JC BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY YMCA

COMPUTER NETWORKS

LAB FILE

VIKAS CHHONKAR

19001015065

B.Tech - Electronics and Computer Engineering

(Sem - 6)

INDEX

EXPERIMENT - 1

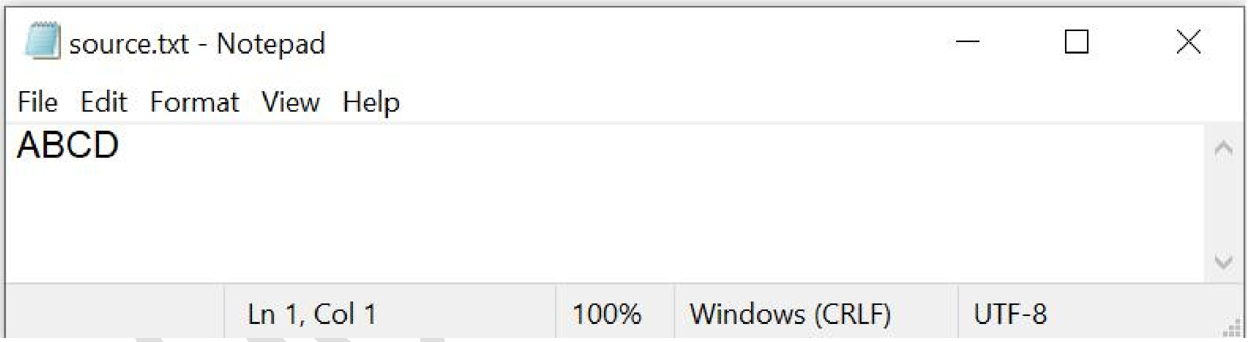
AIM: To implement the Diffie-Hellman Algorithm and encrypt a file on the system using the Secret Key

CODE:

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

void main(){
    char ch;
    //q is prime number and p is primitive root
    int p=5,q=7;

    //xa and xb are private keys
    int xa=4,xb=3;

    FILE *ps, *pd ;

    //x and y are generated keys
    int ya, yb;
    ya = (int)pow(p,xa)%q;
```

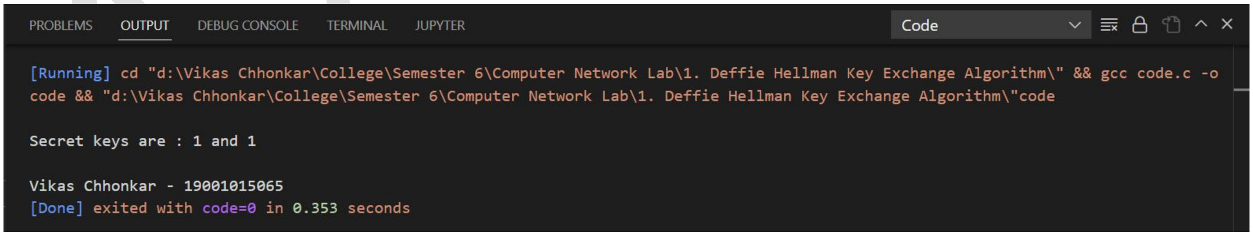
```
        yb = (int)pow(p,xb)%q;

        //  Secret key calculation
        // ka1 and ka2 are secret keys
        int ka1 = (int)pow(yb,xa)%q;
        int ka2 = (int)pow(ya,xb)%q;

        printf("\nSecret keys are : %d and %d",ka1,ka2);

        ps=fopen("D:\\Vikas Chhonkar\\College\\Semester
6\\Computer Network Lab\\1. Deffie Hellman Key Exchange
Algorithm\\source.txt","r");
        pd=fopen("D:\\Vikas Chhonkar\\College\\Semester
6\\Computer Network Lab\\1. Deffie Hellman Key Exchange
Algorithm\\destination.txt","a");

        if(ps==NULL){
            printf("Source File not Found");
            return ;
        }

        if(pd==NULL){
            printf("Destination File not Found");
            return;
        }

        while(1){
            ch=getc(ps);
            if(ch==EOF){
                break;
            }
            putc(ch+ka1,pd);
        }

