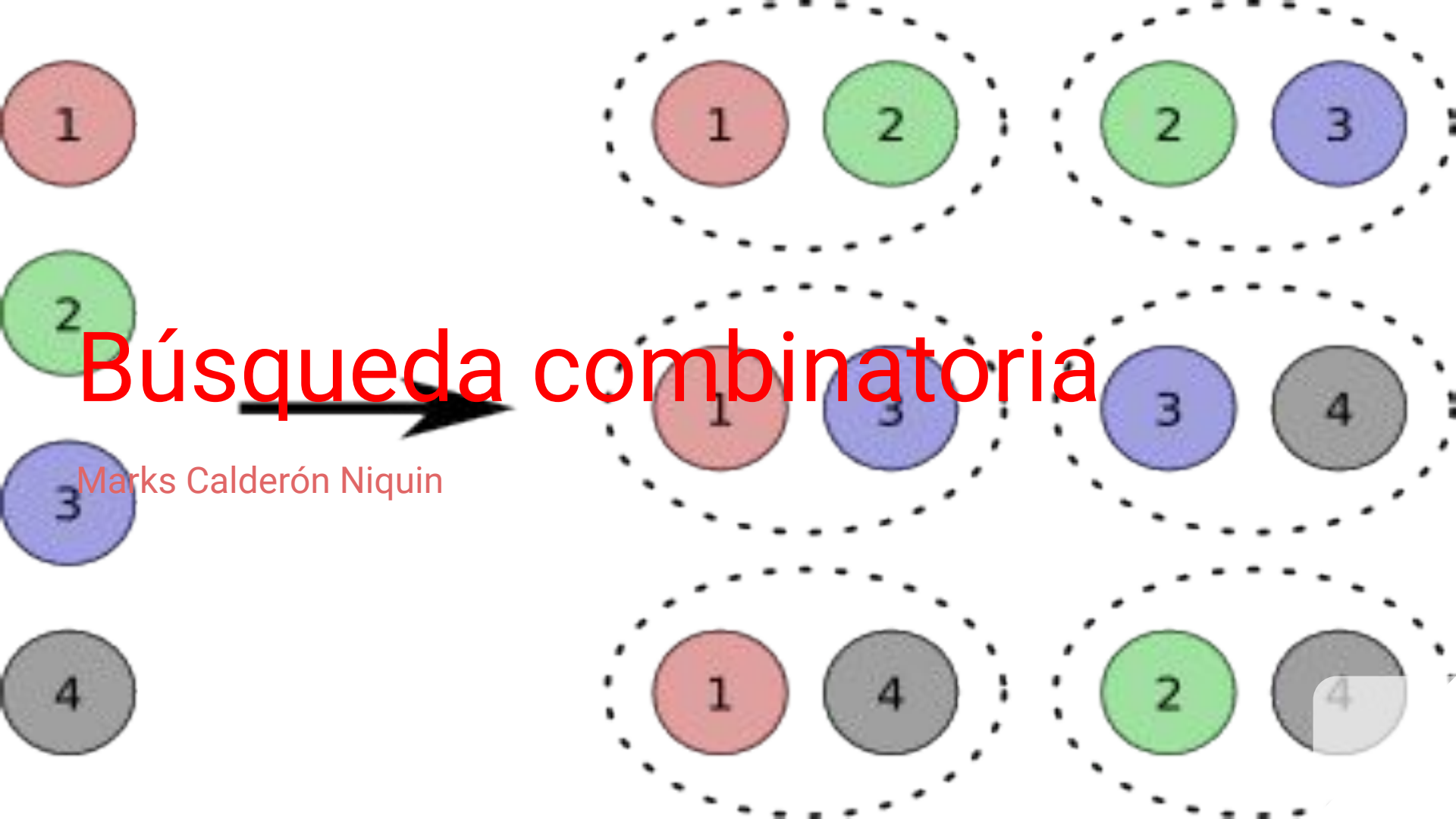


Búsqueda combinatoria

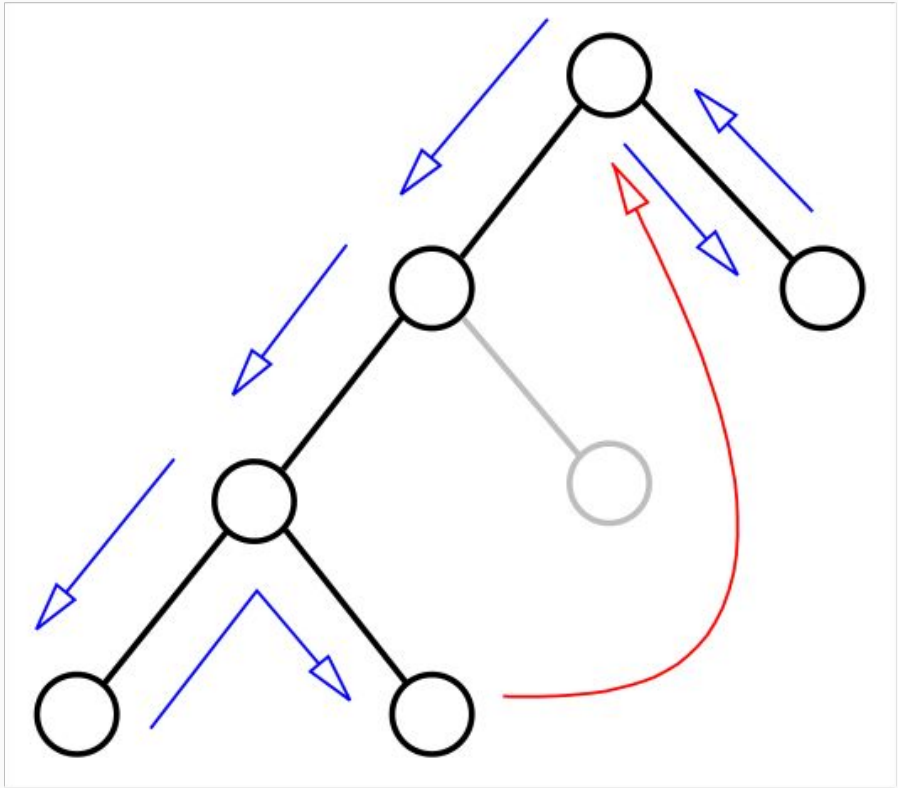
Marks Calderón Niquin

Backtracking

- Backtracking
- Problema de n reinas
- Suma de subconjuntos
- Sudoku

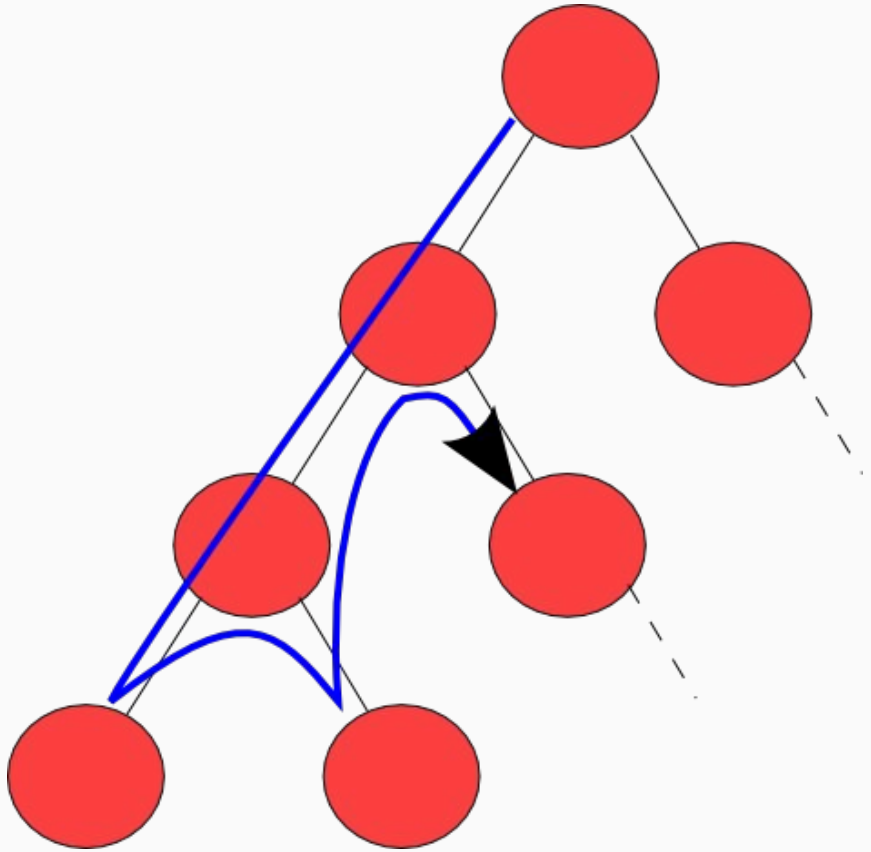
Backtracking

- Suponemos que tenmos una serie de decisiones, sobre diferentes decisiones donde:
  - No tienes la suficiente información conocida de que escoger.
  - Cada decisión lidera a un nuevo conjunto de decisiones
  - Algunas secuencias de elecciones (posiblemente más que uno) puede ser soluciones a tu problema.
- **Backtracking** es un método que intenta diferentes secuencias de decisiones hasta encontrar una que **funcione**



