CN Lab Exam doc

Contents

Lab Programs:

BitStuffing.c:

```
#include <stdio.h>
#include <string.h>

int main() {
    char input[50], stuffed[100] = "01111110"; // Start flag
    int i, count = 0;

    printf("Enter bit stream: ");
    scanf("%s", input);

    for (i = 0; input[i]; i++) {
        stuffed[strlen(stuffed)] = input[i];
        if (input[i] == '1') {
            count++;
            if (count == 5) {
                strcat(stuffed, "0");
                count = 0;
            }
        } else {
            count = 0;
        }
    }

    strcat(stuffed, "01111110"); // End flag
    printf("Bit Stuffed: %s\n", stuffed);
```

```
    return 0;
}
```

Short code(optional):

```
#include<stdio.h>
#include<string.h>
void main(){
char in[50], out[100] = "01111110";
int c = 0;
scanf("%s", in);

for (int i = 0; in[i]; i++) {
    out[strlen(out)] = in[i];
    c = (in[i]=='1') ? c+1 : 0;
    if (c == 5) { strcat(out, "0"); c = 0; }
}
strcat(out, "01111110");
printf("%s", out);}
```

ByteStuffing.c

CharacterStuffing or Bytestuffing both are same

```
#include <stdio.h>
#include <string.h>

#define F '$'
#define E '@'

int main() {
```

```
    char in[50], s[100], d[100];
    int i, j;

    scanf("%s", in);

    // Stuff
    s[0] = F; j = 1;
    for (i = 0; in[i]; i++) {
        if (in[i] == F || in[i] == E) s[j++] = E;
        s[j++] = in[i];
    }
    s[j++] = F; s[j] = 0;

    printf("Stuffed: %s\n", s);

    // De-stuff
    for (i = 1, j = 0; s[i] != F; i++) {
        if (s[i] == E) i++;
        d[j++] = s[i];
    }
    d[j] = 0;

    printf("Destuffed: %s\n", d);
}
```

FixedSizeFraming.c:

```
#include <stdio.h>
#include <string.h>
```

```
#define SIZE 5

int main() {
    char msg[100];
    printf("Enter message: ");
    fgets(msg, sizeof(msg), stdin);
    msg[strcspn(msg, "\n")] = '\0';

    printf("Fixed Frames:\n");
    for (int i = 0; msg[i]; i++) {
        if (i % SIZE == 0) printf("\nFrame: ");
        printf("%c", msg[i]);
    }
    return 0;
}
```

Short code (optional):

```
#include<stdio.h>
#include<string.h>
void main(){
int k=0; char msg[100];
fgets(msg,100,stdin);

for(int i=0; msg[i]; i++){
    if(i%5==0) printf("\nFrame: ");
    printf("%c", msg[i]);
}
}
```

VariableSizeFraming.c:

```
#include <stdio.h>
#include <string.h>

int main() {
    char msg[100];
    int pos = 0, len, frameSize;

    printf("Enter message: ");
    fgets(msg, sizeof(msg), stdin);
    msg[strcspn(msg, "\n")] = '\0';
    len = strlen(msg);

    while (pos < len) {
        printf("Enter frame size (%d left): ", len - pos);
        scanf("%d", &frameSize);
        if (frameSize <= 0 || frameSize > len - pos) continue;

        printf("Frame [%d]: ", frameSize);
        for (int i = 0; i < frameSize; i++)
            printf("%c", msg[pos + i]);
        printf("\n");
        pos += frameSize;
    }
    return 0;
}
```

CRC.c:

```
#include <stdio.h>
#include <string.h>

char data[50], generator[20], temp[20];

void XOR() {
    for (int i = 0; i < strlen(generator); i++)
        temp[i] = (temp[i] == generator[i]) ? '0' : '1';
}

void CRC() {
    int i, j;
    for (j = 0; j < strlen(generator); j++)
        temp[j] = data[j];

    for (i = 0; i <= strlen(data) - strlen(generator); i++) {
        if (temp[0] == '1')
            XOR();
        for (j = 0; j < strlen(generator) - 1; j++)
            temp[j] = temp[j + 1];
        temp[j] = data[i + strlen(generator)];
    }
}

int main() {
    strcpy(generator, "1101");  // Example generator polynomial
    printf("Enter data: ");
```

```
    fgets(data, sizeof(data), stdin);
    data[strcspn(data, "\n")] = '\0';  // Remove newline if present

    int dataLen = strlen(data);
    int genLen = strlen(generator);

    // Append zeros to data
    for (int i = 0; i < genLen - 1; i++)
        data[dataLen + i] = '0';
    data[dataLen + genLen - 1] = '\0';

    printf("Modified data: %s\n", data);
    CRC();

    printf("CRC Remainder: %s\n", temp);

    // Append remainder to original data
    for (int i = 0; i < genLen - 1; i++)
        data[dataLen + i] = temp[i];

    printf("Transmitted data: %s\n", data);
    return 0;
}
```

