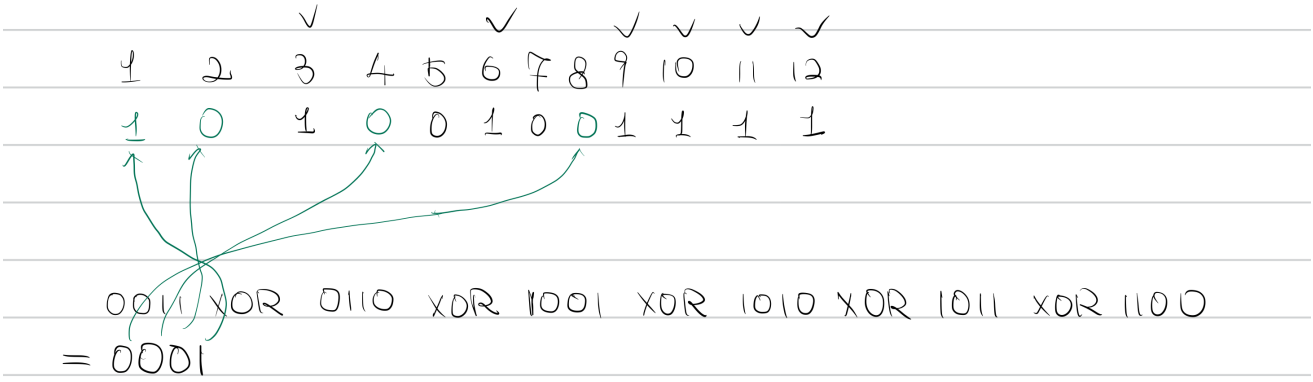
Homework 3

解雲暄 3190105871

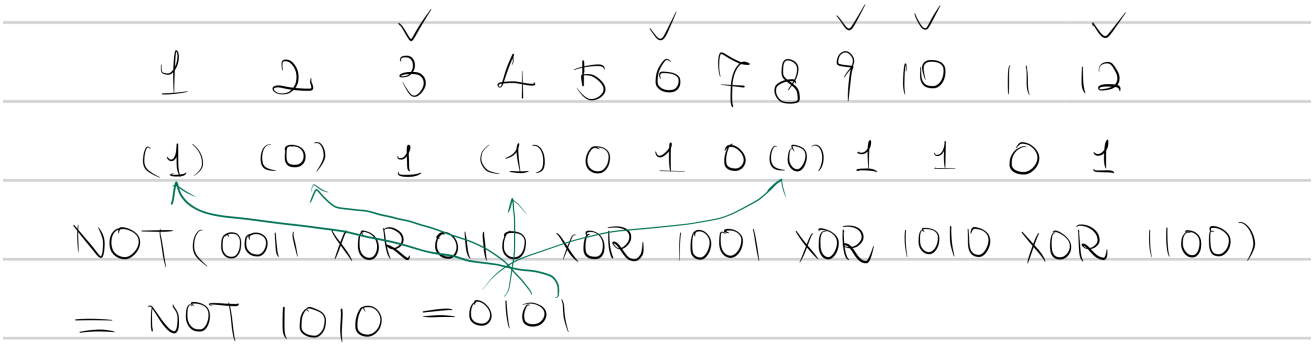
1. Each 111110 will be destuffed into 11111. So the data after destuffing is `0110 0111 1101 1110 1111 11`.

2. The binary value after encoding is `1010 0100 1111`. The approach is as follows:



Even-parity Hamming code is the Hamming code in textbook. For odd-parity Hamming code, we will converse each parity bit.

3. `0xB4D` is `1011 0100 1101` in binary. We can see, there are no bit errors, so the original value is `1010 1101` in binary, or `0xAD` in hexadecimal.



4. No. When we want to *correct* a single error with a distance of 3, there is an important assumption that there will be no double-bit errors; Otherwise, if certain codeword W_0 encounters a double-bit error and become W_0', but with a Hamming distance of 3, W_0' might has a Hamming distance of 1 with another valid codeword W_1, then we will correct W_0' into W_1, which may cause an error. So when we want to *detect* 2 errors, then in this case the *correcting-1-bit* cannot function well.

   Therefore, with a Hamming distance of n, we can detect $n-1$ bit errors, or correct $\lfloor \frac{n-1}{2} \rfloor$ bit errors. But we cannot do both of them at the same time.

5. For each valid message with length m, there should be $(C_n^2 + C_n^1)$ invalid codewords uniquely corresponding to it, where $n = m + r$. So there are totally $(C_n^2 + C_n^1 + 1)2^m$ codewords within 2^n possible values of length n, thus we can get

$$(C_n^2 + C_n^1 + 1)2^m \leq 2^n$$

   i.e.