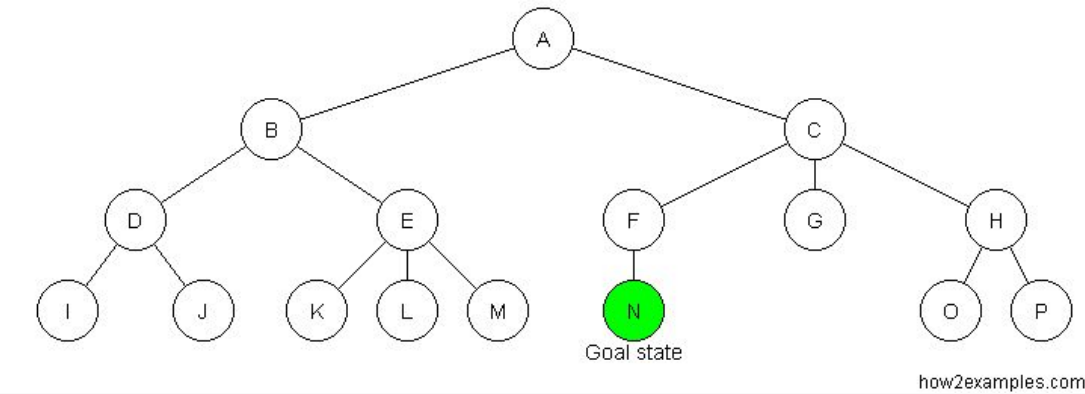
Backtracking

- **Backtracking** usado para resolver problemas donde en una secuencia de decisiones escogida desde un conjunto específico de secuencias que satisface algunos criterios.
- **Backtracking** modificado a **búsqueda primero en profundidad** de un árbol.
- **Backtracking** envuelve solo un arbol de busqueda.
- **Backtracking** es el procedimiento por el cual, después de determinar que un nodo puede conducir a nada más que a la destrucción de nodos, volvemos ("retroceso") al padre del nodo y proceda con la búsqueda del próximo hijo.



Backtracking

- Llamamos a un nodo **no prometedor** si al visitar el nodo, determinamos que no puede conducir a una solución. De lo contrario, lo llamamos **prometedor**.
- **Backtracking** consiste en:
  - Haciendo una búsqueda primero en profundidad del árbol de espacio de espacio.
  - Verificar si cada nodo es **prometedor,** y si no lo es, backtracking al nodo padre.
- Esta técnica es llamado **poda** el espacio de estados del árbol, y el subárbol consiste de los nodos visitados es llamado árbol de espacio de estado podado..

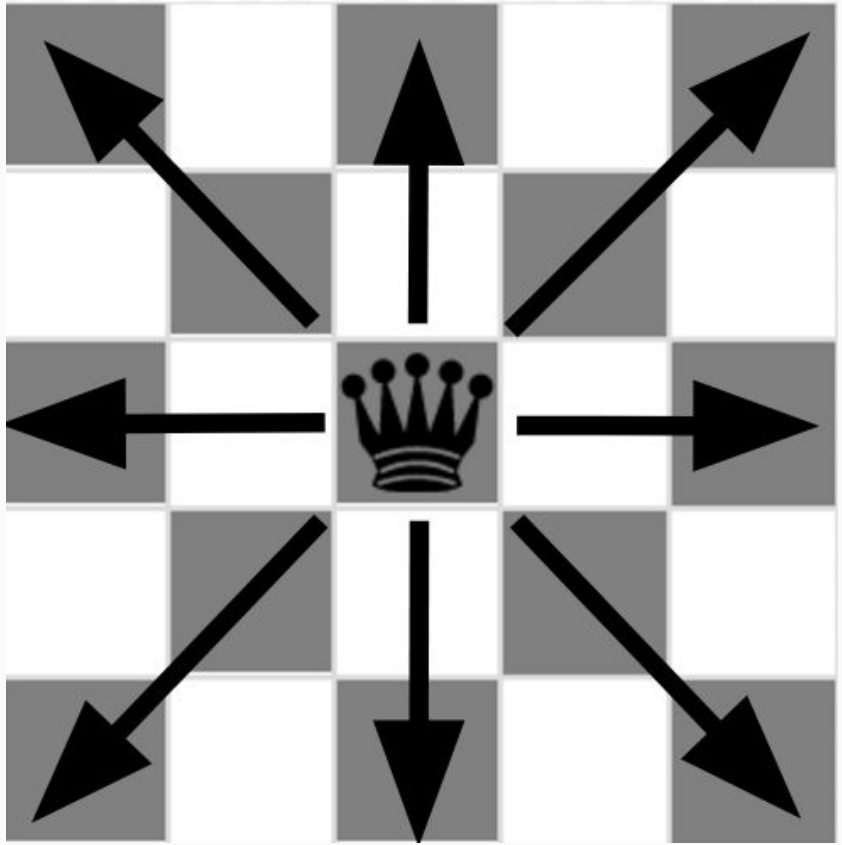


Problema de n reinas

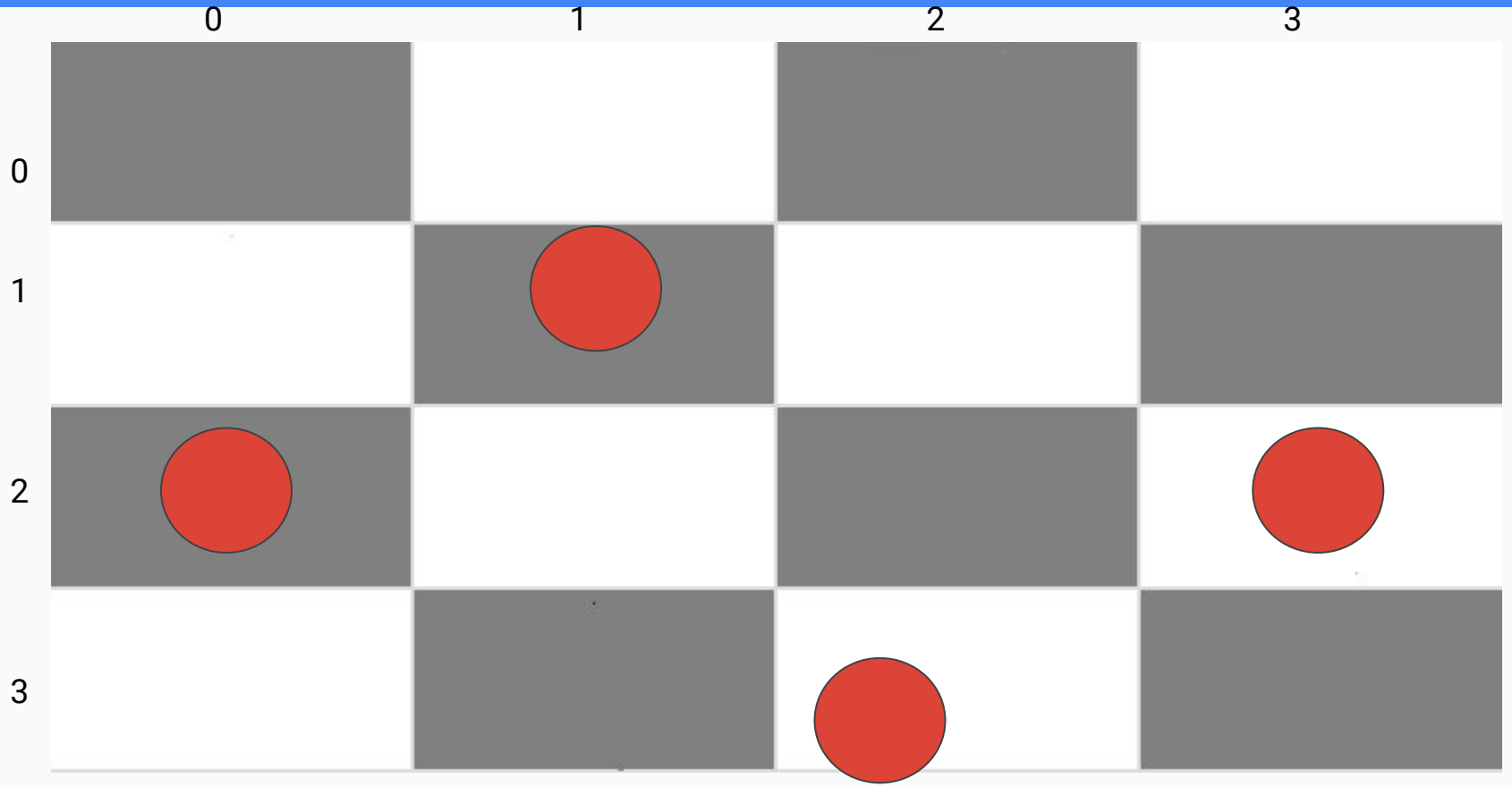
- Backtracking
- Problema de n reinas
- Suma de subconjuntos
- Sudoku