SlidingWindowProtocol.c:

```
#include<stdio.h>

void main()
```

```
{
   int n,frame[100],ack,ch,i;
   printf("Enter The Frame size : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      frame[i]=i+1;
      printf("Frame %d sent data : %d\n",i,frame[i]);
   }
   printf("\nEnter the frame whose acknowledgemet is not recieved : ");
   scanf("%d",&ack);
   printf("\nEnter 1 for GO Back N\nEnter 2 for Selective Repeat\n");
   scanf("%d",&ch);
   if(ch==1)
   {
      printf("\nGO Back N\n");
      printf("Frames sent again : \n");
      for(int i=ack;i<n;i++)
      {
         printf("Frame %d sent data : %d\n",i,frame[i]);
      }
   }
   if(ch==2)
   {
      printf("\nSelective Repeat Protocol\n");
      printf("Frame sent again : \n");
      printf("Frame %d sent data : %d\n",ack,frame[ack]);
   }
}
```

DistanceVectorRouting.c:

```
#include <stdio.h>
#define MAX 10
#define INF 999

typedef struct {
   int dist[MAX], hop[MAX];
} Router;

void init(Router r[], int g[MAX][MAX], int n){
   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++){
         if(i==j) r[i].dist[j]=0, r[i].hop[j]=j;
         else if(g[i][j]!=0) r[i].dist[j]=g[i][j], r[i].hop[j]=j;
         else r[i].dist[j]=INF, r[i].hop[j]=-1;
      }
}

void dv(Router r[], int n){
   int upd;
   do{
      upd=0;
      for(int i=0;i<n;i++)
         for(int j=0;j<n;j++)
            for(int k=0;k<n;k++)
               if(r[i].dist[j] > r[i].dist[k] + r[k].dist[j]){
                  r[i].dist[j] = r[i].dist[k] + r[k].dist[j];
```

```
                r[i].hop[j] = k;
                upd=1;
            }
    }while(upd);
}

int main(){
    int n, g[MAX][MAX];
    Router r[MAX];
    scanf("%d",&n);

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++){
            scanf("%d",&g[i][j]);
            if(i!=j && g[i][j]==0) g[i][j]=INF;
        }

    init(r,g,n);
    dv(r,n);

    for(int i=0;i<n;i++){
        printf("\nRouter %d\n",i);
        for(int j=0;j<n;j++)
            printf("To %d: Dist=%d Hop=%d\n", j, r[i].dist[j], r[i].hop[j]);
    }
}
```

Dijkstra.c:

```
#include <stdio.h>
#define INF 9999
#define MAX 10

void dijkstra(int g[MAX][MAX], int n, int s){
   int cost[MAX][MAX], dist[MAX], vis[MAX]={0}, pred[MAX];

   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++)
         cost[i][j] = g[i][j]==0 ? INF : g[i][j];

   for(int i=0;i<n;i++) dist[i]=cost[s][i], pred[i]=s;

   dist[s]=0; vis[s]=1;

   for(int c=1;c<n;c++){
      int min=INF, nxt=-1;
      for(int i=0;i<n;i++)
         if(!vis[i] && dist[i]<min) min=dist[i],nxt=i;

      vis[nxt]=1;
      for(int i=0;i<n;i++)
         if(!vis[i] && dist[nxt]+cost[nxt][i] < dist[i])
            dist[i] = dist[nxt]+cost[nxt][i], pred[i]=nxt;
   }

   for(int i=0;i<n;i++)
      if(i!=s) printf("To %d = %d\n", i+1, dist[i]);
}
```

```
int main(){
    int n,g[MAX][MAX],s;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++) scanf("%d",&g[i][j]);
    scanf("%d",&s);
    dijkstra(g,n,s-1);
    return 0;
}
```

