Prácticas de Autómatas y Lenguajes. Curso 2017/18

Práctica 1: relaciones como matrices binarias

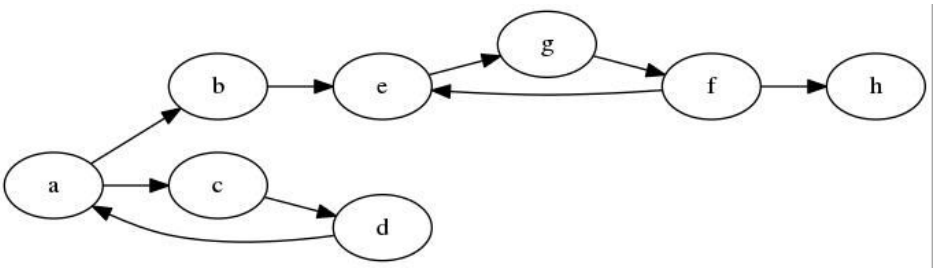
Descripción del enunciado

Esta práctica tiene dos objetivos principales:

1. Que desarrolles con el lenguaje de programación C los tipos de datos necesarios para gestionar relaciones como matrices binarias. Tu profesor de prácticas te explicará que utilizarás estas relaciones para gestionar las transiciones λ_x^x
2. Que esa librería esté preparada para integrarse en el futuro en un simulador de autómatas a pila.

Relaciones binarias homogéneas como matrices binarias

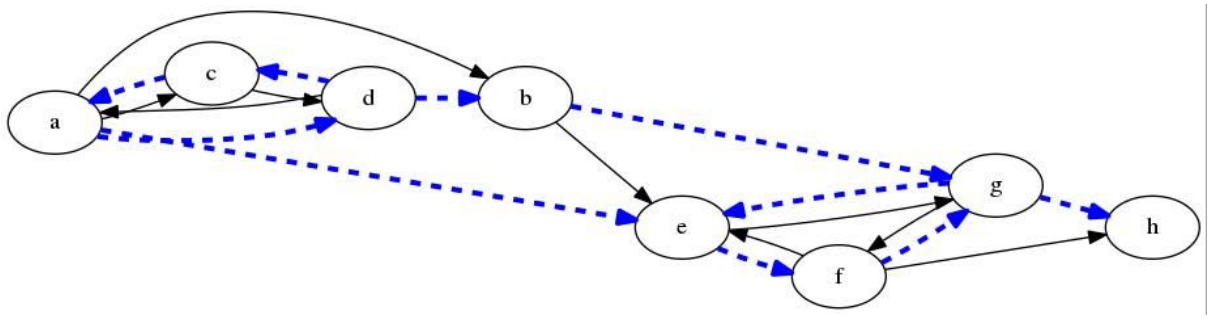
En general cualquier relación puede representarse como una matriz booleana que tiene como entradas los elementos del conjunto y cuyas posiciones guardan un valor lógico **cierto** o **uno** cuando los elementos correspondientes a esa posición están relacionados y **falso** o **cero** cuando no lo están.
Las relaciones que nos interesan en esta parte de la práctica son las **homogéneas**, es decir, aquellas en las que los conjuntos relacionados son el mismo. Un ejemplo que seguro que has analizado en algún momento de tu carrera es la relación de accesibilidad entre los nodos de un grafo. Estarás acostumbrado a referirte a la matriz que describe esa relación como *matriz de adyacencia*.
A continuación tienes un ejemplo de grafo



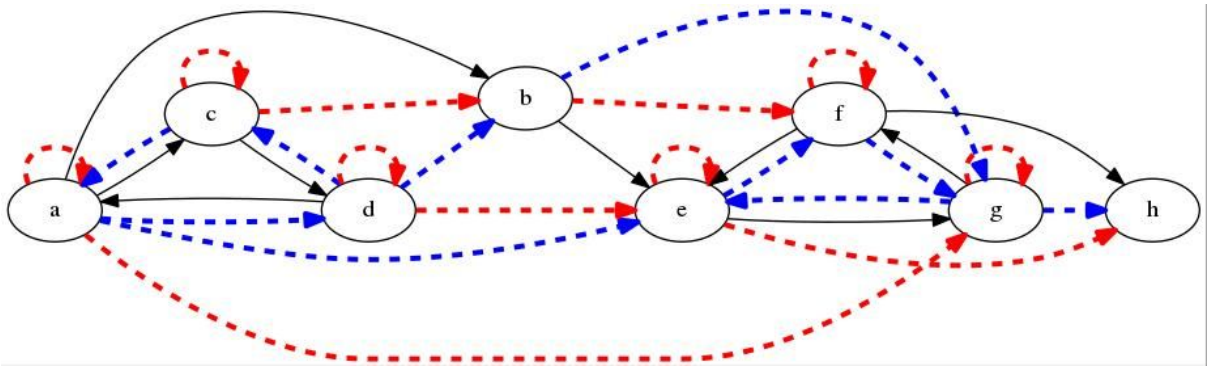
Cuya matriz de adyacencia sería la siguiente