Problema de n reinas

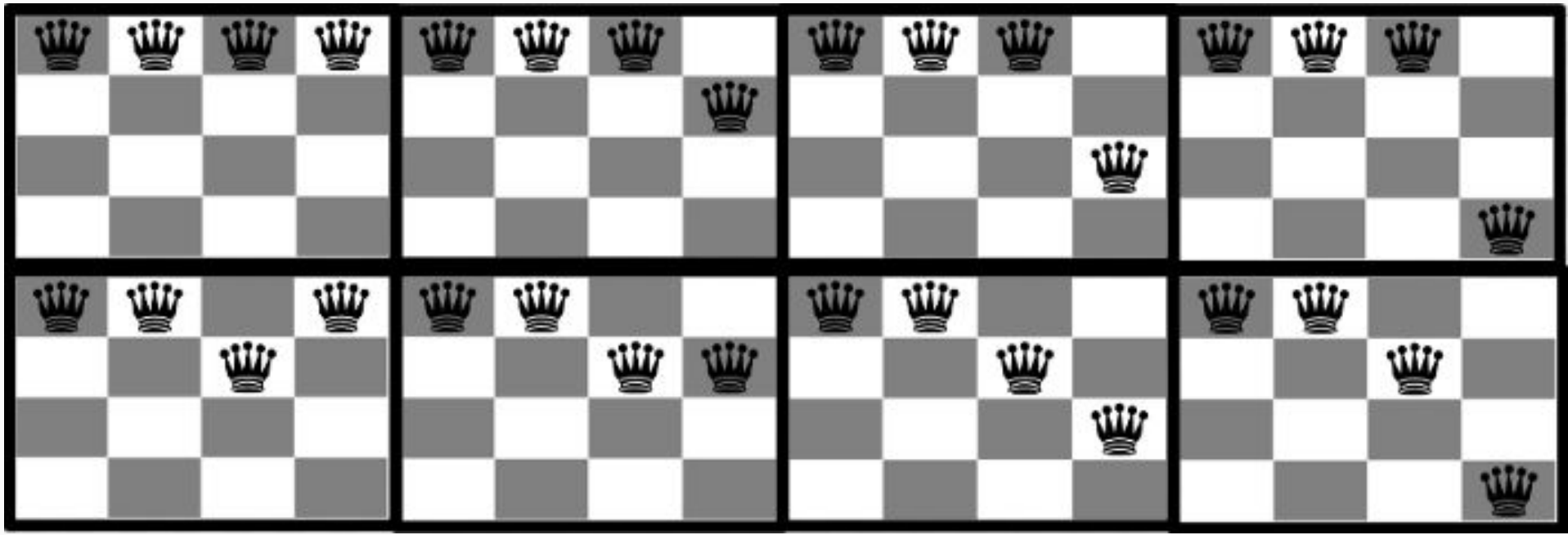
- Intentar poner N reinas en un tablero de N*N tal que ninguna reina pueda atacar a otra reina.
- Recuerde que las reinas pueden moverse en vertical, horizontal, o diagonal de cualquier tamanho.
- Consideremos 8 reinas como ejemplo



Ejemplo: problema de 4 reinas

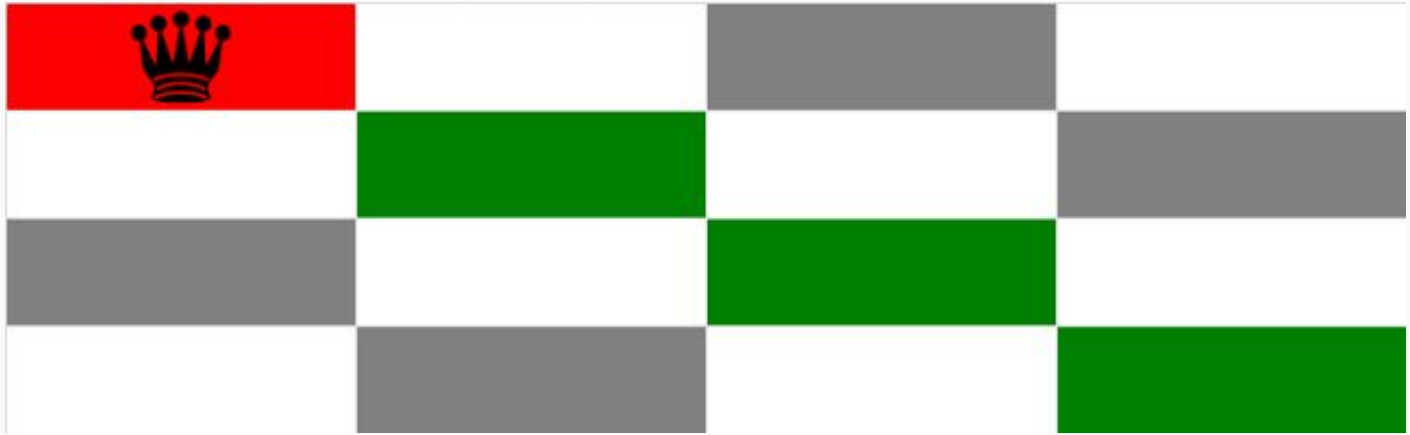


Ejemplo: problema de 4 reinas



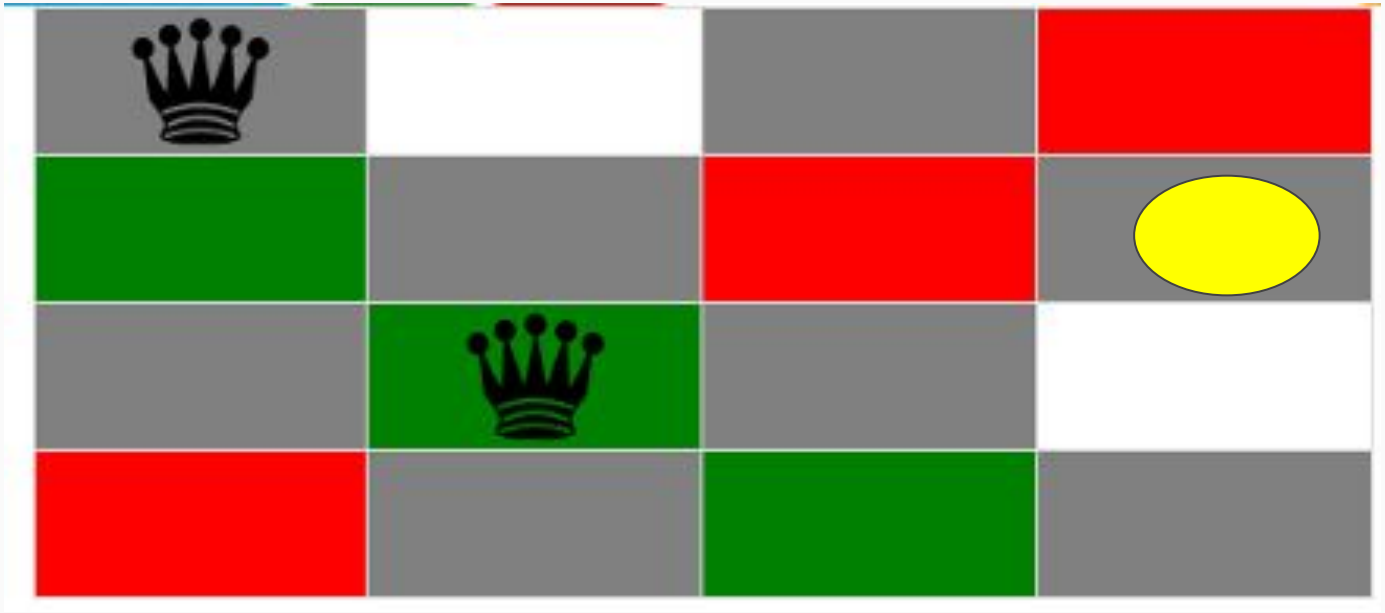
https://medium.com/@andreirtaylor/n-queens-visualized-61b841af753f