        printf("\n\nVikas Chhonkar - 19001015065");
        fclose(ps);
        fclose(pd);
    }
```

OUTPUT:



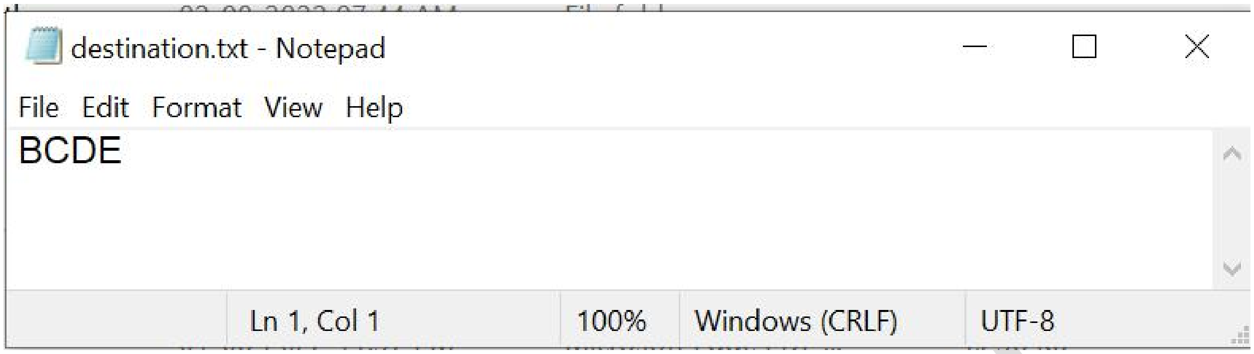
```
ABCD
```

Source File



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\1. Deffie Hellman Key Exchange Algorithm\" && gcc code.c -o
code && "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\1. Deffie Hellman Key Exchange Algorithm\"code

Secret keys are : 1 and 1

Vikas Chhonkar - 19001015065
[Done] exited with code=0 in 0.353 seconds
```

Console Output



```
BCDE
```

Destination File

RESULT:

Hence implemented a program to use the Diffie-Hellman Algorithm and encrypt a system file using the generated Secret Key.

EXPERIMENT - 2

AIM: To implement LRC, VRC, Block parity in 2-D data matrix

CODE:

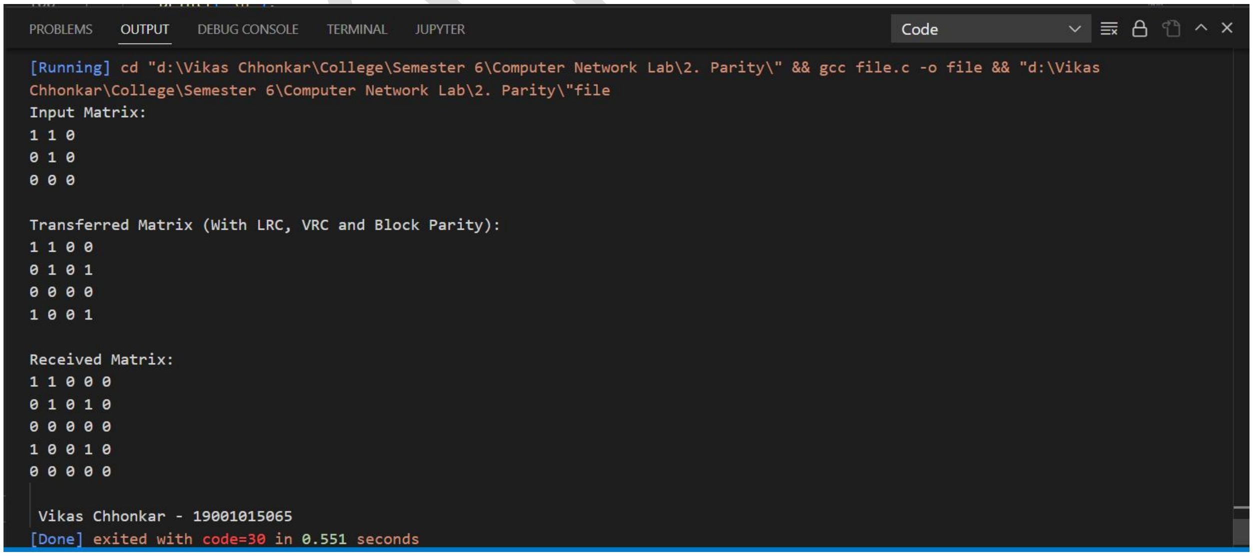
```
#include<stdio.h>
#include<stdlib.h>

int xor1(int, int);

void main(){
    int p, i, j, d[5][5];

    //Input Matrix
    for(i=0; i<3; i++){
        for(j=0; j<3; j++){
            d[i][j] = rand()%2;
        }
    }

    // Lateral Redundancy Check (LRC)
    for(i=0; i<3; i++){
        p=0;
        for(j=0; j<3; j++){
            p=xor1(p,d[i][j]);
        }
        d[i][3] = p;
    }

    // Vertical Redundancy Check (VRC)
    for(j=0; j<3; j++){
        p=0;
        for(i=0; i<3; i++){
            p = xor1(p,d[i][j]);
        }
        d[3][j] = p;
    }

    // Block Parity
    p=0;
    for(i=0; i<3; i++){
        for(j=0; j<3; j++){
            p=xor1(p, d[i][j]);
        }
    }
    d[3][3] = p;
```

```
        // Print 3x3 matrix
        printf("Input Matrix:\n");
        for(i=0; i<3; i++){
            for(j=0; j<3; j++){
                printf("%d ",d[i][j]);
            }
            printf("\n");
        }
        printf("\n");

        // Print 4x4 matrix
        printf("Transferred Matrix (With LRC, VRC and Block Parity):\n");
        for(i=0; i<4; i++){
            for(j=0; j<4; j++){
                printf("%d ",d[i][j]);
            }
            printf("\n");
        }
        printf("\n");

        // At receiver side
        // Checking
        // d[1][1] = 1 - d[1][1];
```

```
for(i=0; i<4; i++){
    p=0;
    for(j=0; j<4; j++){
        p=xor1(p,d[i][j]);
    }
    d[i][4] = p;
}

// VRC
for(j=0; j<4; j++){
    p=0;
    for(i=0; i<4; i++){
        p = xor1(p,d[i][j]);
    }
    d[4][j] = p;
}

// Block Parity
p=0;
for(i=0; i<4; i++){
    for(j-0; j<4; j++){
        p=xor1(p, d[i][j]);
    }
}
d[4][4] = p;
```

```
        printf("Received Matrix: \n");
        for(i=0; i<5; i++){
            for(j=0; j<5; j++){
                printf("%d ", d[i][j]);
            }
            printf("\n");
        }


    }


    int xor1(int a, int b){
        if (a==b) return 0;
        else return 1;
    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\2. Parity\" && gcc file.c -o file && "d:\Vikas
Chhonkar\College\Semester 6\Computer Network Lab\2. Parity\"file
Input Matrix:
1 1 0
0 1 0
0 0 0

Transferred Matrix (With LRC, VRC and Block Parity):
1 1 0 0
0 1 0 1
0 0 0 0
1 0 0 1

Received Matrix:
1 1 0 0 0
0 1 0 1 0
0 0 0 0 0
1 0 0 1 0
0 0 0 0 0

 Vikas Chhonkar - 19001015065
[Done] exited with code=30 in 0.551 seconds
```

