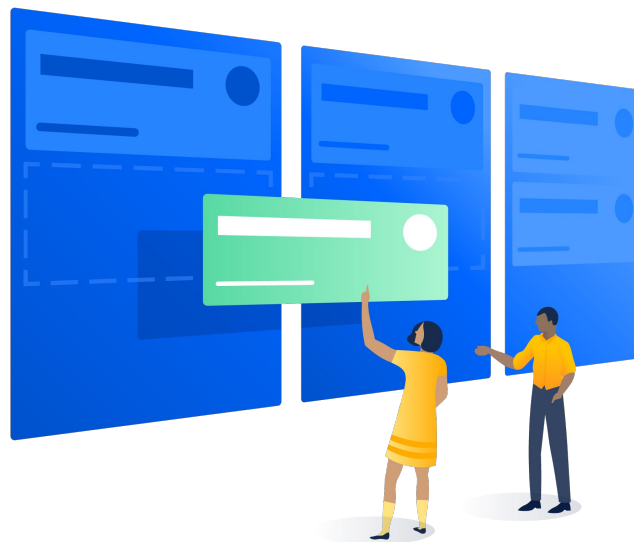


# Backtracking

## Tema 5

# Agenda

- Introducción
- El problema de la mochila (otra vez más)
- El problema de las 8 reinas
- Ramificación y poda



# Búsqueda exhaustiva

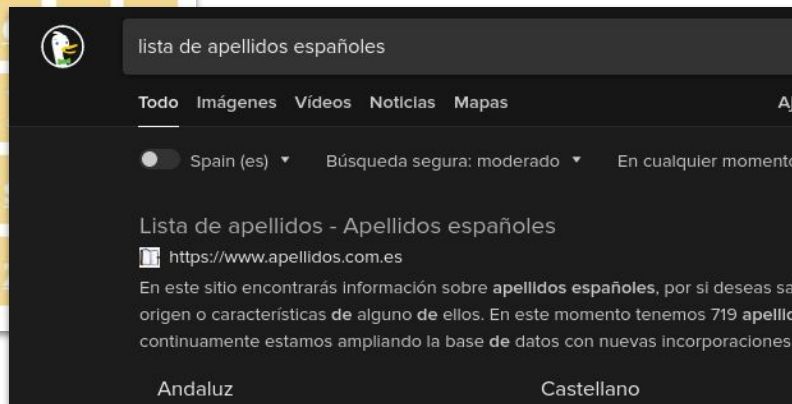
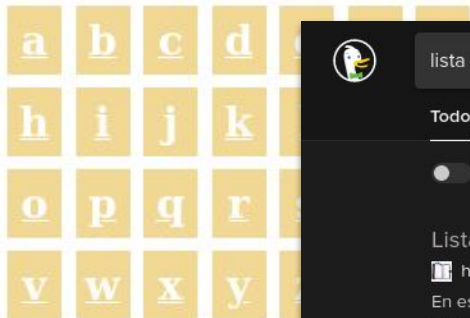
Algunos problemas no pueden resolverse mediante una aproximación algorítmica; i.e.:

1. Paso a paso
2. De forma determinista

En estos casos, puede que no nos quede más opción que realizar una búsqueda exhaustiva, o fuerza bruta.

# Búsqueda exhaustiva

## Apellidos por letra



## Apellidos por longitud

- Apellidos de 3 letras (10)
- Apellidos de 4 letras (55)
- Apellidos de 5 letras (114)
- Apellidos de 6 letras (147)
- Apellidos de 7 letras (170)
- Apellidos de 8 letras (121)
- Apellidos de 9 letras (58)
- Apellidos de 10 letras (27)
- Apellidos de 11 letras (9)

[Listado completo de apellidos](#)

¿Cómo me apellido?

# Búsqueda exhaustiva

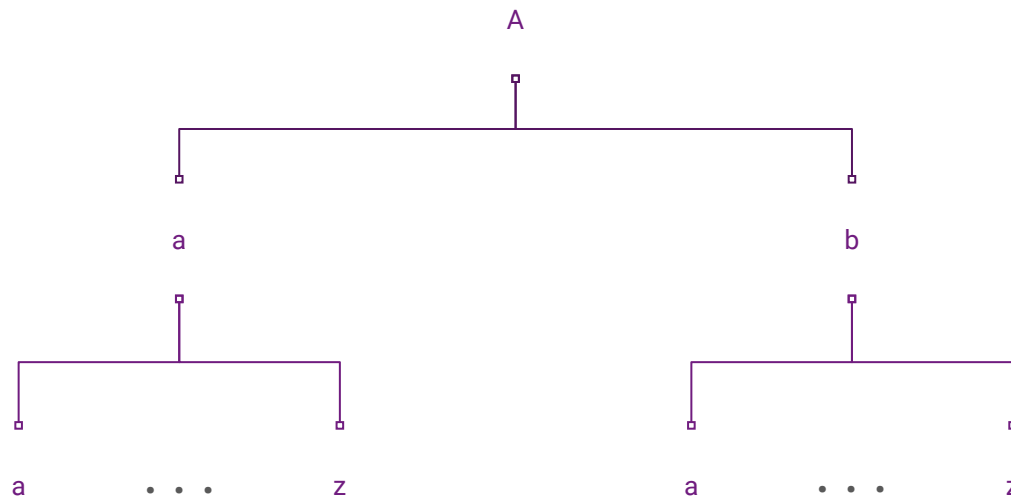
¿A?  
¿Aa?  
¿Aaa?  
¿Aaaa?  
...  
¿Aaaz?  
¿Aab?  
...  
¿Aaz?

¿Ab?  
¿Aba?  
¿Abaa?  
...  
¿Abaz?  
¿Abb?  
¿Abba?  
...  
¿Abbz?

¿Abz?  
...  
¿Abzz?  
¿Az?  
¿Aza?  
¿Azaa?  
...  
¿Azzz?  
¿B?

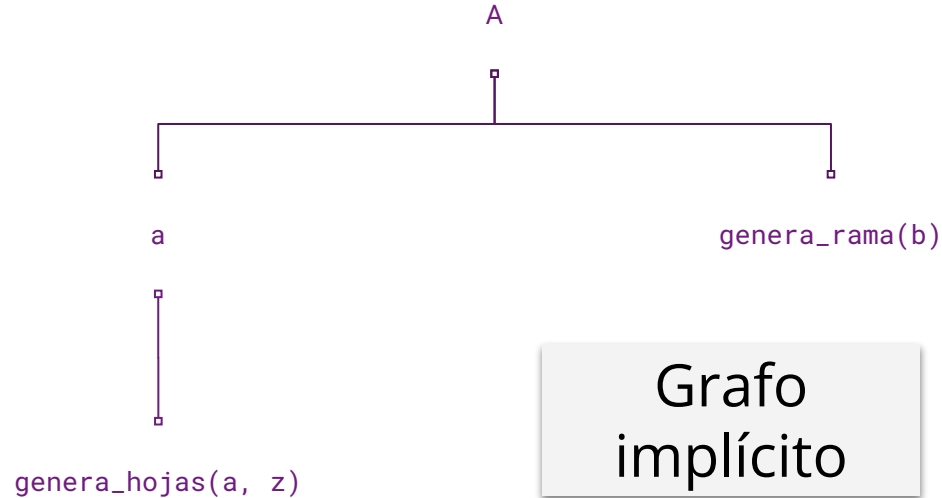
# Búsqueda exhaustiva

¿A?  
¿Aa?  
¿Aaa?  
¿Aaaa?  
...  
¿Aaaz?  
¿Aab?  
...  
¿Aaz?



# Búsqueda exhaustiva

¿A?  
¿Aa?  
¿Aaa?  
¿Aaaa?  
...  
¿Aaaz?  
¿Aab?  
...  
¿Aaz?



Grafo  
implícito

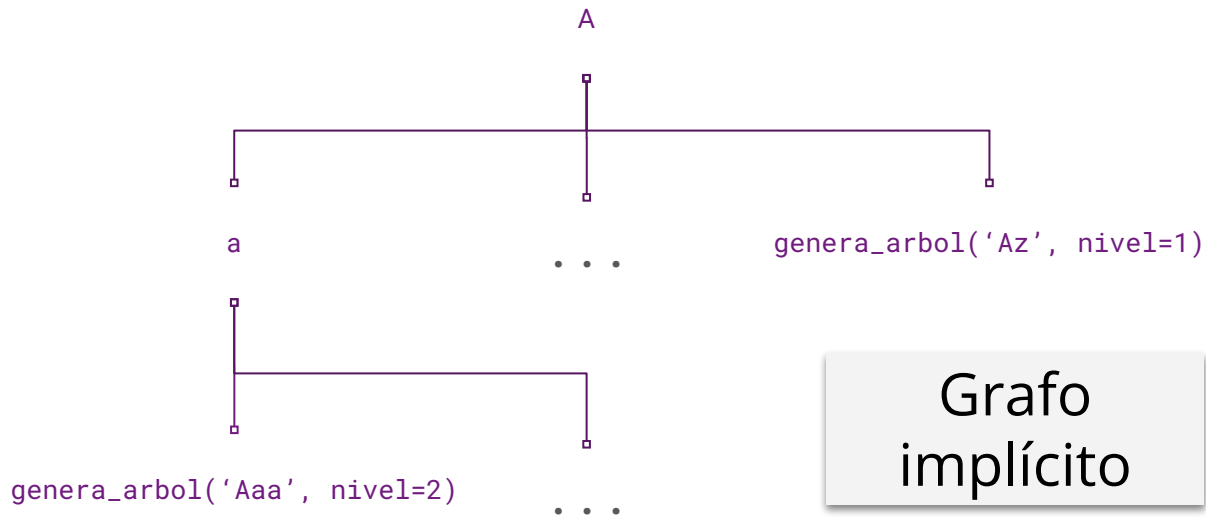
El backtracking es una aproximación (generalmente recursiva) de búsqueda exhaustiva basada en la exploración de grafos implícitos. Para ello, procede de la siguiente forma:

1. Por cada nodo de profundidad  $K$ , basado en una solución parcial  $S$
2. Generamos las derivaciones  $S'$  de  $S$ , y por cada una de ellas
3. Comprobamos si es solución del problema
  - a. Si lo es, almacenar  $S$
  - b. Si no, llamada recursiva al método con  $S'$  y  $K-1$



# Backtracking

¿A?  
¿Aa?  
¿Aaa?  
¿Aaaa?  
...  
¿Aaaz?  
¿Aab?  
...  
¿Aaz?



Grafo  
implícito

# Backtracking

```
# Aprox. incremental
def backtracking(solucion=[], nivel=0):
    if nivel == nivel_max:
        return solucion

    for c in candidatos:
        siguiente = solucion + [ candidato ]
        if es_solucion(siguiente):
            trata_solucion(siguiente)

        backtracking(siguiente, nivel+1)
```

# Backtracking

```
import string

soluciones = []
# Ejemplo de aprox. con cota
def pwdcrack(pwd="", max=10):
    if es_solucion(pwd):
        soluciones.append(pwd)

    if len(pwd) == max:
        return

    for c in string.ascii_letters + string.digits + string.punctuation:
        pwdcrack(pwd + c, max)
```

A este esquema básico pueden incorporarse muchas componentes, en general en forma de parámetros que se añaden en la llamada recursiva:

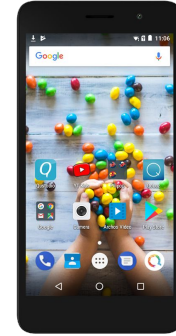
- Nivel de profundidad
- Estructuras de almacenamiento de soluciones parciales
- Estructuras de almacenamiento de entradas parciales
- Elementos de control de flujo (e.g., finalización del proceso, reporte, etc.)

## **El paso de parámetros por valor duplica las estructuras de datos.**

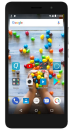
Es mejor el paso por referencia, pero hay que tener cuidado con las modificaciones, o pueden darse casos de corrupción:

- Entre hermanos: el padre aporta distinta información a los hijos
- De padres a hijos (realmente, de hijos a padre): paso de parámetros por referencia, y alteración por parte de los hijos

# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)





# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)





# El problema de la mochila (BT)



# El problema de la mochila (BT)





# El problema de la mochila (BT)



# El problema de la mochila (BT)



# El problema de la mochila (BT)

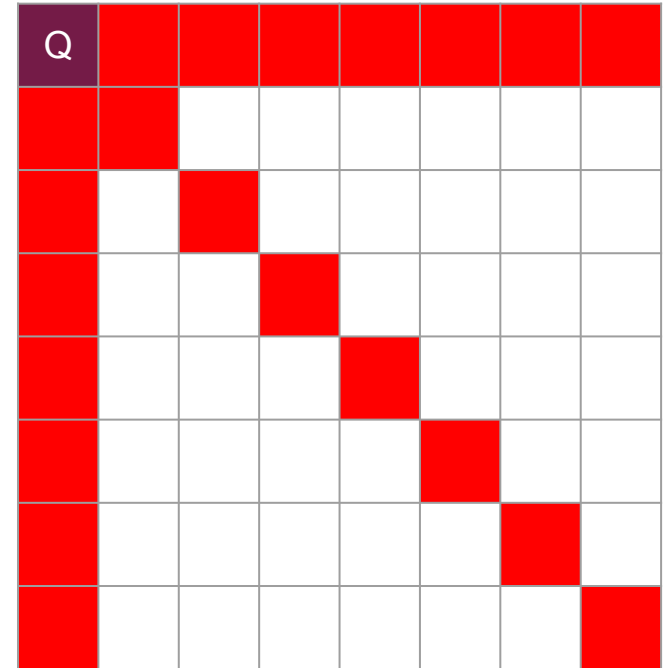


Búsqueda en profundidad hasta 🕶️

# El problema de las 8 reinas

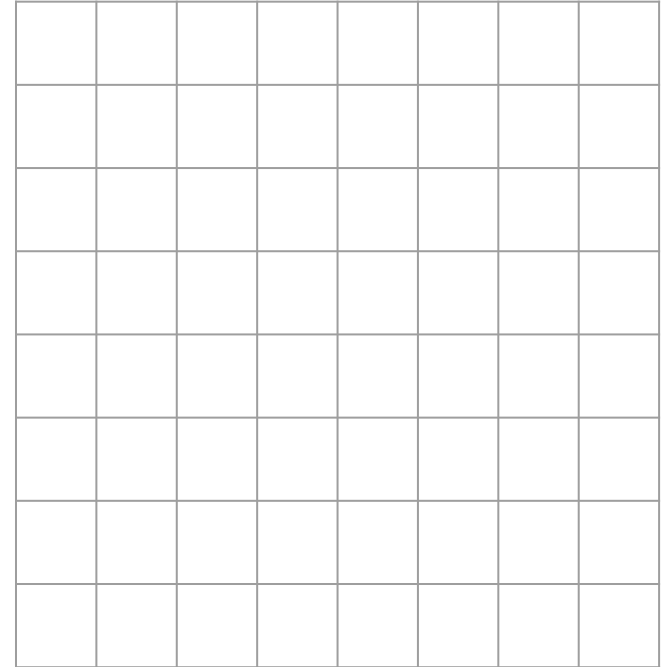
La reina del ajedrez (Q) pone en riesgo a todas las piezas que hay:

- Diagonales
- Horizontal
- Vertical



# El problema de las 8 reinas

El problema de las 8 reinas plantea cómo se podrían introducir 8 reinas en un tablero de 8x8 sin que se amenacen entre sí