Ejemplo: problema de 4 reinas



https://medium.com/@andreirtaylor/n-queens-visualized-61b841af753f

Ejemplo: problema de 4 reinas

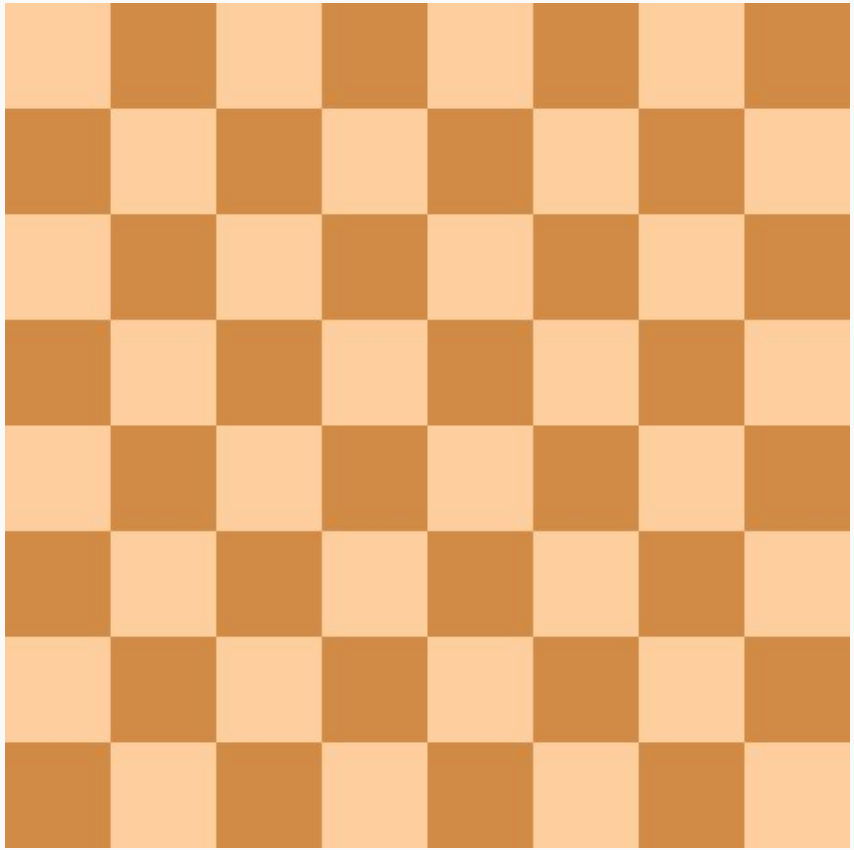


https://medium.com/@andreirtaylor/n-queens-visualized-61b841af753f

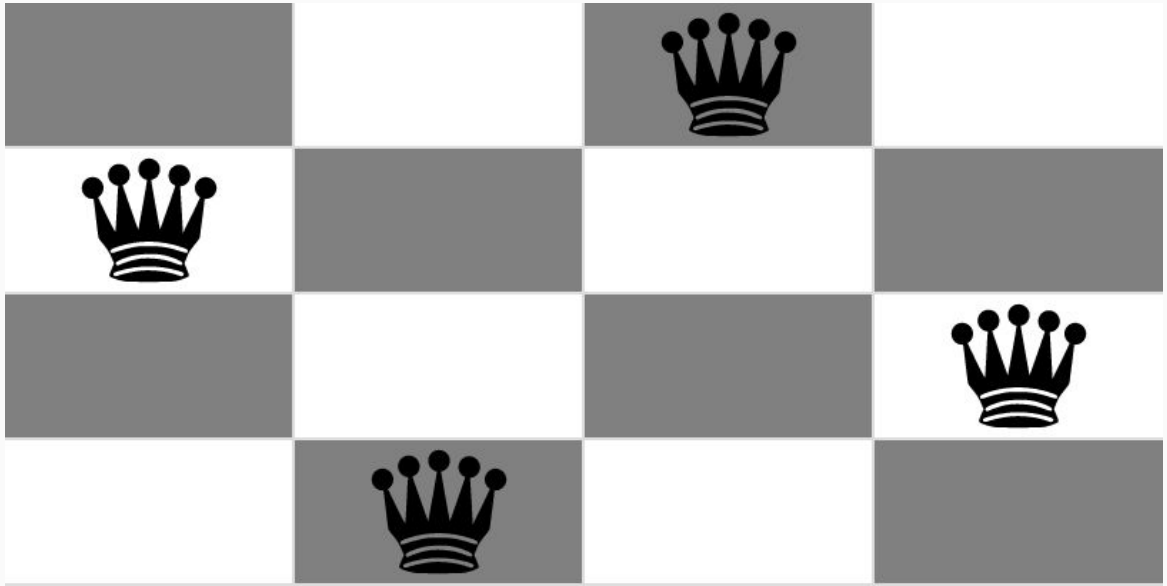
N=4 reinas por fuerza bruta

Idea?

$$N^N$$



Ejemplo: problema de 4 reinas

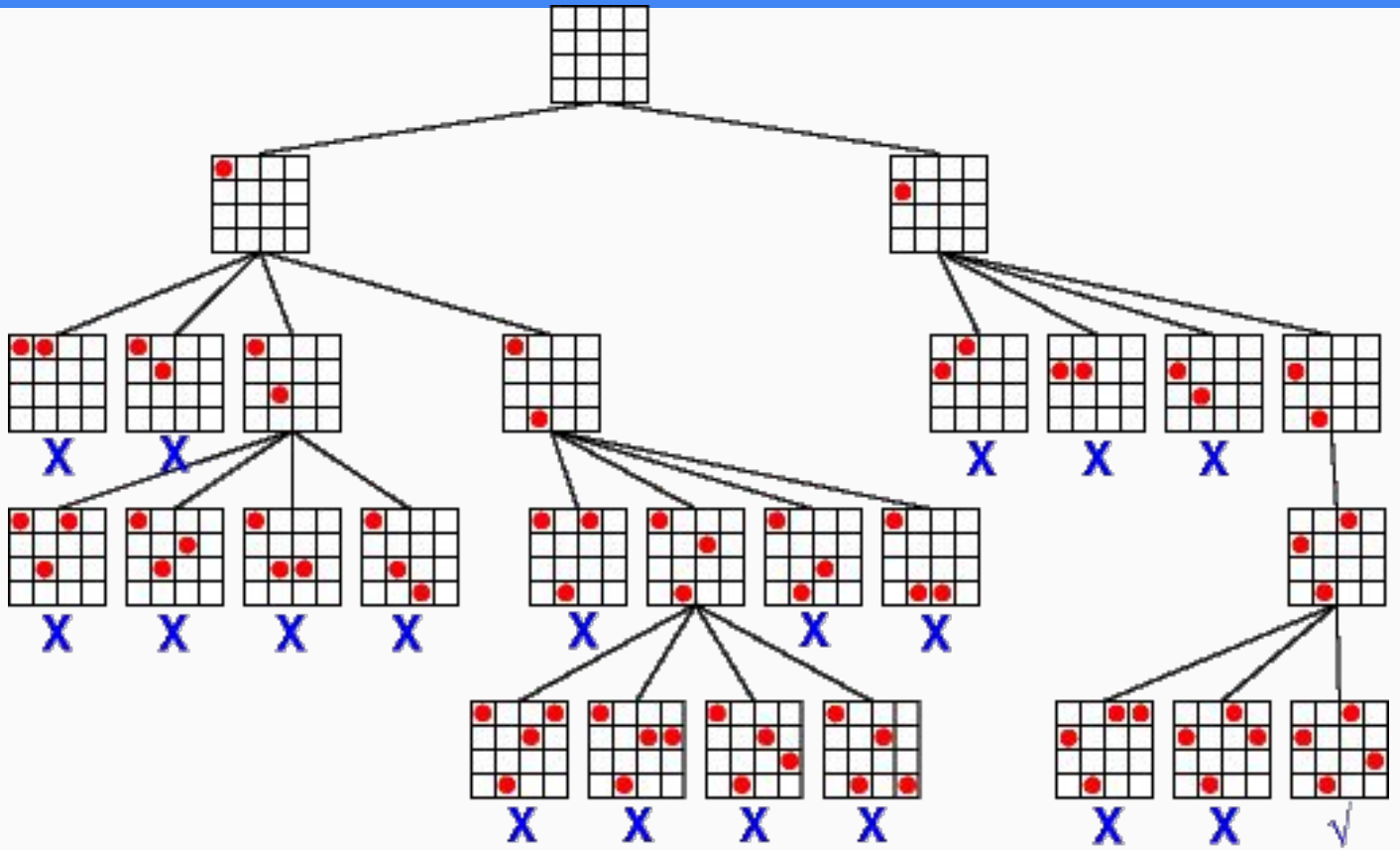


[2,4,1,3]

