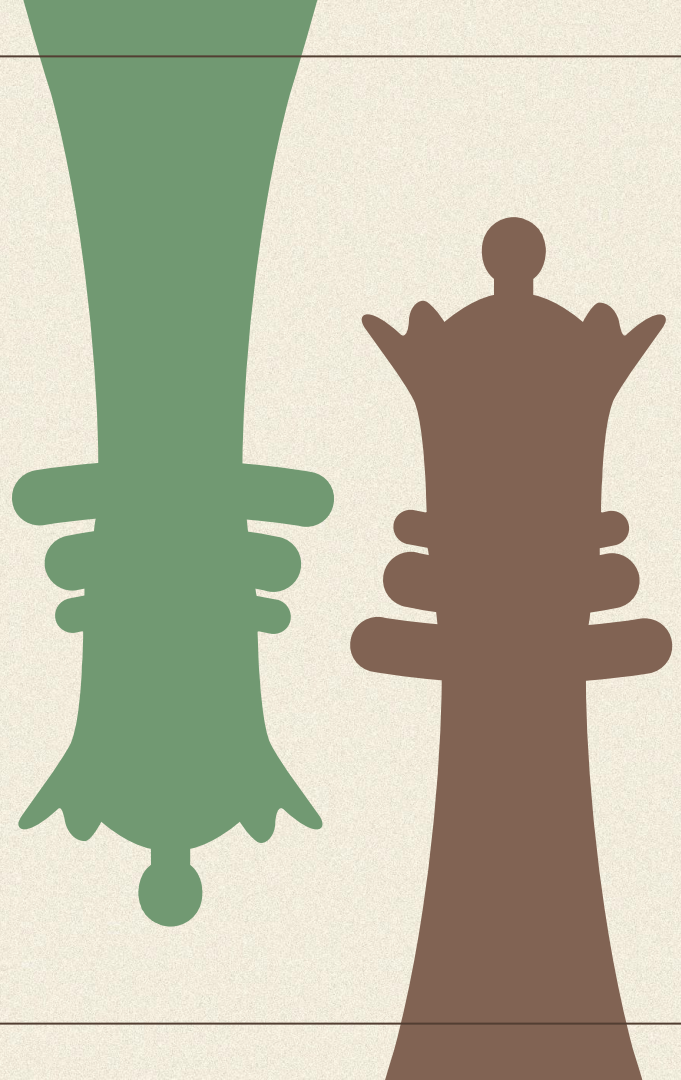


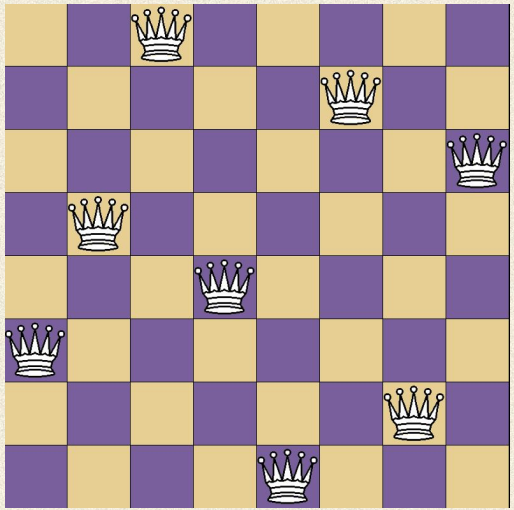
Análisis y Diseño de Algoritmos

Introducción.

El problema de las n-Reinas (n-Queens Problem) es muy antiguo, propuesto por primera vez en el año 1848, consiste en encontrar una asignación a n reinas en un tablero de ajedrez de n x n de tal modo, que éstas no se ataquen.

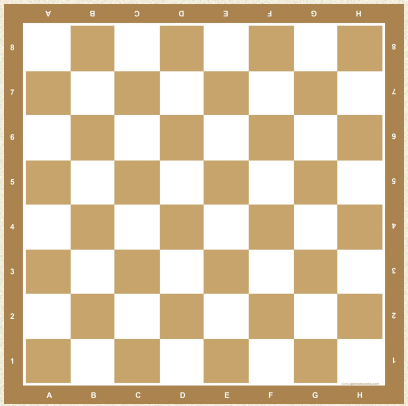
Definición del problema

Problema de las N reinas



Definición del problema

El problema de las n-reinas consiste en encontrar una distribución de n reinas en un tablero de ajedrez de nxn, de tal modo que éstas no se ataquen. Así no pueden encontrarse dos reinas en la misma fila, columna o diagonal.



MOVIMIENTOS GENERALES DE LA REINA DENTRO DE UN TABLERO DE AJEDREZ.