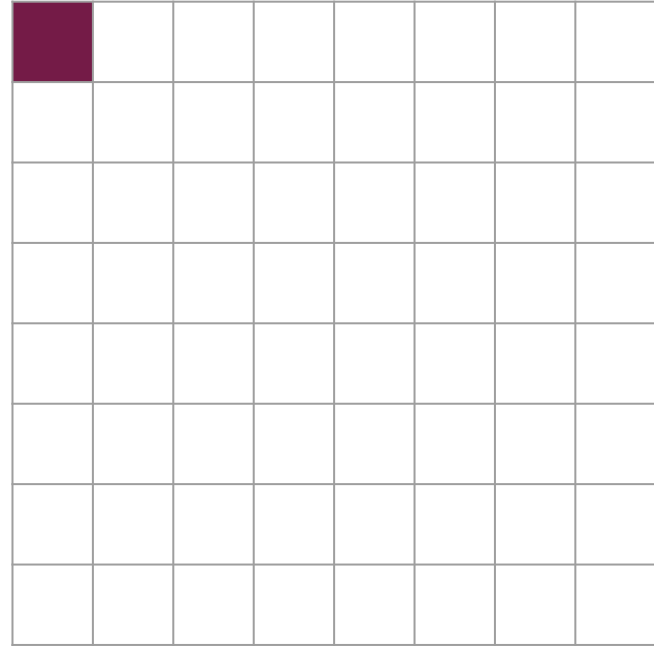


# El problema de las 8 reinas

## 1. Aproximación limpia

$64^8$  combinaciones

- nivel = 1
- iteración = 1

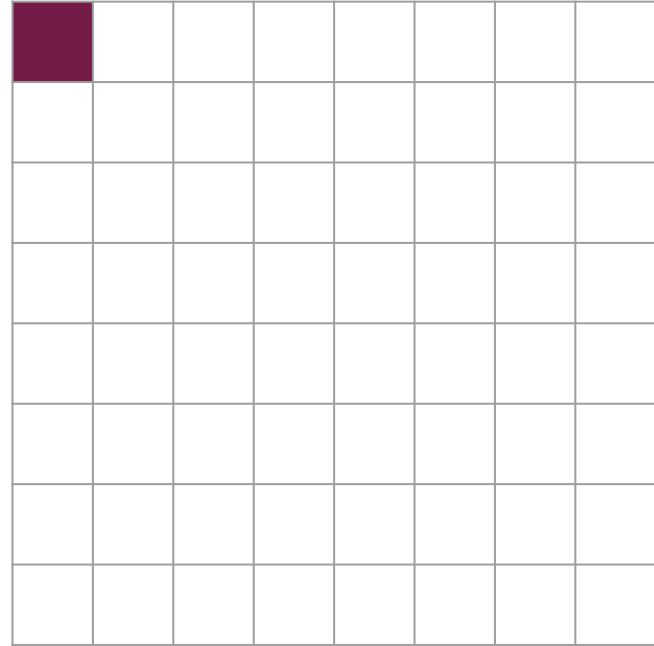


# El problema de las 8 reinas

## 1. Aproximación limpia

$64^8$  combinaciones

- nivel = 2
- iteración = 1

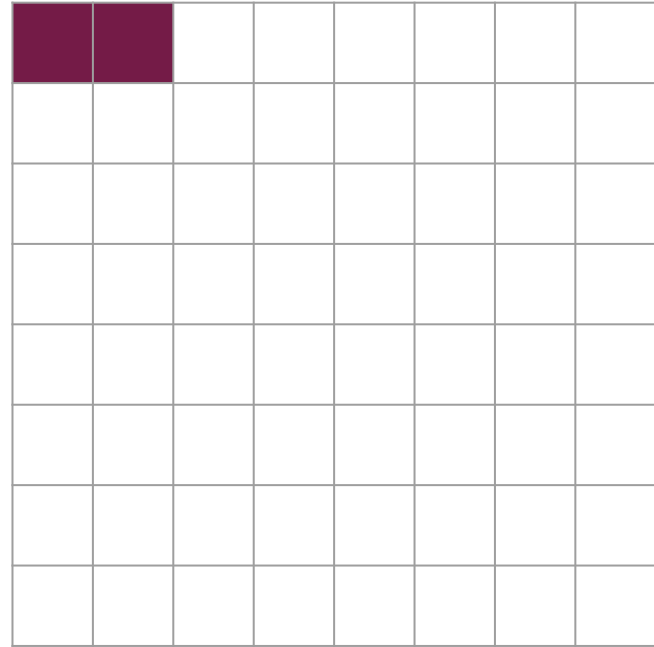


# El problema de las 8 reinas

## 1. Aproximación limpia

$64^8$  combinaciones =  $2^{48}$

- nivel = 2
- iteración = 2





# Límites de la búsqueda exhaustiva

Energía mínima necesaria para realizar una operación\*  $\rightarrow 2.85 \times 10^{-21}$  J

Cantidad de energía en el universo\*\*  $\rightarrow 1.8 \times 10^{47}$  J

Con toda la energía del universo pueden realizarse  $\sim 2^{225}$  operaciones:

$$\log_2 1.8e47 / 2.85e-21$$

Por eso es buena idea trabajar con AES-256 :)

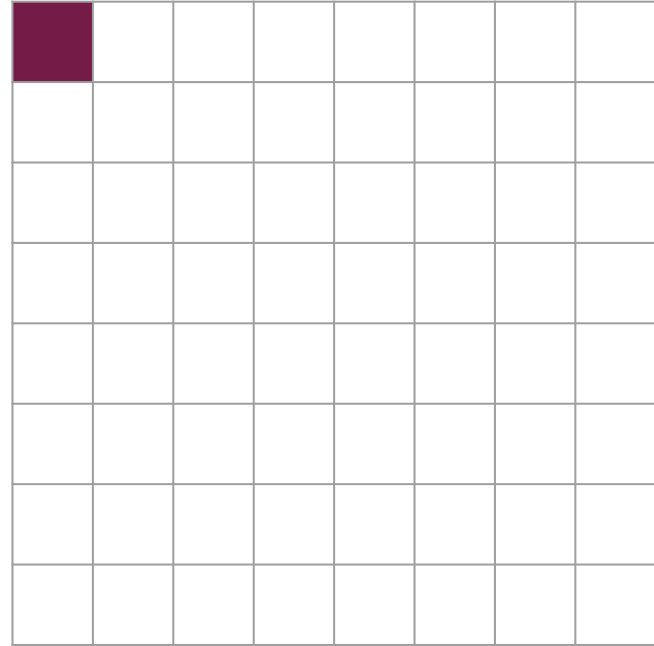
\*Principio de Landauer; \*\* Jim Lochner, *Big Bang Energy*;

# El problema de las 8 reinas

## 2. Sin repeticiones

- $\text{binom}(64, 8)$  combinaciones

- nivel = 1
- iteración = 1

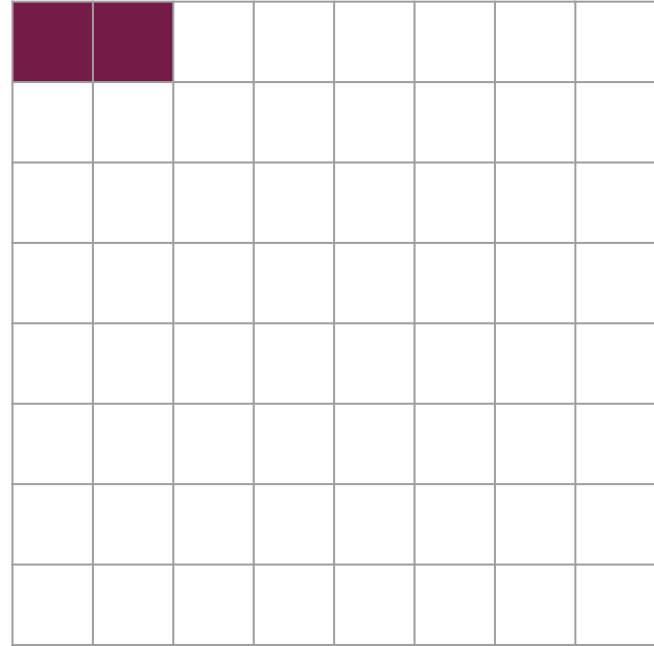


# El problema de las 8 reinas

## 2. Sin repeticiones

- $\text{binom}(64, 8)$  combinaciones

- nivel = 2
- iteración = 1

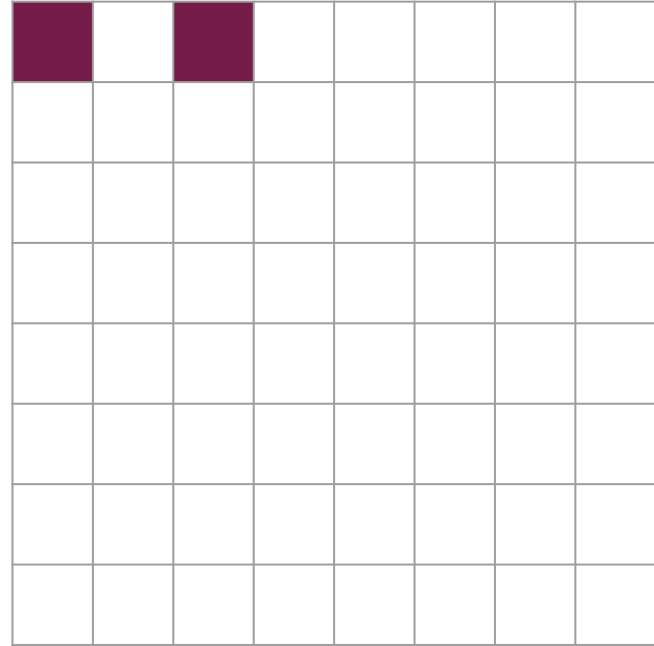


# El problema de las 8 reinas

## 2. Sin repeticiones

- $\text{binom}(64, 8)$  combinaciones

- nivel = 2
- iteración = 2

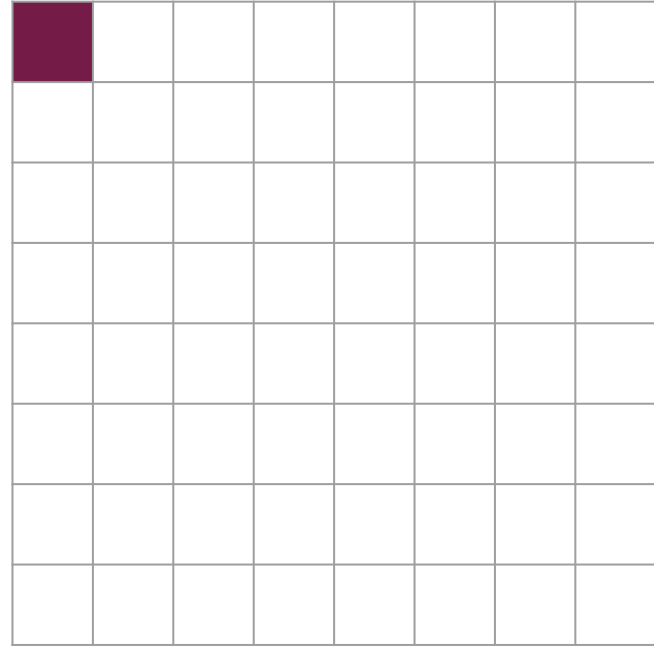


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 1
- iteración = 1

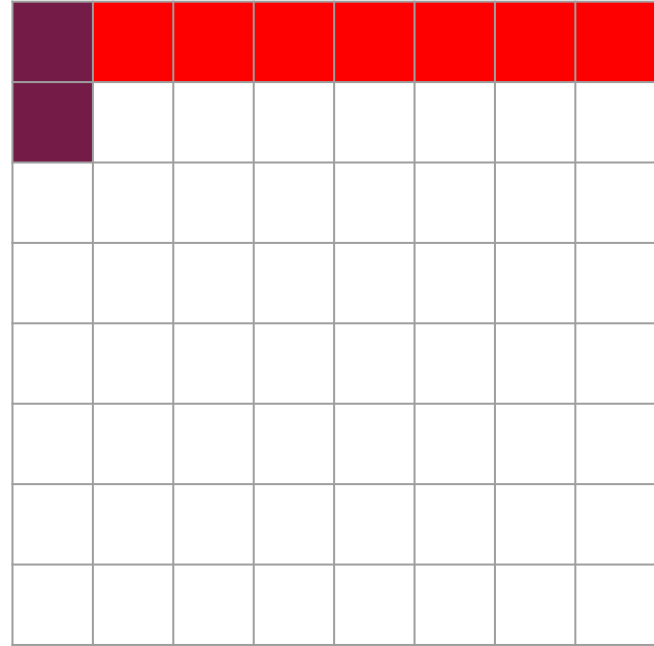


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 2
- iteración = 1

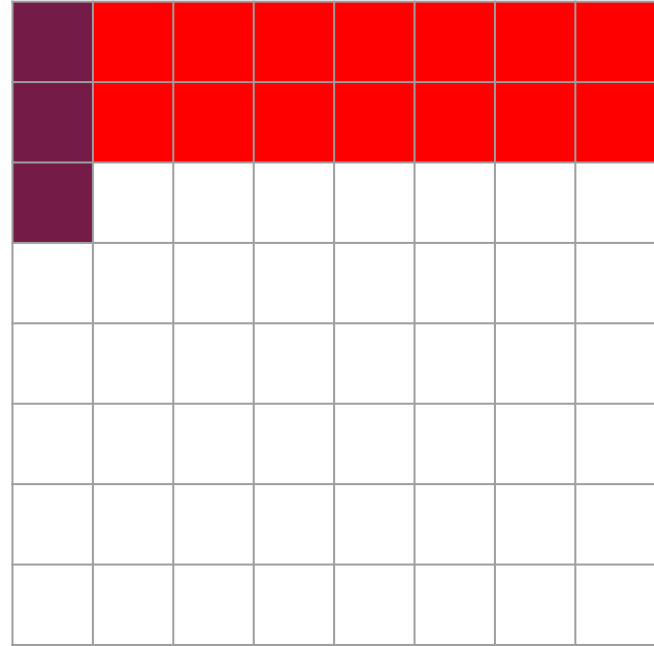


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 3
- iteración = 1

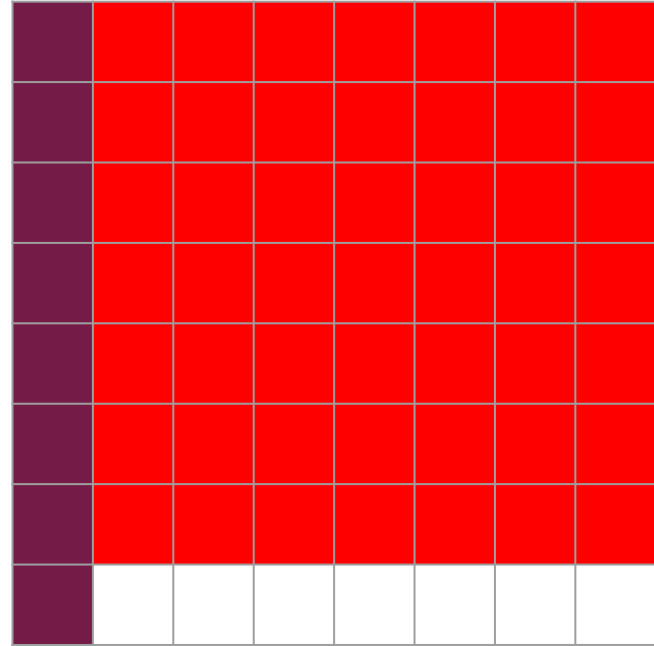


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 1





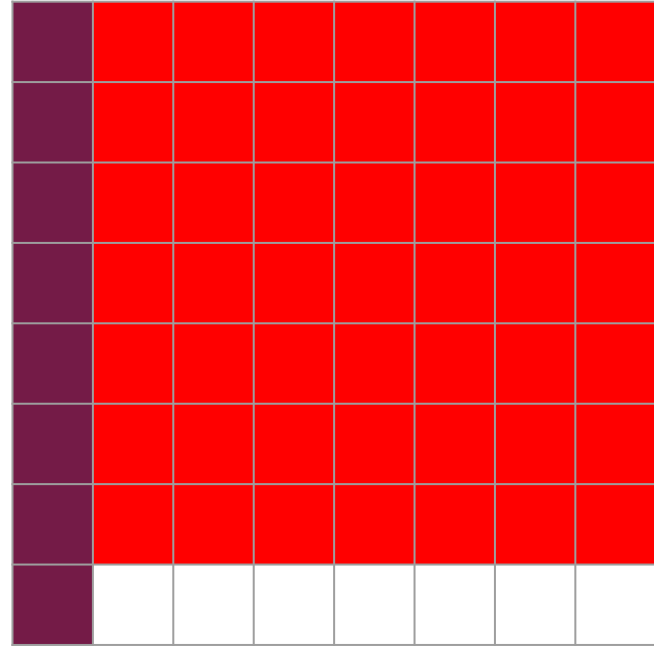
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 1



