SOLUCION EXAMEN COMUNICACION DE DATOS
(ref: scdt9902.doc)
17 de FEBRERO de 1999.

Problema 1:

Se desea monitorear el tiempo de respuesta de un enlace, ejecutando un test durante 4 hs., a intervalos de 30 seg., que permita determinar el tiempo medio de respuesta del enlace. Se cuenta con una red TCP/IP, que brinda servicios de datagrama (*udp_send* y *udp_receive*), utilizables por el test a implementar y un *udp_port* conocido para ser utilizado por el mismo, que se encuentra activo en todos los equipos. Se dispone además de una lista de máquinas (direcciones IP), que son las máquinas a monitorear.

Se pide: Especificar en un lenguaje de alto nivel, el protocolo que debe correr en la máquina origen, y destino, para implementar el test mencionado.

Solución:

El test consiste en enviar un datagrama al *udp_port*, y el servicio que está en la máquina a monitorear lo devuelve como lo recibió. Para poder monitorear el tiempo de respuesta del enlace, se debe tener registro de la hora de envío de cada datagrama.

Supongo que se dispone de una función time_stamp(time), que me devuelve la hora del sistema.
El formato de datagrama a utlizar es el siguiente:

```
datagrm record of
        begin
                tipo: [req,resp];
                dir_origen:  direcciónIP;
                hora_envio: time;
        end;
```

En la máquina destino, debo tener corriendo el siguiente protocolo, que realiza un echo de los paquetes recibidos a la dirección origen.

```
Procedure destino;
// recibido un datagrama, chequea que sea un requerimiento
// y si lo es devuelve un echo del mismo a quien lo envió.
var
        in_dtgrm:datagrm;
        out_dtgrm:datagrm;

begin
        while (true)
        begin
                udp_recive(in_dtgrm);
                if (in_dtgrm.tipo=”req”) then
                begin
                        out_dtgrm.tipo=”resp”;
                        out_dtgrm.dir_origen = in_dtgrm.dir_origen;
                        out_dtgrm.time = in_dtgrm.time;
                        udp_send(in_dtgrm.dir_origen,out_dtgrm);
                end;
        end;
end.
```

En la máquina origen, debo generar un datagrama con tipo REQ, el time_stamp y mi dirección como dirección origen, y proceso los resultados, realizando el cálculo de tiempos al final del proceso. Supongo la solución en un pseudo pascal concurrente:

Los datos los almaceno en la siguiente estructura:

```
est: array [dirIP1 .. dirIPN of  record of
        begin
                cant: integer;
                seg_acum: integer;
        end;


Procedure origen_envio;
// envia los datagramas para el requerimiento de información
// cada 30 segundos durante 4 hrs.
// una vez terminado realiza los cálculos estadísticos
var
        out_dtgrm:datagrm;
        cant:integer;

begin
        cant:=0;
        while (cant <= 480)             // cantidad de veces de consulta 2*60*4
        begin
                for destino in lista_destinatarios
                begin
                        out_dtgrm.tipo=”req”;
                        out_dtgrm.dir_origen = my_dirIP;
                        out_dtgrm.time = time_stamp;
                        udp_send(destino,out_dtgrm);
                end;
                wait(30);               // Espera 30 segundos
        end;
                                        // Entrega los resultados obtenidos
        for destino in lista_destinatarios
        begin
                prom:=est[destino]. seg_acum/ 2 * est[destino].cant;
                print(“promedio nodo ”,destino,” es “,prom)
        end;
end.
```

```
Procedure origen_recibo;
// procesa las respuestas recibidas, y guarda la
// información estadística, para el procesamiento
// final.
var
        in_dtgrm:datagrm;

begin
        while (true)
        begin
                udp_recive(in_dtgrm);
                if (in_dtgr.tipo=”resp”) then
                begin
                        cant_seg:=time_stamp - in_dtgrm.time;
                                // Diferencia en segundos entre salida y llegada
                        est[destino]. seg_acum:= est[destino]. seg_acum+cant_seg;
                        est[destino]. cant:= est[destino]. cant+1;
                end;
        end;
end.
```

Problema 2:

Se desea establecer una conexión segura sobre un protocolo tipo TCP/IP, buscando privacidad en la misma, y verificando que quien se conecta, es el equipo que dice ser.

El servidor cuenta con las primitivas para el manejo de TCP/IP, *a_open*, *p_open*, *send*, *receive* y *close*. Además el servidor cuenta con un puerto conocido de TCP/IP *secure_port*, donde se establecen las conexiones.