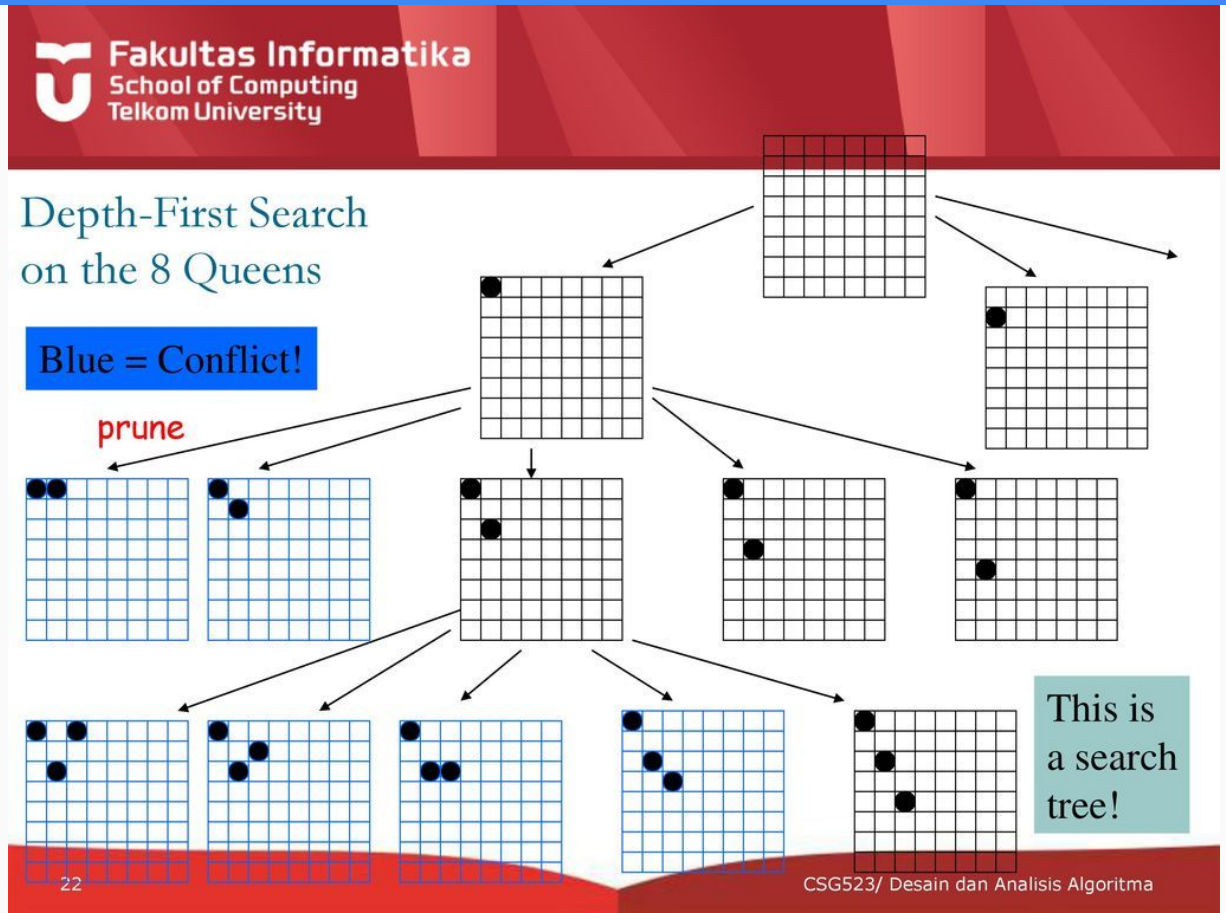
[1,,2,3,4],[.....]

4x4x4x4 = 4^4 = 256

Ejemplo: problema de 4 reinas

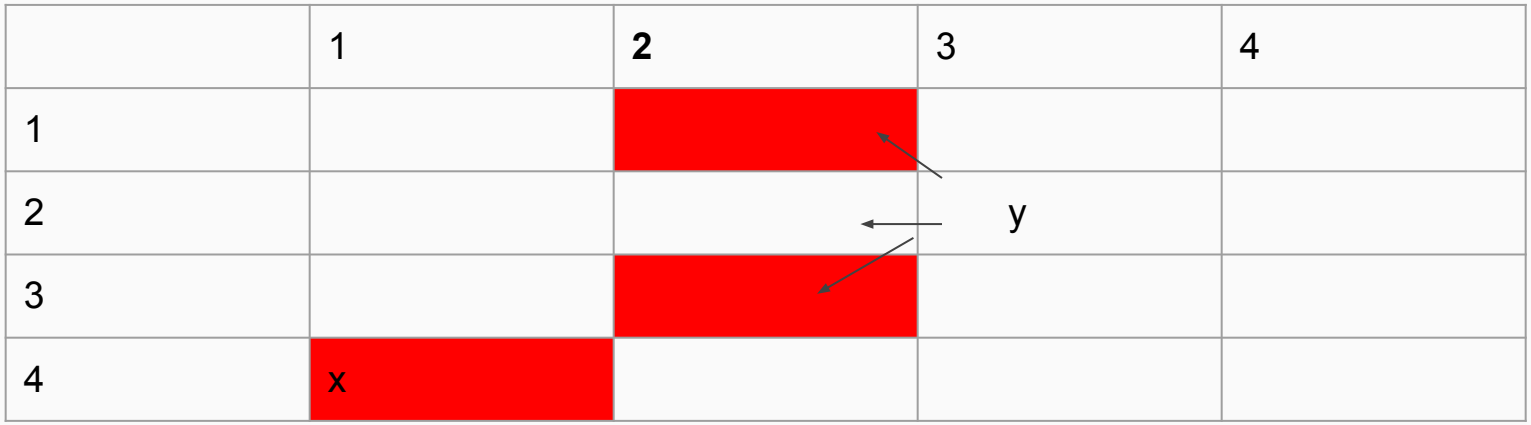


N=8 reinas estrategia



columna 3 , i=3

	1	**2**	3	4
1				
2			y	
3				
4	x			



```
for j=1 hasta i-1

   ....horizontal
  #diagonal
  if(  i - j == || col[j] - col[i]  || )
       conflicto
```

N=8 reinas estrategia

```
function noConflicts(columns, i) {
    for (var j=0; j<i; j++) {
        // same row
        if (columns[j] === columns[i]) {
            return false;
        }

        // diagonal
        if (i-j === Math.abs(columns[j]-columns[i])) {
            return false;
        }
    }
    return true;
}
```

https://thimbleby.gitlab.io/algorithm-wiki-site/wiki/n-queens/

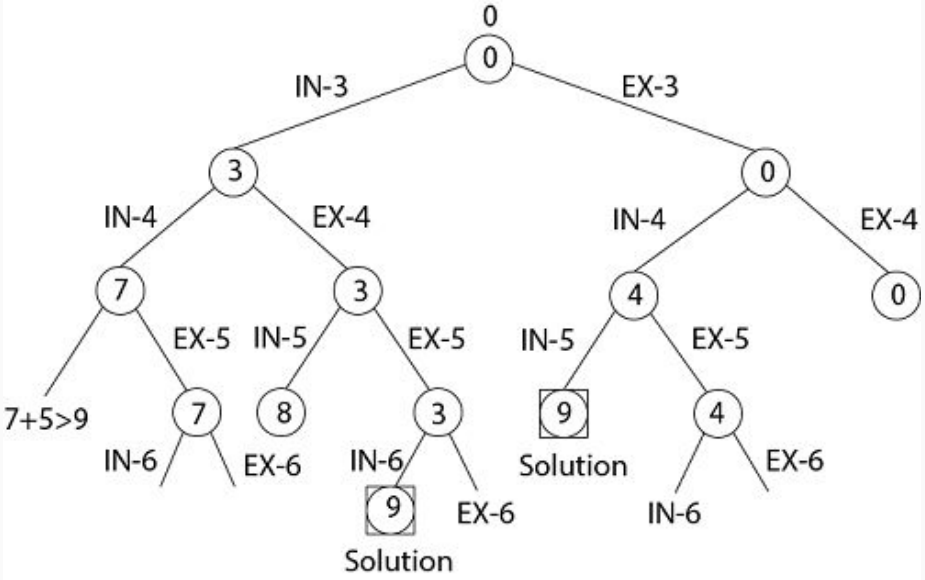
```
// recursively solves the n-queens problem by brute force
function queens(columns, i, n) {
    // we have a complete solution
    if (i === n) {
        return columns;
    }
    // try placing the next queen on every row
    for (var j=0; j<n; j++) {
        columns[i] = j;
        if (noConflicts(columns, i)) {
            var solution = queens(columns, i+1, n);
            if (solution) {
                return solution;
            }
        }
    }
    columns.pop();
    return false;
}

// returns an array that records the queens' position in each
column
// or false if no solution is found
solution = queens([], 0, 8);
```