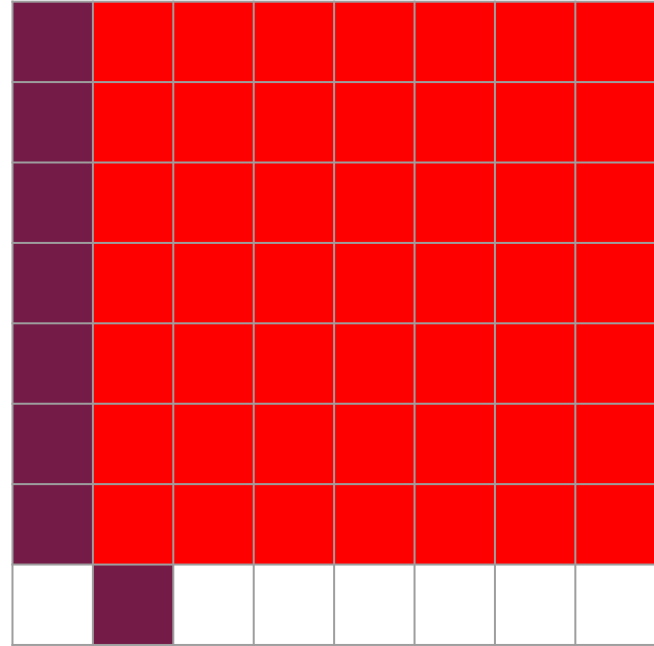
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 2



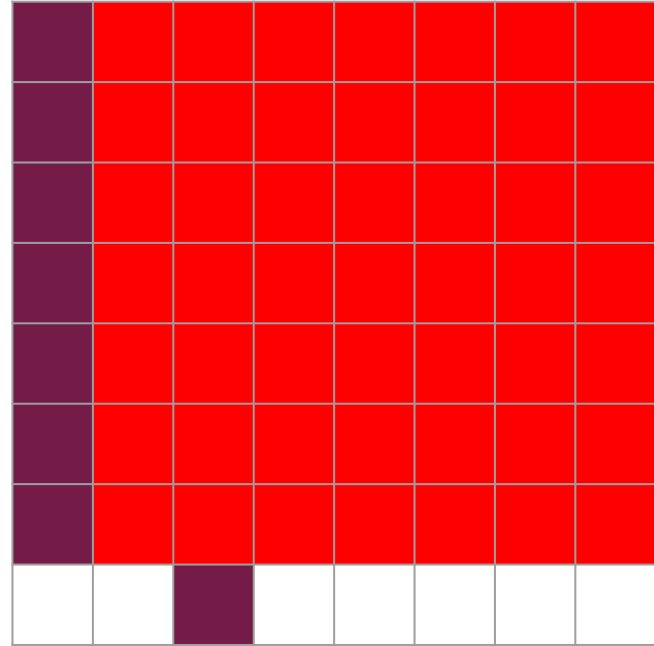
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 3



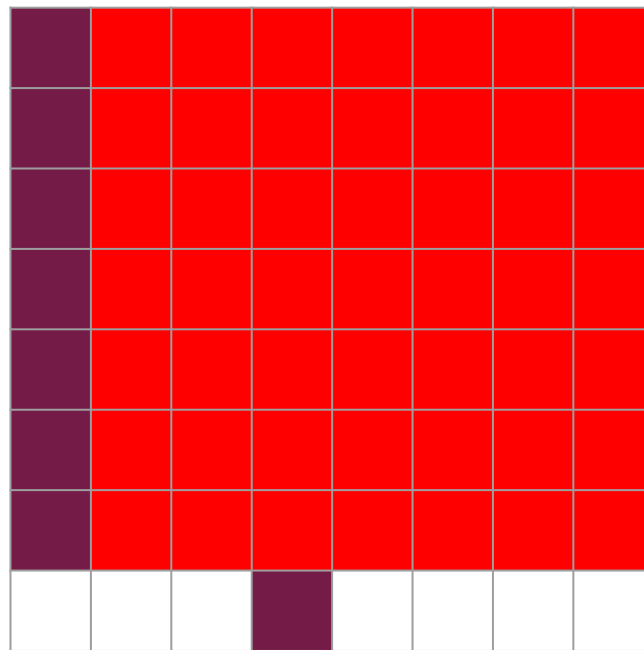
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 4



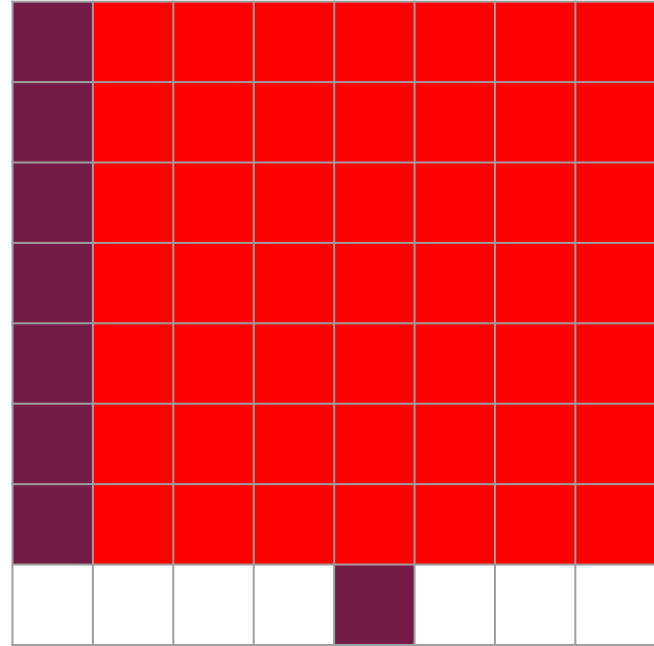
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 5



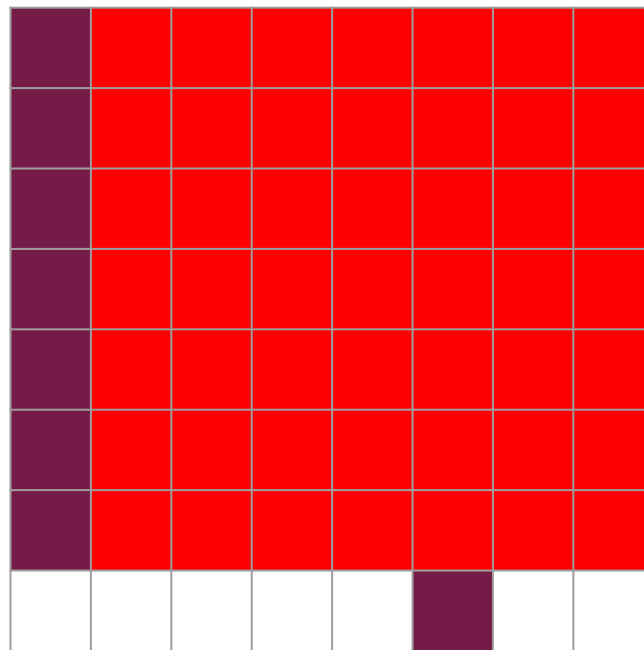
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 6



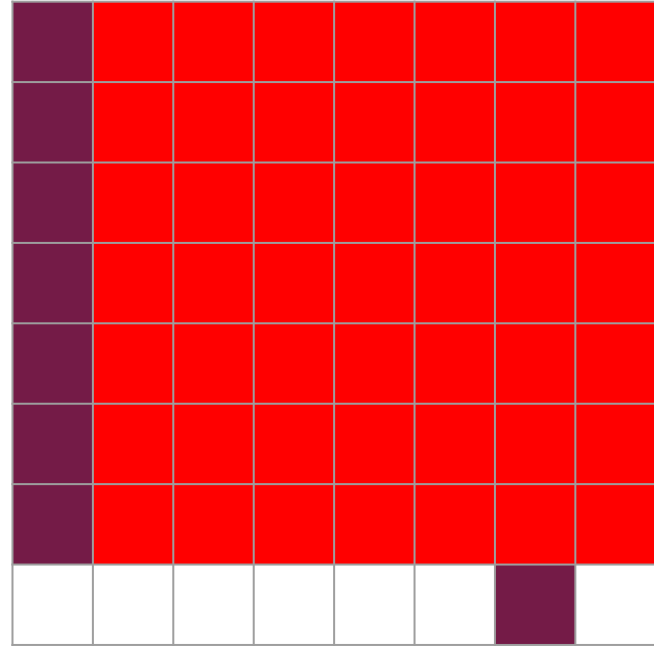
# El problema de las 8 reinas

¿Es solución? No

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 7



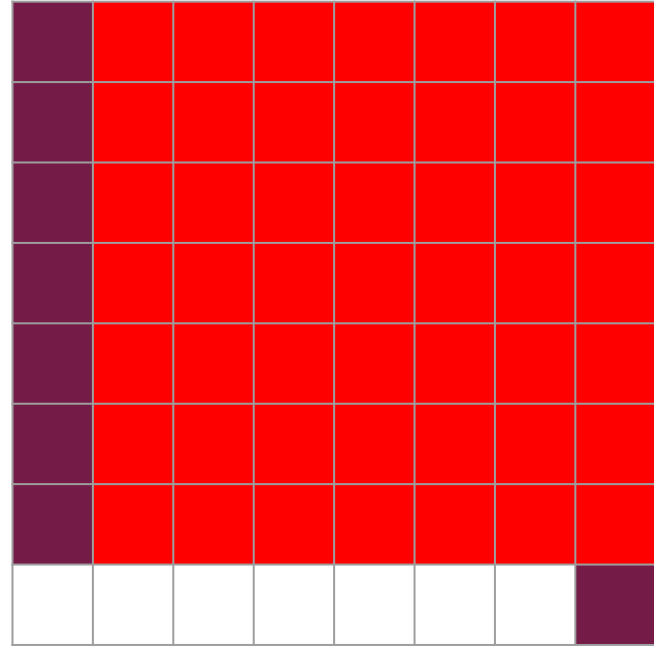
# El problema de las 8 reinas

¿Es solución? No → ¡Backtracking!

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 8



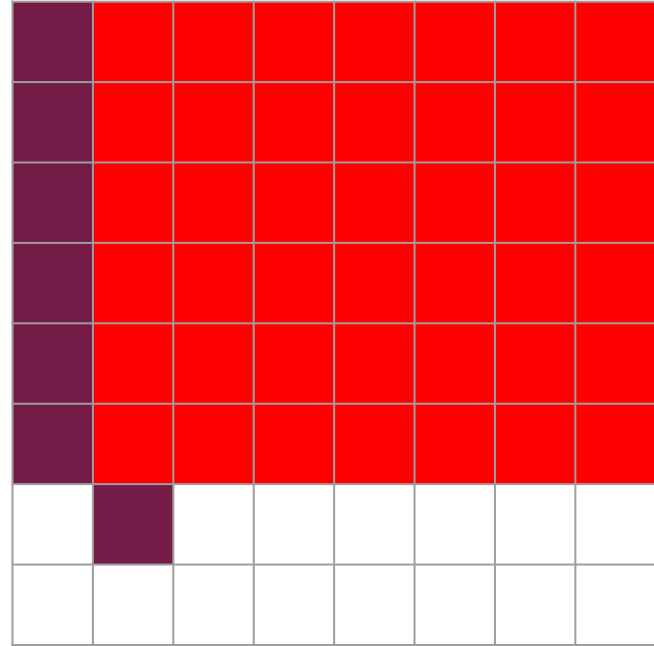


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 7
- iteración = 2



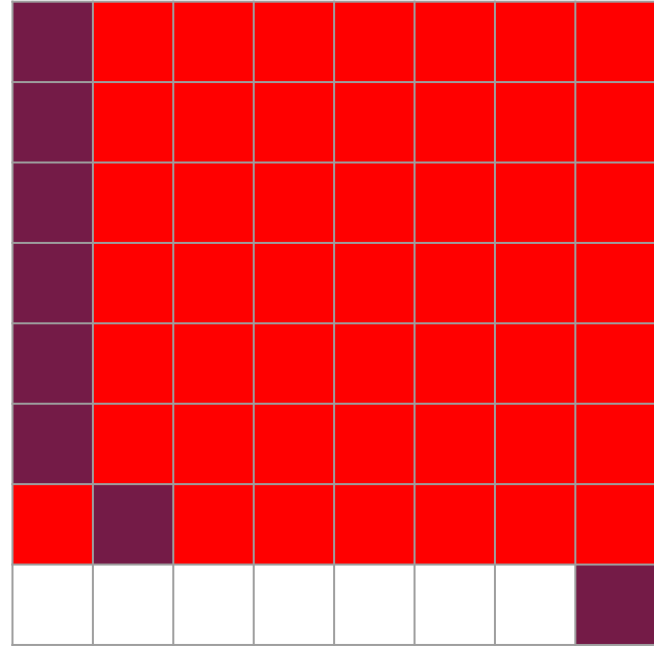
# El problema de las 8 reinas

¿Es solución? No → ¡Backtracking!

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 8



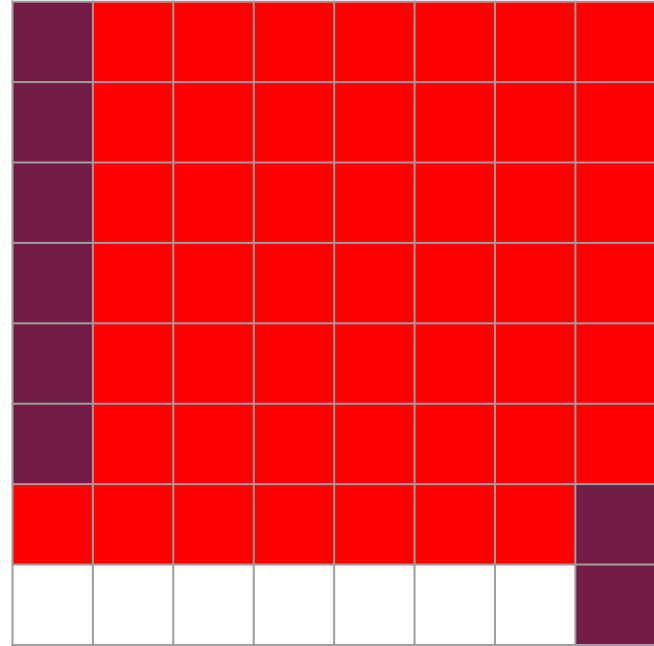
# El problema de las 8 reinas

¿Es solución? No → ¡Backtracking!

