or a few days). The key to success is starting early. You can always take a break, if you finish early.
3. The submitted zipped folder (**<filename>.zip** or **<filename>.tar.gz**) must contain the following:
   (a) The source code files in a folder.
   (b) A report file (**<filename>.DOC** or **<filename>.PDF**) should contain the details like:
      i. Title page with details of the student
      ii. Problem statements
      iii. Algorithms and data structures used in the implementation
      iv. Snapshots of running the codes for each of the problems
4. The submission by each student will be checked with others' submission to identify any copy case (using such detection software). If we detect that the code submitted by a student is a copy (partially or fully) of other's code, then the total marks obtained by one student will be divided by the total number of students sharing the same code.

Instructions for L7:

1. Objective of this Lab Assignment is to make the students familiar with the hardware and software aspects of computer networking.
2. The student will have to demonstrate and explain the coding done for this Lab Assignment in the next laboratory class to be held on **October 3, 2019** for evaluation**.**
3. Please read and follow the **Note** points carefully.

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)

Note:

1. The program should be user interactive to choose which algorithm to check.
2. Transmitted message(s) should be encoded at the sender's side. (Print: Message here)
3. Before transmitting the message, a menu should appear (at sender's side) with options as follows:
    A. Add an error
        i. Manually add an error
            a. Enter number of bits to be flipped: B
            b. Enter bits (B indices) to be flipped
        ii. Randomly add an error
            a. Enter probability of induced error: p_e
            (In this case iterate each bit and generate a random number between 0 and 1 for each bit, if random number is greater than p_e then flip the bit)
    B. Transmit the message (Print transmitted message here)
4. Received message(s) should be decoded at the receiver side

Input:

1. Single Parity Check:
    Length of the Message: N
    N bits binary message: M
2. Two-dimensional Parity Check:
    Length of the Message: N
    Number of segments of message: S (Note: N should be divisible by S)
    S segments of N bits message: M_1, M_2, …, M_S
3. Checksum:
    Length of the Message: N
    Number of segments of message: S (Note: N should be divisible by S)
    S segments of N bits message: M_1, M_2, …, M_S
4. CRC:
    Divisor: D (Should be shared with the receiver before transmitting the final message)
    Length of the Message: N
    Message of N bits: M

Output:

In each case, print:

1. The original message (At sender's terminal window)
2. The transmitted message (At sender's terminal window)
3. The received message (At receiver's terminal window)
4. Yes, if the message is correct else print No (At receiver's terminal window)

Sample Case:

1. Single Parity:

At Sender's Terminal Window	At receiver's terminal window
N: 5 M: 10110 M (after adding parity): 101101 Transmitted M (after adding error): 10**0**101	 M: 100101 Error detected: Yes

2. 2D Parity:

At Sender's Terminal Window	At receiver's terminal window
N: 12 S: 3 M: 1 1 1 0 | 0 0 0 0 | 1 1 0 1 Adding 2D parity: 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0 Adding error: 1 1 1 0 1 0 0 0 **1** 0 1 1 0 1 1 0 0 1 1 0 Transmitted M: 1 1 1 0 1 | 0 0 0 1 0 | 1 1 0 1 1 | 0 0 1 1 0	Received S: 3 Received M: 1 1 1 0 1 | 0 0 0 1 0 | 1 1 0 1 1 | 0 0 1 1 0 Found Error at Row: 2, Col: 4

3. Checksum:

At Sender's Terminal Window	At receiver's terminal window
N: 12 S: 3 M: 1 1 1 0 | 0 0 0 0 | 1 1 0 1 M (with added checksum): 1 1 1 0 | 0 0 0 0 | 1 1 0 1 | 0 0 1 1	Received S: 3 Received M: 1 1 1 0 | 0 0 0 0 | 1 1 0 1 | 0 0 1 1 Checksum: 0 0 0 0 Hence, no error found

4. CRC:

At Sender's Terminal Window	At receiver's terminal window
D: 1001 N: 12 M: 110011110100 M (after CRC): 110011110100111 M (transmitted, after adding error): 110**1**11110100111	Received D: 1001 Received M: 110111110100111 Remainder: 100 Hence, error found

---

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.

