### Write a program for congestion control using Leaky bucket algorithm.

#include<stdlib.h>

#include<stdio.h>

#include<unistd.h>

#define NOF\_PACKETS 10

int rando(int a){

int rn=(random()%10)%a;

return rn==0?1:rn;

}

int main()

{

int packet\_sz[NOF\_PACKETS],i,clk,p\_sz\_rm=0,b\_size,o\_rate,p\_sa,p\_time,op;

for(i=0;i<NOF\_PACKETS;++i)

packet\_sz[i]=rando(6)\*10;

for(i=0;i<NOF\_PACKETS;++i)

printf("\n Packet[%d]:%d bytes\t",i,packet\_sz[i]);

printf("\n enter the output rate : ");

scanf("%d",&o\_rate);

printf("\n enter the bucket size : ");

scanf("%d",&b\_size);

for(i=0;i<NOF\_PACKETS;i++){

if((packet\_sz[i]+p\_sz\_rm)>b\_size)

if(packet\_sz[i]>b\_size)

printf("\n\nIncomming packet size(%d bytes) is greater than bucket capacity (%d bytes)-PACKET REJECTED",packet\_sz[i],b\_size);

else

printf("\n\nBucket capacity exceeded-PACKET REJECTED!!");

else{

p\_sz\_rm+=packet\_sz[i];

printf("\n\nIncomming packet size : %d",packet\_sz[i]);

printf("\nBytes remaining to transmit : %d",p\_sz\_rm);

p\_time=rando(4)\*10;

printf("\nTime left for Transmission : %d",p\_time);

for(clk=10;clk<=p\_time;clk+=10){

sleep(1);

if(p\_sz\_rm){

if(p\_sz\_rm<=o\_rate)

op=p\_sz\_rm,p\_sz\_rm=0;

else

op=o\_rate,p\_sz\_rm-=o\_rate;

printf("\npacket of size %d transmitted",op);

printf("\nBytes remaining to transmit : %d",p\_sz\_rm);

}

else{

printf("\nTime left for transmission : %d units",p\_time-clk);

printf("\nNo of packets to transmit");

}

}

}

}

return 0;

}

**OUTPUT**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Develop a simple data link layer that performs the flow control using the sliding window protocol

#include<stdio.h>

int main(){

int w,i,f,frames[50];

printf("Enter window size: ");

scanf("%d",&w);

printf("\nEnter the number of frames to transmit: ");

scanf("%d",&f);

printf("\nEnter %d frames: ",f);

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

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

}

printf("\nWith siding window protocol the frame will be sent in the following manner\n\n");

printf("\nAfter sending %d frames at each stage sender waits for acknowledgment sent by the reciever\n\n",w);

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

if(i%w==0){

printf("%d\n",frames[i]);

printf("Acknowledgement of the above frames sent is recieved by the sender\n\n");

}

else

{

printf("%d ",frames[i]);

}

}

if(f%w!=0)

printf("\nAcknowledgement of above frames is recieved successfully by the sender\n");

}

**OUTPUT**

Enter window size: 3

Enter number of frames to transmit: 5

Enter 5 frames: 12 5 89 4 6

With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver

12 5 89

Acknowledgement of above frames sent is received by sender

4 6

Acknowledgement of above frames sent is received by sender

### Develop a simple data link layer that performs the flow control using the Go Back N protocol in c

#include<stdio.h>

int main(){

int windosize,sent=0,ack,i;

printf("Enter window size\n");

scanf("%d",&windosize);

while(1){

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

printf("Frame %d has been transmitted.\n",sent);

sent++;

if(sent==windosize)

break;

}

printf("\nPlese enter the last Acknowledgment recieved.\n");

scanf("%d",&ack);

if(ack==windosize)

break;

else

sent=ack;

}

}

**OUTPUT**

enter window size

8

Frame 0 has been transmitted.

Frame 1 has been transmitted.

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.

Please enter the last Acknowledgement received.

2

Frame 2 has been transmitted.

Frame 3 has been transmitted.

Frame 4 has been transmitted.

Frame 5 has been transmitted.

Frame 6 has been transmitted.

Frame 7 has been transmitted.

Please enter the last Acknowledgement received.

