



# Algoritmos y Programación

Programación con  
Restricciones 3

# Programación con restricciones

## REIFICACIÓN

# Reificación

- Ejemplo, series mágicas
- Una serie  $S=(S_0, ..., S_n)$  es mágica si  $S_i$  representa el número de ocurrencias de  $i$ .

	0	1	2	3	4
Occurrences	?	?	?	?	?

- Posible solución: (2,1,2,0,0)
- El objetivo es escribir un programa (modelo) que nos permita obtener series de tamaño  $n$ .

# Series mágicas

```
int n = 5;  
range D = 0..n-1;  
var(int) series[D] in D;  
solve {  
    forall(k in D)  
        series[k] = sum(i in D) (series[i]=k);  
}
```

```
include "globals.mzn";  
  
int: n=5;  
set of int: D = 0..n-1;  
array[D] of var D: series;  
  
constraint  
    forall(k in D)  
        ( series[k] = (sum(i in D)(series[i]=k)));  
  
solve satisfy;
```

# Reificación

La **reificación** (*reification* en inglés) de una restricción, consiste en transformar o crear otra restricción con una variable binaria 0/1. La variable toma el valor 1 si la restricción inicial se cumple y 0 en caso contrario.

```
int n = 5;  
range D = 0..n-1;  
var(int) series[D] in D;  
solve {  
  forall(k in D)  
    series[k] = sum(i in D) (series[i]=k);  
}
```

# Reificación

```
int n = 5;  
range D = 0..n-1;  
var(int) series[D] in D;  
solve {  
  forall(k in D)  
    series[k] = sum(i in D) (series[i]=k);  
}
```

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);  
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);  
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);  
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);  
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
```

Expansión del *forall*

```
1      = 0 + (series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);  
series[1] = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);  
series[2] = 0 + (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);  
series[3] = 0 + (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);  
series[4] = 0 + (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
```

Si por ejemplo,  $\text{series}[0] = 1$