Note:

1. Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is transmitted from the sender to the receiver
2. Message should be encoded at the sender's side
3. An error should be added in the message to be transmitted at the sender's side using an interactive menu as follows:

(a) Add an error
    a) Manually add an error
        1. Enter number of bits to be flipped: B
        2. Enter bits (B indices) to be flipped
    b) Randomly add an error (here randomly generate a number between minimum and maximum index of encoded message and flip the bit)
(b) Transmit the message

4. The message with an error (via point 3) should be transmitted to the receiver
5. Finally, the message should be decoded at the receiver side, and the original message and the detected error should be printed.

Input:

Enter the length of Message: N
Enter the N bit Message: M

Output:

1. Original Message (At sender's terminal window)
2. Redundant bits (At sender's terminal window)
3. Encoded message (At sender's terminal window)
4. Transmitted message (with error if added any) (At sender's terminal window)
5. Received message (At receiver's terminal window)
6. Error Detected, if any (At receiver's terminal window)
7. Decoded message (At receiver's terminal window)

Sample Case:

At Sender's Terminal Window	At receiver's terminal window
N: 7 M: 1100111 Redundant bits: 4 M (after encoding): 11000110101 M (transmitted, after adding error): 11001110101	Received M: 11001110101 Redundant bits: 4 Error is at location: 7 (1 indicates right most bit and 0 implies no error) Corrected M: 11000110101 Original M was: 1100111

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

(STL use is allowed and recommended in this question)

Note:

1. No message passing/transmission is required in this case.
2. The student shall present the compressed message along the code used to compress the message as per the used algorithm (if any).
3. The program should be user interactive with options to choose from the list which algorithm to be used of data compression.
4. In this question the code for each symbol can be different for each student but the length of encoded message would be same (provided correct implementation)

Input:

1. Choice of algorithm

2. A text file (.txt) containing the message

Output:

1. Encoded message with code used for encoding
2. Demonstrate and show the decoded message, also write the encoded message in a file
3. Print the length of encoded message in each case

Sample Case:

1. Huffman:

Input text (in file input.txt)		computer networks lab computer science and engineering iit roorkee
Output	*Huffman codes* 'u': 0000 'g': 00010 'p': 00011 'r': 001 'k': 01000 's': 01001 'w': 010100 'l': 010101 'd': 010110 'b': 010111 't': 0110 'c': 0111 ' ': 100 'm': 10100 'a': 10101 'o': 1011 'e': 110 'i': 1110 'n': 1111	*Encoded message:* 011110111010000010000011 011000110011111001100101 001011001010000101100010 101101010101111000111011 101000001100000110100011 000100101111110110111011 111010010101111010110100 110111100010111011111011 000111101111000101001101 110011010000110111011010 1000110110 *Encoded message length*: 260 *Decoded message:* computer networks lab computer science and engineering iit roorkee

2. Shannon-Fano

Input text (in file input.txt)		computer networks lab computer science and engineering iit roorkee
Output	*Shannon-Fano codes* ' ': 0111 'a': 11001 'b': 11000 'c': 0101 'd': 0110 'e': 111 'g': 01000 'i': 1001 'k': 1010 'l': 00000 'm': 00001 'n': 1011 'o': 0001 'p': 10001 'r': 0011 's': 0010 't': 1101 'u': 01001 'w': 10000	*Encoded message:* 010100010000110001010011 1 011110011011110111111011 000000010011101000100111 0 000011001110000111010100 0 100001100010100111011110 0 110111001001011001111101 1 010111101111100110110110 0 111111101101000100110111 1 111100111001101101000011 1 100110011101011100110001 0 00100111010111111 *Encoded message length*: 267 *Decoded message:* computer networks lab computer science and engineering iit roorkee