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 8

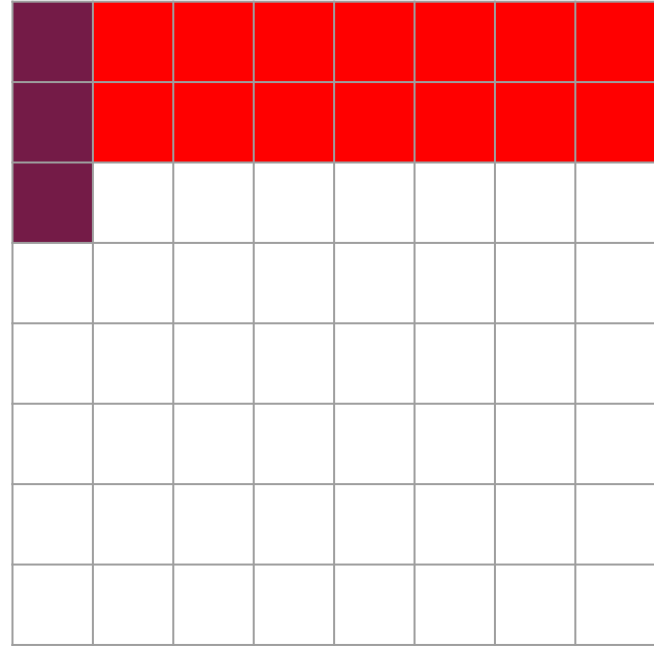


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 3
- iteración = 1

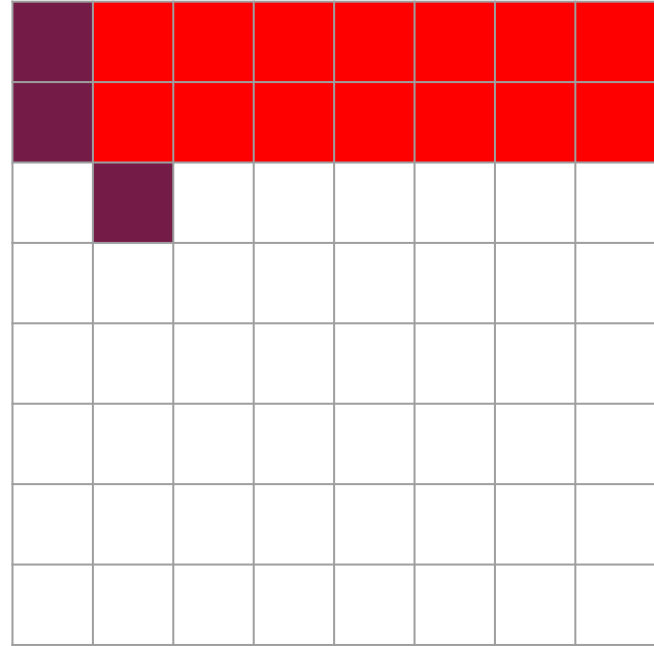


# El problema de las 8 reinas

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 3
- iteración = 2



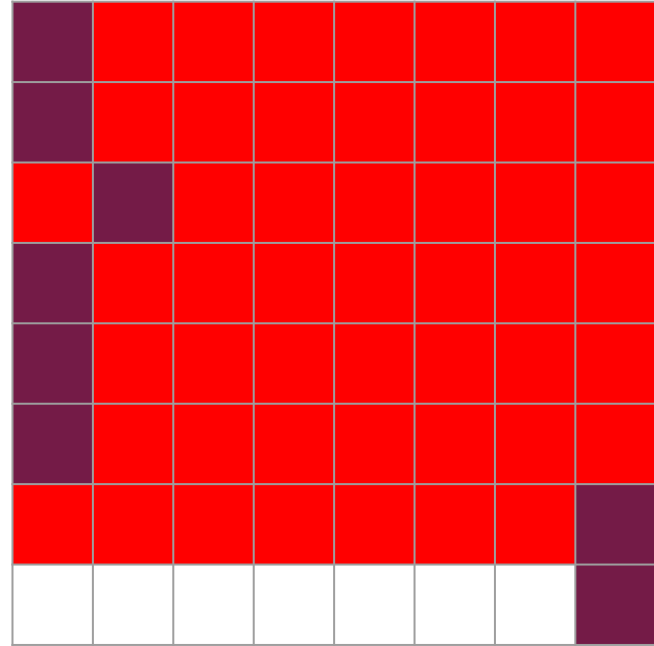
# El problema de las 8 reinas

¿Es solución? No → ¡Backtracking!

## 3. Sin repetir filas

- $8^8$  combinaciones

- nivel = 8
- iteración = 8

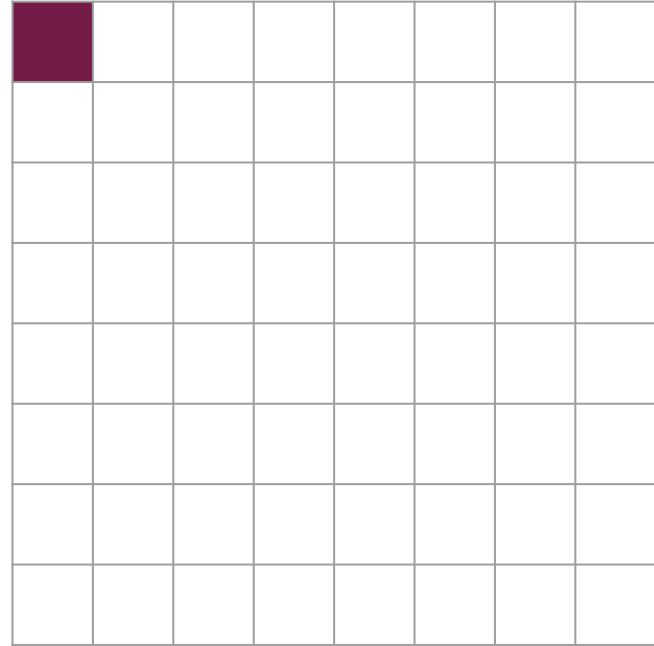


# El problema de las 8 reinas

## 3. Sin repetir filas, ni columnas

- 8! combinaciones

- nivel = 1
- iteración = 1

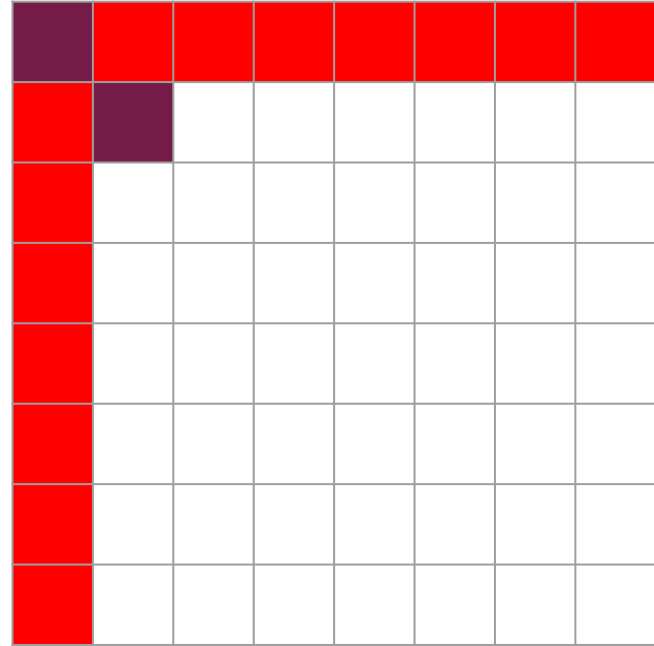


# El problema de las 8 reinas

## 3. Sin repetir filas, ni columnas

- 8! combinaciones

- nivel = 2
- iteración = 1



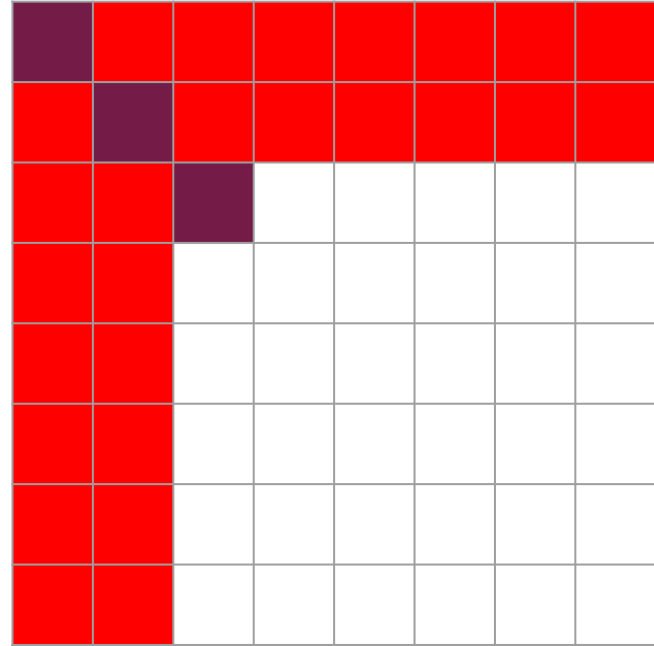


# El problema de las 8 reinas

## 3. Sin repetir filas, ni columnas

- 8! combinaciones

- nivel = 3
- iteración = 1

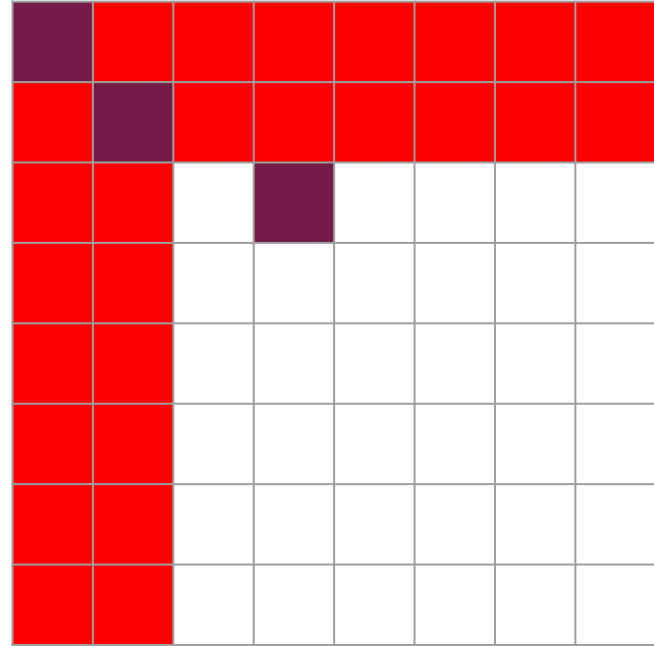


# El problema de las 8 reinas

## 3. Sin repetir filas, ni columnas

- 8! combinaciones

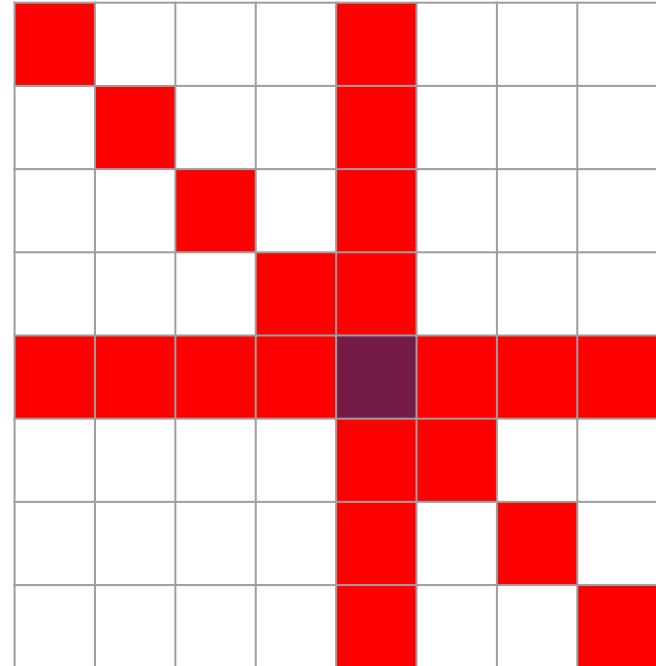
- nivel = 3
- iteración = 2



# El problema de las 8 reinas

## 4. Sin repetir filas, columnas, ni diagonales

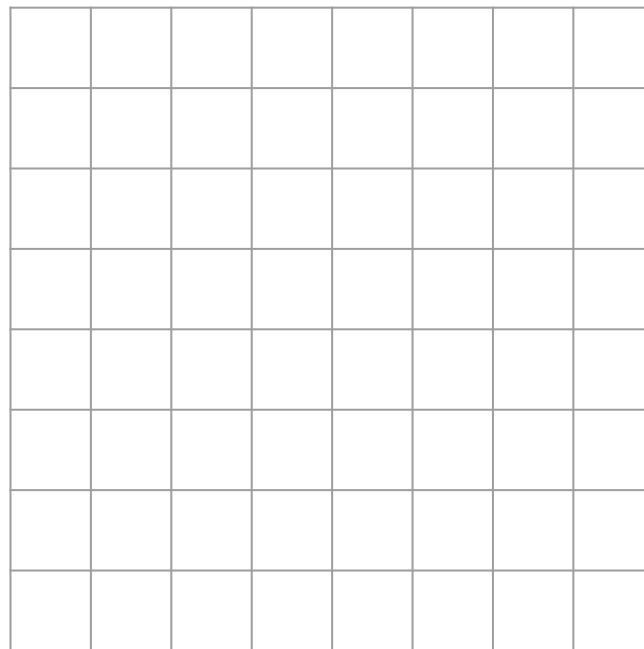
- Cuantas más restricciones pongamos, menos operaciones tendremos que realizar antes de llegar a una solución fallida



# El problema de las 8 reinas

## 4. Sin repetir filas, columnas, ni diagonales

Nivel	1
Iteración	-
Columnas	[ ]
Filas	[ ]
diag45	[ ]
diag135	[ ]



# El problema de las 8 reinas

Nivel	1
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[ ]
diag135	[ ]

	0,1	0,2	0,3				
1,0	1,1	1,2					
2,0	2,1						
3,0							

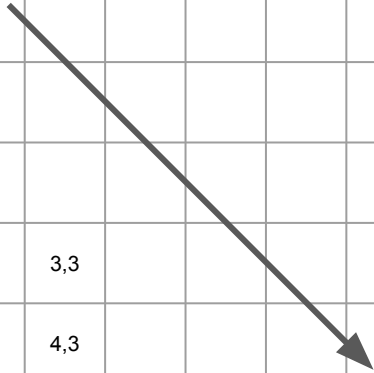
## diag45

Identificada por número de fila de inicio,  
localizable por suma de componentes  
(e.g., (3,0) = 3; (2,1) = 3; (1,2) = 3; etc.)

# El problema de las 8 reinas

Nivel	1
Iteración	1
Columnas	[ 0 ]
Filas	[ 0 ]
diag45	[ ]
diag135	[ ]

0,0						
1,0	1,1					
2,0	2,1	2,2				
	3,1	3,2	3,3			
		4,2	4,3			
			5,3			

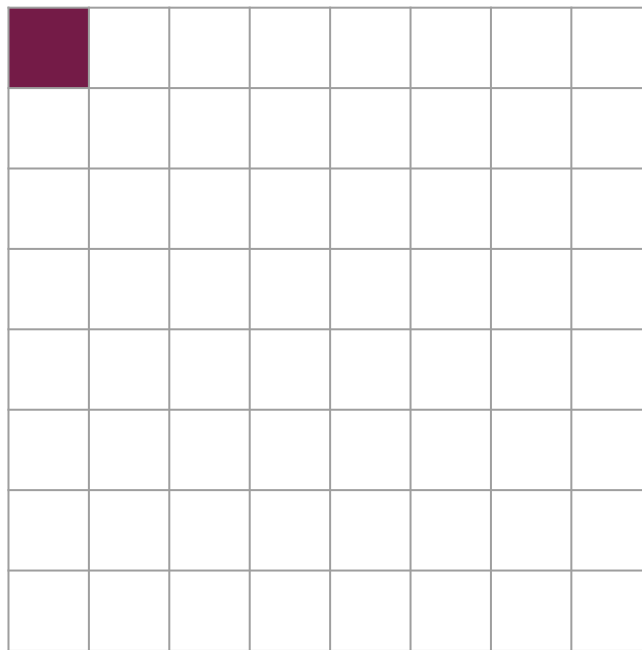


## **diag135**

Identificada por número de fila de inicio,  
localizable por resta de componentes  
(e.g.,  $1 = 1 - 0 = 2 - 1 = 3 - 2$ )

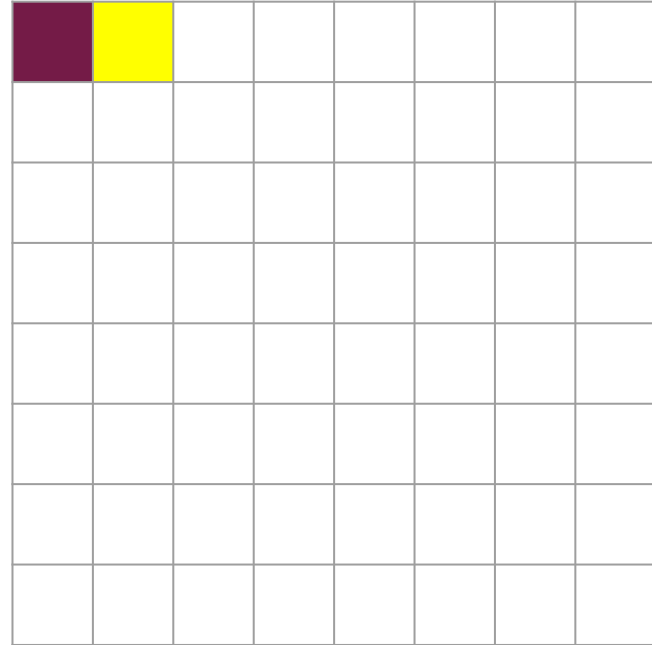
# El problema de las 8 reinas

Nivel	1
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[0]
diag135	[0]



