DEPARTMENT OF ELECTRONICS AND COMMUNICATION

B.M.S COLLEGE OF ENGINEERING

(AUTONOMOUS COLLEGE UNDER VTU, BELAGAVI)

BANGALORE – 560019

2020-21

7TH SEMESTER LAB REPORT

ON

Data Communication Network Lab

Submitted in partial fulfillment for the partial completion of course

DATA COMMUNICATION NETWORK [16EC6DCDCN]

Submitted By

Akshay S Rao | 1BM17EC007

Course Instructor

B. HARSHITHA

Assistant professor, Dept. of ECE

Lab Report: Experiment 1

Problem Statement

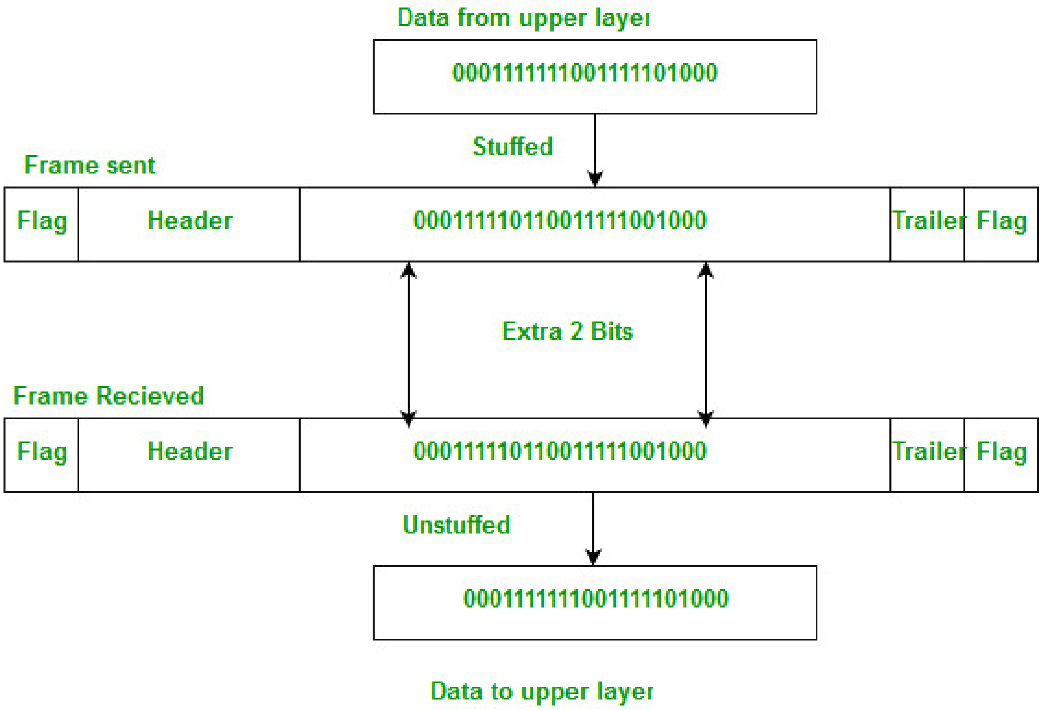
Write a program to demonstrate framing

Introduction

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Theory

Bit stuffing is the insertion of non information bits into data. Note that stuffed bits should not be confused with overhead bits. Overhead bits are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.)



Applications of Bit Stuffing

1. synchronize several channels before multiplexing
2. rate-match two single channels to each other
3. run length limited coding

Run length limited coding – To limit the number of consecutive bits of the same value(i.e., binary value) in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits.

Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e., not corrupted by transmission errors). It is merely a way to ensure that the transmission starts and ends at the correct places.

Disadvantages of Bit Stuffing –
The code rate is unpredictable; it depends on the data being transmitted

Example of bit stuffing –

Bit sequence: 110101111101011111101011111110 (without bit stuffing)

Bit sequence: 110101111100101111101010111110110 (with bit stuffing)

After 5 consecutive 1-bits, a 0-bit is stuffed.

Algorithm

- The input bits are stored inside a linked list.
- After 4 repeated bits of value 1 , a 0 bit is stuffed.

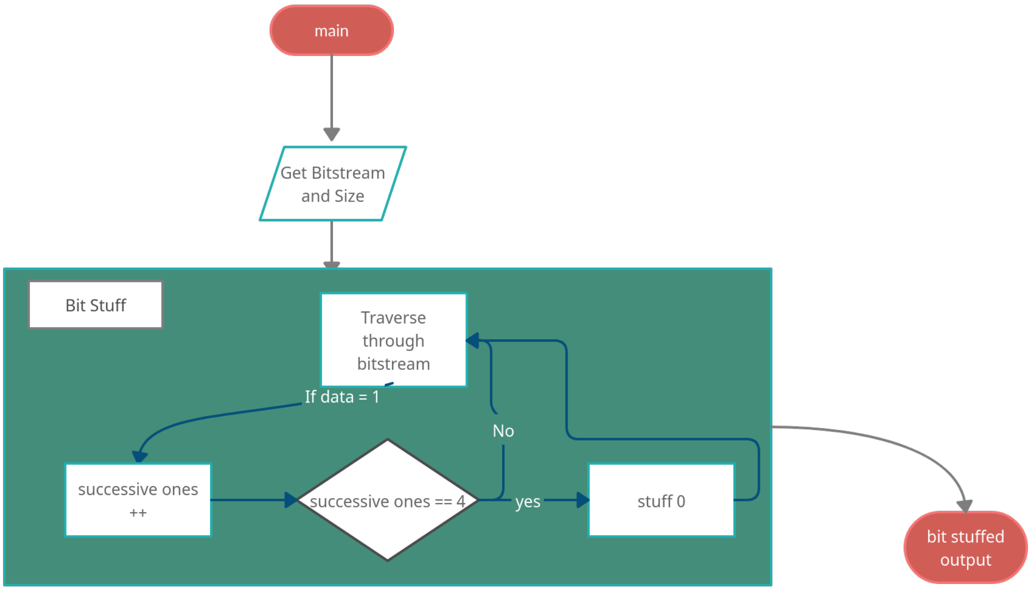
Code

```c
/*

//////////////////////////////////////////////////////////////////////
//Experiment 1 : Demonstration of Data Framing using bit stuffing//
//////////////////////////////////////////////////////////////////////

@author : Akshay S Rao
@usn    : 1BM17EC007
@Batch  : A1
@Date   : 12/10/2020

*/



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

// define the number of ones in the pattern
// after which bit stuffing to be done
#define MAX_REPEATED_BITS 4


// linked list

struct node
{
    char data;
    struct node* next;
};

/**********************************
@Function    : createNode
@input       : char
@returns     : node
@logic       : creates new node
**********************************/

struct node* createNode(char data){
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->next = NULL;
    newNode->data = data;
    return newNode;
}

/**********************************
@Function    : deleteNode
@input       : struct node*
@returns     : void
@logic       : deletes dynamically alloted nodes
**********************************/
void deleteNode(struct node* stream){

    struct node* temp = stream;
    while(temp!=NULL){
        struct node* current = temp;
        temp = temp->next;
        free(current);
    }

}
/**********************************
@Function    : printStream
@input       : struct node*
@returns     : void
@logic       : prints data in the bit stuffed stream
**********************************/
void printStream(struct node* stream){

    struct node* temp = stream;
    while(temp!=NULL){
        printf("%c\t",temp->data);
        temp = temp->next;
    }
    printf("\n");

