Computer Networks

Lab Assignment

BTech (IT) V Semester

Q.1. Huffman Coding

File compression and decompression (You may choose a language of your choice)

It is often possible to make a file significantly smaller without any loss of information whatsoever ("lossless compression"). The trick is to figure out how to do that, and how to reconstruct the original file when needed. Huffman code compression is used today as part of the JPEG image compression and the mp3 audio compression standards etc.

Write a C++ program `compress` that will be invoked with a command line of the form

```
 compress infile outfile
```

When run, this program will read the contents of the file named by its first command line argument, construct a Huffman code for the contents of that file, and use that code to construct a compressed version which is written to a file named by the second command line argument. The input file can contain any data (not just ASCII characters) so it should be treated as a binary file. Your compress program must work for input files up to 10 megabytes in size, so a particular byte value may occur up to 10 million times in the file.

You will also write a C++ program `uncompress` that will be invoked with a command line of the form

```
 uncompress infile outfile
```

When run, this program will read the contents of the file named by its first command line argument, which should be a file that has been created by the `compress` program. It will use the contents of that file to reconstruct the original, uncompressed version, which is written to a file named by the second command line argument. In particular, for *any* file `F`, after running

```
 compress F G
 uncompress G H
```

it must be the case that the contents of `F` and `H` are identical.

Control flow

Taking a top-down design approach to decomposing the problem, you can see that your `compress` program basically needs to go through these steps:

1. Open the input file for reading.
2. Read bytes from the file, counting the number of occurrences of each byte value; then close the file.
3. Use these byte counts to construct a Huffman coding tree.
4. Open the output file for writing.
5. Write enough information (a "file header") to the output file to enable the coding tree to be reconstructed when the file is read by your `uncompress` program.
6. Open the input file for reading, again.
7. Using the Huffman coding tree, translate each byte from the input file into its code, and append these codes as a sequence of bits to the output file, after the header.
8. Close the input and output files.

Thinking along similar lines, your `uncompress` program should go through these basic steps:

1. Open the input file for reading.
2. Read the file header at the beginning of the input file, and reconstruct the Huffman coding tree.
3. Open the output file for writing.
4. Using the Huffman coding tree, decode the bits from the input file into the appropriate sequence of bytes, writing them to the output file.
5. Close the input and output files.

Use a trie.

Q.2. Using the same sequence and thinking along similar lines, write programs in C++ for the following (You may choose a language of your choice)

1. **Parity Check Codes**
2. **CRC codes (coding and decoding)**