RESULT:

Hence implemented LRC, VRC, Block Parity to transmit data and error detection at receiving side.

EXPERIMENT - 3

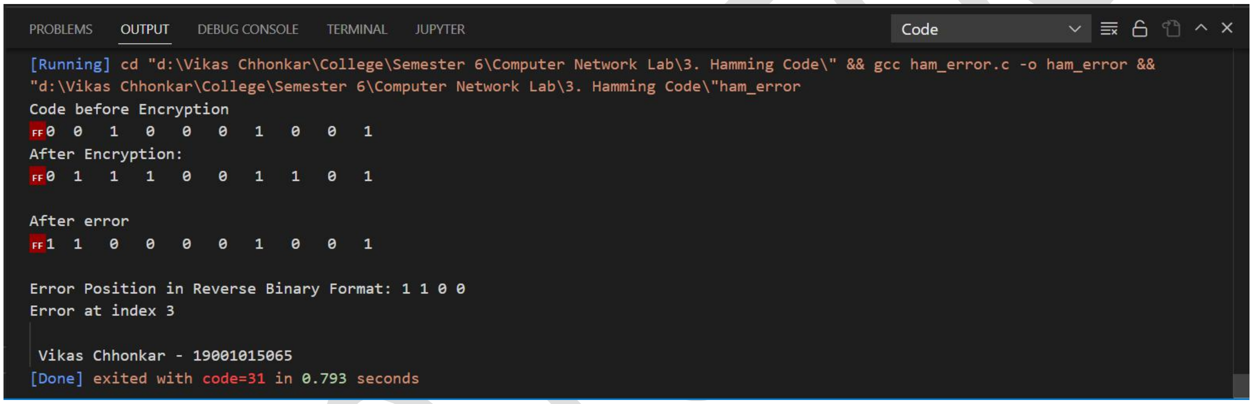
AIM: To implement Hamming Code and error detection

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>

int data[6]={1,0,0,1,0,1};
int b[5];
int code[11];
int n,k;

int check_power_2 (int m){
    int i,r=0;
    for(i=0;i<k;i++){
        if(pow(2,i)==m) r=1;
    }
    return r;
}

int xor1(int a, int b){
    if(a==b) return 0;
    else return 1;
}

void disp(){
    int i;
    printf("\f");
    for(i=1;i<=n+k;i++){
        printf("%d\t", code[i]);
    }
    printf("\n");
}

void convert(int d, int b[5]){
    for(int i=0;i<k;i++){
        b[i]=d%2;
        d=d/2;
    }
}

void main()
{
    int ndx=0,i,j,pos,p;
    n=6; k=1;
    while(pow(2,k)<n+k+1) k++;
    for(i=1;i<=n+k;i++){
        if(check_power_2(i)==1) code[i]=0;
```