El servidor cuenta además con un servicio para encriptar/desencriptar, con una clave pública, y que conoce la clave pública de todos sus clientes. Para acceder a este servicio, se utilizan las siguientes rutinas:

desencript(msg_in,msg_out)	Devuelve en *msg_out*, el mensaje pasado por el parámetro *msg_in*, desencriptado adecuadamente en el receptor.
encript(destino,msg_in,msg_out)	Devuelve en *msg_out* ,el mensaje *msg_in* encriptado con la clave pública del *destino*.

Se pide: Especificar en un lenguaje de alto nivel, un protocolo sobre TCP/IP, el cual se conecta, le entregan mensajes, y los devuelve, y cierra la conexión, pero realizando la transmisión de los mismos con los requerimiento exigidos, sabiendo que puede existir el intercambio inicial de mensajes.

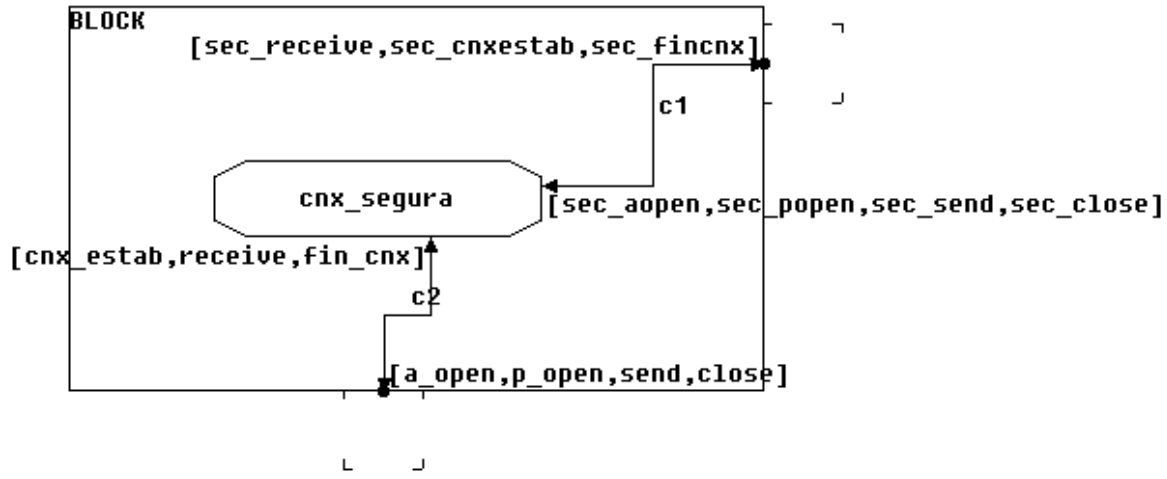
Solución:

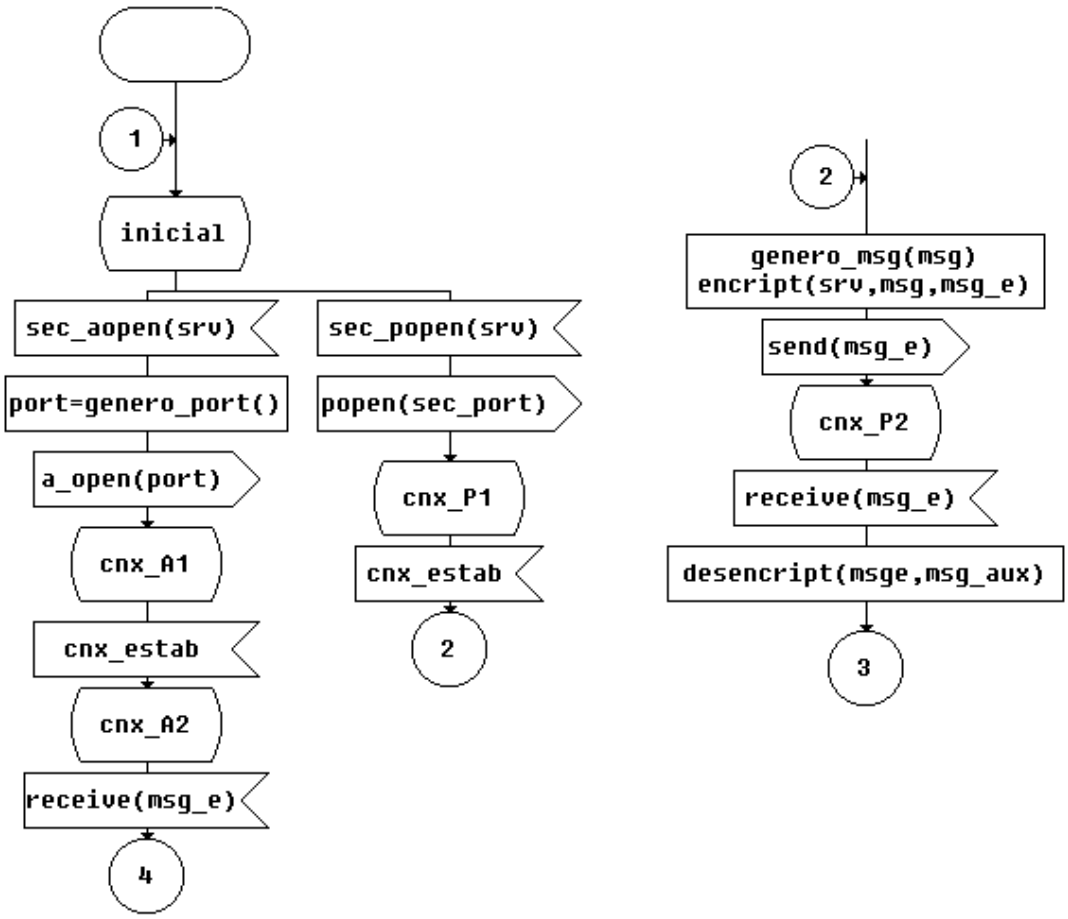
Se plantea la siguiente solución, para establecer conexiones con los requerimientos de seguridad especificados:

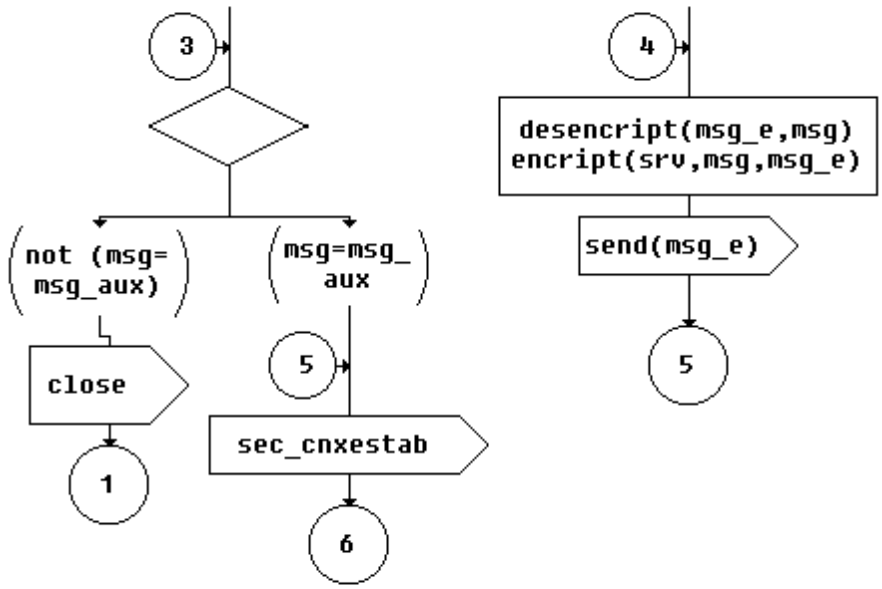
- Todos los mensajes intercambiados, se realizarán encriptados, con la clave publica del destinatario.
- Para verificar la autenticidad del que solicita la conexión, realizo los siguiente, en la instancia de conexión, el lado pasivo de la conexión genera un mensaje aleatorio, lo encripta con la password del lado activo de la conexión, el lado activo al recibirlo, lo desencripta según su clave publica, y lo encripta con la clave publica del lado pasivo, envia el mensaje, y al recibirlo el lado pasivo, si el mensaje original coincide con el mensaje llegado desencriptado, entonces es quien dice ser.