Suma de subconjuntos

- Backtracking
- Problema de n reinas
- Suma de subconjuntos
- Sudoku

Suma de subconjuntos

- El problema de la suma de subconjuntos es encontrar un subconjunto S’ del conjunto S = (S_1, S_2, S_3, ..., S_n) donde el conjunto S son positivos.
- La suma del subconjunto S’ es igual a un entero positivo X .
- La complejidad del algoritmo con fuerza bruta es:
  - $O(2^N)$
- Podemos resolver el problema mediante backtracking. Esto implica un arbol binario.

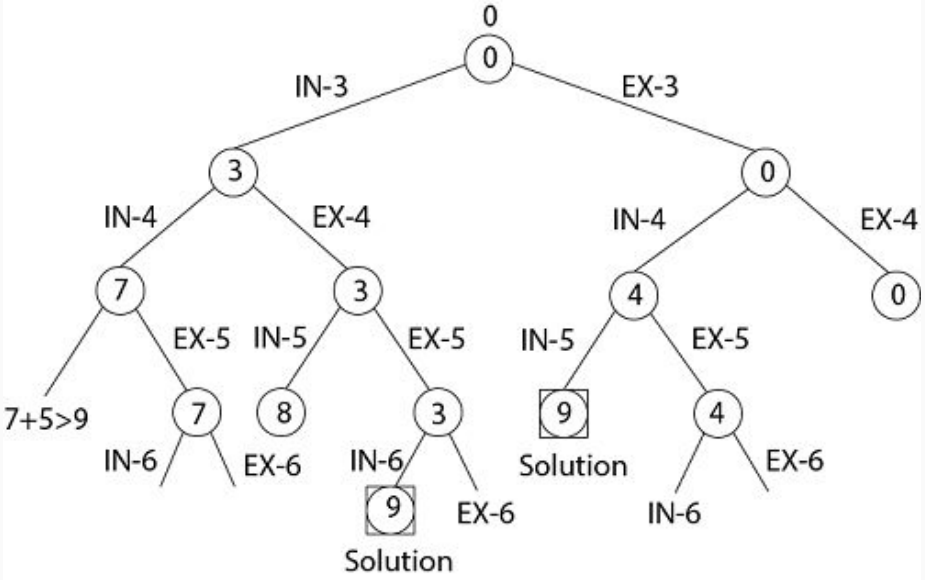


Suma de subconjuntos

- La raiz del arbol es seleccionado de una manera que no tome ninguna entrada.
- Asumimos que los elementos del conjunto dado están ordenados de forma creciente:

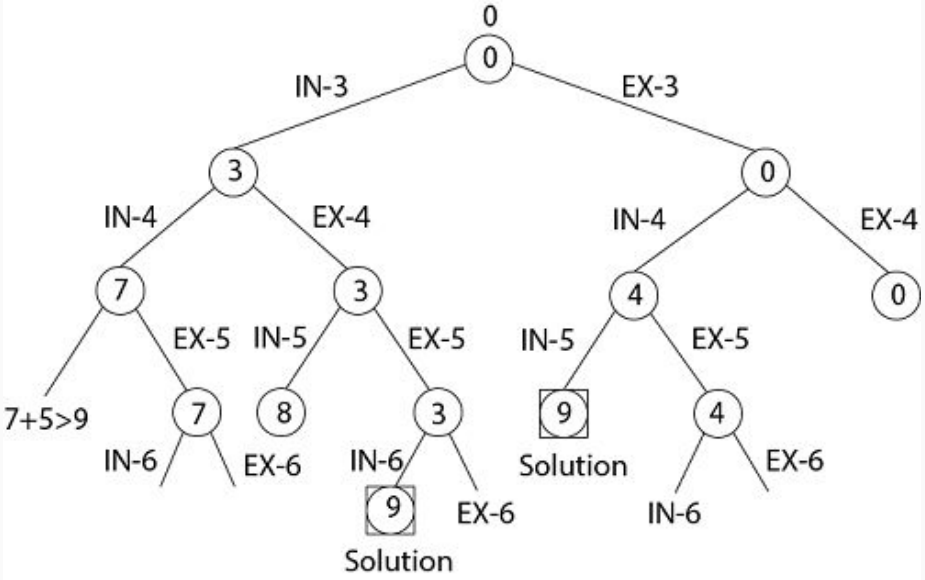
  $S_1 <= S_2 <= S_3 \ldots <= S_n$

- El hijo izquierdo indica que incluimos S_1 al conjunto S y el hijo derecho de la raíz indica que excluimos a S_1
- Cada nodo almacena la solución parcial.
- Si la suma de cualquier estado es la suma equivalente a 'X' entonces la búsqueda es exitosa y termina.
- La poda del árbol es cuando:
  - La suma es muy grande
  - La suma es muy pequeña



Suma de subconjuntos

- Dado un conjunto S = (3,4,5,6) y X = 9. Obtener el subconjunto suma.
- Reto 1: programar si lo encuentra
- Reto 2: que retorne el subconjunto



Sudoku

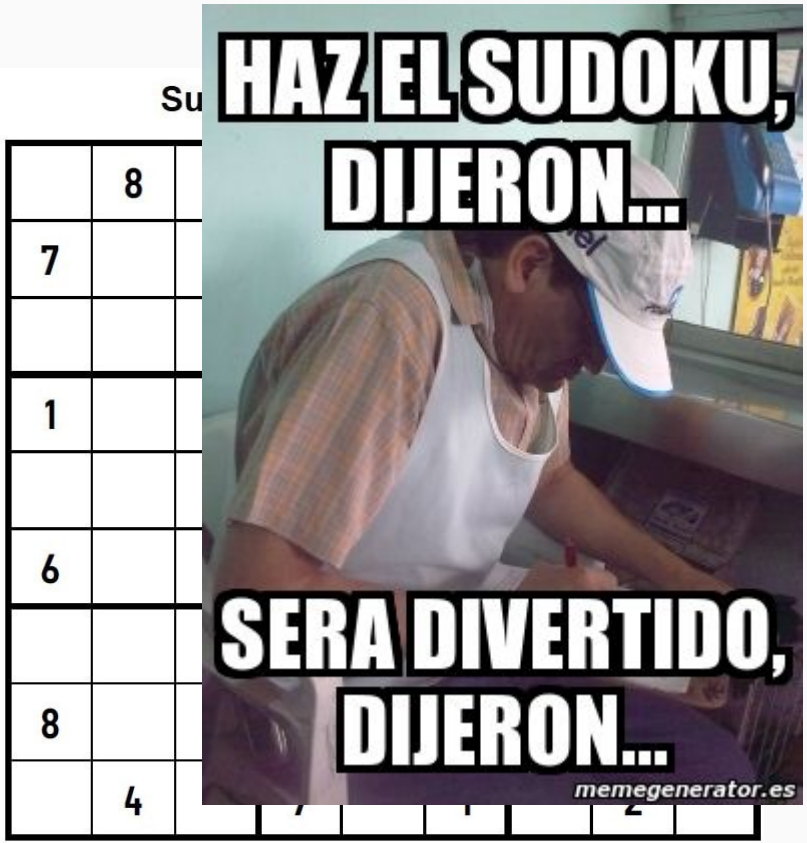
- Backtracking
- Problema de n reinas
- Sudoku

Sudoku

5			1			7		
	2				7	1		
3		1	4			8	5	2
6	1		5	7	2	4		8
		2	9	6				
	4			3		6	2	7
4	5	9		8			7	
1	3					9	8	6
2				1			4	3

8	7	6	9					
	1				6			
	4		3		5	8		
4						2	1	
	9		5					
	5			4		3		6
	2	9						8
		4	6	9		1	7	3
					1			4