```
            else code[i]=data[ndx++];
        }
        printf("Code before Encryption\n");
        disp();

        for(i=0;i<k;i++){
            int stepSize, start;
            stepSize=(int)pow(2,i);
            start=stepSize;
            p=0;
            while(start<=n+k){
                     for(j=start; (j<start+stepSize) &&
j<=n+k ;j++){
                    p=xor1(p,code[j]);
                }
                start=start+2*stepSize;
            }
            code[stepSize]=p;
        }
        printf("After Encryption: \n");
        disp();

        printf("\n");
        printf("After error\n");

        //Error
        code[3]= 1- code[3];
        for(i=0;i<k;i++){
            int stepSize, start;
            stepSize=(int)pow(2,i);
            start=stepSize;
            p=0;
            while(start<=n+k){
                    for(j=start; (j<start+stepSize) && j<=n+k ;j++){
                    p=xor1(p,code[j]);
                }
                start=start+2*stepSize;
            }
            code[stepSize]=p;
        }
        disp();

        int cnt = 0;
        int val = 0;



        printf("\nError Position in Reverse Binary Format: ");
        for(int i=0; i<n+k; i++){
            if(check_power_2(i)==1){
                val += code[i]*pow(2,cnt);
                printf("%d ",code[i]);
                cnt++;
            }
        }
        printf("\n");
        printf("Error at index %d", val);
        printf("\n\n Vikas Chhonkar - 19001015065");
    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\3. Hamming Code\" && gcc ham_error.c -o ham_error &&
"d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\3. Hamming Code\"ham_error
Code before Encryption
FF0  0   1   0   0   0   1   0   0   1
After Encryption:
FF0  1   1   1   0   0   1   1   0   1

After error
FF1  1   0   0   0   0   1   0   0   1

Error Position in Reverse Binary Format: 1 1 0 0
Error at index 3

 Vikas Chhonkar - 19001015065
[Done] exited with code=31 in 0.793 seconds
```

RESULT:

Hence implemented hamming code and its error detection.

EXPERIMENT - 4

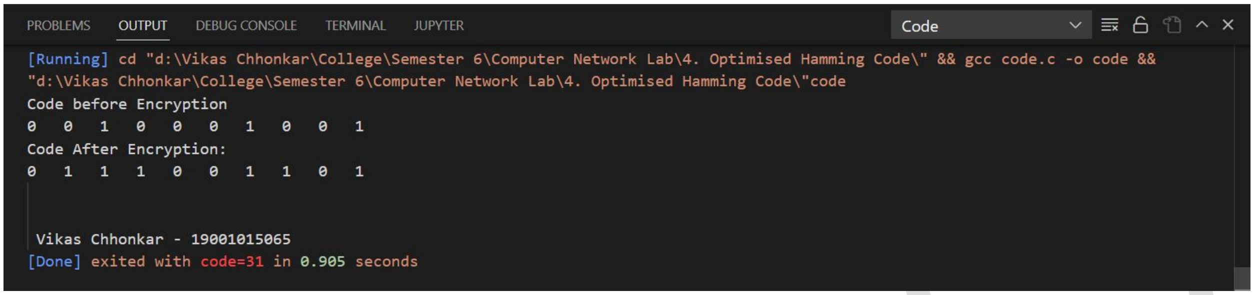
AIM: To implement Hamming Code and error detection (Optimised Version)

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>

int data[6]={1,0,0,1,0,1};
int b[5];
int code[11];
int n,k;

int check_power_2 (int m){
    int i,r=0;
    for(i=0;i<k;i++){
        if(pow(2,i)==m) r=1;
    }
    return r;
}

int xor1(int a, int b){
    if(a==b) return 0;
    else return 1;
}

void disp(){
    int i;
    // printf("\f");
    for(i=1;i<=n+k;i++){
        printf("%d\t", code[i]);
    }
    printf("\n");
}

void convert(int d, int b[5]){
    for(int i=0;i<k;i++){
        b[i]=d%2;
        d=d/2;
    }
}

void main()
{
    int ndx=0,i,j,pos,p;
    n=6; k=1;
    while(pow(2,k)<n+k+1) k++;
    for(i=1;i<=n+k;i++){
        if(check_power_2(i)==1) code[i]=0;
```

```
            else code[i]=data[ndx++];
        }
        printf("Code before Encryption\n");
        disp();

        for(i=0;i<k;i++){
            int stepSize, start;
            stepSize=(int)pow(2,i);
            start=stepSize;
            p=0;
            while(start<=n+k){
                    for(j=start; (j<start+stepSize) && j<=n+k ;j++){
                    p=xor1(p,code[j]);
                }
                start=start+2*stepSize;
            }
            code[stepSize]=p;
        }
        printf("Code After Encryption: \n");
        disp();

        printf("\n\n Vikas Chhonkar - 19001015065");



    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\4. Optimised Hamming Code\" && gcc code.c -o code &&
"d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\4. Optimised Hamming Code\"code
Code before Encryption
0   0   1   0   0   0   1   0   0   1
Code After Encryption:
0   1   1   1   0   0   1   1   0   1


 Vikas Chhonkar - 19001015065
[Done] exited with code=31 in 0.905 seconds
```

RESULT:

Hence implemented the optimised code for Hamming Code and its Error Detection.