	a	b	c	d	e	f	g	h
a	0	1	1	0	0	0	0	0
b	0	0	0	0	1	0	0	0
c	0	0	0	1	0	0	0	0
d	1	0	0	0	0	0	0	0
e	0	0	0	0	0	0	1	0
f	0	0	0	0	1	0	0	1
g	0	0	0	0	0	1	0	0
h	0	0	0	0	0	0	0	0

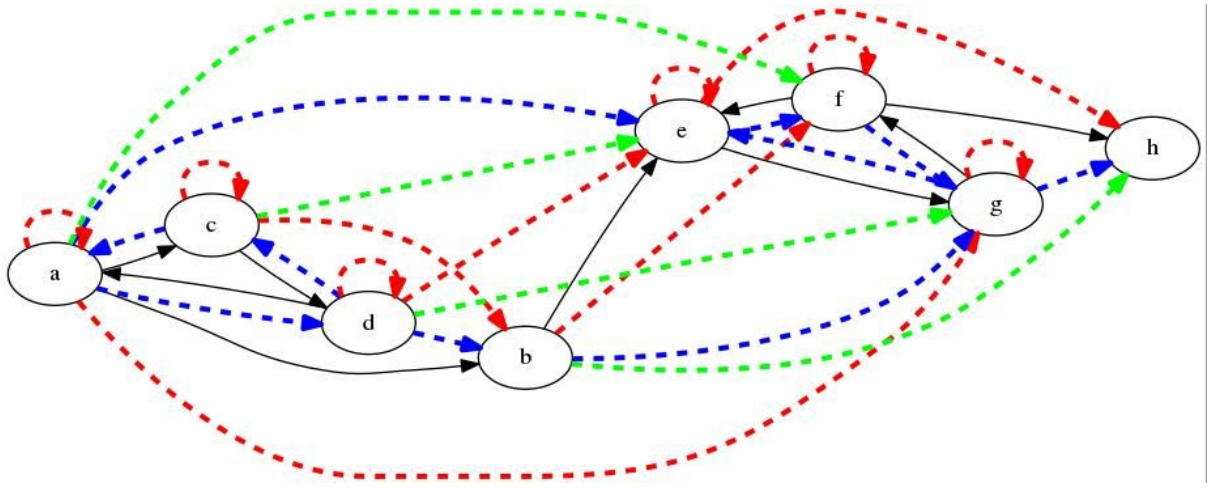
Es frecuente que resulte de interés en este tipo de relaciones preguntarse por su cierre transitivo. En el caso de la accesibilidad del grafo. Esto significa que interesa saber a qué otros nodos se puede acceder en caminos no directos. Por ejemplo, de un paso más a partir de los que aparecen explícitamente en el grafo. Puedes comprobar que esos caminos son los que se añaden al grafo anterior y que se muestran en la siguiente figura con líneas punteadas.