# El problema de las 8 reinas

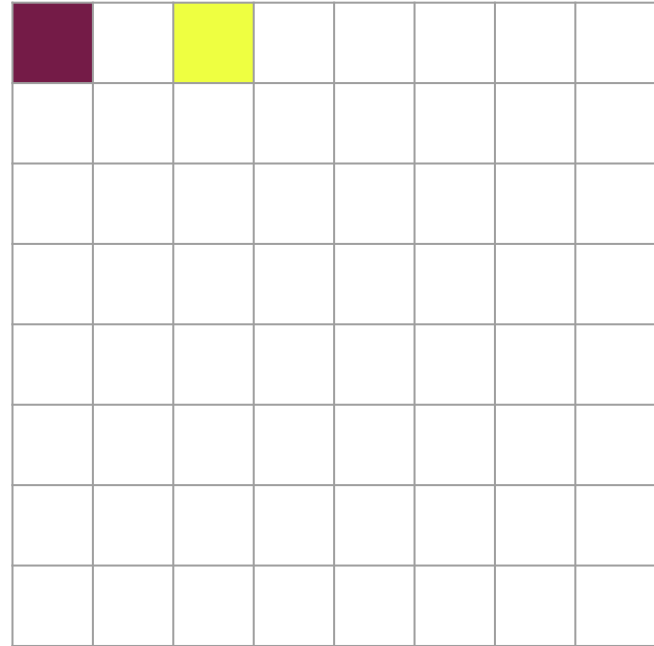
Nivel	2
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[0]
diag135	[0]





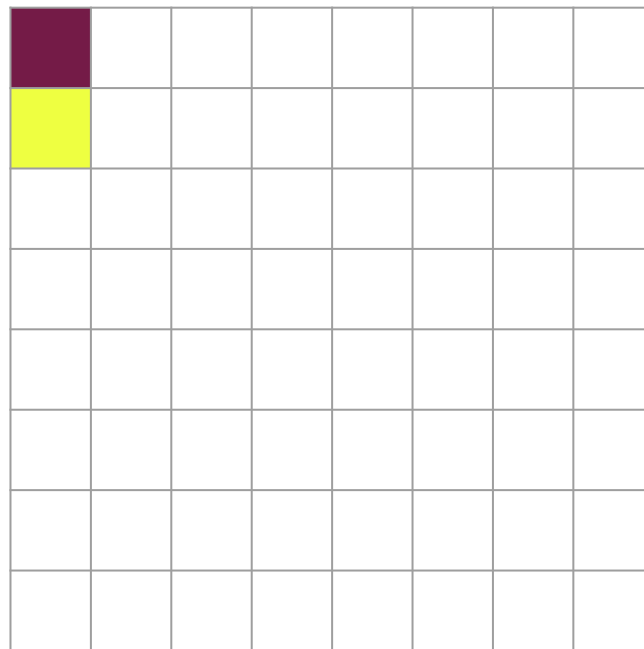
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[0]
diag135	[0]



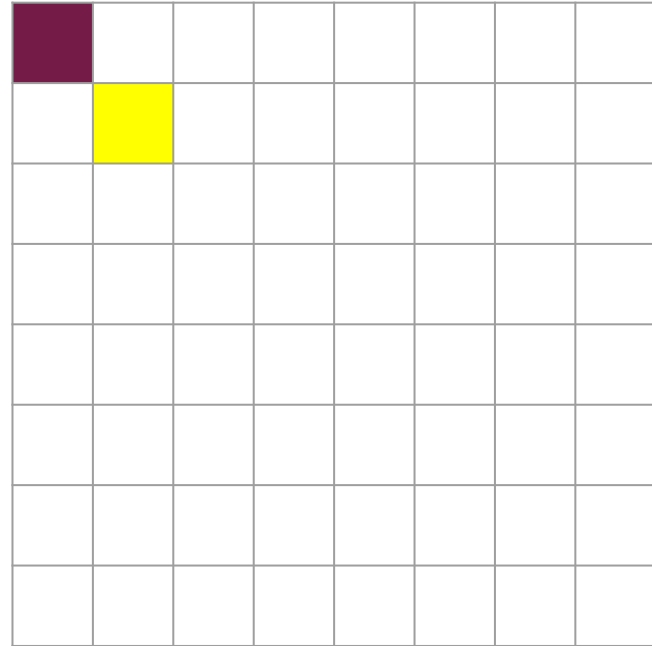
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[0]
diag135	[0]



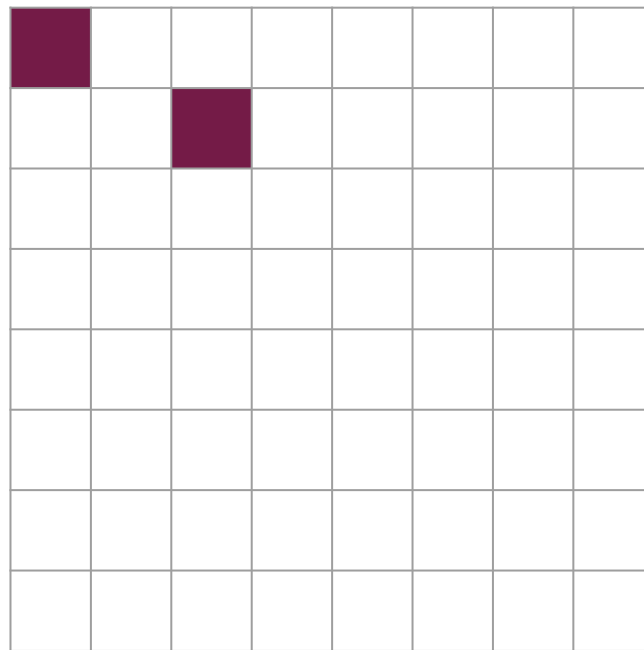
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[0]
Filas	[0]
diag45	[0]
diag135	[0]



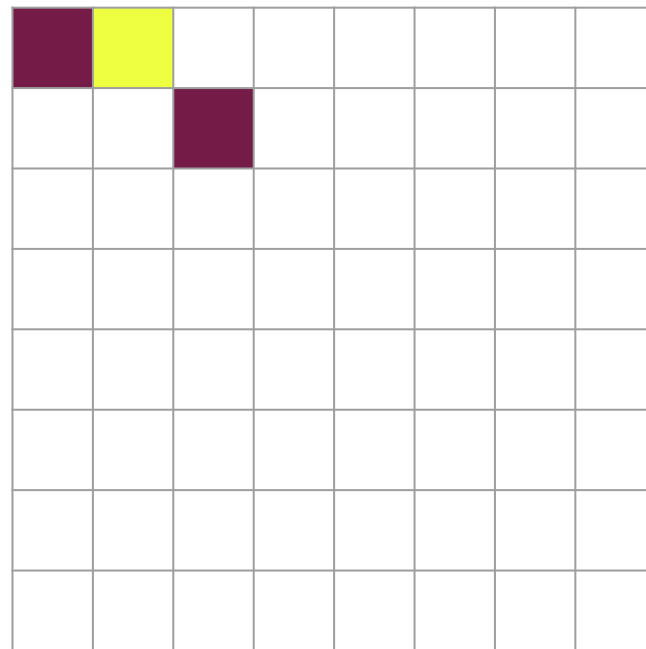
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	$[0, 2]$
Filas	$[0, 1]$
diag45	$[0, 3]$
diag135	$[0, -1]$



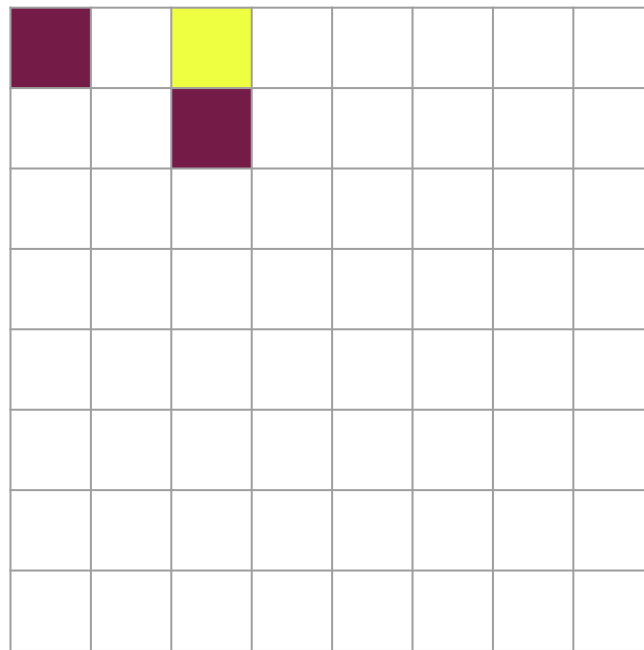
# El problema de las 8 reinas

Nivel	3
Iteración	1
Columnas	$[0, 2]$
Filas	$[0, 1]$
diag45	$[0, 3]$
diag135	$[0, -1]$



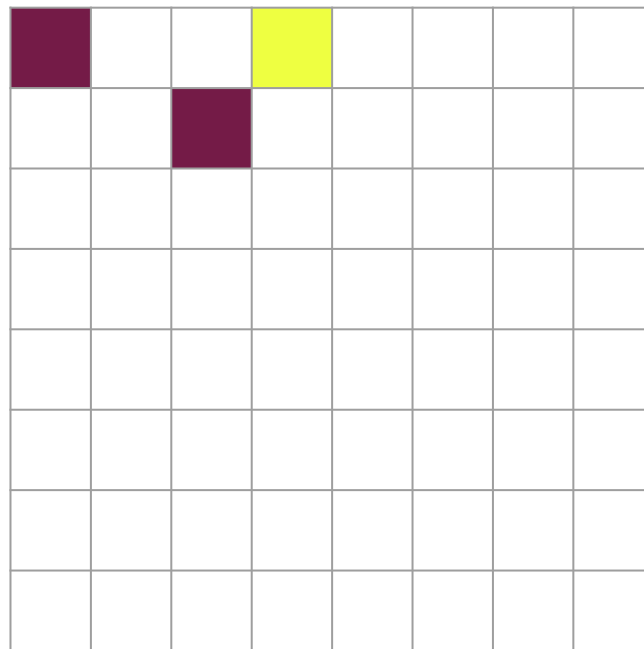
# El problema de las 8 reinas

Nivel	3
Iteración	1
Columnas	$[0, 2]$
Filas	$[0, 1]$
diag45	$[0, 3]$
diag135	$[0, -1]$



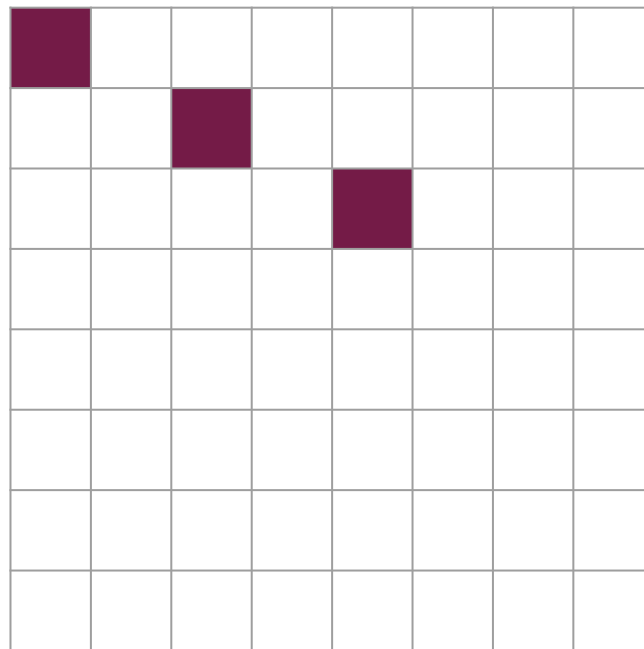
# El problema de las 8 reinas

Nivel	3
Iteración	1
Columnas	$[0, 2]$
Filas	$[0, 1]$
diag45	$[0, 3]$
diag135	$[0, -1]$



# El problema de las 8 reinas

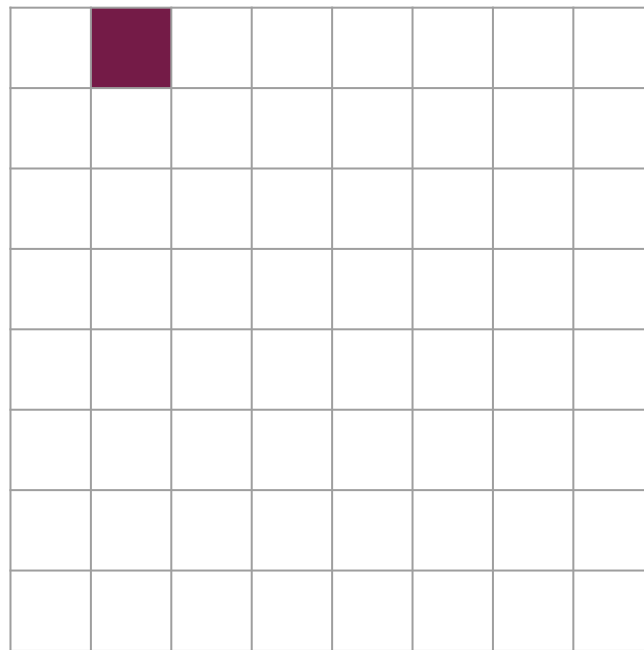
Nivel	3
Iteración	1
Columnas	[0, 2, 4]
Filas	[0, 1, 2]
diag45	[0, 3, 6]
diag135	[0, -1, -2]





# El problema de las 8 reinas

Nivel	1
Iteración	2
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]

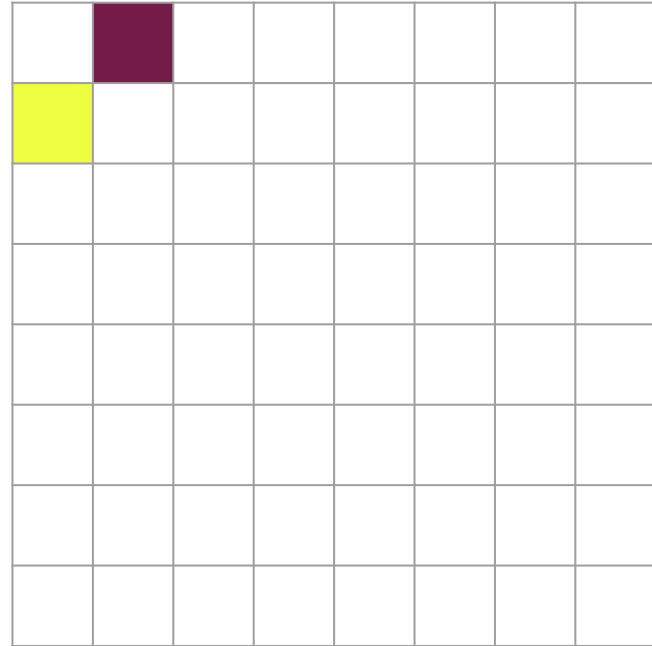


# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]