LinkStateRouting.c:

```
#include <stdio.h>
#define INF 99
#define MAX 10

void dijkstra(int g[MAX][MAX], int src, int n){
    int dist[MAX], vis[MAX]={0};
    for(int i=0;i<n;i++) dist[i]=INF;
    dist[src]=0;

    for(int c=0;c<n-1;c++){
        int u=-1,min=INF;
        for(int i=0;i<n;i++)
            if(!vis[i] && dist[i]<min) min=dist[i],u=i;

        vis[u]=1;
        for(int v=0;v<n;v++)
```

```
            if(!vis[v] && g[u][v] && dist[u]+g[u][v]<dist[v])
                dist[v]=dist[u]+g[u][v];
    }

    for(int i=0;i<n;i++)
        printf("%d -> %d = %d\n", src, i, dist[i]);
}

int main(){
    int n, g[MAX][MAX], src;
    scanf("%d", &n);

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++){
            scanf("%d",&g[i][j]);
            if(i!=j && g[i][j]==0) g[i][j]=INF;
        }

    scanf("%d",&src);
    dijkstra(g, src, n);
    return 0;
}
```

CaesarCipher.c:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

```
int main() {
    char s[100];
    int ch, i;
    printf("1.Encrypt  2.Decrypt: ");
    scanf("%d", &ch);
    scanf("%s", s);

    for(i = 0; s[i]; i++){
        if(isupper(s[i]))
            s[i] = ( (s[i] - 'A' + (ch==1?3:-3) + 26 ) % 26 ) + 'A';
        else
            s[i] = ( (s[i] - 'a' + (ch==1?3:-3) + 26 ) % 26 ) + 'a';
    }

    printf("%s", s);
    return 0;
}
```

LeakyBucket.c:

```
#include <stdio.h>
#define N 10

int main() {
    int p[N], bs, rate, rem = 0;
    printf("Enter output rate: "); scanf("%d", &rate);
    printf("Enter bucket size: "); scanf("%d", &bs);

    for(int i=0;i<N;i++) p[i]=(rand()%5+1)*10;
```

```
    for(int i=0;i<N;i++){
        if(p[i] > bs) { printf("Pkt %d rejected\n", p[i]); continue; }
        if(rem + p[i] > bs) { printf("Overflow! Dropped\n"); continue; }

        rem += p[i];
        printf("Added %d | Rem = %d\n", p[i], rem);

        while(rem > 0){
            int send = rem < rate ? rem : rate;
            rem -= send;
            printf("Sent %d | Rem = %d\n", send, rem);
        }
    }
    return 0;
}
```

Checksum.c:

```
#include <stdio.h>
unsigned short checksum(unsigned short *d, int n, int verify) {
    unsigned int s = 0;
    for (int i = 0; i < n; i++) s += d[i];
    while (s >> 16) s = (s & 0xFFFF) + (s >> 16);
    return verify ? s : ~s & 0xFFFF;   // if verify==1 → return sum, else checksum
}

int main() {
    int n; unsigned short w[25];
```

```
    printf("Enter the no. of 16-bit words: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("word%d: ", i + 1);
        scanf("%hx", &w[i]);
    }

    unsigned short cs = checksum(w, n, 0);
    printf("Calculated checksum: 0x%X\n", cs);

    w[n] = cs;  // append checksum

    printf("%s\n", checksum(w, n + 1, 1) == 0xFFFF ? "No Error." : "Error Detected.");
}
```

ClassfulIPAddressing.c:

```
#include <stdio.h>

int main() {
    int a, b, c, d;
    printf("Enter IP address (format: a.b.c.d): ");
    scanf("%d.%d.%d.%d", &a, &b, &c, &d);

    if (a >= 0 && a <= 255) {
        if (a >= 0 && a <= 127) {
            printf("Class A\n");
```

```
            printf("Network ID: %d.0.0.0\n", a);
            printf("Host ID: 0.%d.%d.%d\n", b, c, d);
        } else if (a >= 128 && a <= 191) {
            printf("Class B\n");
            printf("Network ID: %d.%d.0.0\n", a, b);
            printf("Host ID: 0.0.%d.%d\n", c, d);
        } else if (a >= 192 && a <= 223) {
            printf("Class C\n");
            printf("Network ID: %d.%d.%d.0\n", a, b, c);
            printf("Host ID: 0.0.0.%d\n", d);
        } else if (a >= 224 && a <= 239) {
            printf("Class D (Multicast)\n");
            printf("Multicast Address: %d.%d.%d.%d\n", a, b, c, d);
        } else {
            printf("Class E (Reserved)\n");
            printf("Reserved Address: %d.%d.%d.%d\n", a, b, c, d);
        }
    } else {
        printf("Invalid IP address\n");
    }

    return 0;
}
```