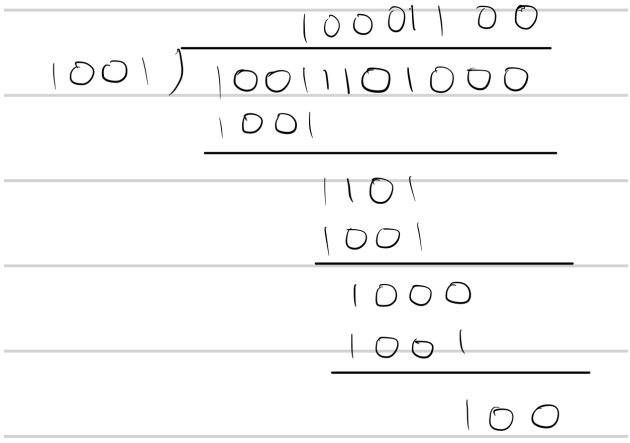
$$(m + r)^2 + m + r + 1 \leq 2^{r+1}$$

6. Internet checksum uses wrap-around carry bit:
   1001 + 1100 = 10101 => 0101 + 1 = 0110
   0110 + 1010 = 10000 => 0000 + 1 = 0001
   0001 + 0011 = 0100

   So the checksum is `0100`.

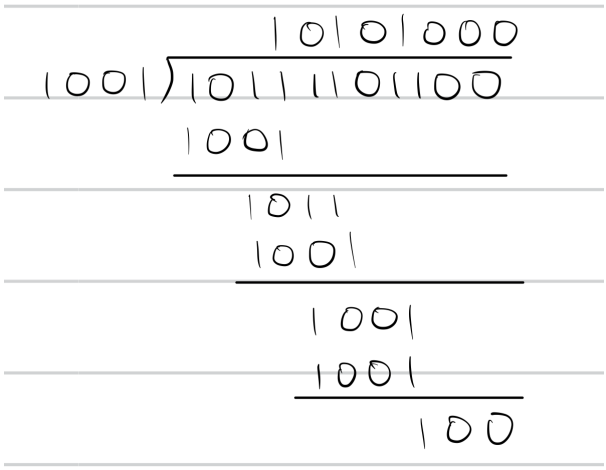
   ~~The 1's complement of 0100 is 1011, so the checksum is `1011`.~~
   # When using the checksum, we use its 1's complement.

7. We calculate `1001 1101 000 / 1001` in binary:



We get `1001 1101 000 mod 1001 = 100`, so **the bit string transmitted is `1001 1101 100`**.

If the 3rd bit is inverted, then the bit string will be `1011 1101 100`, we divide it by `1001`:

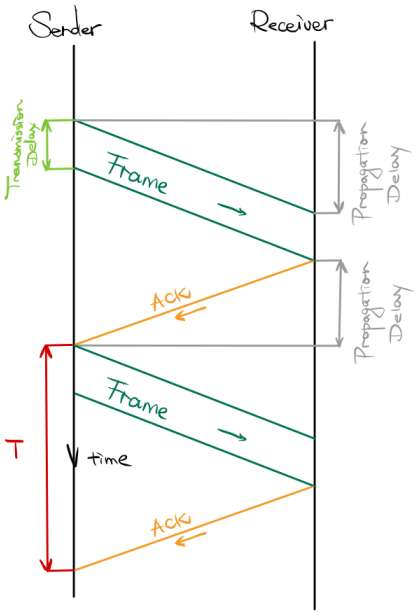


The remainder is not 0, so we can know there is an error.

If the errors make the transmitted bits still divisible by the generator polynomial (i.e. 1001), then the errors cannot be detected. **For example 1001 1100 101 cannot be detected.**

8. Because when we calculate CRC, we will append it to the tail to make the whole codeword be divisible by the generator polynomial.
   (Referenced the answer) And if we put it in the trailer, we can calculate while transmitting the bits and just append the remainder in the end; But if we put it in the header, we should first store the bit stream, calculate the remainder, and then append it in the header, finally transmit the whole frame.