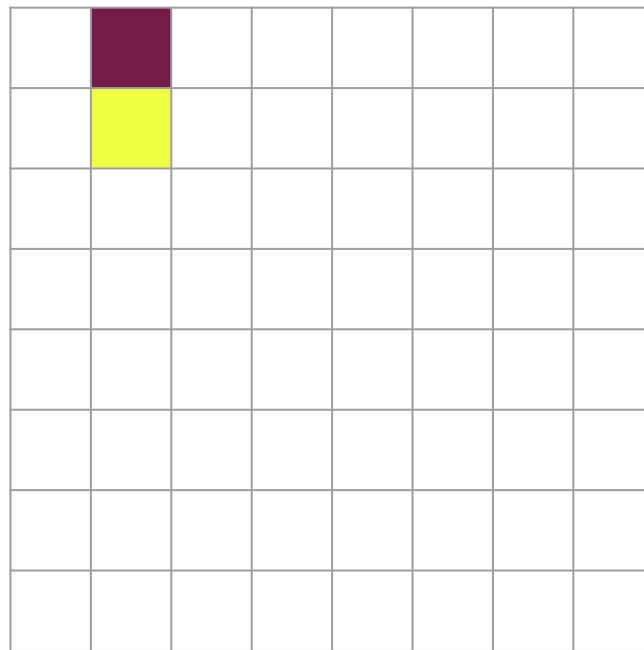

# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]



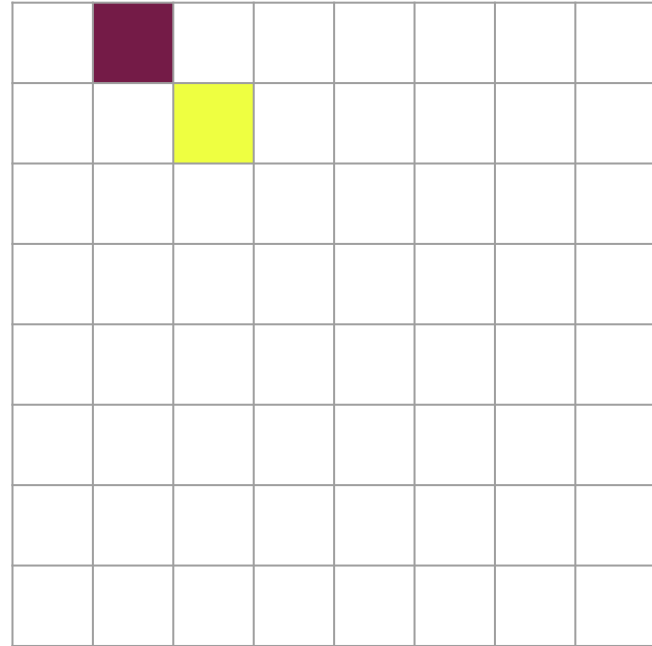
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]



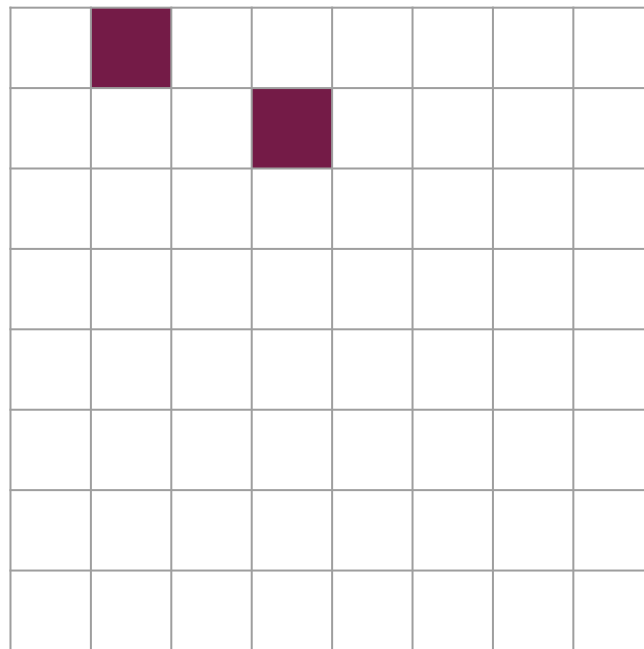
# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]



# El problema de las 8 reinas

Nivel	2
Iteración	1
Columnas	[ 1 ]
Filas	[ 0 ]
diag45	[ 1 ]
diag135	[ -1 ]



# Sudoku

**Support the Guardian**  
Available for everyone, funded by readers  
[Contribute →](#) [Subscribe →](#)

Search jobs Sign in Search

**The Guardian**

[News](#) [Opinion](#) [Sport](#) [Culture](#) [Lifestyle](#) [More v](#)

[UK](#) [UK politics](#) [Education](#) [Media](#) [Society](#) [Law](#) [Scotland](#) [Wales](#) [Northern Ireland](#)

**Shortcuts**  
Newspapers

**Pamela Hutchinson**  
Sun 22 Aug 2010 20.00 BST  
[f](#) [t](#) [e](#)

**The world's hardest sudoku?**  
We asked a reader to put Finnish mathematician Arto Inkala's sudoku to the test

		5	3					
8							2	
	7			1		5		
4					5	3		
	1			7				6

▲ The world's hardest sudoku. Photograph: Guardian

<https://junquera.gitlab.io/sudoku-solver>

Otro problema del backtracking

$$O(n^n)$$

*Podemos mitigarlo, además de mediante la detección temprana de soluciones erróneas, mediante la predicción de soluciones óptimas.*



# Búsqueda en anchura

Podemos pronosticar ramas del grafo más fructíferas mediante la búsqueda en anchura:

- Backtracking → *Depth first*
- Búsqueda en anchura → Análisis de hijos antes de profundizar

Si los combinamos podemos hacer búsqueda en profundidad sólo cuando, tras hacer una búsqueda en anchura, pronosticamos qué hijo puede dar una rama más viable.

*Trabajaremos con pilas y listas de prioridad*

# N tareas, N agentes

Agente \ Coste	T1	T2	T3	T4	T5
A1	2	8	11	9	6
A2	3	2	4	8	9
A3	1	13	3	4	5
A4	10	12	8	3	1
A5	4	3	1	5	2

Asignar tareas a los agentes, de tal forma que cada tarea se realice sólo una vez, y cada agente tenga asignada una sola tarea:

$T[i][j] \rightarrow$  Coste de realizar tarea  $T_j$  por el agente  $A_i$

# N tareas, N agentes

Agente \ Coste	T1	T2	T3	T4	T5
A1	2	8	11	9	6
A2	3	2	4	8	9
A3	1	13	3	4	5
A4	10	12	8	3	1
A5	4	3	1	5	2

1. Crear cota real (marcar un primer límite para ver si hacer poda o no):  
e.g., valores aleatorios (sin repetir A o T)  $\rightarrow$  Coste =  $2 + 2 + 1 + 3 + 5 = 13$

# N tareas, N agentes

Agente \ Coste	T1	T2	T3	T4	T5
A1	2	8	11	9	6
A2	3	2	4	8	9
A3	1	13	3	4	5
A4	10	12	8	3	1
A5	4	3	1	5	2

2. Crear cota mínima por filas (i.e., estimación sin dos valores en una fila):  
e.g., min por fila  $\rightarrow$  Coste =  $2 + 2 + 1 + 1 + 1 = 7$

# N tareas, N agentes

Agente \ Coste	T1	T2	T3	T4	T5
A1	2	8	11	9	6
A2	3	2	4	8	9
A3	1	13	3	4	5
A4	10	12	8	3	1
A5	4	3	1	5	2

3. Crear cota mínima por columnas (i.e., estimación sin dos valores en una columna):  
e.g., min por columna  $\rightarrow$  Coste =  $1 + 2 + 1 + 3 + 1 = 8$

# N tareas, N agentes

Agente \ Coste	T1	T2	T3	T4	T5
A1	2	8	11	9	6
A2	3	2	4	8	9
A3	1	13	3	4	5
A4	10	12	8	3	1
A5	4	3	1	5	2

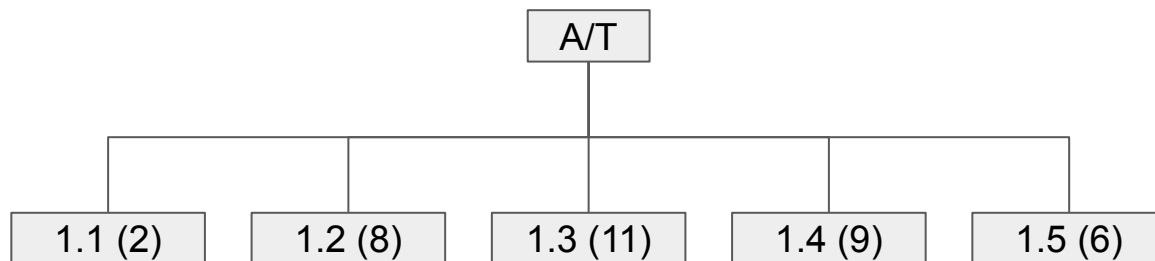
4. Cota mínima-máxima =  $\max\{7, 8\} = 8$

# N tareas, N agentes

Rango de valores válidos = [8, 13]

## 1. Análisis de primera tarea

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

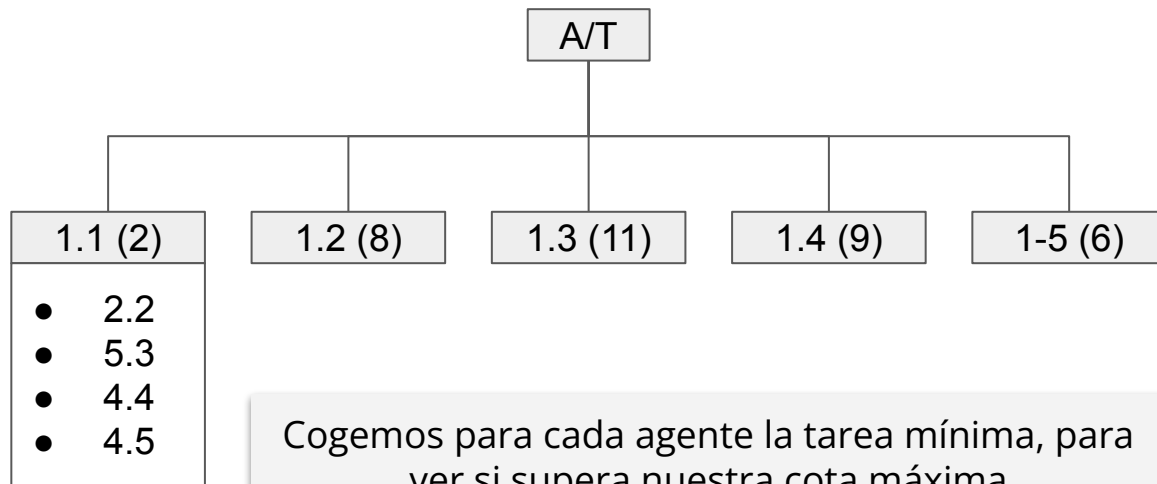


# N tareas, N agentes

Rango de valores válidos = [8, 13]

2. Estimación de siguientes tareas óptimas  
(i.e., ¿a quién le asignamos lo que queda por hacer?)

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2



Cogemos para cada agente la tarea mínima, para ver si supera nuestra cota máxima

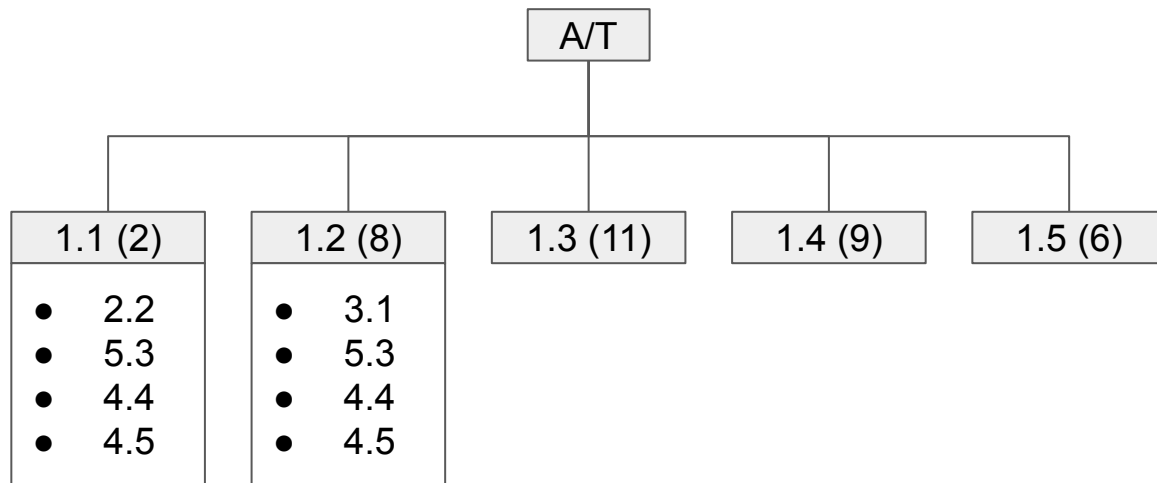


# N tareas, N agentes

Rango de valores válidos = [8, 13]

2. Estimación de siguientes tareas óptimas  
(i.e., ¿a quién le asignamos lo que queda por hacer?)

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

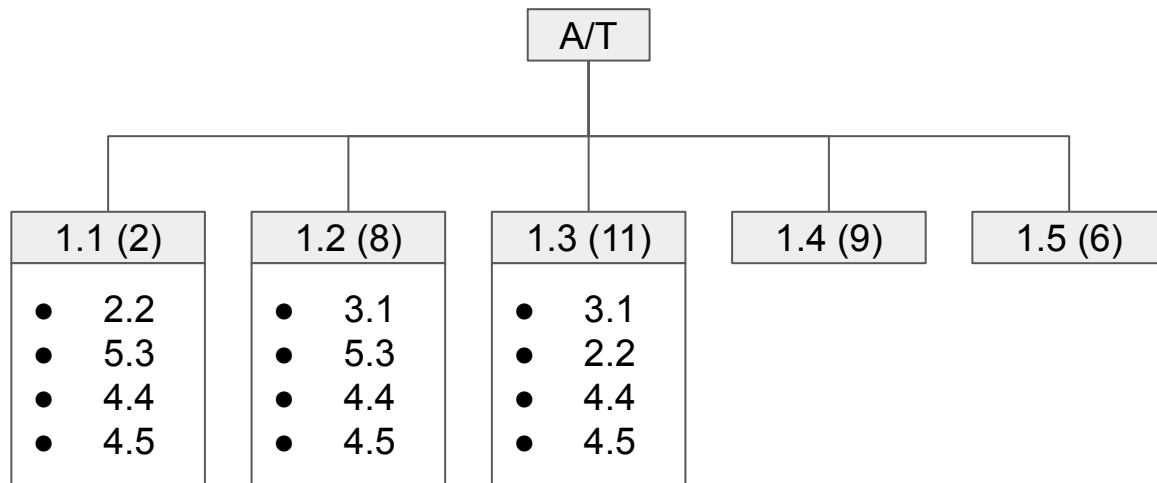


# N tareas, N agentes

Rango de valores válidos = [8, 13]

2. Estimación de siguientes tareas óptimas  
(i.e., ¿a quién le asignamos lo que queda por hacer?)

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

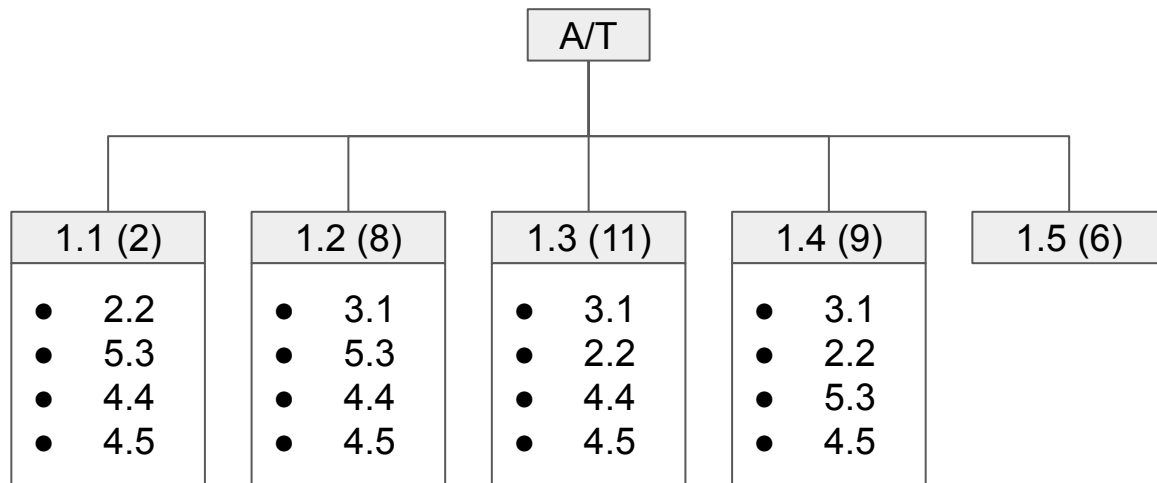


# N tareas, N agentes

Rango de valores válidos = [8, 13]