```
1      = (series[2]=0)+(series[3]=0)+(series[4]=0);  
series[1] = 1 + (series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);  
series[2] = (series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);  
series[3] = (series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);  
series[4] = (series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
```

Como vemos, ahora  $\text{series}[1] > 0$ , por tanto podemos eliminar el término de la primera ecuación ( $\text{series}[1]=0$ ):

# Reificación

- Permite modelar reglas del tipo:
  - Al menos 2 de entre un grupo de restricciones debe cumplirse
  - La suma de estos valores debe estar 2 y 4
  - El número de dispositivos periféricos puede ser a lo sumo 5
  - ...

# Parejas estables

Tenemos dos conjuntos: mujeres y hombres, con el mismo número de mujeres que de hombres, y además cada uno tiene su propia lista de preferencias. La pregunta que debemos responder es:

¿Cuál es la mejor manera de emparejarlos para que las parejas sean estables?



# Parejas estables, ejemplo

María	Carlos	Marco	Juan
Ana	Juan	Marco	Carlos
Lucía	Juan	Carlos	Marco

*Matriz de preferencias de las mujeres*

Carlos	Ana	María	Lucía
Marco	María	Ana	Lucía
Juan	María	Lucía	Ana

*Matriz de preferencias de los hombres*

- Ejemplo de emparejamiento estable:  $\{(María, Carlos), (Ana, Marco), (Lucía, Juan)\}$
- Ejemplo de emparejamiento inestable:  $\{(María, Juan), (Ana, Marco), (Lucía, Carlos)\}$ 
  - *Pareja bloqueante:* (María, Carlos)

# Parejas estables

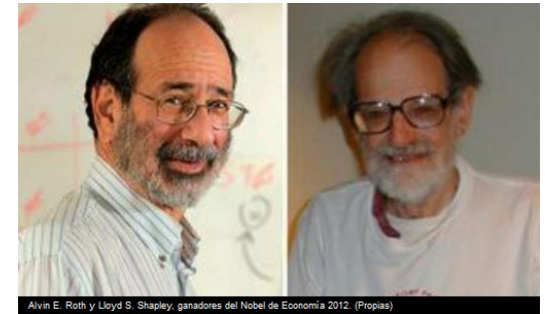
- Historia
  - Mediados del siglo XIX, EEUU
  - Los hospitales hacían ofertas a los estudiantes en los que estaban interesados
  - Los estudiantes por miedo a no conseguir mejores ofertas aceptaban la primera
  - Esto generaba todo tipo de problemas que se solucionó con un organismo intermedio
  - El estudiante debía enviar al organismo intermedio su lista de preferencias
  - Los hospitales debían hacer lo mismo
  - El problema era encontrar el emparejamiento perfecto. En el año 1951 se creó el Programa Nacional de Emparejamiento de Residentes (PNER)

# Parejas estables

- Gale y Shapley (1962) plantearon y resolvieron el problema sin conocer los trabajos del PNER.
- Obtener el número máximo de parejas estables es NP-Completo
- Investigaciones posteriores de Shapley, junto a Alvin Roth les vale el premio Nobel de economía en el año 2012.

“...por su trabajo sobre **asignaciones estables** y sus teorías de **rediseño de mercados económicos**.

Según subraya la Academia en su declaración, ambos abordaron un problema fundamental: cómo coordinar de la mejor forma posible los diferentes actores presentes en la sociedad, por ejemplo para que los alumnos sean asignados a escuelas o los órganos donados lleguen a los pacientes que necesitan un trasplante”



# Algoritmo de Gale y Shapley

Cada persona está libre

Mientras algún hombre  $h$  esté libre hacer

Comienzo

$m :=$  primera mujer de la lista de preferencias de  $h$  que no ha sido propuesta

Si  $m$  está libre entonces

Asignar  $h$  a  $m$

sino

Si  $m$  prefiere a  $h$  antes que a su pareja actual  $h'$

romper la relación  $m-h'$  y asociar  $m$  con  $h$

sino

$m$  rechaza a  $h$  y  $h$  permanece libre

Fin

El matching obtenido es estable.

# Ejemplo

- $H=\{h_0, h_1, h_2, h_3\}$  (hombres)
- $M=\{m_0, m_1, m_2, m_3\}$  (mujeres),

## Preferencias:

$h_0$ :	$m_0$ $m_1$ $m_2$ $m_3$
$h_1$ :	$m_2$ $m_1$ $m_3$ $m_0$
$h_2$ :	$m_2$ $m_1$ $m_3$ $m_0$
$h_3$ :	$m_1$ $m_2$ $m_3$ $m_0$

$m_0$ :	$h_0$ $h_2$ $h_3$ $h_1$
$m_1$ :	$h_0$ $h_1$ $h_2$ $h_3$
$m_2$ :	$h_1$ $h_2$ $h_0$ $h_3$
$m_3$ :	$h_3$ $h_0$ $h_2$ $h_1$

$M_1=\{(h_0, m_0), (h_1, m_2), (h_2, m_1), (h_3, m_3)\}$  es estable

# Emparejamientos estables

Los siguientes matchings son estables:

$$M_1 = \{(h_0, m_0), (h_1, m_1), (h_2, m_2), (h_3, m_3)\}$$

$$M_2 = \{(h_0, m_1), (h_1, m_0), (h_2, m_2), (h_3, m_3)\}$$

$$M_3 = \{(h_0, m_0), (h_1, m_1), (h_2, m_3), (h_3, m_2)\}$$

$$M_4 = \{(h_0, m_1), (h_1, m_0), (h_2, m_3), (h_3, m_2)\}$$

$$M_5 = \{(h_0, m_1), (h_1, m_3), (h_2, m_0), (h_3, m_2)\}$$

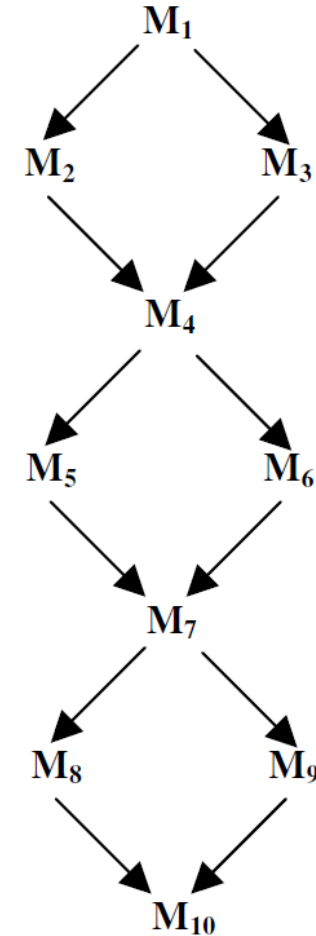
$$M_6 = \{(h_0, m_2), (h_1, m_0), (h_2, m_3), (h_3, m_1)\}$$

$$M_7 = \{(h_0, m_3), (h_1, m_2), (h_2, m_0), (h_3, m_1)\}$$

$$M_8 = \{(h_0, m_2), (h_1, m_3), (h_2, m_1), (h_3, m_0)\}$$

$$M_9 = \{(h_0, m_2), (h_1, m_3), (h_2, m_1), (h_3, m_0)\}$$

$$M_{10} = \{(h_0, m_3), (h_1, m_2), (h_2, m_1), (h_3, m_0)\}$$



# Parejas estables: Programación declarativa

wrank

María	Carlos	Marco	Juan
Ana	Juan	Marco	Carlos
Lucía	Juan	Carlos	Marco

*Matriz de preferencias de las mujeres*

mrnk

Carlos	Ana	María	Lucía
Marco	María	Ana	Lucía
Juan	María	Lucía	Ana

*Matriz de preferencias de los hombres*