```

```c
/***********************************
@Function   : bitStuff
@input      : a bitstream (char*), bitstream's size (int)
@returns    : bitstuffed data frame
@logic      : matches pattern and breaks pattern
***********************************/
struct node* bitStuff(char* bitStream, int streamSize){

    struct node* head = NULL;
    struct node* tail = NULL;

    for(int index = 0; index < streamSize; index++){
        if(bitStream[index] == '0'){
            if(head == NULL){
                head = createNode(bitStream[index]);
                tail = head;
            }
            else{

                tail->next = createNode(bitStream[index]);
                tail = tail->next;
            }
        }
        else{
            int cntRepeatedBits = 0;
            while((index < streamSize) && (bitStream[index] == '1')
                    && (cntRepeatedBits < MAX_REPEATED_BITS))
            {

                tail->next = createNode(bitStream[index]);
                tail = tail->next;
                cntRepeatedBits++;

                if(cntRepeatedBits == MAX_REPEATED_BITS){
                    tail->next = createNode('0');
                    tail = tail->next;

                    break;
                }
                index++;

            }
        }

    }
    return head;
```

```c
/***********************************
@Function   : main
@input      : nothing
@returns    : nothing
@logic      : takes input bitstream and bitstream size
***********************************/
int main(){

    int inputSize;
    printf("Please enter size of bit stream: ");
    scanf("%d",&inputSize);printf("\n");

    char* bitStream = (char*)malloc(sizeof(char)*inputSize);
    printf("Please enter bit stream:\n");

    int bufferSize = 0;
    while(bufferSize!=inputSize){
        //insert new bit in new line
        scanf("%s",&bitStream[bufferSize]);
        printf("\n");
        bufferSize++;
    }

    struct node* stuffedStream = bitStuff(bitStream, inputSize);
    printf("stuffedStream is ");
    printStream(stuffedStream);

    free(bitStream);
    deleteNode(stuffedStream);


return 0;
}
```

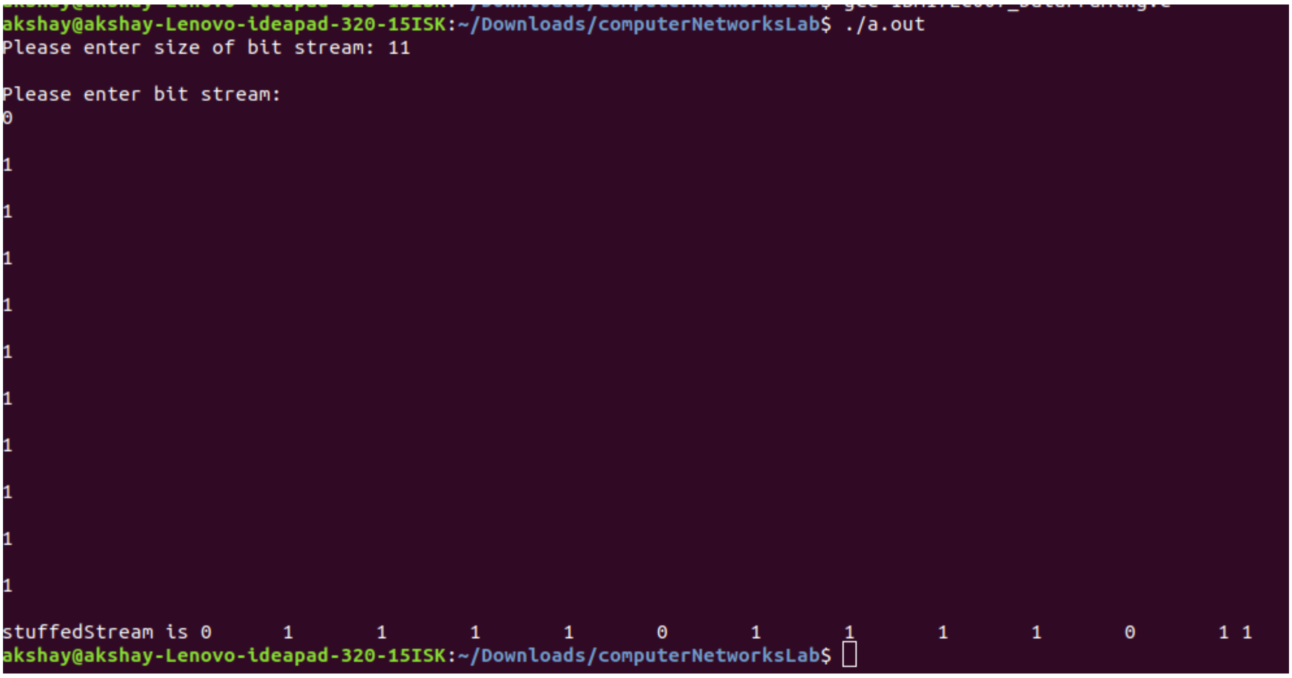
Flowchart



Result

A bit stream of size 11 is entered. The bit stream entered is 01111111111.

after stuffing the result is 01111**0**1111**0**11. As we can see after repetition of four 1's a 0 is stuffed



Lab Report: Experiment 2

Problem Statement

Write a program to generate CRC code for checking error.

Introduction

Cyclic codes are special linear block codes with one extra property. In cyclic code, if a codeword is cyclically shifted, the result is another codeword. For example if 1011000 is a codeword an we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a_0 to a_6 and the bits in the second word b_0 to b_6 we can shift the bits by using the following:

$$b_1 = a_0 \quad b_2 = a_1 \quad b_3 = a_2 \quad b_4 = a_3 \quad b_5 = a_4 \quad b_6 = a_5 \quad b_0 = a_6$$

In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

Theory and Algorithm

We can create cyclic codes to correct errors. We simply discuss a category of cyclic codes called the cyclic redundancy check (CRC) that is used in networks such as LANs and WANs.