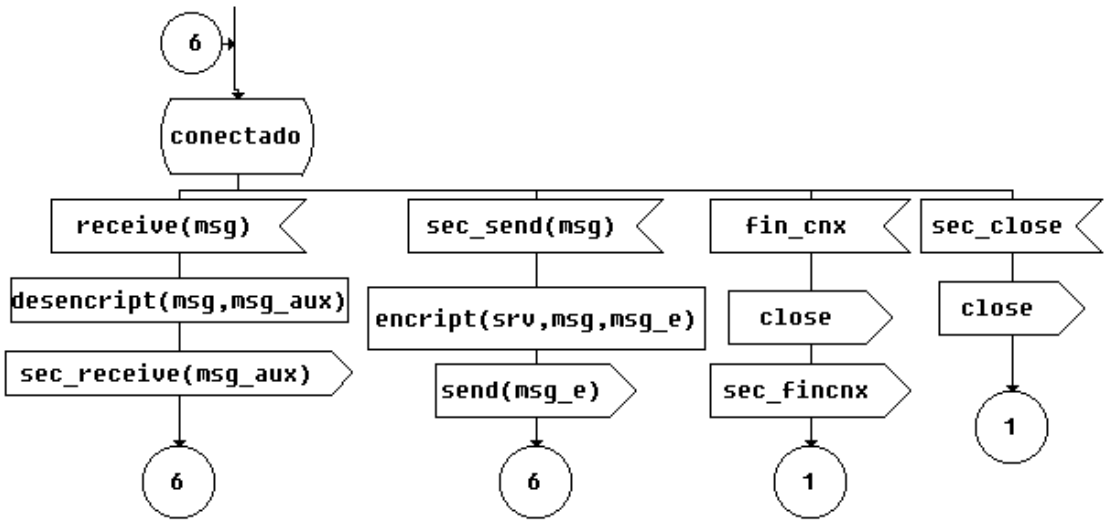
Las primitivas a implementar son las siguientes:

- sec_aopen realiza un open activo seguro
- sec_popen realiza un open pasivo seguro
- sec_send realiza un send seguro
- sec_receive realiza un receive seguro
- sec_close cierra una conexión segura.


BLOCK
[sec_receive,sec_cnxestab,sec_fincnx]
c1
cnx_segura
[sec_aopen,sec_popen,sec_send,sec_close]
[cnx_estab,receive,fin_cnx]
c2
[a_open,p_open,send,close]


1
inicial
sec_aopen(srv)
port=genero_port()
a_open(port)
cnx_A1
cnx_estab
cnx_A2
receive(msg_e)
4
sec_popen(srv)
popen(sec_port)
cnx_P1
cnx_estab
2
2
genero_msg(msg)
encript(srv,msg,msg_e)
send(msg_e)
cnx_P2
receive(msg_e)
desencript(msge,msg_aux)
3


3
not (msg= msg_aux)
msg=msg_ aux
close
1
5
sec_cnxestab
6
4
desencript(msg_e,msg)
encript(srv,msg,msg_e)
send(msg_e)
5


6
conectado
receive(msg)
desencript(msg,msg_aux)
sec_receive(msg_aux)
6
sec_send(msg)
encript(srv,msg,msg_e)
send(msg_e)
6
fin_cnx
close
sec_fincnx
1
sec_close
close
1

Problema 3:

Sea un protocolo de ruteo en redes, que busca el pasaje de información de rutas, a otros ruteadores. La distancia de las distintas rutas es medida en *hops*, cantidad de saltos en la red, y se consideran como máximo 15 saltos.

La información disponible en cada ruteador, tiene el siguiente formato:

dirección red	*dirección de nodo*	*nro de saltos*	*puerto salida*

La *dirección red*, contiene la dirección de la red a acceder. La *dirección de nodo*, es la dirección dentro de la subred, o 0 si se está considerando toda la subred. El *puerto salida* contiene la dirección IP del puerto de salida hacia la red.

El ruteador además conoce los diferentes enlaces que tiene activos, para conectarse con el resto de la red.

Se pide: Especificar un protocolo, que permita a un ruteador, interrogar sobre todas, o una ruta en particular, definida en el equipo, y con la respuesta recibida se actualicen las rutas existentes, utilizando un servicio de datagramas existentes.