```
include "globals.mzn";
```

```
enum Women = {Maria, Ana, Lucia};
enum Men = {Carlos, Marco, Juan};
```

```
array[Women, Men] of int: wrnk=
array[Men, Women] of int: mrnk=
```

```
array[Men] of var Women: wife;
array[Women] of var Men: husband;
```

Por ejemplo, mrnk[Carlos, Ana] = 1

parámetros

	María (C,M,J)	Ana (C,M,J)	Lucía (C,M,J)
Carlos (M,A,L)	1	3	2
Marcos (M,A,L)	2	2	3
Juan (M,A,L)	3	1	1

# Parejas estables

array de variables de decisión

wife[m]: Variable de decisión

```
constraint
forall(m in Men)(husband[wife[m]] = m) /\
forall(w in Women)(wife[husband[w]] = w) /\
```

**inverse(wife, husband)**

El matrimonio es recíproco, si x está casado con y, y está casado con x,  
o lo que es lo mismo:  $husband[wife[m]] = m$  y  $wife[husband[w]] = w$ .

Compiling stable\_matching0.mzn

Running stable\_matching0.mzn

wife : [Maria, Lucia, Ana]

husband: [Carlos, Juan, Marco]

-----

Finished in 154msec

Al uso de variables de decisión como  
índices de array se le conoce en inglés  
como *element constraint*



# Parejas estables

constraint