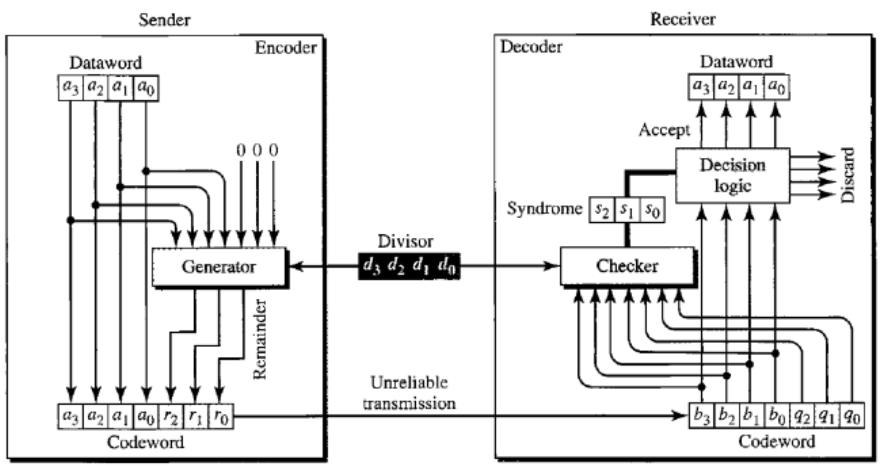
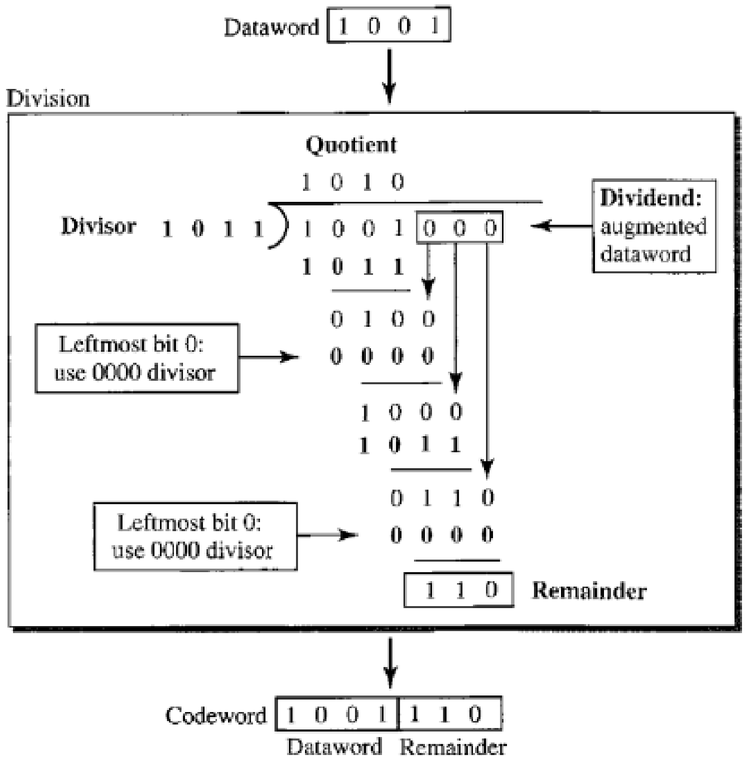


Figure 10.14 *CRC encoder and decoder*

Figure 10.15 *Division in CRC encoder*



In the encoder, the dataword has k bits (4 here) ; the codeword has n bits (7 here) . The size of the dataword is augmented by adding n - k(3 here) 0s to the right-hand side of the word. The n bit result is fed into the generator. The generator uses a divisor of size n-k+1(4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo 2 division). The quotient of the division is discarded; the remainder ($r_2r_1r_0$) is appended to the dataword to create the codeword.

Algorithm

- A string of n 0's is appended to the data unit to be transmitted.
- Here, n is one less than the number of bits in CRC generator.
- Binary division is performed of the resultant string with the CRC generator.
- After division, the remainder so obtained is called as CRC.
- It may be noted that CRC also consists of n bits.
- The CRC is obtained after the binary division.
- The string of n 0's appended to the data unit earlier is replaced by the CRC remainder.

Code

```c
/*

//////////////////////////////////////////////////////////////////
Experiment 2: Write a program to generate CRC code for checking error.
//////////////////////////////////////////////////////////////////

@author : Akshay S Rao
@usn    : 1BM17EC007
@Batch  : A1
@Date   : 19/10/2020

*/



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


struct Sender{

    int data[50];
    int div[16];
    int rem[16];
    int dataLength = 0;
    int divLength = 0;
};

/*********************************
@Function   : generator
@input      : (Struct)Sender*
@returns    : nothing
@logic      : does modulo 2 division to generate
              remainder and appends it to data
*********************************/


void
generator(struct Sender* sender){
    // calculate remainder
    int i = 0;
    int dataLength = sender->dataLength;
    int divLength =  sender->divLength;

    for(; i < divLength; i++)
        {sender->rem[i] = sender->data[i];}

    int k = divLength - 1;
    while(k < dataLength){
        if(sender->rem[0] == 1){
            for(i = 0; i < divLength; i++){
```

fig1

```c

/*********************************
@Function   : generator
@input      : (Struct)Sender*
@returns    : nothing
@logic      : does modulo 2 division to generate
              remainder and appends it to data
*********************************/


void
generator(struct Sender* sender){
    // calculate remainder
    int i = 0;
    int dataLength = sender->dataLength;
    int divLength =  sender->divLength;

    for(; i < divLength; i++)
        {sender->rem[i] = sender->data[i];}

    int k = divLength - 1;
    while(k < dataLength){
        if(sender->rem[0] == 1){
            for(i = 0; i < divLength; i++){
                sender->rem[i] = sender->rem[i] ^ sender->div[i];
            }
        }
        else{
            if(k == dataLength-1)break;
            // left shift remainder
            for(i = 0; i < divLength-1;i++){
                sender->rem[i] = sender->rem[i+1];
            }
            sender->rem[i] = sender->data[++k];
        }
    }
    int j = 1;
    // append remainder
    for(i = (dataLength - divLength + 1); i < dataLength; i++){
        sender->data[i] = sender->rem[j++];
    }

}
```

fig2

```c
/*********************************
@Function   : get
@input      : array*
@returns    : returns length of input data
@logic      : get input from standard input
*********************************/

int
get(int* array){

    int length = 0;
    char ch;
    while((ch = fgetc(stdin))!='\n'){
        if(ch == '1')array[length] = 1;
        else array[length] = 0;

        length++;
    }
    printf("\n");
    return length;
}

/*********************************
@Function   : get_divisor
@input      : (Struct)Sender*
@returns    : returns length of input data
@logic      : get divisor input from standard input
*********************************/

// //gets divisor and returns its length
int
get_divisor(struct Sender* sender){
    printf("Enter divisor: ");
    return get(sender->div);


}
```