Solución:

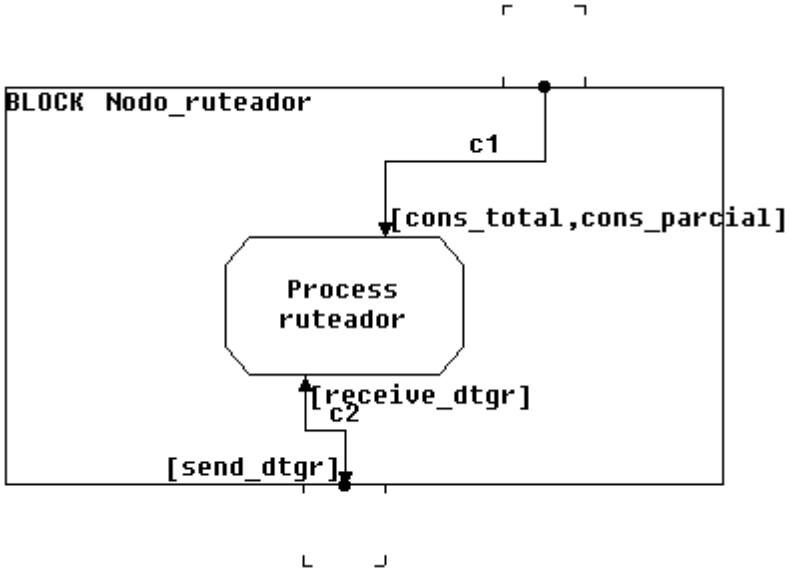
El protocolo a especificar tiene las siguientes características:

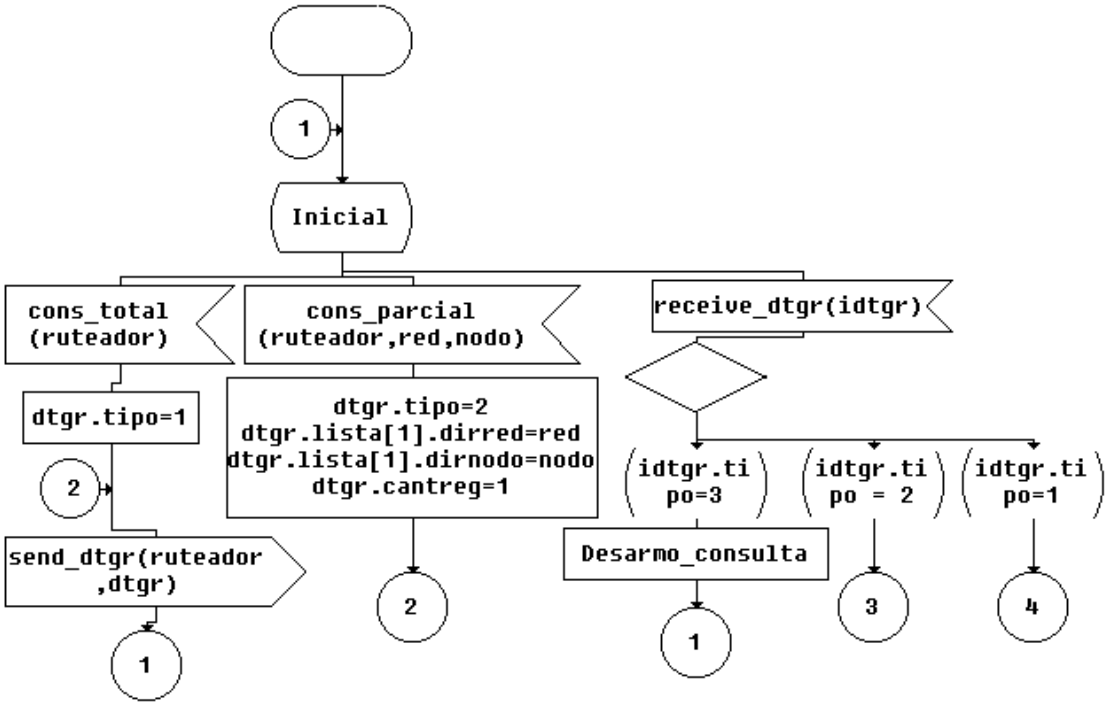
- Voy a tener dos primitivas cons_total(ruteador), que transfiere todas las rutas, y cons_parcial(ruteador,red,nodo) que transfiere las rutas hacia este nodo.
- En el ruteador esta ejecutando un proceso que ante las consultas, devuelve los resultados.
- Ante las respuestas de *cons_total*, y *cons_parcial*, actualiza la tabla del ruteado colocando la dirección red, direcciòn de nodo, nro de saltos (el resultado de consulta +1), y en puerto salida la dirección del enlace de salida.
- Cuento con el servicio de datagramas *send_dtgr*, y *receive_dtgr*.

