

**Grupo Miércoles 17:00-19:00 semanas A**

**– Práctica 4 –**

**Autor:** Víctor Marteles Martínez

**NIP:928927**

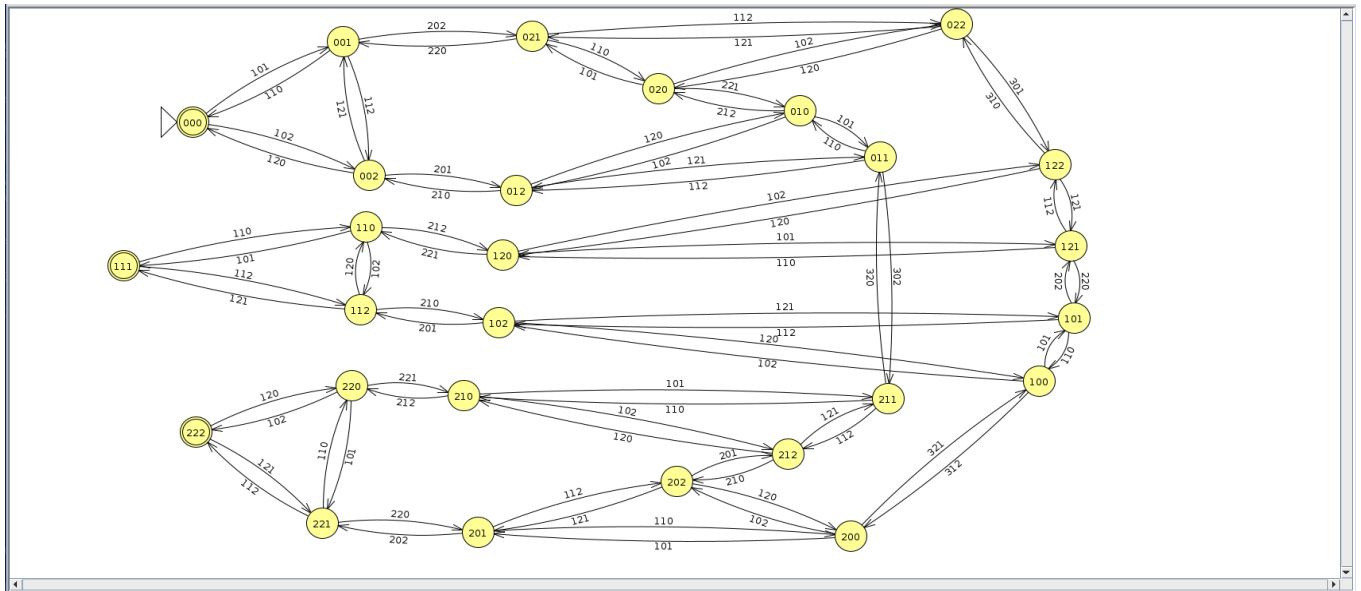
**Autor:** Javier Martínez Virto

**NIP:930853**

## Ejercicio 1:

Para resolver el ejercicio 1, hemos seleccionado la siguiente codificación:

- **Estados:** Los nombres de los estados se componen de tres números de rango [0-2]. La posición de cada número representa el disco de tamaño posición y su valor del número representa el palo en el que se ubica. Por ejemplo, si tenemos el estado 201, la situación actual es que el disco de tamaño 3 está en el palo 2; el disco de tamaño 2 está en el palo 0; y el disco de tamaño 1 está en el palo 1.
- **Transiciones:** Las transiciones se componen de tres números de rango [0-2]. Su significado es el siguiente (teniendo xyz como nodo): muevo el disco x, que estaba en y, y lo pongo en z.



## Ejercicio 2:

### 1. Resumen

La gramática elegida para ha sido la de descripción de grafos DOT, que se compone de algo así:

```
graph un_automata
{
p0 ->p1(a);
p1 ->p1(a),p1(b),p2(b);
p2 ->p2(b),p0();
}
```

Nuestro lenguaje desarrollado de acuerdo a este formato es el siguiente:

```
grafo : GRAFO EOL INI EOL origen FIN EOL
      ;
```

```
origen : ESTADO FLECHA transiciones PCOMA EOL
      | origen ESTADO FLECHA transiciones PCOMA EOL
      ;
```

```
transiciones : ESTADO IPAR ESTADO FPAR
             | ESTADO IPAR ESTADO FPAR COMA transiciones
             ;
```

significado de los TOKENS:

- GRAFO: Nombre del grafo. ( e.g. graph un\_automata )
- EOL: End of line. ( '\n' )
- INI: Corchete comienzo de grafo. ( '{' )
- FIN: Corchete fin de grafo. ( '}' )
- ESTADO: Estado del autómeta. ( e. g. '000' )
- FLECHA: Flecha hacia transiciones ( '->' )
- PCOMA: Punto y coma de final de transiciones. ( ';' )
- IPAR: Inicio paréntesis. ( '(' )
- FCOMA: Fin paréntesis. ( ')' )
- COMA: Coma que separa las transiciones. ( ',' )

La gramática de nuestro grafo:

```
graph ej1prac4
{
000 ->001(101),002(102);
001 ->021(202),000(110),002(112);
002 ->000(120),012(201),001(121);
110 ->120(212),111(101),112(102);
112 ->110(120),111(121),102(210);
012 ->010(120),002(210),011(121);
021 ->020(110),001(220),022(112);
111 ->110(110),112(112);
222 ->220(120),221(121);
221 ->222(112),220(110),201(220);
220 ->210(221),221(101),222(102);
102 ->101(121),100(120),112(201);
120 ->122(102),121(101),110(221);
201 ->221(202),202(112),200(110);
211 ->212(112),210(110),011(320);
210 ->220(212),212(102),211(101);
011 ->012(112),010(110),211(302);
022 ->020(120),122(301),021(121);
100 ->200(312),102(102),101(101);
122 ->121(121),120(120),022(310);
200 ->100(321),201(101),202(102);
010 ->011(101),012(102),020(212);
```

```

020 ->010(221),022(102),021(101);
101 ->100(110),102(112),121(202);
121 ->101(220),120(110),122(112);
202 ->201(121),200(120),212(201);
212 ->202(210),211(121),210(120);
}

```

### Ejercicio 3:

#### 1. Resumen

Para solucionar el problema del ejercicio 3 hemos escrito un programa cuya parte central es la siguiente:

```

while (strcmp(pot[elni][eFin], "") == 0 && k < DIM) {
    multiplicar(tablaTr, anterior, pot); // pot = C * C^(k-1)
    copiar(pot, anterior);              // anterior = C^k
    k++;
}

```

Siendo “tablaTr” la tabla de adyacencia, “anterior” la potencia k-1 de C y “pot” la potencia k de C.

El bucle “while” sigue mientras NO haya camino de longitud k desde “elni” hasta “eFin” y, además, que k no supere DIM (dimensión de la matriz de adyacencia).

Dentro del bucle, primero se hace “multiplicar(tablaTr, anterior, pot)” (básicamente  $pot = C * C^{(k-1)}$ ). Después, se hace “copiar(pot, anterior)” ( $anterior = C^{(k-1)}$ ). Por último, hacemos “k++” (incrementando la longitud del camino).

#### 2. Pruebas

./th <thP3D3.txt

ENTRADA	SALIDA
Nodo inicial : 000 Nodo final : 212	Movimientos: 102-201-121-302-112
Nodo inicial : 001 Nodo final : 021	Movimientos: 202
Nodo inicial : 111 Nodo final : 220	Movimientos: 112-210-120-312-101-202-110

--	--