Short code (optional):

```
#include <stdio.h>

int main() {
    int a, b, c, d;
```

```
    scanf("%d.%d.%d.%d", &a, &b, &c, &d);

    if (a < 0 || a > 255) return printf("Invalid IP\n");

    if (a <= 127) {
        printf("Class A\nNetwork: %d.0.0.0\nHost: 0.%d.%d.%d\n", a, b, c, d);
    } else if (a <= 191) {
        printf("Class B\nNetwork: %d.%d.0.0\nHost: 0.0.%d.%d\n", a, b, c, d);
    } else if (a <= 223) {
        printf("Class C\nNetwork: %d.%d.%d.0\nHost: 0.0.0.%d\n", a, b, c, d);
    } else if (a <= 239) {
        printf("Class D (Multicast)\n");
    } else {
        printf("Class E (Reserved)\n");
    }
}
```

ClasslessIPAddressing.c:

```
#include <stdio.h>

int main() {
    int a, b, c, d, prefix;
    unsigned int ip, mask, network, broadcast, firstHost, lastHost;

    printf("Enter IP address and prefix (e.g., 192.168.1.1 24): ");
    scanf("%d.%d.%d.%d %d", &a, &b, &c, &d, &prefix);

    if (prefix < 0 || prefix > 32) {
```

```
    printf("Invalid prefix\n");
    return 1;
}

ip = (a << 24) | (b << 16) | (c << 8) | d;
mask = prefix == 0 ? 0 : (~0U << (32 - prefix));
network = ip & mask;
broadcast = ip | ~mask;

if (prefix == 32) {
    firstHost = lastHost = ip;
} else if (prefix == 31) {
    firstHost = network;
    lastHost = broadcast;
} else {
    firstHost = network + 1;
    lastHost = broadcast - 1;
}

printf("Subnet Mask: %u.%u.%u.%u\n",
       (mask >> 24) & 0xFF, (mask >> 16) & 0xFF,
       (mask >> 8) & 0xFF, mask & 0xFF);

printf("First Host: %u.%u.%u.%u\n",
       (firstHost >> 24) & 0xFF, (firstHost >> 16) & 0xFF,
       (firstHost >> 8) & 0xFF, firstHost & 0xFF);

printf("Last Host: %u.%u.%u.%u\n",
       (lastHost >> 24) & 0xFF, (lastHost >> 16) & 0xFF,
```

```
        (lastHost >> 8) & 0xFF, lastHost & 0xFF);

    return 0;
}
```

Short code (optional):

```
#include <stdio.h>

int main() {
    int a,b,c,d,p;
    unsigned ip, mask;

    scanf("%d.%d.%d.%d %d", &a,&b,&c,&d,&p);
    ip = (a<<24)|(b<<16)|(c<<8)|d;

    mask = p==0 ? 0 : (~0U << (32-p));
    unsigned net = ip & mask;
    unsigned bc = ip | ~mask;

    printf("Mask: %u.%u.%u.%u\n",
        mask>>24, mask>>16 & 255, mask>>8 & 255, mask & 255);

    printf("Network: %u.%u.%u.%u\n",
        net>>24, net>>16 & 255, net>>8 & 255, net & 255);

    printf("Broadcast: %u.%u.%u.%u\n",
        bc>>24, bc>>16 & 255, bc>>8 & 255, bc & 255);
}
```

Viva Questions:

1. What is bit stuffing?

Bit stuffing inserts a **0 after every sequence of five consecutive 1s** to make sure the flag 01111110 never appears accidentally inside data.

2. Why specifically after *five* 1s?

Because the HDLC flag is 01111110.
This contains **six 1s in a row** inside its core, so stuffing after 5 prevents the receiver from ever seeing 6 consecutive 1s.

3. What happens when the receiver sees 5 consecutive 1s?

It **discards the next bit** (if it is 0). That 0 is known to be stuffed padding.

4. What if the data itself contains 6 or 7 consecutive 1s?

Stuffing continues repeatedly.
Example:
Data = 111111 (6 ones)
Process:

- After first 5 → insert 0
- Then count rest → if again 5 → insert 0
  Stuffing handles long runs automatically.