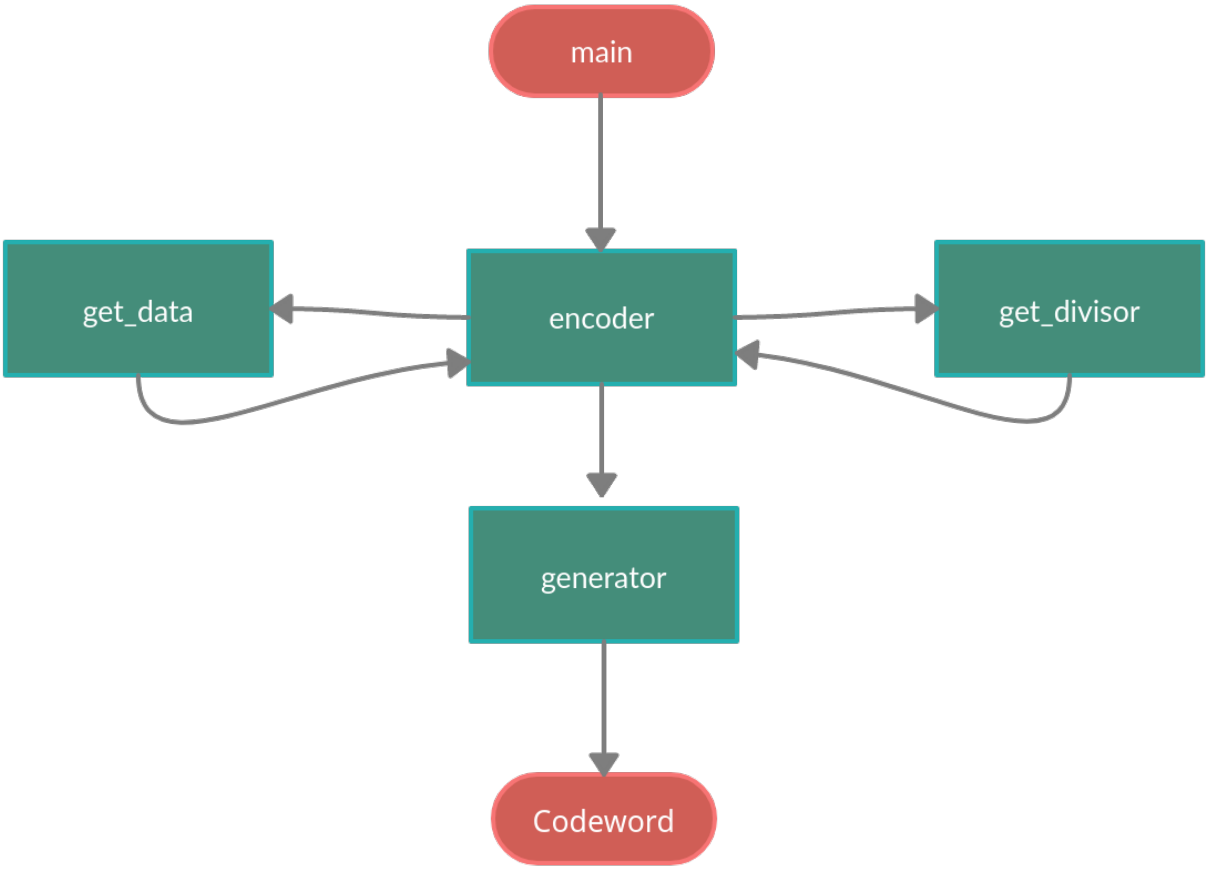
fig 3

```c
/*********************************
@Function   : get_data
@input      : (Struct)Sender*
@returns    : returns length of input data
@logic      : get data input from standard input
*********************************/

// //gets input data and returns its length
int
get_data(struct Sender* sender){
    printf("Enter data: ");
    return get(sender->data);
}

/*********************************
@Function   : encoder
@input      : (Struct)Sender*
@returns    : none
@logic      : calls get and generate functions to produce codeword
*********************************/

void
encoder(struct Sender* sender){


    sender->dataLength = get_data(sender);
    sender->divLength = get_divisor(sender);


    int dataLength = sender->dataLength;
    int divLength = sender->divLength;

    // append zeros
    for(int index = dataLength; index < (dataLength + divLength -1); index++)
        {sender->data[index] = 0;}

    // generate remainder
    sender->dataLength = dataLength + divLength -1;
    generator(sender);

    printf("codeword created\n");
}
```

fig 4

```c
void
encoder(struct Sender* sender){


    sender->dataLength = get_data(sender);
    sender->divLength = get_divisor(sender);


    int dataLength = sender->dataLength;
    int divLength = sender->divLength;

    // append zeros
    for(int index = dataLength; index < (dataLength + divLength -1); index++)
        {sender->data[index] = 0;}

    // generate remainder
    sender->dataLength = dataLength + divLength -1;
    generator(sender);

    printf("codeword created\n");
}


int main()
{
    /* code */

    struct Sender sender;
    encoder(&sender);

    for(int i = 0; i < sender.dataLength; i++){printf("%d",sender.data[i]);}

    printf("\n");

    return 0;
}
```

Flow Chart



Result

The data entered is 10101111 and the divisor is 1011 resulted in codeword 10101111110 with 110 as remainder.

```
7th sem/Networks/lab and self study/computerNetworksLab$ ./a.out
Enter data: 10101111

Enter divisor: 1011

codeword created
10101111110
akshay@akshay-Lenovo-ideapad-320-15ISK:/media/akshay/LENOVO/Textbooks and Notes/
```

Lab Report: Experiment 3

Problem Statement

Write a program to simulate Shortest Path Routing Algorithm

Introduction

Dijkstra's algorithm is an algorithm for finding the shortest path between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Theory and Algorithm

Let the node at which we are starting be called the **initial node**. Let the **distance of node Y** be the distance from the **initial node** to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbour B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").