```
inverse(wife, husband) /\nforall(m in Men, w in Women)\n    (wrank[w,m] < wrank[w,husband[w]] -> mrank[m,wife[m]] < mrank[m,w]) /\nforall(w in Women, m in Men)\n    (mrnk[m,w] < mrank[m,wife[m]] -> wrank[w,husband[w]] < wrank[w,m]);
```

solve satisfy;

1 -> mayor prioridad

Si  $m$  prefiere a  $w$ , por encima de su esposa

entonces

$w$  debe preferir a su esposo antes que a  $m$

# Propagación y poda con *element constraint*

## Ejemplo:

- $x, y$ : variables de decisión  $x \in \{3,4,5\}$   
 $y \in \{0,1,2,3,4,5\}$
- $c$  es un array de enteros (en casos más complejos, podría ser un array de variables)

Array c

i	0	1	2	3	4	5
c[i]	3	4	5	5	4	3

- restricción:  $x = c[y]$

# Propagación y poda con *element constraint*


## Ejemplo:

Si por ejemplo, nos llega información de que  $x \neq 3$ :

$$x \in \{\textcolor{red}{3}, 4, 5\}$$

Array c

i	0	1	2	3	4	5
c[i]	3	4	5	5	4	3



$$x \in \{3, 4, 5\}$$
$$y \in \{0, 1, 2, 3, 4, 5\}$$

- **restricción:**  $x = c[y]$

$$y \in \{\textcolor{red}{0}, 1, 2, 3, 4, \textcolor{red}{5}\}$$


# Propagación y poda con *element constraint*

## Ejemplo:

Si por ejemplo, nos llega información de que  $y \neq 4$ :

$$y \in \{\textcolor{red}{0}, 1, 2, 3, \textcolor{red}{4}, \textcolor{red}{5}\}$$

Array c



i	0	1	2	3	4	5
c[i]	3	4	5	5	4	3

- $x \in \{3, 4, 5\}$
- $y \in \{0, 1, 2, 3, 4, 5\}$