En la Fig.01 se muestran los movimientos posibles de una reina en un tablero de 4x4 (movimientos horizontales, verticales y diagonales).

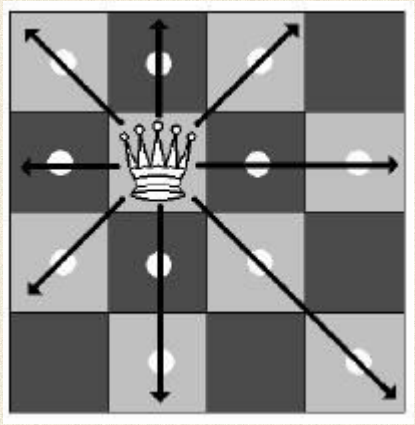


Fig.01

“POSICIÓN NO VÁLIDA DE UNA SEGUNDA REINA DENTRO DEL TABLERO”

Mientras que la Fig.02 muestra un ejemplo de dos reinas que pueden comerse.

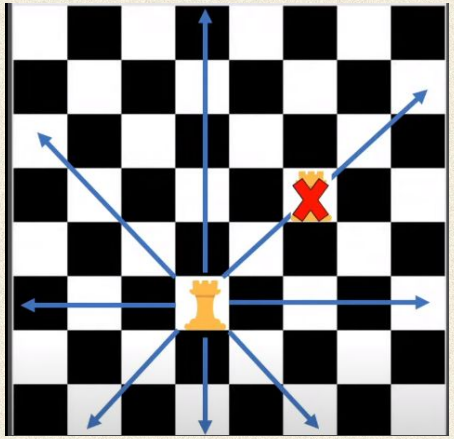


Fig.02

“POSICIÓN VÁLIDA DE UNA SEGUNDA REINA DENTRO DEL TABLERO”

En Fig.03, es un claro en donde se llega a cumplir la condición, la cual no llega a interponerse en ningún movimiento de la primera reina.

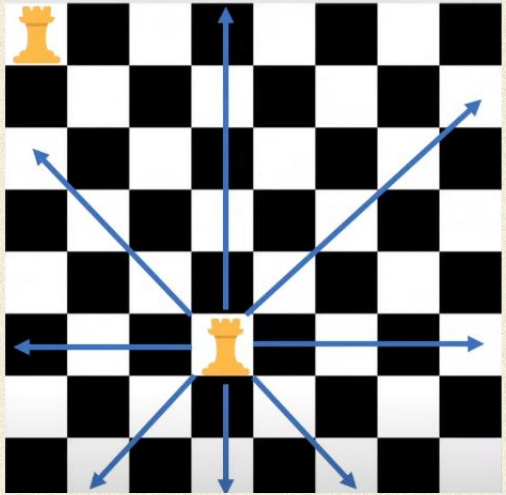


Fig.03

“INSERCIÓN DE MÁS REINAS DENTRO DEL TABLERO”

Podemos llegar a una primera problemática, si se coloca una reina más dentro del tablero, es mucho más difícil mientras más reinas tengamos dentro de este mismo tablero.