Algorithm

- Create a set sptSet (shortest path tree set) that keeps track of vertices included in
- shortest path tree, i.e., whose minimum distance from source is calculated and finalized.
- Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as
- INFINITE.
- Assign distance value as 0 for the source vertex so that it is picked first.
- While sptSet doesn't include all vertices

  Pick a vertex u which is not there in sptSet and has minimum distance value.

  Include u to sptSet.

  Update distance value of all adjacent vertices of u.

- To update the distance values, iterate through all adjacent vertices.
- For every adjacent vertex v, if sum of distance value of u (from source) and weight of
- edge u-v, is less than the distance value of v, then update the distance value of v.

Code

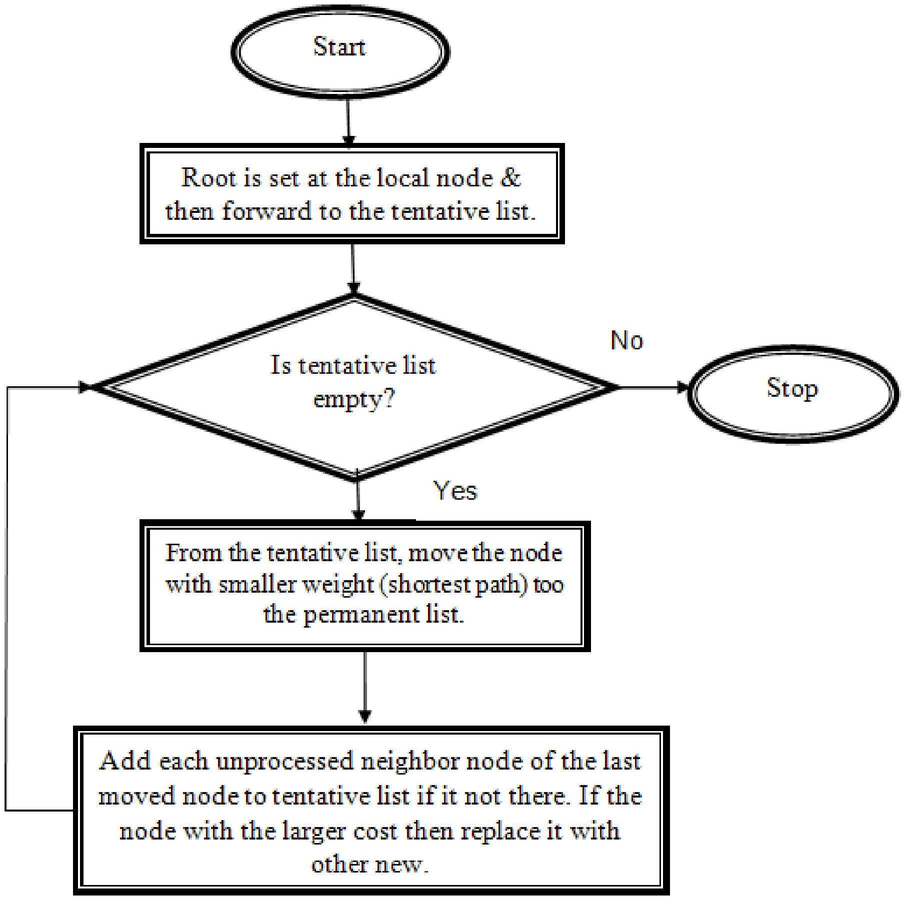
```c
/*
//////////////////////////////////////////////////////////////////////
Write a program to simulate Shortest Path Routing Algorithm
/////////////////////////////////////////////////////////////////////


@author : Akshay S Rao
@usn    : 1BM17EC007
@Batch  : A1
@Date   : 2/11/2020

*/




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

#define MAX_NODES 10
#define INFI 9999


/***********************************
@Function   : dijkstra
@input      : graph (2 dimensional array), number of nodes (int), start node
@returns    : void
@logic      : uses dijkstra algorithm to compute single source shortest path
***********************************/
void dijkstra(int graph[MAX_NODES][MAX_NODES], int n, int start_node){
    int cost[MAX_NODES][MAX_NODES], distance[MAX_NODES], pred[MAX_NODES];
    int visited[MAX_NODES], count, mindistance, nextnode,i,j;
    //pred[] stores the predecessor of each node
    //count gives the number of nodes seen so far
    //create the cost matrix
    for(i = 0; i < n; i++){
        for (j = 0; j < n; ++j)
        {
            /* code */
            if(graph[i][j] == 0)cost[i][j] = INFI;
            else cost[i][j] = graph[i][j];
        }
    }

    for (i = 0; i < n; ++i)
    {
        /* code */
        distance[i] = cost[start_node][i];
        pred[i] = start_node;
        visited[i] = 0;
    }
    distance[start_node] = 0;
    visited[start_node] = 1;
    count = 1;
```

: [✔], Line 110, Column 13 - Field 1 of 3

```c
    while(count < n -1){
        mindistance = INFI;
        // relaxation step
        for(i = 0; i < n; i++){
            if(distance[i] < mindistance && !visited[i]){
                mindistance = distance[i];
                nextnode = i;
            }
        }
        // choose next smallest distance node
        visited[nextnode] = 1;
        for(i = 0; i < n; i++){
            if(!visited[i]){
                if(mindistance + cost[nextnode][i] < distance[i]){
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
            }
        }count++;
    }

//print the path and distance of each node
    for(i=0;i<n;i++){
        if(i!=start_node){
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);
            j=i;
            do{
                j=pred[j];
                printf("<-%d",j);
            }while(j!=start_node);
        }
    }
}
```

```c
int main(){

    int G[MAX_NODES][MAX_NODES],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    printf("\nEnter the starting node:");
    scanf("%d",&u);

    dijkstra(G,n,u);
    printf("\n");

    return 0;
}
```

Flowchart



Result

```
akshay@akshay-Lenovo-ideapad-320-15ISK:/media/akshay/LENOVO/Textbooks and Notes/
7th sem/Networks/lab and self study/computerNetworksLab$ ./a.out
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
```

Lab Report: Experiment 4

Problem Statement

Write a program to encrypt and decrypt a given message using substitution cypher method.

Introduction

The two basic building blocks of all encryption techniques are substitution and transposition. A substitution technique is one in which the letters of plaintext are replacedby other letters or by numbers or symbols. If the plaintext is viewed as a sequenceof bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

Theory

Caesar Cipher

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example

plain:	meet	me	after	the	toga	party
cipher:	PHHW	PH	DIWHU	WKH	WRJD	SDUWB

plain:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
cipher:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Then the algorithm can be expressed as follows. For each plaintext letter p , substitute the ciphertext letter C:

$$C = E(3, p) = (p + 3) \bmod 26$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = E(k, p) = (p + k) \bmod 26$$

where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$$p = D(k, C) = (C - k) \bmod 26$$

Algorithm

- Traverse the given text one character at a time .
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Return the new string generated.