5. Does bit stuffing increase frame size?

Yes, but only when long runs of 1s appear.
Worst case expansion = **~1 bit for every 5 bits**.

6. Where is bit stuffing used?

Bit-oriented protocols: **HDLC, PPP, LAPB**, etc.

7. What is the flag used?

01111110 (binary) → 0x7E (hex).

---

1. What is byte stuffing?

If data contains **FLAG or ESC**, prefix with ESC so receiver knows it's data, not control.

2. Why do we need ESC?

ESC tells the receiver:
“Next byte is not a control byte; treat as data.”

3. What happens if data is @@@$?

Stuffed becomes:
$ @ @ @ @ $

4. What if FLAG is $ and data contains $?

Stuffed: @$
Receiver sees @ → reads next char as data.

5. Difference between bit stuffing and byte stuffing?

- Bit stuffing: operates on bits, used in bit-oriented protocols.
- Byte stuffing: operates on bytes/characters, used in **character-oriented protocols**.

---

1. Why framing is needed?

To decide **where a message begins and ends** inside a continuous stream.

2. What is fixed-size framing?

Divide message into equal-sized chunks (e.g., 5 bytes per frame).
No boundary markers needed.

3. Disadvantage of fixed-size framing?

Last frame may need **padding**, causing overhead.

4. What is variable-size framing?

Each frame has different length; receiver must know frame length using:

- Length field
- Special delimiter
- Escape characters.

5. Which is more efficient?

Variable-size framing (no padding), but requires parsing.

---

1. What is the purpose of CRC?

To detect errors using **polynomial division**.

2. Mathematical meaning of generator polynomial?

If G(x) is degree r, then append **r zeros** to data for division.

3. Important CRC formula

Transmitted frame:

T(x) = M(x) * x^r + R(x)

Where R(x) = remainder of dividing M(x)×x^r by G(x).

4. What happens at the receiver?

Receiver divides T(x) by G(x).
If remainder = 0 → **No error**.
Else → **Error detected**.

5. Why XOR is used?

Because subtraction in GF(2) = XOR.

6. What types of errors CRC can detect?

- All **single-bit** errors
- All **double-bit** errors
- All **odd number of bit errors**
- All **burst errors ≤ r bits**

7. What is burst error?

A sequence of corrupted bits, not necessarily consecutive flips.

8. If generator is 1101 (degree 3), how many zeros appended?

r = 3 → append **3 zeros**.

9. What is the remainder size?

Always **r bits** for a generator of degree r.

---

1. Why sliding window protocol?

To send multiple frames before waiting for ACK → improves throughput.

2. Define window size.

Number of frames allowed “in flight” (unacknowledged).

3. Go-Back-N retransmission rule

If frame k lost, resend frames **k, k+1, k+2, ..., lastSent**.

4. Selective Repeat retransmission rule

Only resend **that particular lost frame**.

5. Formula: Maximum window size for Selective Repeat?

W ≤ (Sequence Space Size / 2)

If sequence numbers = 0..N,
W ≤ (N+1)/2.

6. Why SR has that constraint?

To avoid ambiguity between old and new frames after wrap-around.

7. Which protocol has better efficiency?

Selective Repeat → fewer retransmissions.

8. Why GBN is simpler?

Only one timer and no out-of-order buffers.

9. What if ACK for frame 0 is lost?

Sender waits → timeout → retransmits starting from 0.

---

1. Update formula for DVR

D[i][j] = min(D[i][k] + D[k][j]) for all neighbors k

2. What is hop count?

Next router on shortest path.

3. What is count-to-infinity problem?

When routers take many iterations to learn a link is broken, continuously increasing distance metrics.

4. Remedies?

- Split horizon
- Poison reverse
- Triggered updates.

5. Complexity of Bellman-Ford?

$O(V^3)$ for full table updates.

6. Who uses DVR?

RIP (Routing Information Protocol).

---

1. Why link-state routing?

Fast convergence, less prone to count-to-infinity errors.

2. Algorithm used?

Dijkstra's shortest path algorithm.

3. Key idea

Pick the **nearest unvisited** node, relax its edges.

4. Complexity

With adjacency matrix: $O(V^2)$
With priority queue: $O(E \log V)$

5. Requirements for Dijkstra

All edge weights must be **non-negative**.