EXPERIMENT - 5

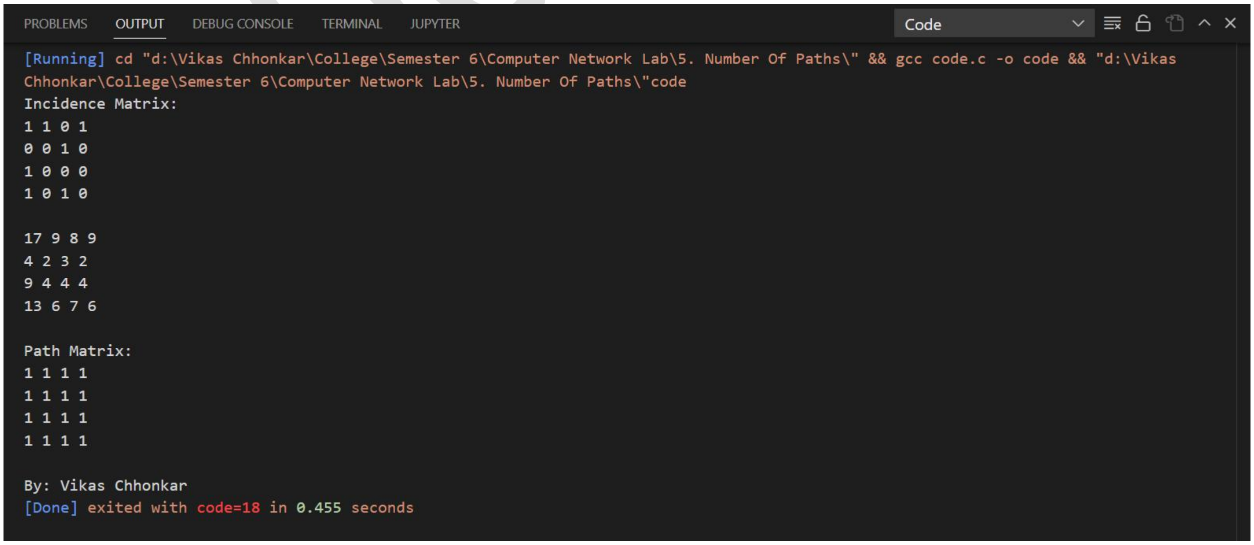
AIM: To find the number of paths for each pair of vertices (u, v) in a directed graph using matrix exponentiation and matrix addition

CODE:

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


// Function for Displaying Matrix
void disp(int a[4][4]){
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}

// Function for Matrix Multiplication
void matmul(int a[4][4], int b[4][4], int c[4][4]){
    int temp[4][4];
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            temp[i][j] = 0;
            for(int k = 0; k < 4; k++){
                temp[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            c[i][j] = temp[i][j];
        }
    }
}

// Function for Matrix Addition
void matadd(int a[4][4], int b[4][4], int c[4][4]){
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
```

```
// Main Function
void main(){
    int A[4][4] = {1,1,0,1,0,0,1,0,1,0,0,0,1,0,1,0};
    int I[4][4] = {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};

    int C[4][4], P[4][4];

    printf("Incidence Matrix: \n");
    disp(A);
    printf("\n");

    for(int i=0; i<4; i++){
        for(int j=0; j<4; j++){
            C[i][j] = 0;
        }
    }

    for(int i=0; i<4; i++){
        matmul(I,A,I);
        matadd(C,I,C);
    }

    disp(C);
    printf("\n");

    for(int i=0; i<4; i++){
        for(int j=0; j<4; j++){
            if(C[i][j]==0){
                P[i][j] = 0;
            }
            else{
                P[i][j] = 1;
            }
        }
    }

    printf("Path Matrix: \n");
    disp(P);
    printf("\n");

    printf("By: Vikas Chhonkar");
}
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\5. Number Of Paths\" && gcc code.c -o code && "d:\Vikas
Chhonkar\College\Semester 6\Computer Network Lab\5. Number Of Paths\"code
Incidence Matrix:
1 1 0 1
0 0 1 0
1 0 0 0
1 0 1 0