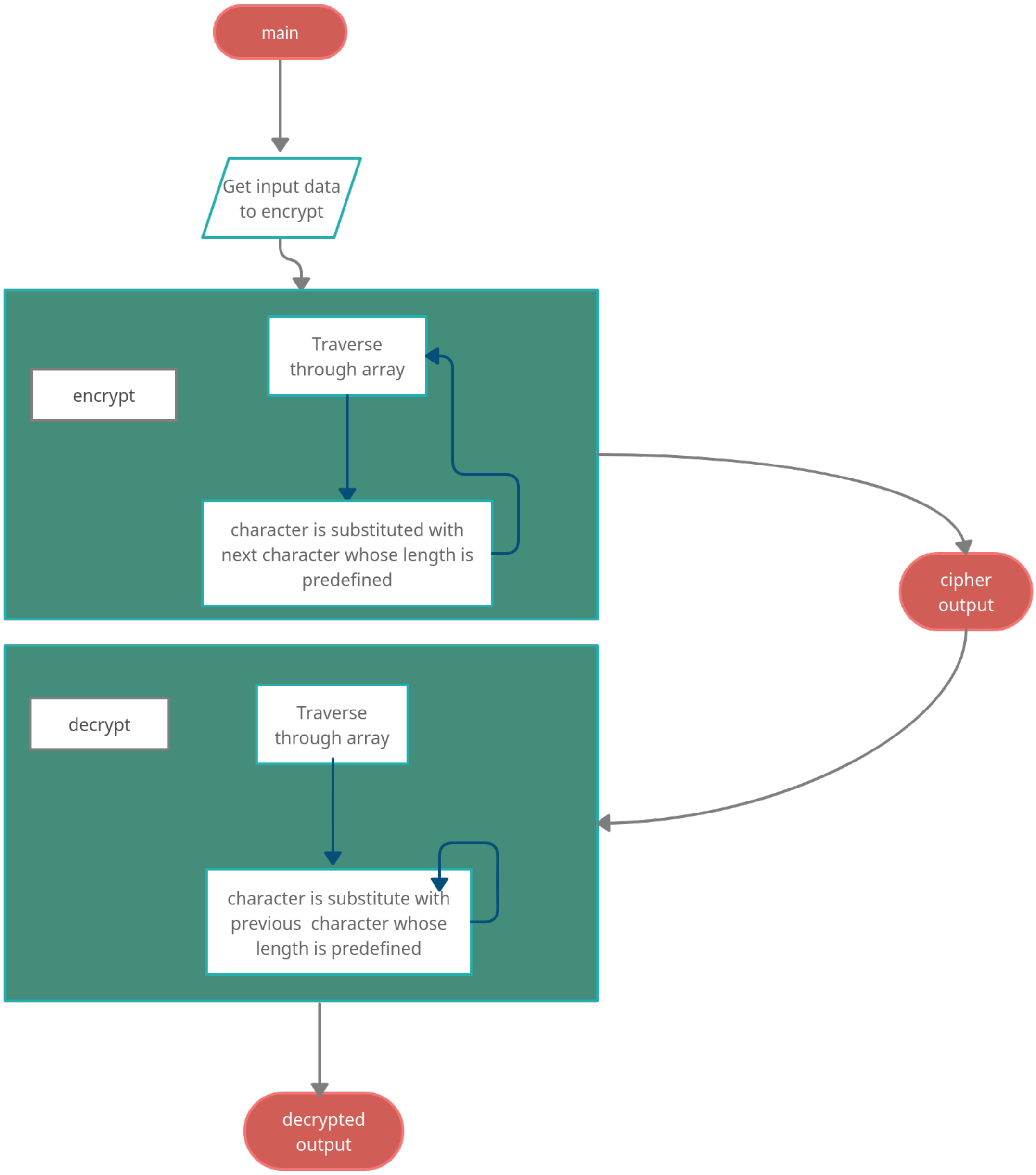
Code

```c
/*

////////////////////////////////////////////////////////////////////
Encrypt and decrypt a given message using substitution cypher method.
///////////////////////////////////////////////////////////////////


@author : Akshay S Rao
@usn    : 1BM17EC007
@Batch  : A1
@Date   : 25/10/2020

*/




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

#define MAX_BUFFER_SIZE 50
#define SUBSTITUTION_LENGTH 3


/**********************************
@Function     : caesar
@input        : array (char*), length(int)
@returns      : void
@logic        : substitutes original letter with
                a letter SUBSTITUTION_LENGTH apart
**********************************/

void encrypt(char* array, int length){

    for(int i = 0; i < length; i++){
        char new_char = array[i] +  SUBSTITUTION_LENGTH;
        if(new_char - 'z' > 0)array[i] = 'a' + (new_char-'z');
        else array[i] = new_char;

    }


}
```

```c
/*********************************
@Function   : decrypt
@input      : array (char*), length(int)
@returns    : void
@logic      :  substitutes original letter with
            a letter SUBSTITUTION_LENGTH behind
**********************************/

void decrypt(char* array, int length){


    for(int i = 0; i < length; i++){
        char new_char = array[i] -  SUBSTITUTION_LENGTH;
        if('a' - new_char  > 0)array[i] = 'z' - ('a' - new_char);
        else array[i] = new_char;

    }

}

/*********************************
@Function   : get_input
@input      : array (char*)
@returns    : length of input
@logic      : takes input as long as you dont press new line
**********************************/

int get_input(char* array){
    int length = 0;
    char ch;
    printf("enter small case letters only\n");
    while((ch = fgetc(stdin))!='\n'){
        if((ch - 'a') < 0 || (ch - 'a') > 25){
            continue;
        }
        else{
            array[length] = ch;
            length++;
        }

        if(length >= MAX_BUFFER_SIZE)break;

    }

    return length;

}
```

```c
int main(){

    char buffer[50];
    int inputLength = get_input(buffer);

    printf("your data is: ");
    for(int i = 0 ; i < inputLength; i++){
        printf("%c",buffer[i]);
    }

    printf("\n");
    encrypt(buffer, inputLength);

    printf("your cipher is: ");

    for(int i = 0 ; i < inputLength; i++){
        printf("%c",buffer[i]);
    }
    printf("\n");

    decrypt(buffer, inputLength);
    printf("your decrypte data is: ");

    for(int i = 0 ; i < inputLength; i++){
        printf("%c",buffer[i]);
    }


    printf("\n");



return 0;
}
```