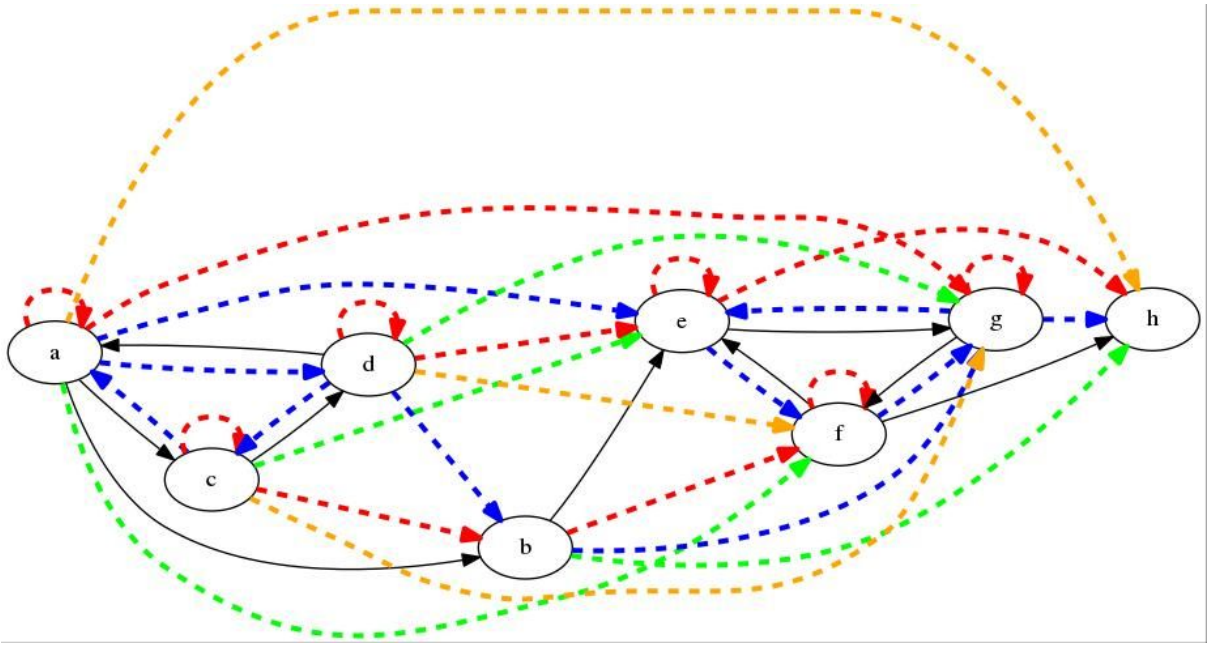
También puedes comprobar que si ahora se añadieran los que dan un paso más son los que se muestran a continuación en color rojo



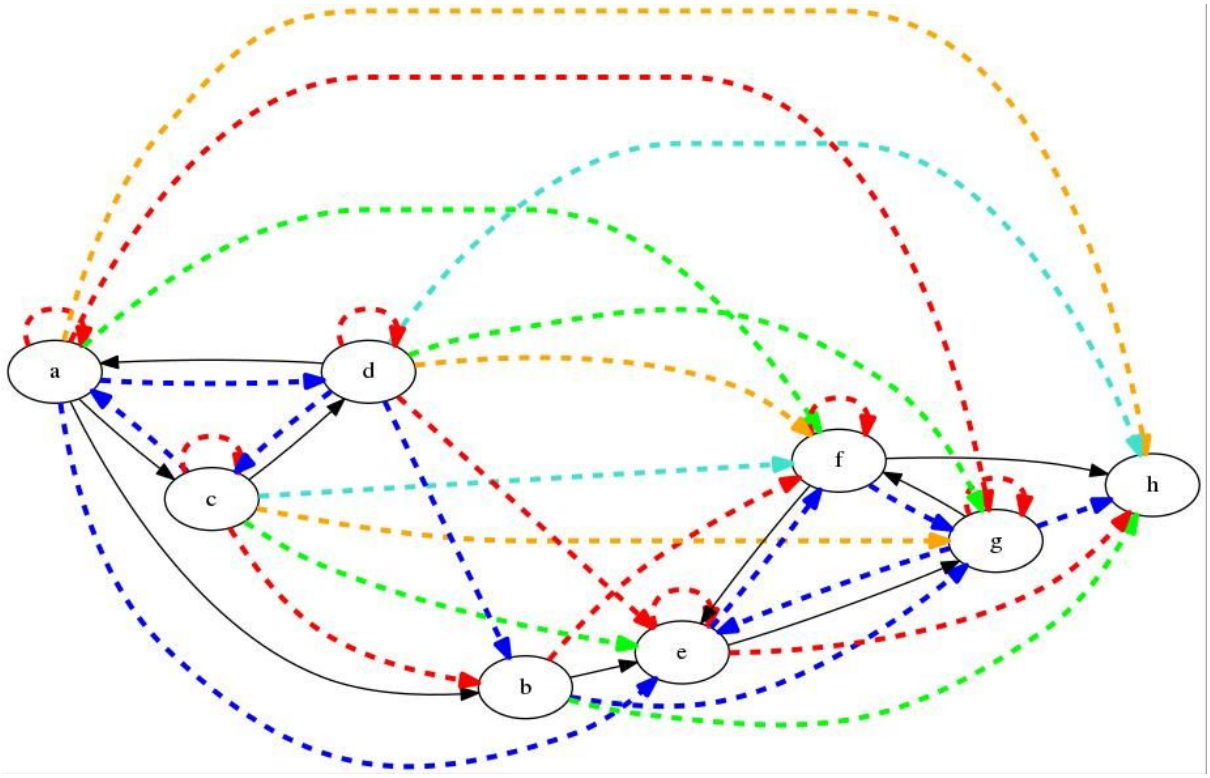
Y si se da un paso más se añadirían los que aparecen ahora en verde



