**GO BACK N**

#include<stdio.h>

int main(){

int ack,sent = 0,i,window;

printf("enter the window size\n");

scanf("%d",&window);

while(1){

for(i = 1;i<=window;i++){

printf("\nframe %d has been sent",sent);

sent++;

if(window == sent){

break;

}

}

printf("enter the acknowelgement ");

scanf("%d",&ack);

if(ack == window){

break;

}

sent = ack;

}

return 0;

}

**DISTANCE VECTOR**

#include<stdio.h>

int main(){

int nn[10],rt[50][50],delay[10];

int i,j,s,n,r,min,node;

printf("entert the number of nodes in network");

scanf("%d",&n);

printf("enter the node to be updated\n");

scanf("%d",&s);

printf("how many neighbour nodes ");

scanf("%d",&r);

for(i = 0;i<r;i++){

printf("\nenter the node:");

scanf("%d",&nn[i]);

printf("\ndelay");

scanf("%d",&delay[i]);

}

printf("\nenter the neightbour nodes routing table\n");

for(i = 0;i<r;i++){

for(j = 0;j<n;j++){

scanf("%d",&rt[i][j]);

rt[i][j]+=delay[i];

}

}

printf("\nTO|DELAY|LINE\n");

for(i = 0;i<n;i++){

min = 32767;

for(j = 0;j<r;j++){

if(rt[j][i]<min){

min = rt[j][i];

node = nn[j];

}

}

if(s!=i){

printf("\n%c|%d|%c",i+65,min,node+65);

}

else{

printf("\n%c|0|-",s+65);

}

}

return 0;

}

**CYCLICK REDUNDANCY CHECK**

#include<stdio.h>

#include<string.h>

int main(){

char m[50],g[50],tm[50];

int ml,gl,i,j,f=0;

printf("enter the messege\n");

scanf("%s",m);

ml = strlen(m);

printf("enter the generator\n");

scanf("%s",g);

gl = strlen(g);

for(i = 0;i<gl-1;i++){

m[ml+i] = '0';

}

m[ml+i] = '\0';

for(i = 0;i<gl;i++){

tm[i] =m[i];

}

for(i = 0;i<ml;i++){

if(tm[0] == '1'){

for(j = 0;j<gl;j++){

if(tm[j] == g[j]){

tm[j] = '0';

}

else{

tm[j] = '1';

}

}

}

for(j = 0;j<gl-1;j++){

tm[j] = tm[j+1];

}

tm[gl-1] = m[gl+i];

}

tm[gl-1] = '\0';

printf("remainder is %s",tm);

for(i = 0;i<gl-1;i++){

m[ml+i] = tm[i];

}

m[ml+i] = '\0';

printf("\nsended messege %s\n",m);

printf("\nRECEIVED SIDE\n");

printf("enter the received messege ");

scanf("%s",m);

for(i = 0;i<gl;i++){

tm[i] =m[i];

}

for(i = 0;i<ml;i++){

if(tm[0] == '1'){

for(j = 0;j<gl;j++){

if(tm[j] == g[j]){

tm[j] = '0';

}

else{

tm[j] = '1';

}

}

}

for(j = 0;j<gl-1;j++){

tm[j] = tm[j+1];

}

tm[j] = m[gl+i];

}

tm[gl-1]='\0';

printf("remainder is %s",tm);

for(i = 0;i<gl-1;i++){

if(tm[i] != '0')

f = 1;

}

if(f == 1)

printf("error occured");

else

printf("received sucessfully");

return 0;

**MINIMUM SPANNING TREE BROAD CAST TREE**

#include<stdio.h>

int main(){

int nn[50][50],v[50];

int i,j,c,n,min = 3767,k,m;

printf("enter the number od nodes\n");

scanf("%d",&n);

printf("enter -1 if no connection\n");

for(i = 0;i<n;i++){

v[i] = 0;

for(j = 0;j<n;j++){

scanf("%d",&nn[i][j]);

if(nn[i][j] != -1 && nn[i][j]<min){

min = nn[i][j];

k = i;

m = j;