Flowchart



Result

Input given is akshay and it is substituted with a character 3 length apart to get cipher text dnvkdc which is decrypted again to get original data akshay

```
akshay@akshay-Lenovo-ideapad-320-15ISK:/media/akshay/LENOVO/Textbooks and Notes/7th sem/Networks/lab and self study/computerNetworksLab$ ./a.out
enter small case letters only
akshay
your data is: akshay
your cipher is: dnvkdc
your decrypte data is: akshay
```

Lab Report: Experiment 5

Problem Statement

Write a program to demonstrate Diffie-Hellman algorithm

Introduction

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

Theory

- First, both Alice and Bob agree upon a prime number and another number that has no factor in common. Lets call the prime number as p and the other number as g. Note that g is also known as the generator and p is known as prime modulus.
- Now, since eve is sitting in between and listening to this communication so eve also gets to know p and g.
- Now, the modulus arithmetic says that r = (g to the power x) mod p. So r will always produce an integer between 0 and p.
- The first trick here is that given x (with g and p known) , its very easy to find r. But given r (with g and p known) its difficult to deduce x.
- One may argue that this is not that difficult to crack but what if the value of p is a very huge prime number? Well, if this is the case then deducing x (if r is given) becomes almost next to impossible as it would take thousands of years to crack this even with supercomputers.
- This is also called the discrete logarithmic problem.
- Coming back to the communication, all the three Bob, Alice and eve now know g and p.

- Now, Alice selects a random private number xa and calculates (g to the power xa) mod p= ra.This resultant ra is sent on the communication channel to Bob. Intercepting in between, eve also comes to know ra.
- Similarly Bob selects his own random private number xb, calculates (g to the power xb) mod p= rb and sends this rb to Alice through the same communication channel. Obviously eve also comes to know about rb.
- So eve now has information about g, p, ra and rb.
- Now comes the heart of this algorithm. Alice calculates (rb to the power xa) mod p = Final key which is equivalent to (g to the power (xa*xb)) mod p .
- Similarly Bob calculates (ra to the power xb) mod p = Final key which is again equivalent to (g to the power(xb * xa)) mod p.
- So both Alice and Bob were able to calculate a common Final key without sharing each others private random number and eve sitting in between will not be able to determine the Final key as the private numbers were never transferred.

Algorithm

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated = $x = G^a mod P$	Key generated = $y = G^b mod P$
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = $k_a = y^a mod P$	Generated Secret Key = $k_b = x^b mod P$
Algebraically it can be shown that $k_a = k_b$	
Users now have a symmetric secret key to encrypt	

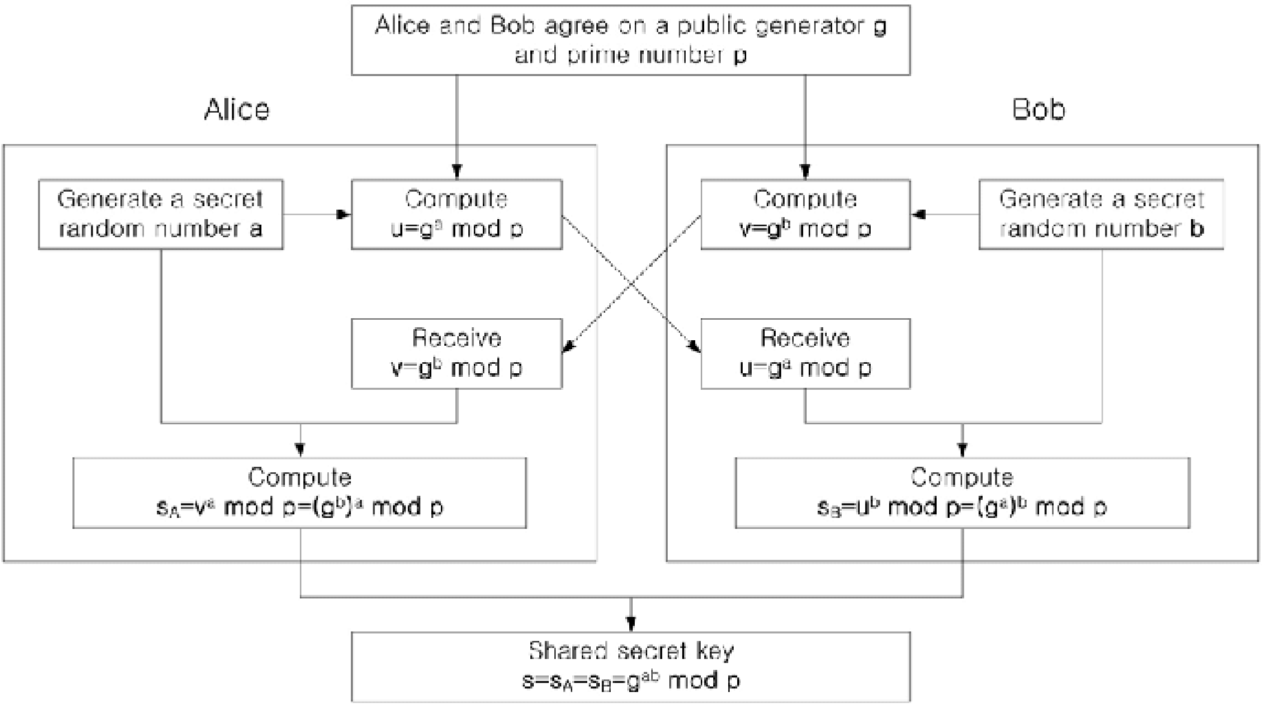
-

Code

```c
/*

///////////////////////////////////////////////////////////////////
Write a program to implement Diffie-Hellman Algorithm.
//////////////////////////////////////////////////////////////////


@author : Akshay S Rao
@usn    : 1BM17EC007
@Batch  : A1
@Date   : 9/11/2020

*/




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



/**********************************
@Function   : power
@input      : a (long long int) : g in theory,
              b (long long int) - private number:x or y in theory
@returns    : a power b
@logic      : nothing special here
***********************************/
long long int power(long long int a, long long int b){

    if (b==1){return a;}
    else return ((long long int)pow(a,b));
}

int main(){

    srand(time(NULL));

    long long int p, g, x, y, Ka, Kb, keyAllice, keyBob;

    printf("enter a prime number: ");
    scanf("%lld", &p);
    printf("\n");

    printf("choosing a random number for g.......");
    g = 9;
    printf("the chosen number is %lld\n", g);

    x = 4;
    y = 3;
```

```c
    srand(time(NULL));

    long long int p, g, x, y, Ka, Kb, keyAllice, keyBob;

    printf("enter a prime number: ");
    scanf("%lld", &p);
    printf("\n");

    printf("choosing a random number for g.......");
    g = 9;
    printf("the chosen number is %lld\n", g);

    x = 4;
    y = 3;

    printf("the x chosen by alice is %lld\n", x);
    printf("the y chosen by bob is %lld\n", y);

    Ka = power(g, x)%p;
    Kb = power(g, y)%p;

    printf("the encrypted message by alice is  %lld\n", Ka);
    printf("the encrypted message by bob is %lld\n", Kb);

    keyAllice = power(Kb, x)%p;
    keyBob = power(Ka, y)%p;

    printf("the key obtained by  alice is  %lld\n", keyAllice);
    printf("the key obtained by bob is %lld\n", keyBob);




    return 0;
}
```