8

### Implement the data link layer framing method such as Character Stuffing

### Program:

#include<stdio.h>

#include<string.h>

#include<conio.h>

void main()

{

char a[30],fs[50]=" ",t[3],sd,ed,x[3],s[3],d[3],y[3]; int i,j,p=0,q=0;

printf("Enter characters to be stuffed:");

scanf("%s",a);

printf("\nEnter a character that represents starting delimiter:");

getchar();

scanf("%c",&sd);

printf("\nEnter a character that represents ending delimiter:");

getchar();

scanf("%c",&ed);

x[0]=s[0]=s[1]=sd;

x[1]=s[2]='\0';

y[0]=d[0]=d[1]=ed;

d[2]=y[1]='\0';

strcat(fs,x);

for(i=0;i<strlen(a);i++)

{

t[0]=a[i];

t[1]='\0';

if(t[0]==sd)

strcat(fs,s);

else if(t[0]==ed)

strcat(fs,d);

else

strcat(fs,t);

}

strcat(fs,y);

printf("\n After stuffing:%s",fs);

}

**OUTPUT**

Enter characters to be stuffed: goodday

Enter a character that represents starting delimiter: d

Enter a character that represents ending delimiter: g

After stuffing: dggooddddayg.

### Implement the data link layer framing method such as bit stuffing.

### #include <stdio.h>

### #include<string.h>

### #include <conio.h>

### void main()

### {

### char a[20],fs[50]="",t[6],r[5]; int i,j,p=0,q=0;

### printf("enter bit string : ");

### scanf("%s",a);

### strcat(fs,"01111110");

### if(strlen(a)<5)

### {

### strcat(fs,a);

### }

### else

### {

### for(i=0;i<strlen(a)-4;i++)

### {

### for(j=i;j<i+5;j++)

### {

### t[p++]=a[j];

### }

### t[p]='\0';

### if(strcmp(t,"11111")==0)

### {

### strcat(fs,"111110"); i=j-1;

### }

### else

### {

### r[0]=a[i];

### r[1]='\0';

### strcat(fs,r);

### }

### p=0;

### }

### for(q=i;q<strlen(a);q++)

### {

### t[p++]=a[q];

### }

### t[p]='\0';

### strcat(fs,t);

### }

### strcat(fs,"01111110");

### printf("After stuffing : %s",fs);

### getch();

### }

### Output:

Enter bit string: 10101111110

After stuffing: 0111111010101111101001111110

Enter bit string: 1011111011110111110

After stuffing: 0111111010111110011110111110001111110

### Implement Dijkstra’s algorithm to compute the shortest path through a graph.

#include<stdio.h>

#include<conio.h>

void main()

{

int path[6][6],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;

printf("enter the cost matrix\n");

for(i=1;i<=5;i++)

for(j=1;j<=5;j++)

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

printf("enter number of paths\n");

scanf("%d",&p);

printf("enter possible paths\n");

for(i=1;i<=p;i++)

for(j=1;j<=5;j++)

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

for(i=1;i<=p;i++)

{

t[i]=0;

stp=st;

for(j=1;j<=5;j++)

{

edp=path[i][j+1];

t[i]=t[i]+a[stp][edp];

if(edp==ed)

break;

else

stp=edp;

}

}

min=t[st];

index=st;

for(i=1;i<=p;i++)

{

if(min>t[i])

{

min=t[i];

index=i;

}

}

printf("minimum cost %d",min);

printf("\n minimum cost path ");

for(i=1;i<=5;i++)

{

printf("--> %d",path[index][i]);

if(path[index][i]==ed)

break;

}

getch();

}

**Or**

#include <limits.h>

#include <stdio.h>

#include <stdbool.h>

// Number of vertices in the graph

#define V 9

// A utility function to find the vertex with minimum distance value, from

// the set of vertices not yet included in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// A utility function to print the constructed distance array

void printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t %d\n", i, dist[i]);

}

// Function that implements Dijkstra's single source shortest path algorithm

// for a graph represented using adjacency matrix representation

void dijkstra(int graph[V][V], int src)

{

int dist[V]; // The output array. dist[i] will hold the shortest

// distance from src to i

bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest

// path tree or shortest distance from src to i is finalized

// Initialize all distances as INFINITE and stpSet[] as false

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices not

// yet processed. u is always equal to src in the first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the picked vertex.

for (int v = 0; v < V; v++)

// Update dist[v] only if is not in sptSet, there is an edge from

// u to v, and total weight of path from src to v through u is

// smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// print the constructed distance array

printSolution(dist);

}

// driver program to test above function

int main()

{

/\* Let us create the example graph discussed above \*/

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

**OUTPUT**

A screenshot of a computer

Description automatically generated