- **restricción:**  $x = c[y]$

# Propagación y poda con *element constraint*

## Ejemplo:

Si por ejemplo, si además  $y \neq 1$ :

$$y \in \{\textcolor{red}{0}, \textcolor{red}{1}, 2, 3, \textcolor{red}{4}, \textcolor{red}{5}\}$$

↓ Array c      ↓

i	0	1	2	3	4	5
c[i]	3	4	5	5	4	3

$$x \in \{\textcolor{red}{3}, \textcolor{red}{4}, 5\}$$

$$x = 5$$

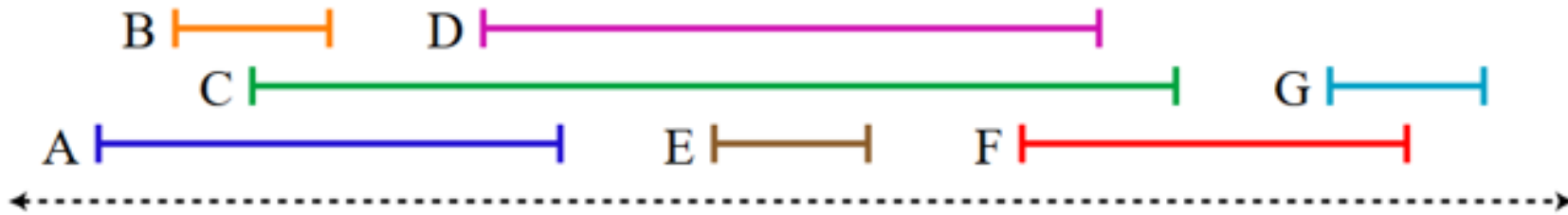
- $x \in \{3, 4, 5\}$
- $y \in \{0, 1, 2, 3, 4, 5\}$
- restricción:  $x = c[y]$

# Resumiendo

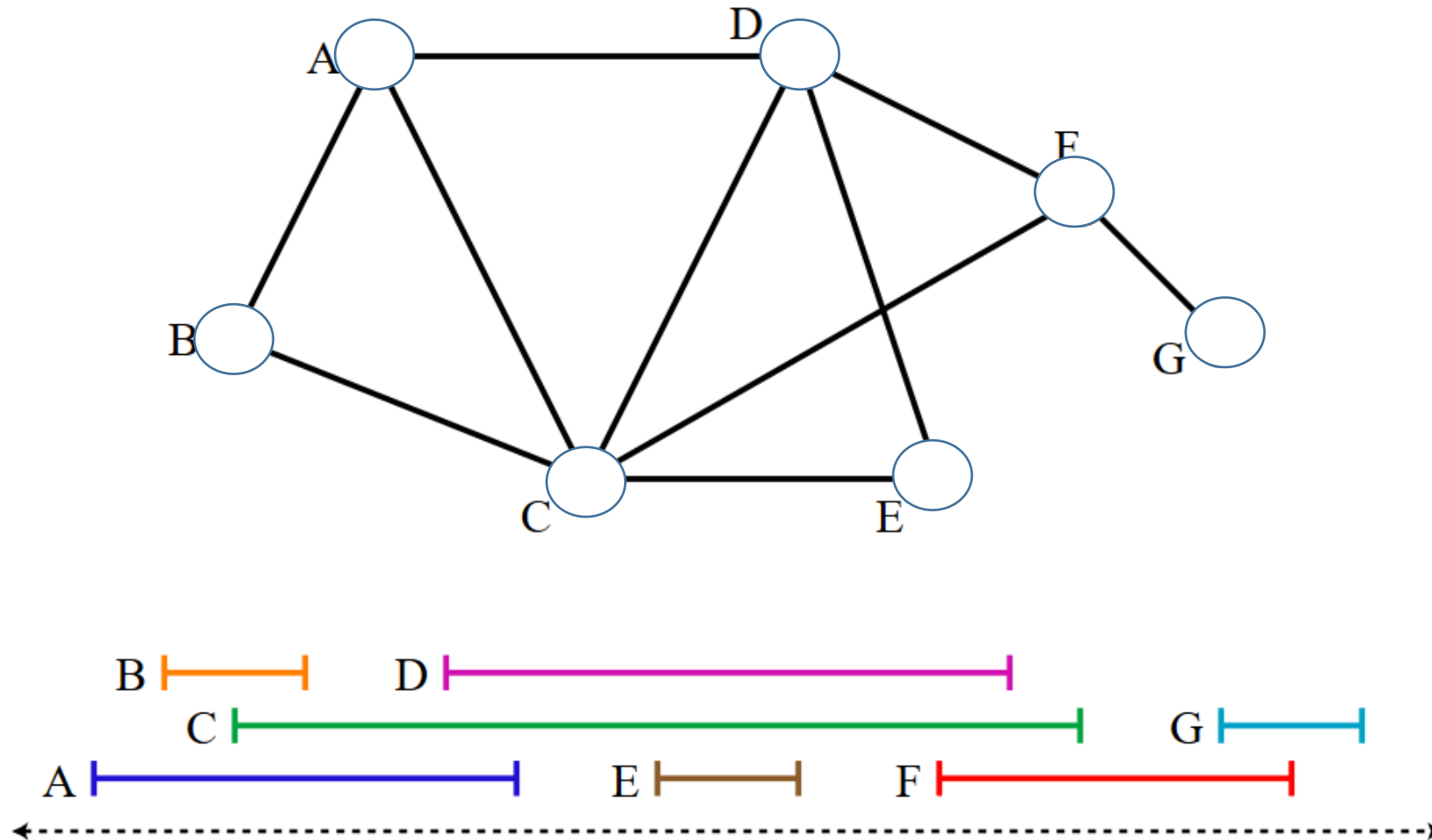
Este ejemplo nos permite demostrar la utilización de 2 características interesantes de la programación con restricciones:

1. Uso de variables de decisión como índices en *arrays* (a este concepto se le llama en inglés ***element constraint***)
2. Combinación lógica de restricciones

# Grafos de Intervalo



# Grafos de Intervalo





# Problema de las 8 reinas

- Plantear el problema de las 8 reinas mediante programación con restricciones
  - En general, existen muchas formas de modelar un problema
  - Asociamos una variable de decisión para cada columna, y la solución vendría dada indicando, para esa columna, un valor que indicaría la fila donde ubicamos la reina en el tablero.
  - Además tenemos 3 restricciones, aplicadas a filas, diagonal superior y diagonal inferior.

# Solución