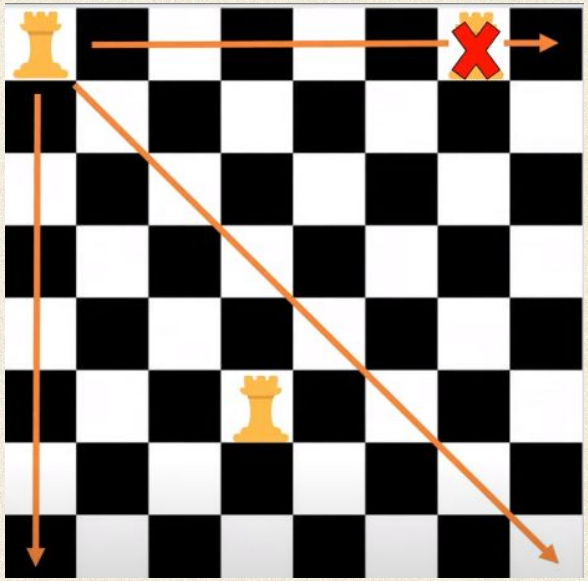


Fig.04

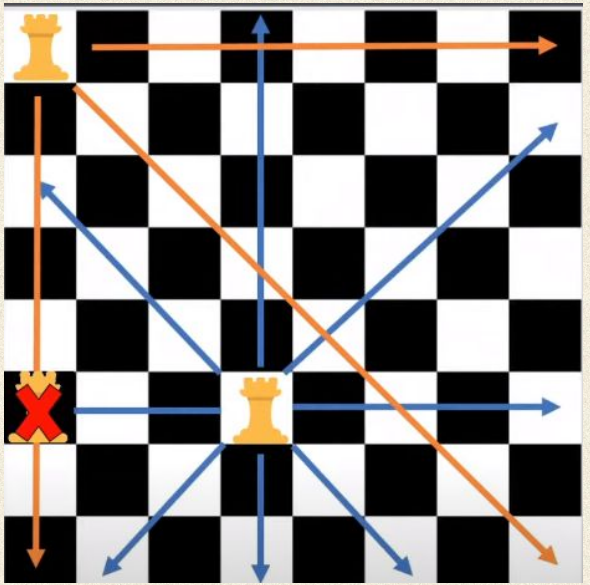


Fig.05

Resolución del Problema 4x4

Vamos a trabajar con un tablero de 4x4 para encontrar una solución más rápida. A continuación se muestra la representación de los datos que contendrá dicho tablero.

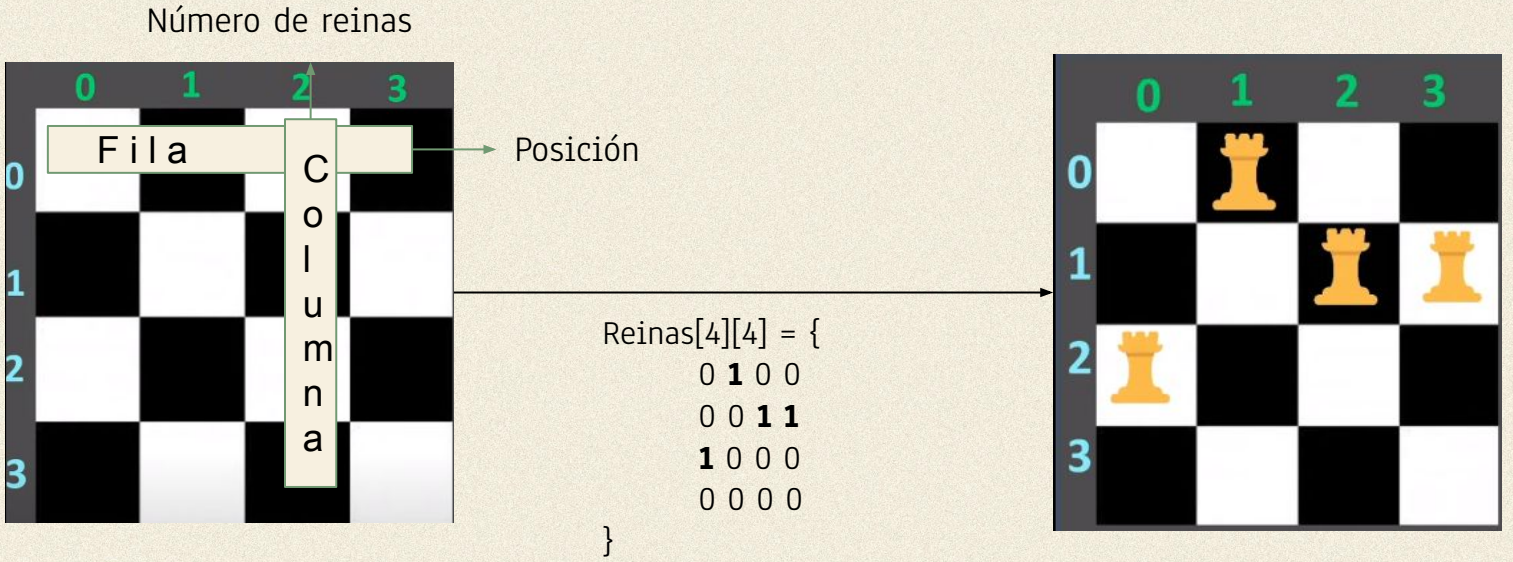


Fig.06

Fig.07

“Problema de representar los datos en un arreglo bidimensional”

Los puntos importantes de por qué no representar esta situación en un arreglo bidimensional:

- Lo único que nos llega a interesar son las ubicaciones de las reinas, que estas son representadas por el número “1”.
- Podemos representar a dos o más reinas dentro de una misma columna.