9. 27kbps. Solution:



Each 1 Transmission Delay + 2 Propagation Delay forms a cycle, i.e.

$$T = TD + 2PD$$

Where Transmission Delay

$$TD = \frac{\text{Frame Size}}{\text{Bandwidth}} \triangleq \frac{F}{W}$$

Within a cycle, Stop-and-wait protocol conveys F bits, thus the efficiency

$$\eta = \frac{F}{T \cdot W}$$

i.e. $\eta = \frac{F}{TD \cdot W + 2PD \cdot W} = \frac{F}{F + 2PD \cdot W}$

$\therefore W = \frac{(1-\eta)F}{2\eta PD} = \frac{75\% \times 900\,\text{bit}}{2 \times 25\% \times 50\text{ms}} = 27\,\text{kbps}$

10. # Protocol 5 is go-back-n protocol; Bandwidth of T1 trunk is 1.544 Mbps.

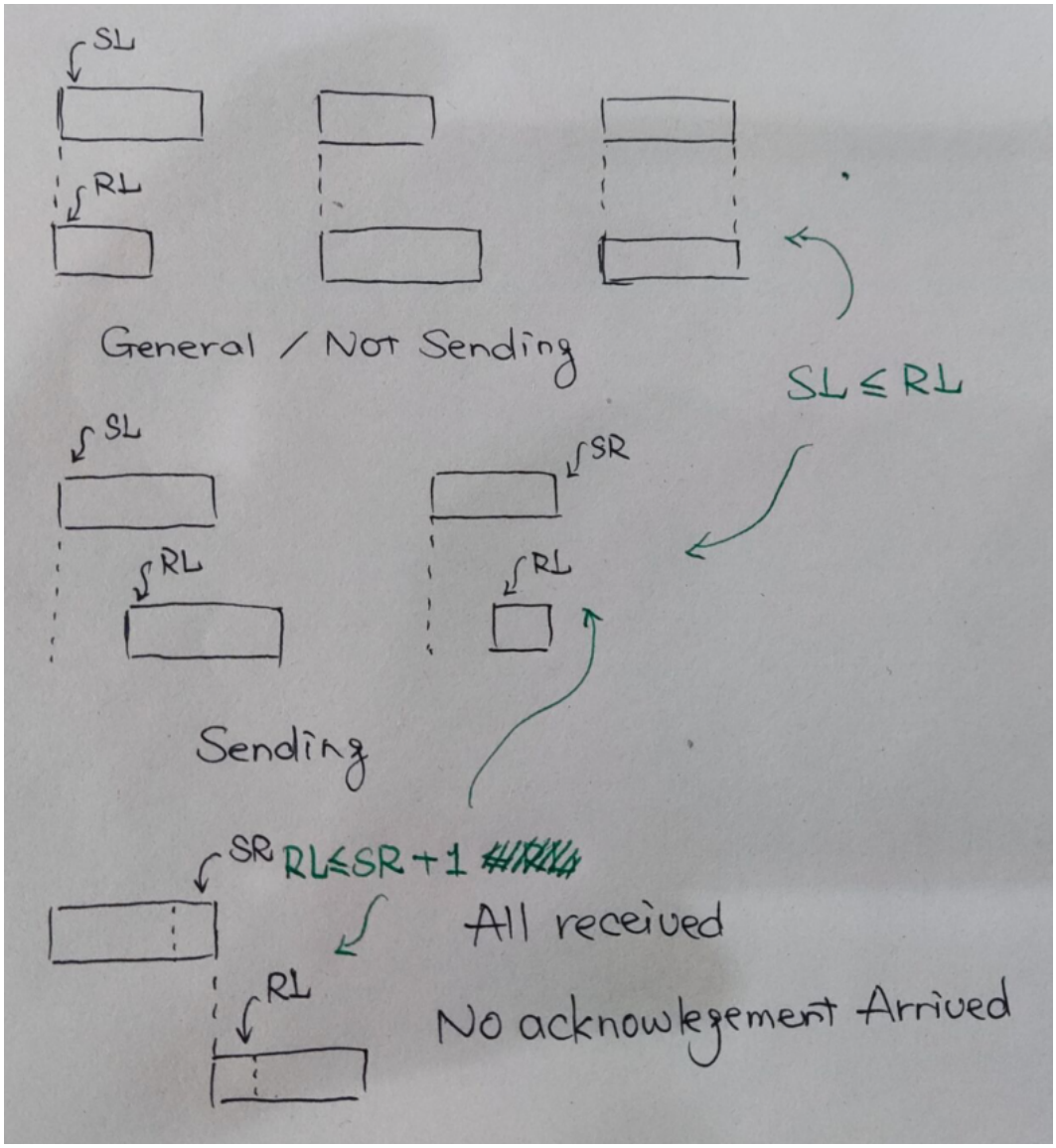
Propagation delay $PD = 3000\text{km} \times 6\mu\text{s/km} = 18\text{ms}$,

Transmission delay $TD = \frac{64\times 8 \text{ bits}}{1.544 \text{ Mbps}} = 331.6\mu\text{s} = 0.33\text{ms}$.