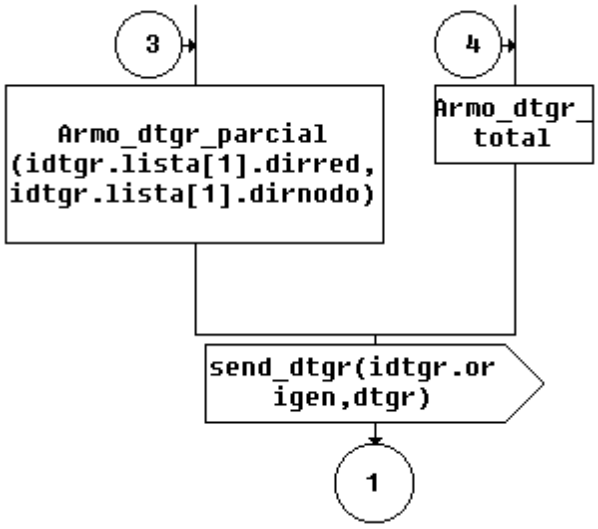
Formato de los datagramas es el siguiente:

```
struct {
        tipo_comando integer;   // tipo 1 requerimiento consulta total
                                // tipo 2 requerimiento consulta parcial
                                // tipo 3 respuesta consulta
        cant_reg integer:
        lista registro[cant_reg] ;
}datagrama;

struct registro {
        dir_red char[100];
        dir_nodo char[100];
        hops integer;
};
```


BLOCK Nodo_ruteador
c1
[cons_total,cons_parcial]
Process
ruteador
[receive_dtgr]
c2
[send_dtgr]


1
Inicial
cons_total
(ruteador)
cons_parcial
(ruteador,red,nodo)
receive_dtgr(idtgr)
dtgr.tipo=1
dtgr.tipo=2
dtgr.lista[1].dirred=red
dtgr.lista[1].dirnodo=nodo
dtgr.cantreg=1
idtgr.ti
po=3
idtgr.ti
po = 2
idtgr.ti
po=1
2
send_dtgr(ruteador
,dtgr)
Desarmo_consulta
2
3
4
1
1




```
Armo_dtgr_total
// Arma un datagrama con todos los registros de la
// Tabla_ruteo del nodo.
begin
        dtgr.cantreg:=cantidad_registros_tabla_ruteo;
        for i=1 to dtgr.cantreg
        begin
                dtgr.lista(i).dirred:=Tabla_ruteo(i).dirred;
                dtgr.lista(i).dirnodo:=Tabla_ruteo(i).dirnodo;
                dtgr.lista(i).hops:=Tabla_ruteo(i).hops;
        end;
end.


Armo_dtgr_parcial(red,nodo)
// Arma un datagrama con todos los registros de la
// Tabla_ruteo que se corresponden con la red y el
// nodo buscado.
begin
        j:=0;
        for i=1 to cantidad_registros_tabla_ruteo
        begin
                if (red= Tabla_ruteo(i).dirred and
                        nodo= Tabla_ruteo(i).dirnodo) then
                begin
                        dtgr.lista(j).dirred:=Tabla_ruteo(i).dirred;
                        dtgr.lista(j).dirnodo:=Tabla_ruteo(i).dirnodo;
                        dtgr.lista(j).hops:=Tabla_ruteo(i).hops;
                end;
        end;
        dtgr.cantreg:=j;
end.
```

```
Desarmo_consulta
// Actualiza la tabla local de ruteo, con la
// información llegada en el datagrama.
begin
        j=1;
        for i=1 to idtgr.cantreg
        begin
                while (Tabla_ruteo(j).dirred <> idtgr.lista(i).dirred and
                        Tabla_ruteo(j).dirnodo <> idtgr.lista(i).dirnodo and
                        not fin Tabla_ruteador)
                        j:=j+1;
                if (fin Tabla_ruteador) then
                        // hay que insertarlo en la tabla del ruteador
                        // incremento hops en 1 y coloco la dir_enlace de salida.
                        insert(Tabla_ruteador, idtgr.lista(i).dirred, idtgr.lista(i).dirnodo,
                                idtgr.lista(i).hops+1,dir_enlace(idtgr.origen));
                else
                        // hay que modificar la tabla del ruteador
                        // incremento hops en 1 y coloco la dir_enlace de salida.
                        modify(Tabla_ruteador,
dtgr.lista(i).hops+1,dir_enlace(idtgr.origen));
        end;
end.
```