FlowChart



Result:

```
akshay@akshay-Lenovo-ideapad-320-15ISK:/media/akshay/LENOVO/Textbooks and Notes/
7th sem/Networks/lab and self study/computerNetworksLab$ ./1BM17EC007_Diffie_Hel
lman
enter a prime number: 29

choosing a random number for g.......the chosen number is 9
the x chosen by alice is 4
the y chosen by bob is 3
the encrypted message by alice is  7
the encrypted message by bob is 4
the key obtained by  alice is  24
the key obtained by bob is 24
```

Lab Report: Experiment 6

Problem Statement

To study the Basic Networking Commands

1. Discover IP Address, Subnet Mask , Default Gateway of your system

Subnet mask

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:3f:99:12:68  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 54:e1:ad:97:f8:81  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 16266  bytes 1394444 (1.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16266  bytes 1394444 (1.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:96:91:0a  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.184  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::9cba:d51:93c2:f94a  prefixlen 64  scopeid 0x20<link>
        inet6 2406:7400:73:1a44:d98d:8157:ecb1:8fc8  prefixlen 64  scopeid 0x0<global>
        inet6 2406:7400:73:1a44:873:caa5:3804:648b  prefixlen 64  scopeid 0x0<global>
        ether 64:6e:69:d9:90:eb  txqueuelen 1000  (Ethernet)
        RX packets 222942  bytes 246792313 (246.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 75925  bytes 17770571 (17.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

IP address

Default gateway

Default gateway

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ ip route | grep default
default via 192.168.0.1 dev wlp3s0 proto dhcp metric 600
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    600    0        0 wlp3s0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 virbr0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.0.0     0.0.0.0         255.255.255.0   U     600    0        0 wlp3s0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

2. Discover your default server and your system's address: cat *etc/resolve.conf*

DNS server

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
# 127.0.0.53 is the systemd-resolved stub resolver.
# run "systemd-resolve --status" to see details about the actual nameservers.

nameserver 127.0.0.53
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:3f:99:12:58  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 54:e1:ad:97:f8:81  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 18170  bytes 1568697 (1.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18170  bytes 1568697 (1.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:96:91:0a  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0-nic: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 52:54:00:96:91:0a  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.184  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::9cba:d518:93c2:f94a  prefixlen 64  scopeid 0x20<link>
        inet6 2406:7400:73:1a44:d98d:8157:ecb1:8fc8  prefixlen 64  scopeid 0x0<global>
        inet6 2406:7400:73:1a44:873:caa5:3804:648b  prefixlen 64  scopeid 0x0<global>
        ether 64:6e:69:d9:90:eb  txqueuelen 1000  (Ethernet)
        RX packets 234710  bytes 254555356 (254.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 82376  bytes 18901164 (18.9 MB)
```

3. Find out a command which resolve addresses to hostnames: host *hostname*

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ host www.google.com
www.google.com has address 216.58.197.68
www.google.com has IPv6 address 2404:6800:4007:812::2004
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ host 216.58.197.68
68.197.58.216.in-addr.arpa domain name pointer maa03s21-in-f4.1e100.net.
68.197.58.216.in-addr.arpa domain name pointer maa03s21-in-f68.1e100.net.
```

4. Find maximum number of hops to a given target: traceroute *hostname*

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ traceroute www.google.com
traceroute to www.google.com (216.58.197.68), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)  7.427 ms  7.322 ms  7.302 ms
 2  * * *
 3  14.142.183.201.static-Bangalore.vsnl.net.in (14.142.183.201)  9.274 ms  9.223 ms  9.146 ms
 4  * 172.31.167.50 (172.31.167.50)  16.642 ms 172.31.167.46 (172.31.167.46)  16.476 ms
 5  121.240.1.46 (121.240.1.46)  16.436 ms  16.354 ms  17.385 ms
 6  108.170.253.97 (108.170.253.97)  16.242 ms  10.834 ms  10.815 ms
 7  108.170.236.197 (108.170.236.197)  11.482 ms 108.170.237.95 (108.170.237.95)  10.197 ms 108.170.236.197 (108.17
0.236.197)  10.347 ms
 8  maa03s21-in-f4.1e100.net (216.58.197.68)  9.953 ms  11.382 ms  10.109 ms
```

5. Find time to reach a server from a client: ping *hostname*

```
akshay@akshay-Lenovo-ideapad-320-15ISK:~$ ping www.google.com
PING www.google.com(maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004)) 56 data bytes
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=1 ttl=117 time=26.1 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=2 ttl=117 time=28.8 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=3 ttl=117 time=28.9 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=4 ttl=117 time=28.7 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=5 ttl=117 time=29.1 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=6 ttl=117 time=26.5 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=7 ttl=117 time=29.0 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=8 ttl=117 time=29.2 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=9 ttl=117 time=28.6 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=10 ttl=117 time=31.9 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=11 ttl=117 time=28.7 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=12 ttl=117 time=29.2 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=13 ttl=117 time=28.7 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=14 ttl=117 time=30.3 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=15 ttl=117 time=29.2 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=16 ttl=117 time=30.4 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=17 ttl=117 time=25.9 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=18 ttl=117 time=32.7 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=19 ttl=117 time=28.8 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=20 ttl=117 time=29.2 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=21 ttl=117 time=28.6 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=22 ttl=117 time=28.6 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=23 ttl=117 time=28.7 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=24 ttl=117 time=31.1 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=25 ttl=117 time=29.2 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=26 ttl=117 time=30.6 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=27 ttl=117 time=28.0 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=28 ttl=117 time=28.6 ms
64 bytes from maa05s10-in-x04.1e100.net (2404:6800:4007:808::2004): icmp_seq=29 ttl=117 time=28.7 ms
```