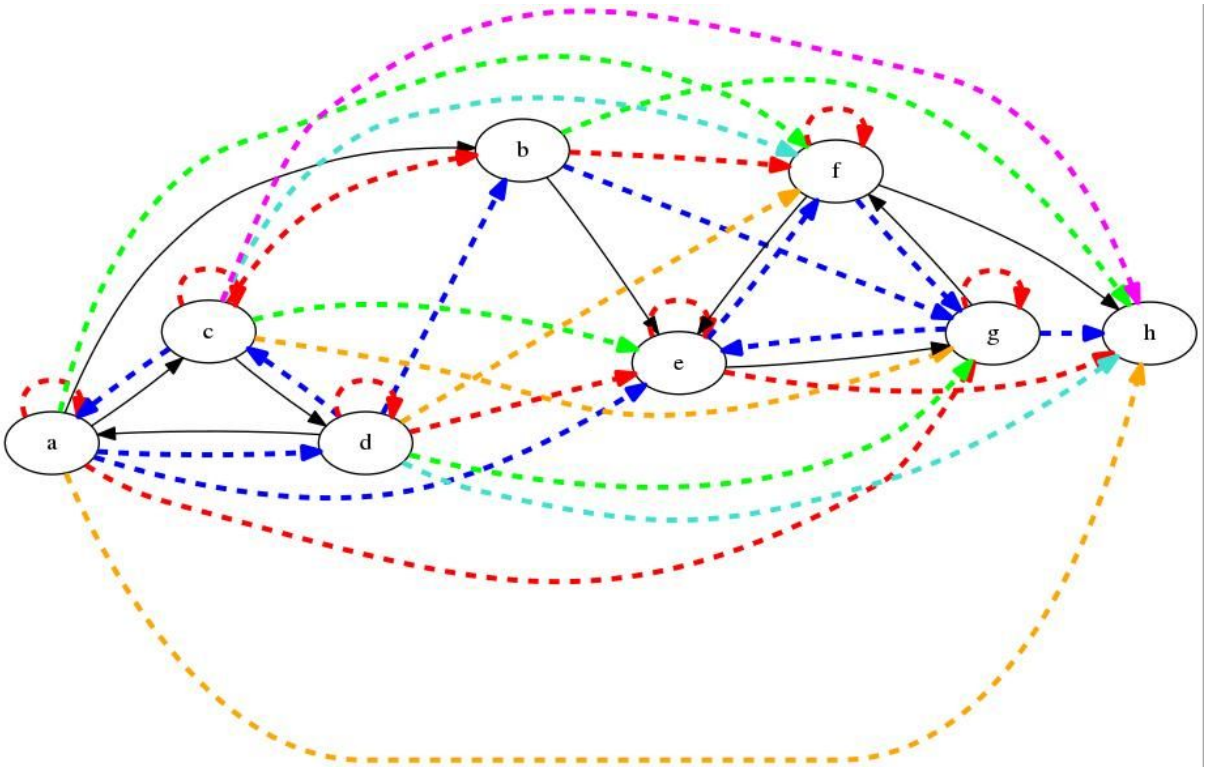
Si se da un paso más se añadirían los que aparecen en naranja



Y si se da un paso más aparecen las que se muestran en azul turquesa.



Y, finalmente, sólo faltaría añadir los que dan un paso más, que aparecen en morado



Puedes comprobar que ya no puedes añadir más caminos indirectos al grafo.

El objetivo de esta práctica es que implementes un tipo de datos para manejar relaciones justamente en el sentido expuesto en el ejemplo anterior.

Lo usarás más adelante para calcular a qué conjunto de configuraciones (configuración no determinista) puedes transitar sin modificar la pila ni consumir símbolos de la entrada mediante transiciones del tipo $\lambda_{\lambda}^{\lambda}$.

Para ello te centrarás en el uso de las representaciones matriciales de las relaciones.