6. Who uses link-state routing?

OSPF, IS-IS.

---

1. Formula for Caesar Cipher?

Encryption:

$C = (P + K) \bmod 26$

Decryption:

$P = (C - K + 26) \bmod 26$

2. What is K in Caesar cipher?

Shift key (here K = 3).

3. What about uppercase vs lowercase?

Treat separately, because ASCII codes differ.

4. Is Caesar cipher secure?

No — only 26 possible keys → brute-force in seconds.

---

1. Purpose of leaky bucket algorithm?

To shape/tame traffic rate — smooth out bursts.

2. Output rate meaning?

Maximum **drain rate** — number of bytes sent per unit time.

3. What happens when bucket overflows?

Packets are **discarded**.

4. Difference: Token Bucket vs Leaky Bucket?

- **Token Bucket** allows bursts.
- **Leaky Bucket** forces a steady rate.

5. Can leaky bucket reorder packets?

No — first in, first out.

---

1. What is Internet checksum?

1's complement **sum** of 16-bit words, followed by 1's complement of the result.

2. Why use end-around carry?

To ensure result fits in 16 bits while preserving error detection ability.

3. Verification rule

If the receiver adds all words including checksum and gets **0xFFFF** → no error.

4. Formula

Checksum = ~(Sum_of_all_16bit_words)

5. Why 1's complement instead of 2's complement?

1's complement allows unidirectional error detection → sum of all words + checksum should be all 1s.

6. What type of errors it detects?

- Most single-bit errors
- Many multi-bit errors

- Burst errors up to limited sizes
  (Not as strong as CRC)

---

1. What is classful addressing?

A fixed-block method where the first octet decides the class of the IP.

2. What are the ranges of Class A/B/C/D/E?

- A: 0–127
- B: 128–191
- C: 192–223
- D: 224–239
- E: 240–255

3. Why was Classful addressing replaced?

It wasted IPs.
A single Class A network gives **16 million** IPs — too large for most users.

4. Does Class D have Network and Host IDs?

No — Class D is for **multicast**, not unicast communication.

5. Does Class E get used in real networks?

No — reserved for research.

6. Why is 127.x.x.x special?

It is **loopback / localhost** range.
Example: 127.0.0.1

7. How many hosts possible?

- Class A: $2^{24} - 2$
- Class B: $2^{16} - 2$
- Class C: $2^{8} - 2$

Subtract 2 for network & broadcast.

---

1. What is CIDR?

Classless Inter-Domain Routing — allows variable-length subnet masks.

2. Why do we use CIDR?

To save IP addresses and aggregate routes.

3. How to calculate subnet mask from prefix?

Subnet mask has **prefix** number of ones.
Example: /26 → 26 ones followed by 6 zeros.

4. How to find number of hosts in a subnet?

Formula:

Hosts = 2^(32 – prefix) – 2

Example: /24
Hosts = 2^8 – 2 = 254.

5. Exceptions?

- /31 → 2 usable hosts (point-to-point).
- /32 → 1 host (no network/broadcast).

6. Difference between Network Address and Broadcast Address?

- Network: first address, identifies the subnet.
- Broadcast: last address, message to all hosts.

7. Example viva question

Q: Find network & broadcast of 10.0.5.17/20
Steps:
Host bits = 12 → mask = 255.255.240.0
Network = 10.0.0.0
Broadcast = 10.0.15.255

8. What is route aggregation?

Combining multiple networks into a single summarized route.

Example:
192.168.0.0/24
192.168.1.0/24
→ aggregated as
192.168.0.0/23

9. Why bitwise operations used in CIDR calculation?

Because subnet masks and IP calculations are **bit-level** processes.

==========

---

1. Difference between Error Detection & Correction?

- Detection: Identify presence of error (CRC, checksum, parity).
- Correction: Recover original message (Hamming Code).

2. Why do we add padding in fixed-size framing?

To fill last frame if data is not divisible by frame size.

3. Why framing is needed even in TCP?

TCP is a stream-oriented protocol → application must define message boundaries.

4. What is maximum theoretical expansion in byte stuffing?

Worst case:
Every byte is FLAG or ESC → stuffed message becomes **2× original + 2 flags**.

5. Why routing algorithms need INF?

To denote unreachable nodes and prevent accidental overflow when adding costs.

6. What if CRC generator = 1001?

Degree = 3 → append **3 zeros**.
Receiver also expects remainder of **3 bits**.