Sudoku



Combinaciones posibles

6,670,903,752,021,072,936,960

(6.67×10^{21})

Sudoku: solución

	8	4	1	9	6	2	3	5
3	2	1	5	8	4	6	9	7
9	5	6	3	2	7	4	1	8
2	9	7	4	5	1	3	8	6
8	4	5	9	6	3	1	7	2
6	1	3	8	7	2	9	5	4
1	7	9	6	4	5	8	2	3
4	3	2	7	1	8	5	6	9
5	6	8	2	3	9	7	4	1

función sudoku(mat)

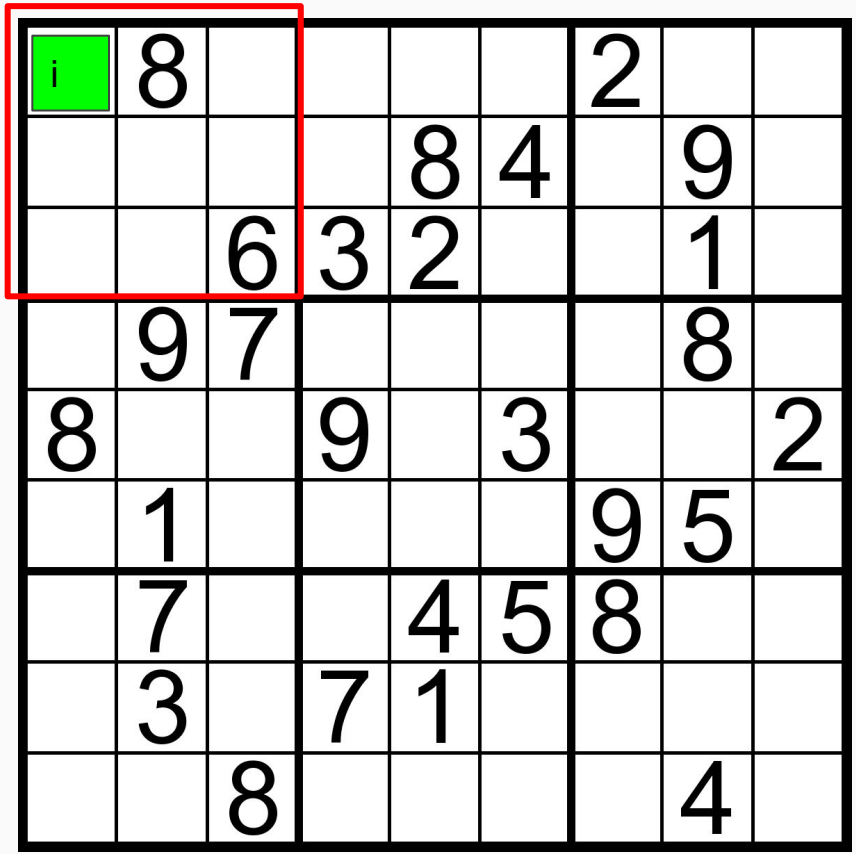
pos = Encontrar **posición** de una **celda vacía** (mat)

Sudoku: solución

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

```
función sudoku(mat)

    pos = Encontrar posición de una celda vacía (mat)

    Si pos no existe

        retornar True

    sino

        x , y = pos
```

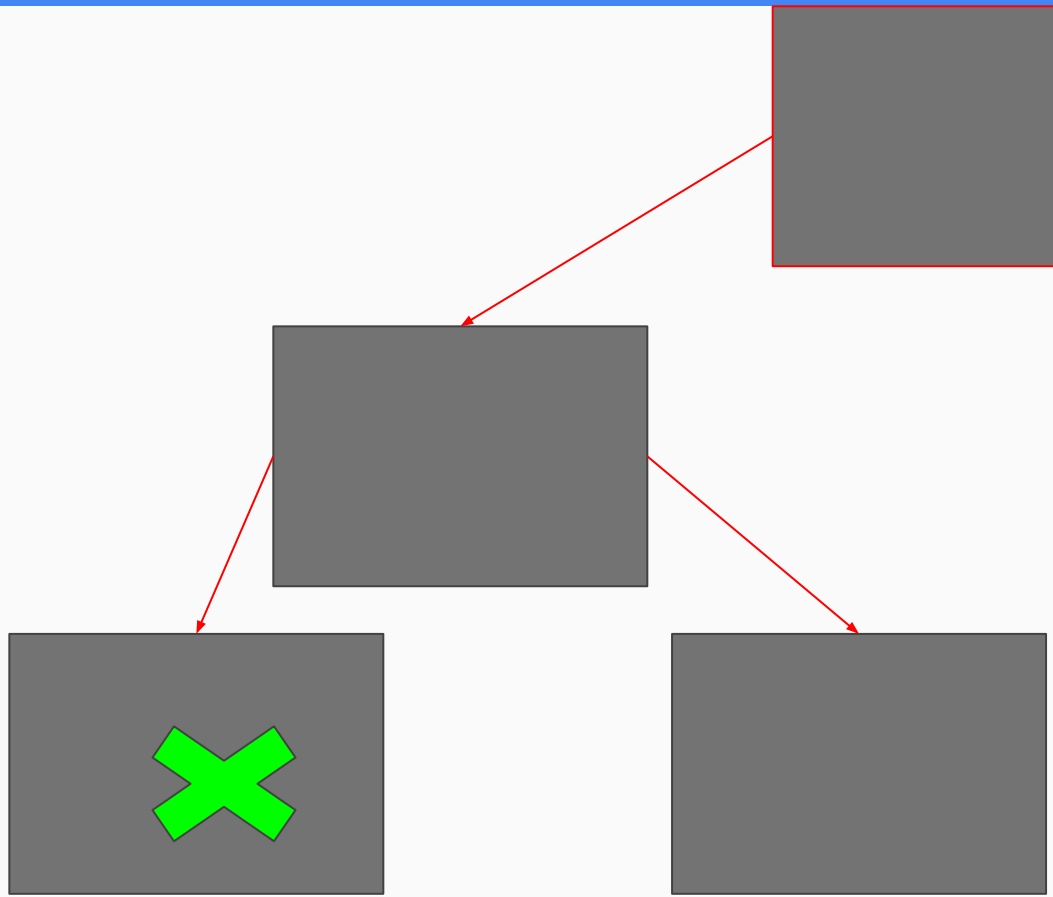
Sudoku: solución

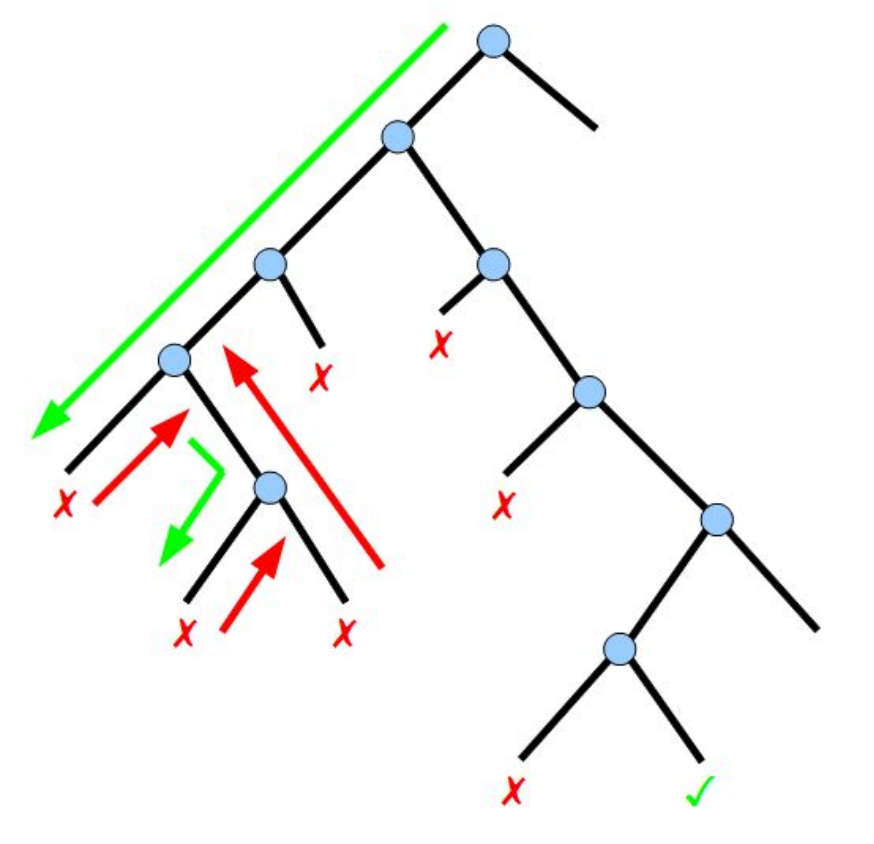



```
función sudoku(mat)
        pos = Encontrar posición de una celda vacía (mat)
        Si pos no existe
                retornar True
        sino
                x , y = pos

        Para i=1 hasta 9
                Si Valido(mat, i, x,y)
                        mat[x,y] = i

                        Si sudoku(mat)
                                retornar True

                        mat[x,y] = 0

        retornar Falso
```