17 9 8 9
4 2 3 2
9 4 4 4
13 6 7 6

Path Matrix:
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

By: Vikas Chhonkar
[Done] exited with code=18 in 0.455 seconds
```

RESULT:

Hence implemented a program to find the number of paths for each pair of vertices (u, v) in a directed graph using matrix exponentiation and matrix addition where element at (i, j) in C matrix represents the number of paths between the pair of vertex i and vertex j.

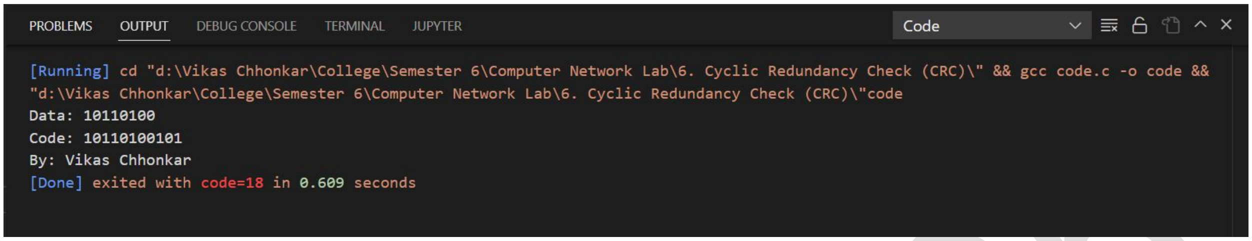
EXPERIMENT - 6

AIM: To implement Cyclic Redundancy Check

CODE:

```
#include<stdio.h>

int xor1(int a, int b){
    if(a==b) return 0;
    return 1;
}

//Main Function
void main(){
    int data[8] = {1, 0, 1, 1, 0, 1, 0, 0};
    int div[4] = {1, 1, 0, 1};
    int code[11];

    for(int i=0; i<8; i++) {
        code[i] = data[i];
    }
    for(int i=8; i<11; i++) {
        code[i] = 0;
    }

    for(int i=0; i<8; i++){
        int m = code[i];
        for(int j=0; j<4; j++){
            code[i+j] = xor1(code[i+j], m*div[j]);
        }
    }

    printf("Data: ");
    for(int i=0; i<8; i++) {
        printf("%d", data[i]);
    }
    printf("\n");

    printf("Code: ");
    for(int i=0; i<8; i++) {
        printf("%d", data[i]);
    }
    for(int i=8; i<11; i++){
        printf("%d", code[i]);
    }
    printf("\n");

    printf("By: Vikas Chhonkar");
}
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\6. Cyclic Redundancy Check (CRC)\" && gcc code.c -o code &&
"d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\6. Cyclic Redundancy Check (CRC)\"code
Data: 10110100
Code: 10110100101
By: Vikas Chhonkar
[Done] exited with code=18 in 0.609 seconds
```

RESULT:

Hence implemented Cyclic Redundancy Check

EXPERIMENT - 7

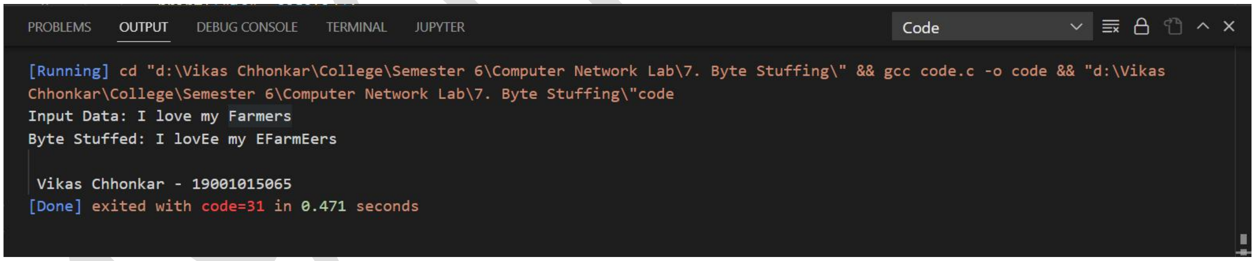
AIM: To implement Byte Stuffing

CODE:

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>


void main(){

    char data[] = "I love my Farmers";

    char code[50];

    int i=0;

    int j = 0;

    while(i<17){

        char temp = data[i];

         if(temp=='E'||temp=='F'||temp=='e'||temp=='f'){

            code[j++] = 'E';

        }

        code[j++]=temp;

        i++;

    }
```

```
        printf("Input Data: ");
        for(i=0; i<17; i++){
            printf("%c", data[i]);
        }
        printf("\nByte Stuffed: ");
        for(int i=0; i<20; i++){
            printf("%c", code[i]);
        }

        printf("\n\n Vikas Chhonkar - 19001015065");
    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\7. Byte Stuffing\" && gcc code.c -o code && "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\7. Byte Stuffing\"code
Input Data: I love my Farmers
Byte Stuffed: I lovEe my EFarmEers

 Vikas Chhonkar - 19001015065
[Done] exited with code=31 in 0.471 seconds
```

RESULT:

Hence implemented byte stuffing using Byte character ‘E’

EXPERIMENT - 8

AIM: To implement Bit Stuffing

CODE:

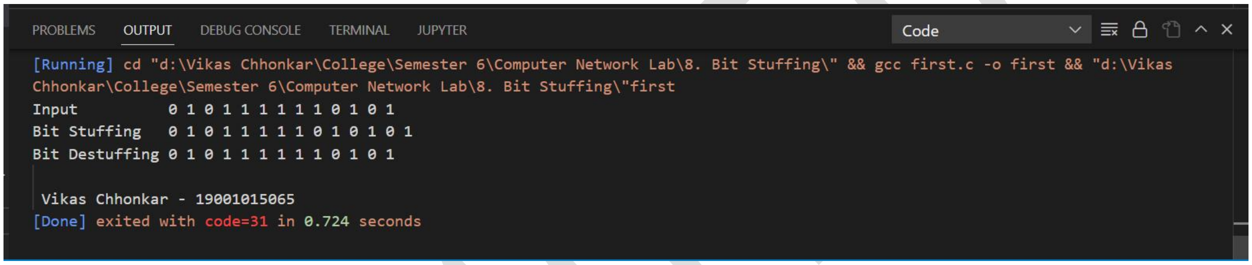
```
#include<stdio.h>

int test1(int arr[]){
    int i=0;
    if(arr[0]==1) return 0;
    for(int i=1; i<=5; i++){
        if(arr[i]==0) return 0;
    }
    return 1;
}

int test2(int arr[]){
    int i=0;
    if(arr[0]==1) return 0;
    for(int i=1; i<=5; i++){
        if(arr[i]==0) return 0;
    }
    return 1;
```

```
}


void main(){
    int data[] = {0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1};
    int code[14];
    int n = sizeof(data)/sizeof(data[0]);
    int k = 0;
    int i=0;
    int j;
    while(i<=n-6){
        if(test1(&data[i])==1){
            for(int j=i; j<=i+5; j++){
                code[k++] = data[j];
            }
            code[k++] = 0;
            i+=6;
        } else code[k++] = data[i++];
    }
    for(j=i; j<=n-1; j++){
        code[k++] = data[j];
    }
    printf("Input          ");
    for(int i=0; i<n; i++){
        printf("%d ", data[i]);
    }
```

```
printf("\n");
printf("Bit Stuffing   ");
for(int i=0; i<k; i++){
    printf("%d ", code[i]);
}

printf("\n");

//Bit Destuffing
int ans[50];
n = sizeof(code)/sizeof(code[0]);
k = 0;
i = 0;
while(i<=n-7){
    if(test2(&code[i])==1){
        for(int j=i; j<=i+5; j++){
            ans[k++] = code[j];
        }
        i+=7;
    } else ans[k++] = code[i++];
}
for(j=i; j<=n-1; j++){
    ans[k++] = code[j];
}

printf("Bit Destuffing ");
```

```
        for(int i=0; i<k; i++){
            printf("%d ", ans[i]);
        }

        printf("\n\n Vikas Chhonkar - 19001015065");
    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\8. Bit Stuffing\" && gcc first.c -o first && "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\8. Bit Stuffing\"first
Input          0 1 0 1 1 1 1 1 1 0 1 0 1
Bit Stuffing   0 1 0 1 1 1 1 1 0 1 0 1 0 1
Bit Destuffing 0 1 0 1 1 1 1 1 1 0 1 0 1

 Vikas Chhonkar - 19001015065
[Done] exited with code=31 in 0.724 seconds
```

RESULT:

Hence implemented bit stuffing in an array and then reverting it to original form

EXPERIMENT - 9

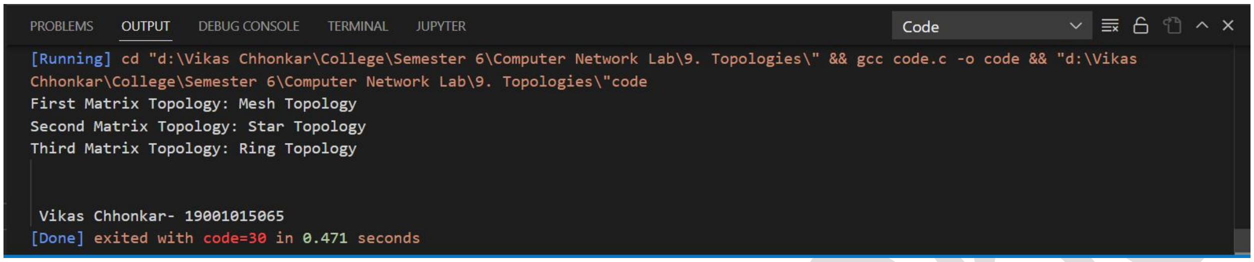
AIM: Write code to find Topology of given network