So in a cycle (see Problem 9), we can transmit $N = \frac{T}{TD} = \frac{TD+2PD}{TD} = 110$ frames.

As $2^6 = 64 < 110 < 128 = 2^7$, **we use 7 bits' sequence number**.

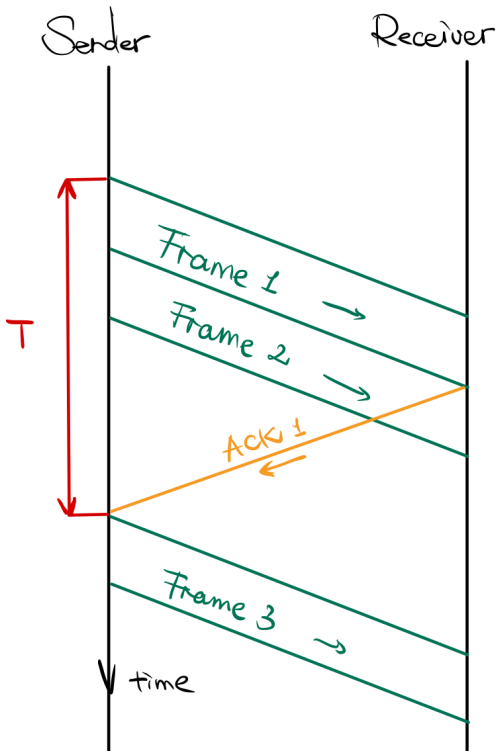
11. Sender Left $\leq$ Receiver Left $\leq$ Sender Right + 1. Solution:



12. # Protocol 6 is selective repeat protocol.

    Yes. Suppose the sender sent several frames, say No.0 ~ N, and the receiver received these frames, sent back an accumulative ACK N and started to waiting for Frame N+1. But the ACK was lost, then the receiver thought that all the frames were lost, so it sent No.0 ~ N again. However, the receiver wanted Frame N+1, so it sent NAK N+1 back. But the NAK was lost again. As the receiver would not send NAK again, the sender would be permanently sending No.0 ~ N, which is a severe error.

13. This problem is similar to Problem 9. We consider cases where sending window size is not 1. We here assume that the sending window size is 2, and the time cost for transmitting these frames is shorter than the round-trip delay. See the figure below:



(As the problem says "The headers are very short", we can recognize the acknowledgement very quickly even though it is piggybacked.)

We can see that, when the ACK of the 1st frame arrives, the 3rd frame can be sent. So still each *one* but not more transmission delay and 2 propagation delay forms a cycle. But as the window size doubled, the channel utilization doubled.

The time period of a cycle is $T = TD + 2PD = \frac{1000\text{bits}}{1\text{Mbps}} + 2 \times 270\text{ ms} = 541\text{ ms}$. In this period of time, maximumly $B = T \times W = 541\text{ ms} \times 1\text{ Mbps} = 541\text{ kb}$ data can be sent. Therefore:

1. For stop-and-wait protocol, in each cycle, only 1 frame is sent. So the channel utilization is
$$\eta_1 = \frac{N \times F}{B} = \frac{1 \times 1000\text{ bits}}{541\text{ kb}} = 0.185\%$$
2. For protocol 5 (GBN), the maximum sender window size can be 7. So in each cycle, 7 frames are sent. So the channel utilization is
$$\eta_2 = \frac{N \times F}{B} = \frac{7 \times 1000\text{ bits}}{541\text{ kb}} = 1.294\%$$
3. For protocol 6 (SR), the maximum sender window size can be 4. So in each cycle, 4 frames are sent. So the channel utilization is
$$\eta_3 = \frac{N \times F}{B} = \frac{4 \times 1000\text{ bits}}{541\text{ kb}} = 0.739\%$$

14. (Referenced the answer) PPP was designed to be implemented in software, but bit-stuffing, which is more suitable to use in hardware while transmitting, is kind of hard to implement in software, which is usually byte-oriented.

15. The fixed overhead bytes are 2 Flag bytes (front and end). Address and Control bytes (1 byte each) can be negotiated to omit; Protocol bytes can be 1 or 2 (default) bytes; Checksum bytes can be 2 (default) or 4 bytes.
    Therefore, the minimum overhead is 2 + 1 + 2 = 5 bytes; The maximum overhead is 2 + 2 + 2 + 4 = 10 bytes.