2. Estimación de siguientes tareas óptimas  
(i.e., ¿a quién le asignamos lo que queda por hacer?)

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

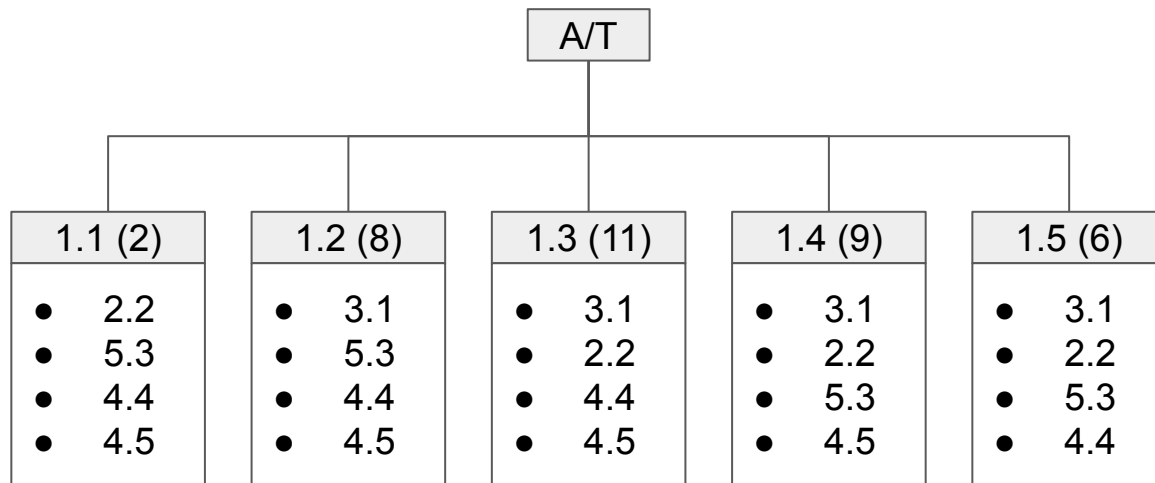


# N tareas, N agentes

Rango de valores válidos = [8, 13]

2. Estimación de siguientes tareas óptimas  
(i.e., ¿a quién le asignamos lo que queda por hacer?)

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

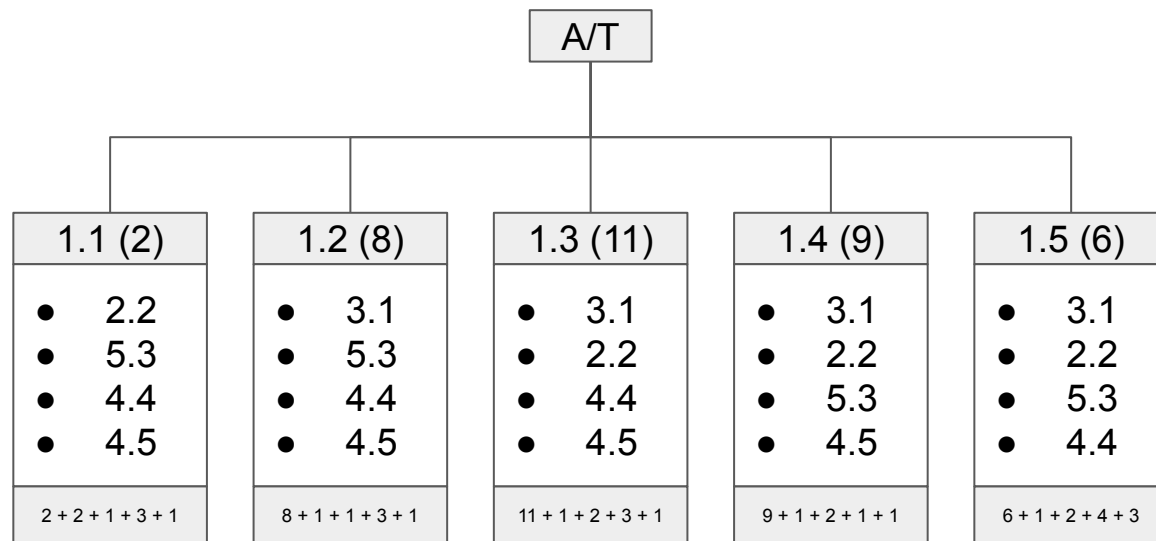


# N tareas, N agentes

Rango de valores válidos = [8, 13]

## 3. Cálculo de nueva cota

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

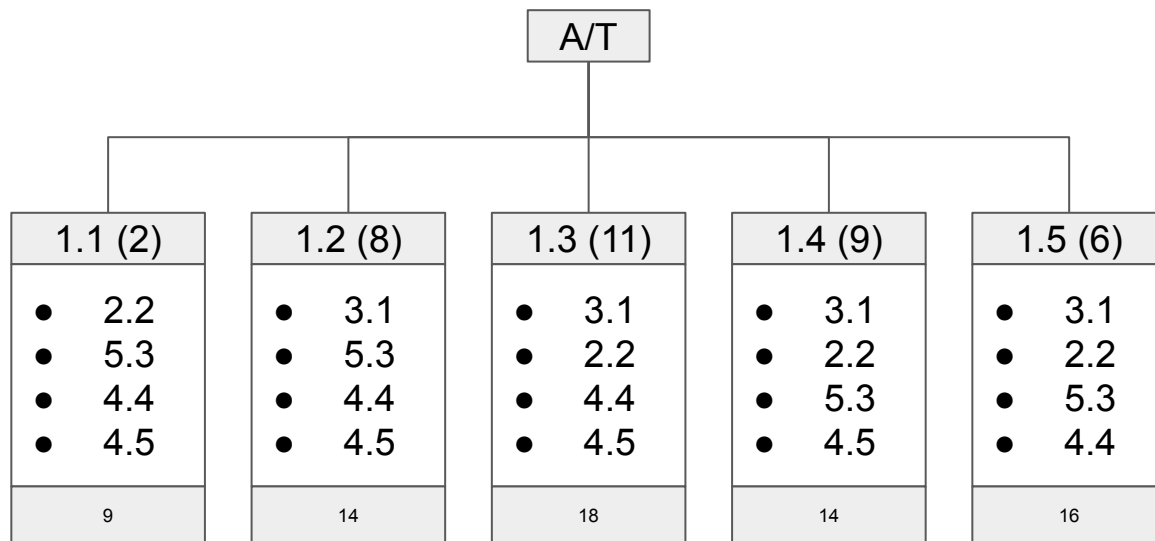


# N tareas, N agentes

## 3. Cálculo de nueva mínima

Rango de valores válidos = [8, 13]

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

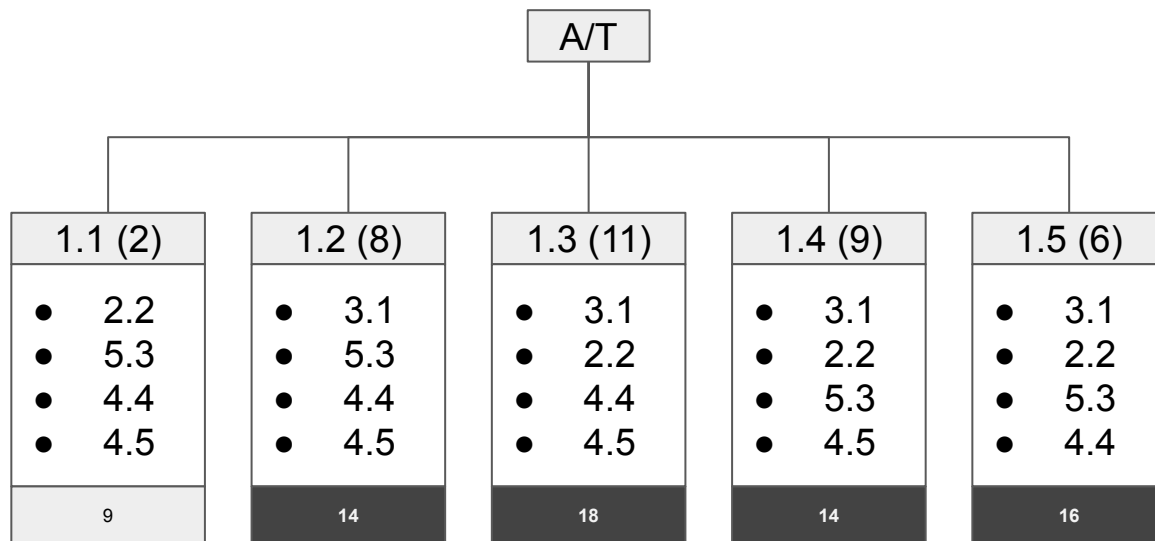


# N tareas, N agentes

Rango de valores válidos = [8, **13**]

## 4. Poda de valores que superen cota de referencia

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

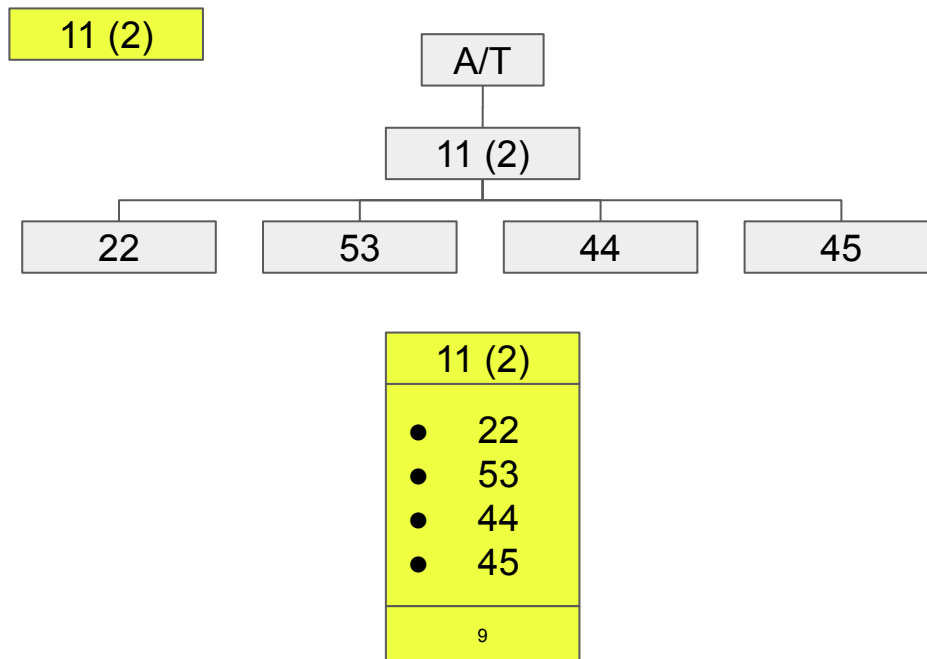


# N tareas, N agentes

Rango de valores válidos = [8, 13]

5. Repetir procedimiento explorando nodos supervivientes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2



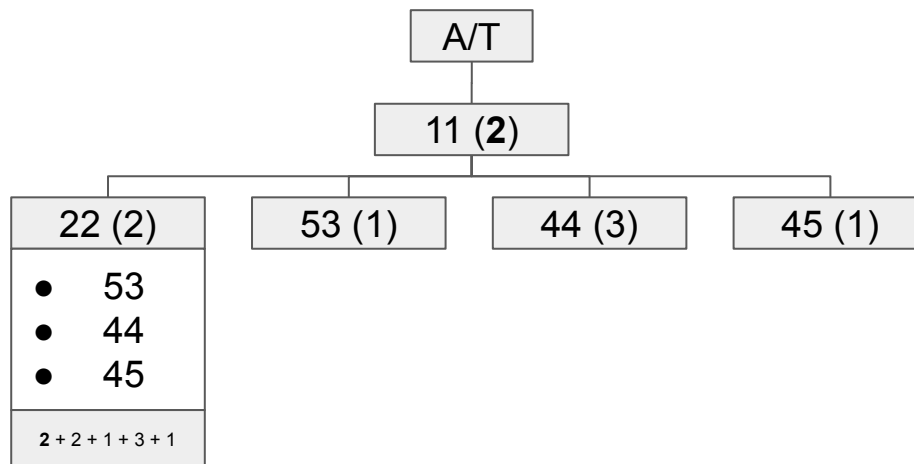


# N tareas, N agentes

Rango de valores válidos = [8, 13]

5. Repetir procedimiento explorando nodos supervivientes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

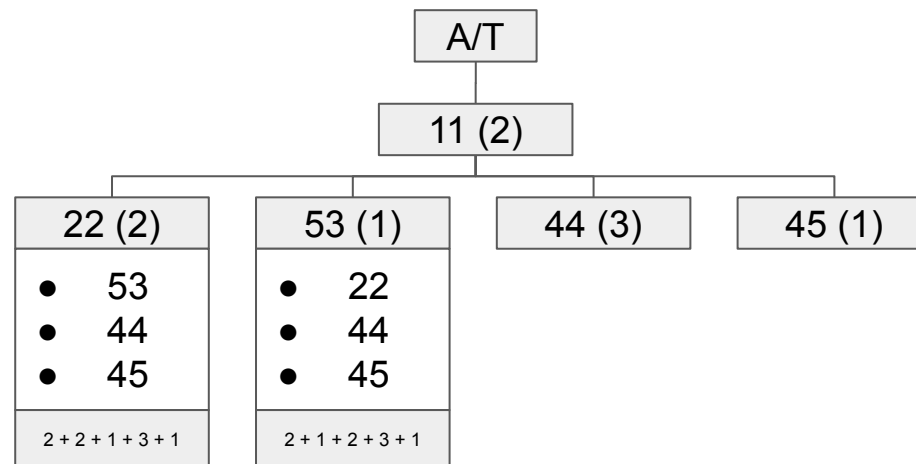


# N tareas, N agentes

Rango de valores válidos = [8, 13]

5. Repetir procedimiento explorando nodos supervivientes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

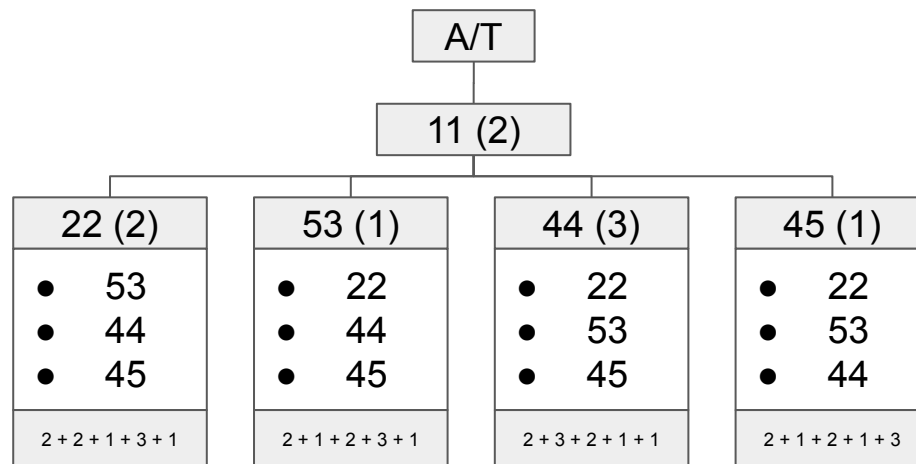


# N tareas, N agentes

Rango de valores válidos = [8, 13]