Fíjate que cada uno de esos pasos consiste en añadir los 1s que corresponden con las transiciones de más de un paso pendientes de añadir (en el ejemplo, respectivamente de 2, 3, 4, 5, 6 y 7)

A continuación te mostramos la matriz en la que el convenio de colores te indica los 1s que añadimos en cada paso.

	a	b	c	d	e	f	g	h
a	1	1	1	1	1	1	1	1
b	0	0	0	0	1	1	1	1
c	1	1	1	1	1	1	1	1
d	1	1	1	1	1	1	1	1
e	0	0	0	0	1	1	1	1
f	0	0	0	0	1	1	1	1
g	0	0	0	0	1	1	1	1
h	0	0	0	0	0	0	0	0

Un resultado que igual recuerdes de la asignatura de álgebra, garantiza que es suficiente para añadir los 1s en cada uno de estos casos, con multipilcar la matriz de la relación hasta el paso i-1 por la matriz de partida (en ese orden) y acumular los 1s que aparezcan de nuevas. Esta operación es el producto de matrices en el que la disyunción e intersección lógica hacen el papel de la suma y el producto en el producto de matrices numéricas. Llegará un momento (como muy tarde tras tantos pasos como filas o columnas hay en la matriz) en la que ya no añadirás más 1s y habrás obtenido la matriz del cierre transitivo.

El tipo de dato relación

Tu relación debe guardar suficiente información como para poder almacenar simultáneamente la relación original y el cierre transitivo.

Un detalle específico de tu librería es que se te pide que añadas un nombre (`char *`)

Se te recomienda utilizar algún tipo de convenio para que el nombre refleje el origen de la relación.

Por ejemplo, el programa principal que tienes más adelante puede resumirse de la siguiente manera:

- Se crea una relación de nombre “grafo1” (es la relación r1)
- Posteriormente se crea otra (la r2) que es una copia de la primera.
- Posteriormente se calcula el cierre transitivo de la primera.

El convenio de nombres elegido es el siguiente:

- Cuando se crea una relación proporcionando su nombre, éste es el utilizado (por ejemplo r1 tiene como nombre “grafo1”)
- Cuando se copia una relación, el nombre de la copia es el mismo que la original al que se le añade un carácter “‘“ final (en nuestro ejemplo, r2 tiene como nombre “grafo1’”)
- Cuando se cierra una relación, se añade algún indicador a su nombre
    - Por ejemplo se te sugiere que añadas “+”
    - O tal vez, como en el programa que se te muestra, un “+” por cada paso que has dado al hacer el cierre. Este último convenio es un poco más farragoso pero conserva más información.

A continuación se describe las cabeceras de las funciones que debes implementar.

```
typedef struct _Relacion Relacion;

Relacion * relacionNueva(char * nombre, int num_elementos);
/*
Crea una nueva relación a la que se le da como nombre el proporcionado como
argumento para un conjunto con el número de elementos que se prorpociona como
argumento.
Debe reservar memoria propia para todas las componentes que decidas utilizar en
la relación.
*/
void relacionImprime(FILE * fd, Relacion * p_r);
/*
Muestra por el FILE * la relación. Puedes suponer el formato de salida utilizado
en los ejemplos.
*/
void relacionElimina(Relacion * p_r);
/*
Libera toda la memoria asociada con la relación.
*/
Relacion * relacionCopia(Relacion * p_r1);
```

```
/*
Genera en memoria nueva una copia de la relación proporcionada como argumento y
la devuelve.
*/
Relacion * relacionInserta(Relacion * p_r, int i, int j);
/*
Modifica la relación proporcionada como argumento para que tenga constancia de
que el elemento i está relacionado con el j.
Se está suponiendo que los elementos están dispuestos en un orden precondebido y
conocido por el usuario de la librería.
Una vez modificada, la relación es también devuelta.
*/

int  relacionTamano(Relacion * p_r);
/*
Devuelve el cardinal del conjunto sobre el que está definida la relación.
*/

Relacion * relacionCierreTransitivo(Relacion * p_r);

/*
Modifica la relación para conservar el ella su cierre transitivo. Devuelve la
relación como retorno.
*/
int  relacionIJ(Relacion * p_r,int i, int j);
/*
Devuelve 1 si el elemento i está relacionado originalmente con el j y 0 en caso
contrario.
*/
int relacionCierreIJ(Relacion * p_r, int i, int j);
/*
Devuelve 1 si el elemento i está relacionado (en el cierre transitivo) con el j y
0 en el caso contrario.
*/
```