}

}

}

printf("\n%c - %c = %d",k+65,m+65,min);

v[k] = 1;

v[m] = 1;

c = 1;

while(c<=n-2){

min = 3767;

for(i = 0;i<n;i++){

if(v[i] == 1){

for(j = 0;j<n;j++){

if(nn[i][j] != -1 && v[j] != 1 && nn[i][j]<min){

min = nn[i][j];

k = i;

m = j;

}

}

}

}

v[m] = 1;

printf("\n%c - %c = %d",k+65,m+65,min);

c++;

}

return 0;

}

**DIJESCTRA ALGORITHEM**  
#include <stdio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main() {

int G[MAX][MAX], i, j, n, startnode;

// Input number of vertices

printf("Enter the number of vertices: ");

scanf("%d", &n);

// Input adjacency matrix

printf("Enter the adjacency matrix (use -1 for no edge):\n");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

scanf("%d", &G[i][j]);

if (G[i][j] == -1) {

G[i][j] = INFINITY;

}

}

}

// Input the starting node

printf("Enter the starting node (0 to %d): ", n - 1);

scanf("%d", &startnode);

// Call the Dijkstra function

dijkstra(G, n, startnode);

return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode) {

int distance[MAX], pred[MAX];

int visited[MAX] = {0}; // Array to track visited nodes

int i, j, count, mindistance, nextnode;

// Initialize distance and predecessor arrays

for (i = 0; i < n; i++) {

distance[i] = G[startnode][i];

pred[i] = startnode;

}

distance[startnode] = 0;

visited[startnode] = 1; // Mark startnode as visited

count = 1;

// Dijkstra's algorithm

while (count < n - 1) {

mindistance = INFINITY;

// Find the node with the minimum distance

for (i = 0; i < n; i++) {

if (distance[i] < mindistance && visited[i] == 0) {

mindistance = distance[i];

nextnode = i;

}

}

// Mark the nextnode as visited

visited[nextnode] = 1;

// Update the distance and predecessor arrays

for (i = 0; i < n; i++) {

if (visited[i] == 0 && mindistance + G[nextnode][i] < distance[i]) {

distance[i] = mindistance + G[nextnode][i];

pred[i] = nextnode;

}

}

count++;

}

// Print the shortest distance and path

for (i = 0; i < n; i++) {

if (i != startnode) {

printf("\nDistance from node %d to node %d = %d", startnode, i, distance[i]);

printf("\nPath: %d", i);

// Print the path

j = i;

while (j != startnode) {

j = pred[j];

printf(" <- %d", j);

}

}

}

}

**BIT STUFFING**

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{

char ip[50],op[100];

int i,ln,j;

//clrscr();

printf("\n enter a text:");

gets(ip);

//ln=strlen(ip);

for(i=0,j=0;ip[i]!='\0';i++,j++)

{

op[j]=ip[i];

if(strncmp(ip+i,"11111",5)==0)

{

op[j]='\0';

strcat(op,"111110");

i=i+4;

j=j+5;

}

}

op[j]='\0';

printf("\n output is:01111110 %s 01111110",op);

getch();

}

**CHARACTER STUFFING**

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{

char ip[100],op[100];

int i,ln,j;

//clrscr();

printf("\n enter a text:");

gets(ip);

//ln=strlen(ip);

for(i=0,j=0;ip[i]!='\0';i++,j++)

{

op[j]=ip[i];

if( strncmp(ip+i,"FLAG",4)==0 )

{

op[j]='\0';

strcat(op,"ESC FLAG");

i=i+3;

j=j+7;

}

if( strncmp(ip+i,"ESC",3)==0 )

{

op[j]='\0';

strcat(op,"ESC ESC");

i=i+2;

j=j+6;

}

}

op[j]='\0';

printf("\n output is:FLAG %s FLAG",op);

getch();

}