5. Repetir procedimiento explorando nodos supervivientes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

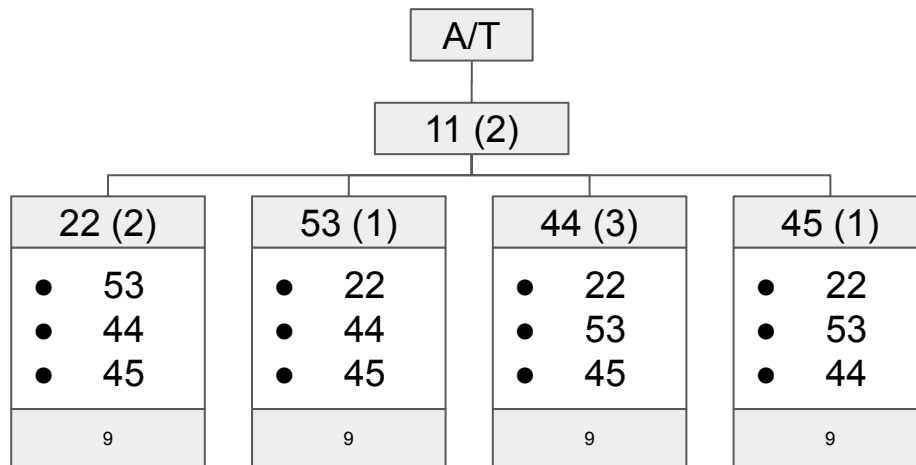


# N tareas, N agentes

Rango de valores válidos = [8, 13]

5. Repetir procedimiento explorando nodos supervivientes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

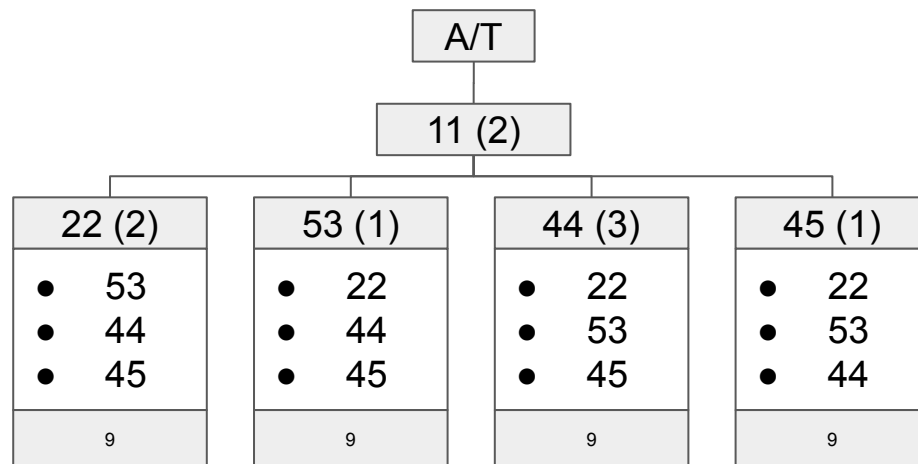


# N tareas, N agentes

Rango de valores válidos = [8, 13]

6. Si no se producen descartes, crear pila (lista FIFO) de exploración

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2



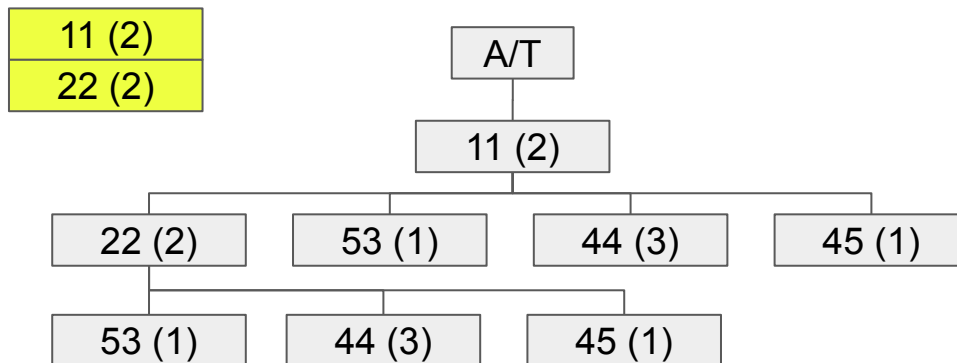
11 (2)	11 (2)	11 (2)	11 (2)
45 (1)	44 (3)	53 (1)	22 (2)

# N tareas, N agentes

Rango de valores válidos = [8, 13]

7. Y extraer elementos 1 a 1, repitiendo el procedimiento

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2



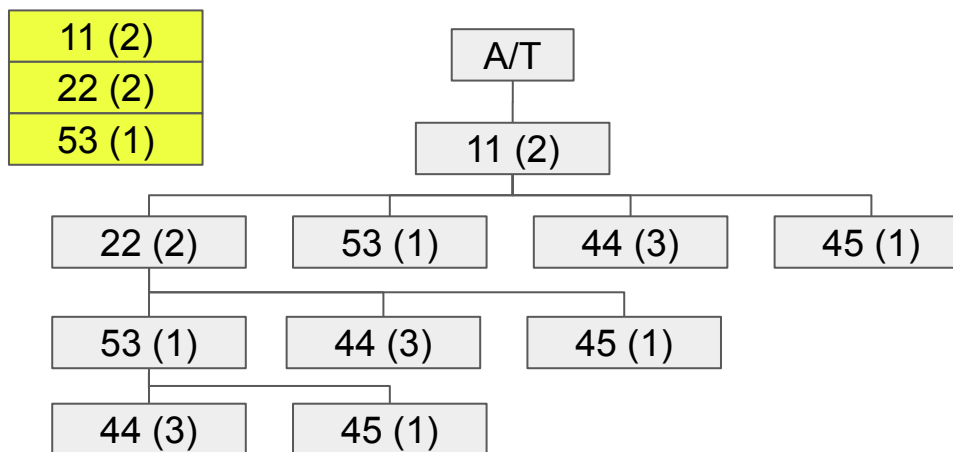
11 (2)	11 (2)	11 (2)	11 (2)	11 (2)	11 (2)
45 (1)	44 (3)	53 (1)	22 (2)	22 (2)	22 (2)
			45 (1)	44 (3)	53 (1)

# N tareas, N agentes

Rango de valores válidos = [8, 13]

7. Y extraer elementos 1 a 1, repitiendo el procedimiento

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2



11 (2)	11 (2)	11 (2)	11 (2)	11 (2)
45 (1)	44 (3)	53 (1)	22 (2)	22 (2)
			45 (1)	44 (3)

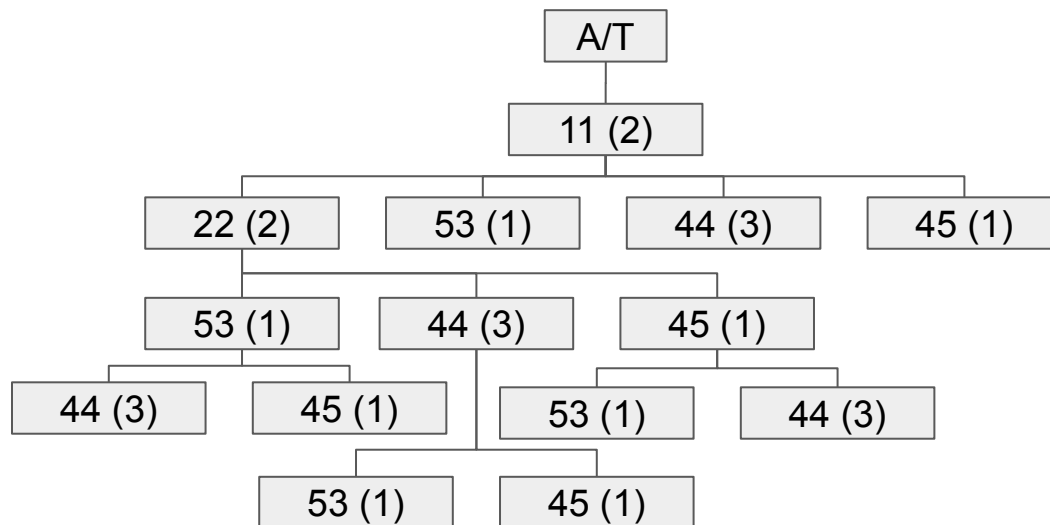
# N tareas, N agentes

Rango de valores válidos = [8, 13]

7. Y extraer elementos 1 a 1, repitiendo el procedimiento

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

11 (2)	11 (2)	11 (2)
45 (1)	44 (3)	53 (1)



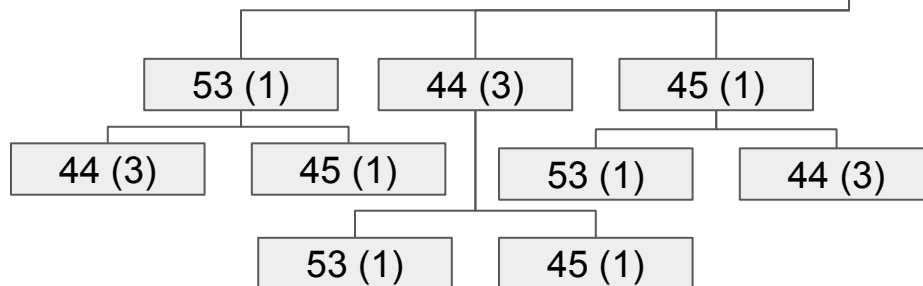
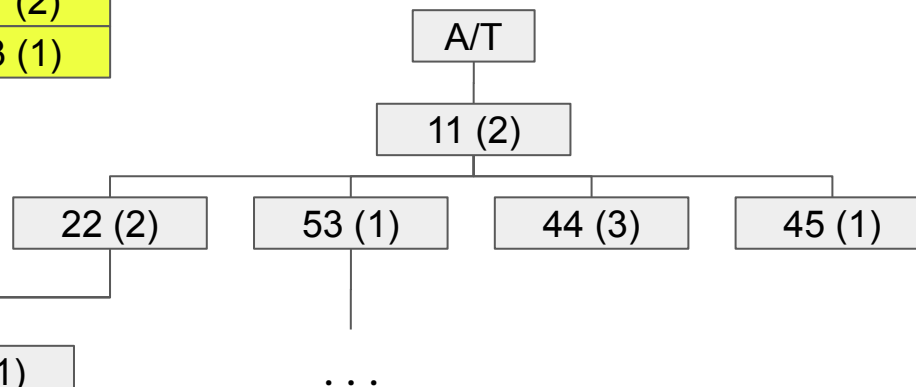


# N tareas, N agentes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

7. Y extraer elementos 1 a 1, repitiendo el procedimiento

11 (2)
53 (1)



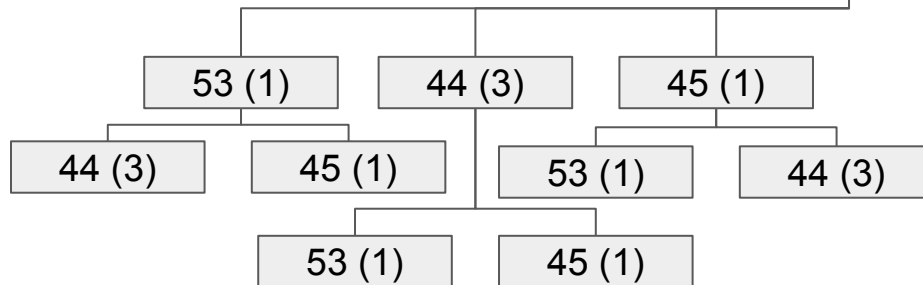
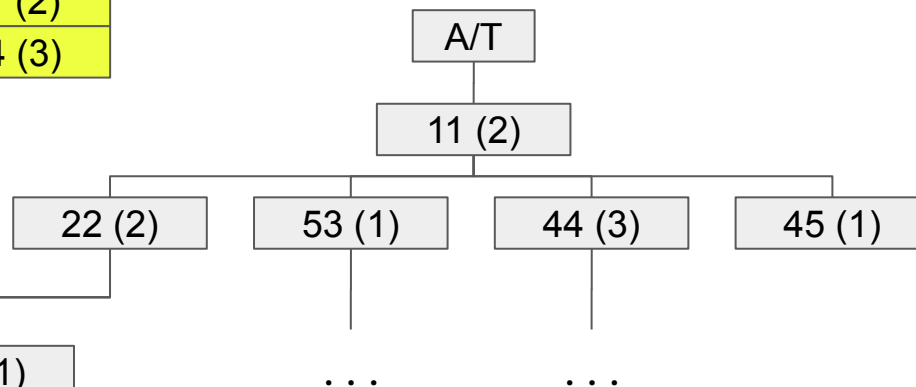
11 (2)	11 (2)
45 (1)	44 (3)

# N tareas, N agentes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

7. Y extraer elementos 1 a 1, repitiendo el procedimiento

11 (2)
44 (3)



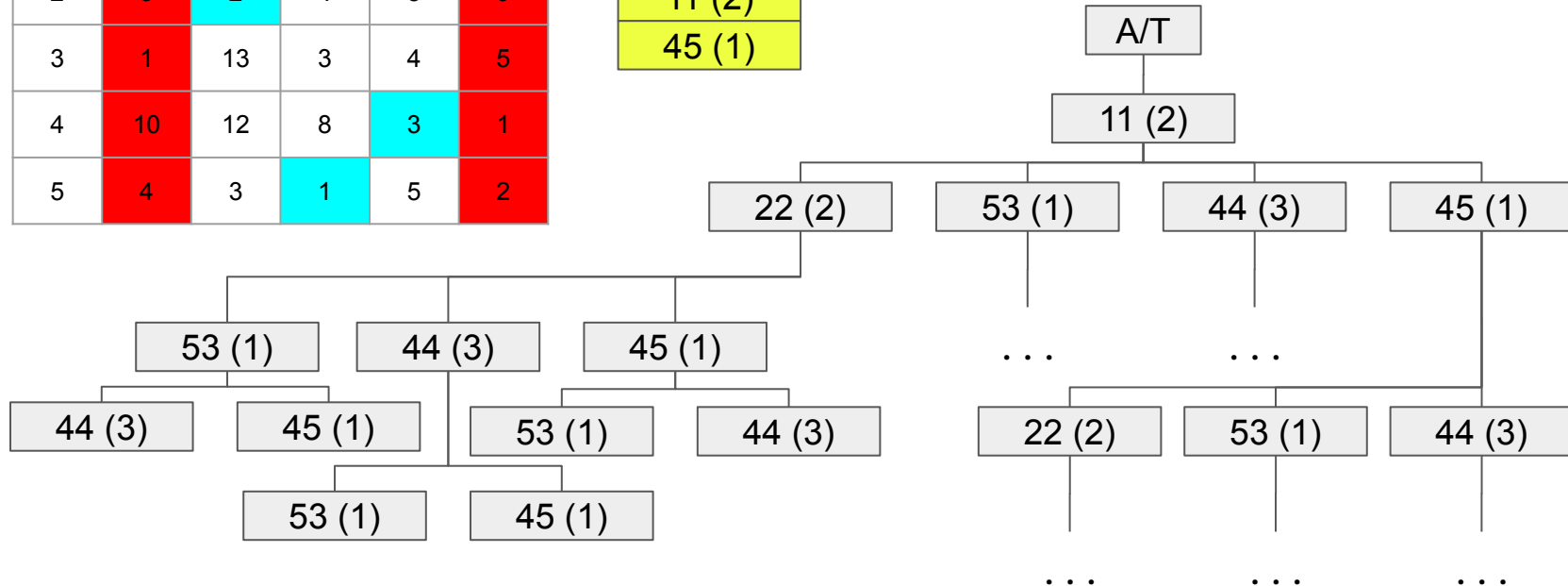
11 (2)
45 (1)

# N tareas, N agentes

A\CT	1	2	3	4	5
1	2	8	11	9	6
2	3	2	4	8	9
3	1	13	3	4	5
4	10	12	8	3	1
5	4	3	1	5	2

11 (2)
45 (1)

7. Y extraer elementos 1 a 1, repitiendo el procedimiento





¿Preguntas?