Ejemplos

A continuación se te muestra un programa principal de ejemplo.

```
#include "relacion.h"


int main(int argc, char ** argv)
{
       Relacion * r1;
       Relacion * r2;

       int i=0;

       r1 = relacionNueva("grafo1",8);


       relacionInserta(r1,0,1);
       relacionInserta(r1,0,2);
       relacionInserta(r1,1,4);
       relacionInserta(r1,2,3);
       relacionInserta(r1,3,0);
       relacionInserta(r1,4,6);
       relacionInserta(r1,5,4);
       relacionInserta(r1,5,7);
       relacionInserta(r1,6,5);
```

```
        fprintf(stdout,"R1\n");
        relacionImprime(stdout,r1);

        r2 = relacionCopia(r1);

        fprintf(stdout,"R2 = copia(R1)\n");
        fprintf(stdout,"R2\n");
        relacionImprime(stdout,r2);

        fprintf(stdout,"R1*\n");
        relacionCierreTransitivo(r1);

        relacionImprime(stdout,r1);



        relacionElimina(r1);
        relacionElimina(r2);

        return 0;
}
```

Y una salida que podría producir.

```
R1
grafo1={
                [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
                CIERRE
        [0]     0       1       1       0       0       0       0       0
        [1]     0       0       0       0       1       0       0       0
        [2]     0       0       0       1       0       0       0       0
        [3]     1       0       0       0       0       0       0       0
        [4]     0       0       0       0       0       0       1       0
        [5]     0       0       0       0       1       0       0       1
        [6]     0       0       0       0       0       1       0       0
        [7]     0       0       0       0       0       0       0       0

                RELACION INICIAL i
        [0]     0       1       1       0       0       0       0       0
        [1]     0       0       0       0       1       0       0       0
        [2]     0       0       0       1       0       0       0       0
        [3]     1       0       0       0       0       0       0       0
        [4]     0       0       0       0       0       0       1       0
        [5]     0       0       0       0       1       0       0       1
        [6]     0       0       0       0       0       1       0       0
        [7]     0       0       0       0       0       0       0       0
}
R2 = copia(R1)
R2
grafo1'={
                [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
                CIERRE
        [0]     0       1       1       0       0       0       0       0
        [1]     0       0       0       0       1       0       0       0
        [2]     0       0       0       1       0       0       0       0
        [3]     1       0       0       0       0       0       0       0
        [4]     0       0       0       0       0       0       1       0
        [5]     0       0       0       0       1       0       0       1
        [6]     0       0       0       0       0       1       0       0
        [7]     0       0       0       0       0       0       0       0
```

```
               RELACION INICIAL i
       [0]     0       1       1       0       0       0       0       0
       [1]     0       0       0       0       1       0       0       0
       [2]     0       0       0       1       0       0       0       0
       [3]     1       0       0       0       0       0       0       0
       [4]     0       0       0       0       0       0       1       0
       [5]     0       0       0       0       1       0       0       1
       [6]     0       0       0       0       0       1       0       0
       [7]     0       0       0       0       0       0       0       0
}
R1*
grafo1+++++++={
               [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
               CIERRE
       [0]     1       1       1       1       1       1       1       1
       [1]     0       0       0       0       1       1       1       1
       [2]     1       1       1       1       1       1       1       1
       [3]     1       1       1       1       1       1       1       1
       [4]     0       0       0       0       1       1       1       1
       [5]     0       0       0       0       1       1       1       1
       [6]     0       0       0       0       1       1       1       1
       [7]     0       0       0       0       0       0       0       0

               RELACION INICIAL i
       [0]     0       1       1       0       0       0       0       0
       [1]     0       0       0       0       1       0       0       0
       [2]     0       0       0       1       0       0       0       0
       [3]     1       0       0       0       0       0       0       0
       [4]     0       0       0       0       0       0       1       0
       [5]     0       0       0       0       1       0       0       1
       [6]     0       0       0       0       0       1       0       0
       [7]     0       0       0       0       0       0       0       0
}
```

En esta práctica se pide:

- Cada grupo debe realizar una entrega
- Tu profesor te indicará la tarea Moodle donde debes hacer la entrega.
- Es imprescindible que mantengas los nombres especificados en este enunciado **sólo para las funciones que se piden de manera obligatoria**.
- Cualquier práctica que no compile (con los flags de compilación -Wall -ansi -pedantic), o no ejecute correctamente será considerada como no entregada.
- Cualquier práctica que deje “lagunas de memoria” será penalizada de manera explícita en función de la gravedad de esas lagunas.