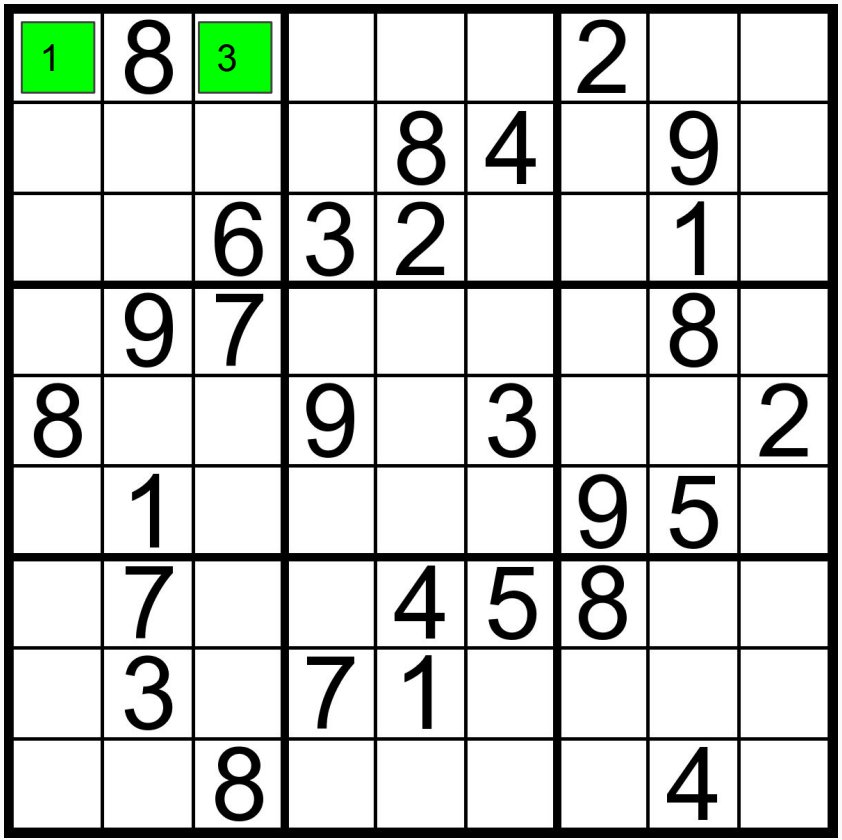
Sudoku: solución

```
función sudoku(mat)
        pos = Encontrar posición de una celda vacía (mat)
        Si pos no existe
                retornar True
        sino
                x , y = pos

        Para i=1 hasta 9
                Si Valido(mat, i, x,y)
                        mat[x,y] = i

                        Si sudoku(mat)
                                retornar True

                        mat[x,y] = 0

        retornar Falso
```





Sudoku: solución




```
función sudoku(mat)
        pos = Encontrar posición de una celda vacía (mat)
        Si pos no existe
                retornar True
        sino
                x , y = pos

        Para i=1 hasta 9
                Si Valido(mat, i, x,y)
                        mat[x,y] = i

                        Si sudoku(mat)
                                retornar True

                        mat[x,y] = 0

        retornar Falso
```

Implementación Sudoku

Sudoku: solución

```
funcion encontrar_posicion_vacia(mat)
    para i=1 hasta 9
        para j=1 hasta 9
            Si mat[i,j] == 0
                return (i,j)
    return Nulo
```

```
función sudoku(mat)
    pos = Encontrar posición de una celda vacía (mat)
    Si pos no existe
        retornar True
    sino
        x , y = pos

    Para i=1 hasta 9
        Si Valido(mat, i, x,y)
            mat[x,y] = i

            Si sudoku(mat)
                retornar True

            mat[x,y] = 0

    retornar Falso
```

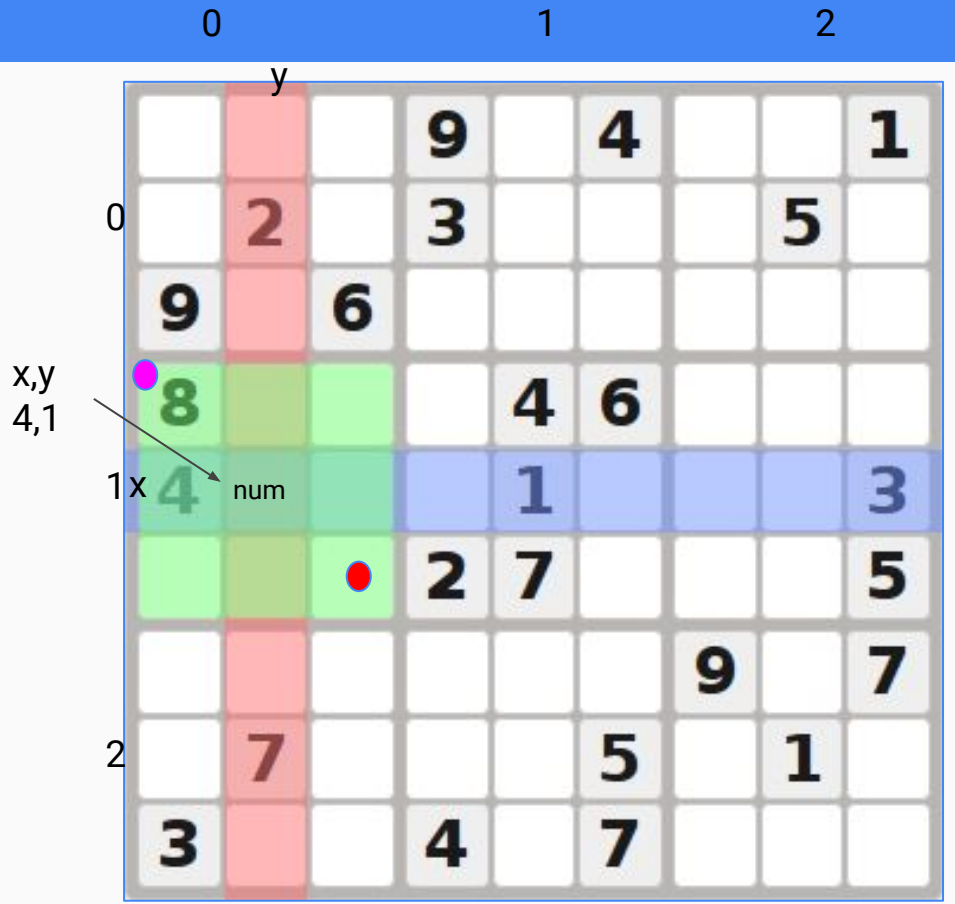
Sudoku: solución


```
función Valido(M, num,x,y)
    //fila
    para i=1 hasta 9
        Si M[x,i] == num && y != i
            return false
    //columna
    para i=1 hasta 9
        Si M[i,y] == num && x != i
            return false
    //checar celda
    idx = x//3
    idy = y//3
    para i= idx*3 hasta idx*3 + 2
        para j= idy*3 hasta idy*3 + 2
            Si M[i,j] == num and i!=x and j!=y
                return false
    return true
```

```
función sudoku(mat)
    pos = Encontrar posición de una celda vacía (mat)
    Si pos no existe
        retornar True
    sino
        x , y = pos

    Para i=1 hasta 9
        Si Valido(mat, i, x,y)
            mat[x,y] = i

            Si sudoku(mat)
                retornar True

            mat[x,y] = 0

    retornar Falso
```

Sudoku: solución

```
función Valido(M, num,x,y)
    //fila
    para i=1 hasta 9
        Si M[x,i] == num && y != i
            return false
    //columna
    para i=1 hasta 9
        Si M[i,y] == num && x != i
            return false
    //checar celda
    idx = x//3
    idy = y//3
    para i= idx*3 hasta idx*3 + 2
        para j= idy*3 hasta idy*3 + 2
            Si M[i,j] == num and i!=x and j!=y
                return false
    return true
```



Sudoku

http://norvig.com/sudoku.html

https://spin.atomicobject.com/2012/06/18/solving-sudoku-in-c-with-recursive-backtracking/