CODE:

```
#include <stdio.h>

void calcTopology(int mat[4][4], int n){
    int ones = 0;
    int twos = 0;
    int n_1 = 0;
    for(int i=0; i<n; i++){
        int temp = 0;
        for(int j=0; j<n; j++){
            temp = temp + mat[i][j];
        }
        if(temp ==1) ones++;
        else if(temp==2) twos++;
        else if(temp==(n-1)) n_1++;
    }

    if(ones==0 && twos==n && n_1 ==0)
```

```
        printf("Ring Topology\n");
    else if(ones==0 && twos==0 && n_1 ==n){
        printf("Mesh Topology\n");
    }
    else if(ones==n-1 && twos==0 && n_1 ==1){
        printf("Star Topology\n");
    }
    else{
        printf("No Standard Topology");
    }
}

void main(){
    //Mesh
    int mat1[4][4]={
        {0,1,1,1},
        {1,0,1,1},
        {1,1,0,1},
        {1,1,1,0}
    };

    //Star
    int mat2[4][4] = {
        {0,1,1,1},
        {1,0,0,0},
        {1,0,0,0},
        {1,0,0,0}
    };

    //Ring
    int mat3[4][4] = {
        {0,1,0,1},
        {1,0,1,0},
        {0,1,0,1},
        {1,0,1,0}
    };

    printf("First Matrix Topology: ");
    calcTopology(mat1, 4);

    printf("Second Matrix Topology: ");
    calcTopology(mat2, 4);

    printf("Third Matrix Topology: ");
    calcTopology(mat3, 4);
}
printf("\n\n Vikas Chhonkar- 19001015065");
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\9. Topologies\" && gcc code.c -o code && "d:\Vikas
Chhonkar\College\Semester 6\Computer Network Lab\9. Topologies\"code
First Matrix Topology: Mesh Topology
Second Matrix Topology: Star Topology
Third Matrix Topology: Ring Topology


 Vikas Chhonkar- 19001015065
[Done] exited with code=30 in 0.471 seconds
```

RESULT:

Hence implemented code to check the Topology of given Network.

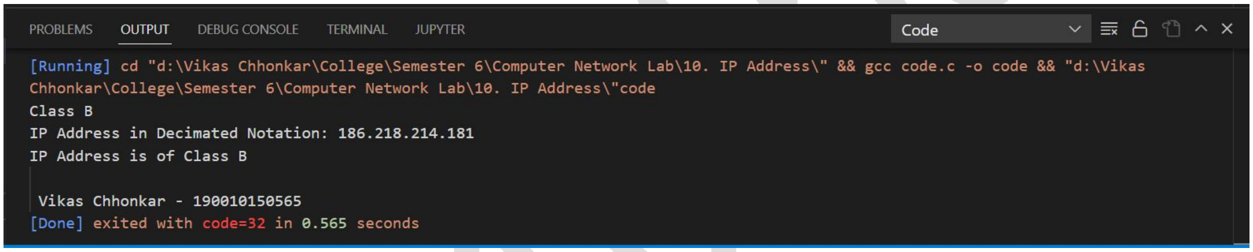
EXPERIMENT - 10

AIM: Write a program to find class of IP address

CODE:

```
    #include<stdio.h>

    #include<string.h>

    #include<math.h>


    int convert (int mat[]){

        int ans = 0;

        for(int i=7; i>=0; i--){

            ans = ans + mat[i]*pow(2,7-i);

        }

        return ans;

    }


    void main(){

        int                  mat[32]                          =
{1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,
1,0,1,0,1};

        if(mat[0]==0) printf("Class A");
        else{
            if(mat[1]==0) printf("Class B");
            else{
                if(mat[2]==0) printf("Class C");
                else{
                    if(mat[3]==0) printf("Class D");
                    else{
                        printf("Class E");
                    }
                }
            }
        }
        int a = convert(&mat[0]);
        int b = convert(&mat[8]);
        int c = convert(&mat[16]);
        int d = convert(&mat[24]);
        printf("\nIP        Address        in        Decimated
Notation: %d.%d.%d.%d", a, b, c, d);
        printf("\nIP Address is of Class ");
        if(a<128){
            printf("A");
        }
        else if(a<192){
            printf("B");
        }
        else if(a<224){
            printf("C");
        }
        else{
            printf("D");
        }

        printf("\n\n Vikas Chhonkar - 190010150565");
    }
```

OUTPUT:



```
[Running] cd "d:\Vikas Chhonkar\College\Semester 6\Computer Network Lab\10. IP Address\" && gcc code.c -o code && "d:\Vikas
Chhonkar\College\Semester 6\Computer Network Lab\10. IP Address\"code
Class B
IP Address in Decimated Notation: 186.218.214.181
IP Address is of Class B

 Vikas Chhonkar - 190010150565
[Done] exited with code=32 in 0.565 seconds
```

RESULT:

Hence implemented the code to Find the Class of IP Address.