```
range R = 1..8;
var(int) row[R] in R;
solve {
    forall(i in R, j in R: i < j) {
        row[i] ≠ row[j];
        row[i] ≠ row[j] + (j - i);
        row[i] ≠ row[j] - (j - i);
    }
}
```

```
int: n = 8;
set of int: R = 1..n;
array [R] of var R: row;

constraint forall(i in R, j in i+1..n)(
    row[i] != row[j] /\
    row[i] != row[j] + (j-i) /\
    row[i] != row[j] - (j-i)
);

solve satisfy;
```

Filas

Diagonal inferior

Diagonal superior

```
range R = 1..8;  
var{int} row[R] in R;  
solve {  
  forall(i in R, j in R: i < j) {  
    row[i] ≠ row[j];  
    row[i] ≠ row[j] + (j - i);  
    row[i] ≠ row[j] - (j - i);  
  }  
}
```

Filas

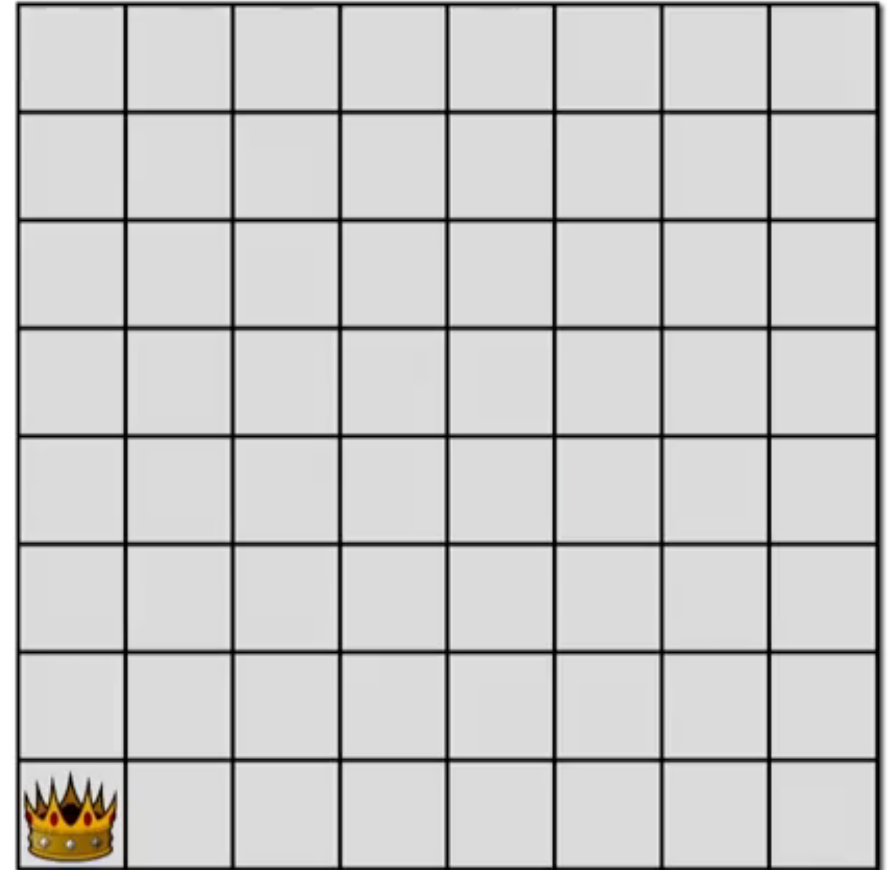
Diagonal inferior

Diagonal superior

```
range R = 1..8;  
var{int} row[R] in R;  
solve {  
  forall(i in R, j in R: i < j) {  
    → row[i] ≠ row[j];  
    → row[i] ≠ row[j] + (j - i);  
    → row[i] ≠ row[j] - (j - i);  
  }  
}
```

```
row[1] ≠ row[2];  
...  
row[1] ≠ row[8];  
  
row[1] ≠ row[2] + 1;  
...  
row[1] ≠ row[8] + 7;  
  
row[1] ≠ row[2] - 1;  
...  
row[1] ≠ row[8] - 7;
```

```
row[1] ≠ row[2];  
...  
row[1] ≠ row[8];  
  
row[1] ≠ row[2] + 1;  
...  
row[1] ≠ row[8] + 7;  
  
row[1] ≠ row[2] - 1;  
...  
row[1] ≠ row[8] - 7;
```



```
row[1] ≠ row[2];  
...  
row[1] ≠ row[8];  
  
row[1] ≠ row[2] + 1;  
...  
row[1] ≠ row[8] + 7;  
  
row[1] ≠ row[2] - 1;  
...  
row[1] ≠ row[8] - 7;
```