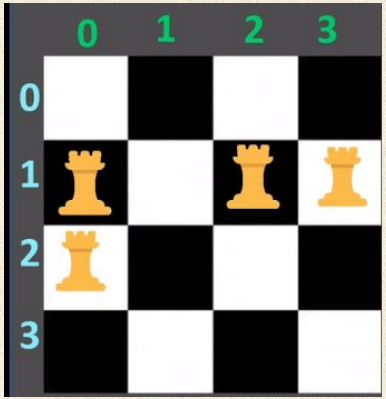
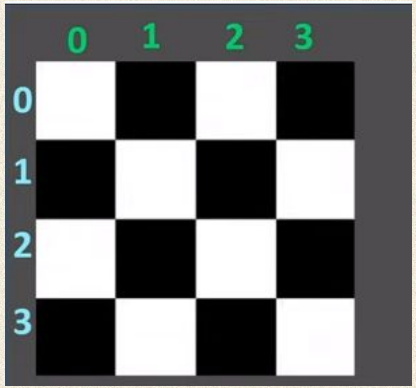


Fig.08

Resolución Más Eficiente Para Representar Los Datos

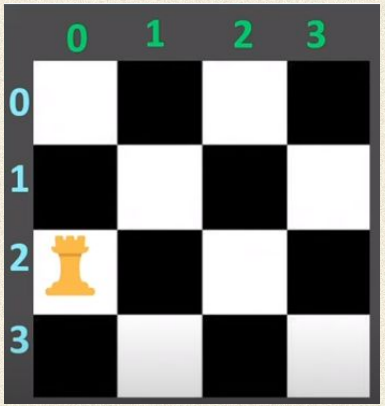
Entonces, ya se entendió que, trabajar con un arreglo bidimensional se vuelve un poco más complicado. Sin embargo, podemos tomar un camino más sencillo y también eficiente, y es pasar de este arreglo bidimensional a un arreglo unidimensional, ya que este tipo de representación nos permite tener las posiciones en dónde se encontrarán las reinas.



reinas[4]

Fig.09

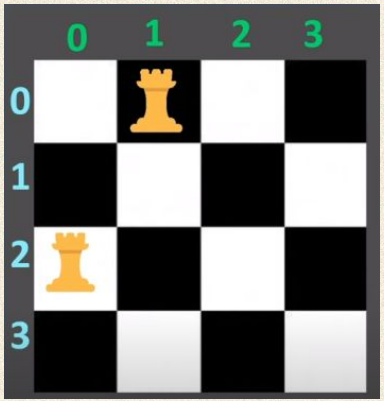
Fig.10



Reina[0] = 2

Reina[4] = {2}

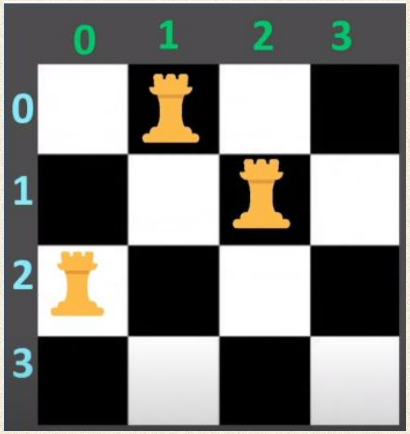
Fig.11



Reina[0] = 2
Reina[1] = 0

Reina[4] = {2,0}

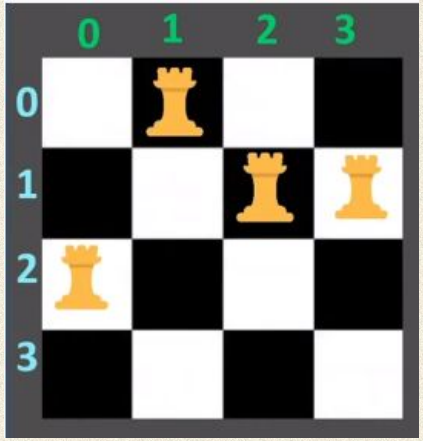
Fig.12



Reina[0] = 2
Reina[1] = 0
Reina[2] = 1

Reina[4] = {2,0,1}

Fig.13



Reina[0] = 2
Reina[1] = 0
Reina[2] = 1
Reina[3] = 1

Reina[4] = {2,0,1,1}

- Es una manera eficiente, ya que reducimos el número de espacios dentro del arreglo.
- Esta manera de representar los datos no nos permite representar dos reinas dentro de una misma columna.

Colisión en la misma fila

Si *“reinas[i] = reinas[k]”* podemos decir que hay dos reinas en la misma fila y habrá colisiones.

reinas[0] = reinas [3]
1 = 1

Generalización:
reinas[i] = reinas [k]

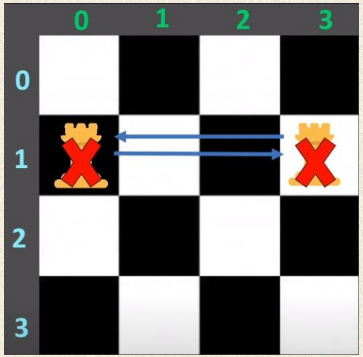


Fig.14

En este caso tenemos una restricción de que estos no pueden ser iguales.

reinas[1] = reinas [2]
3 = 3

Generalización:
reinas[i] = reinas [k]

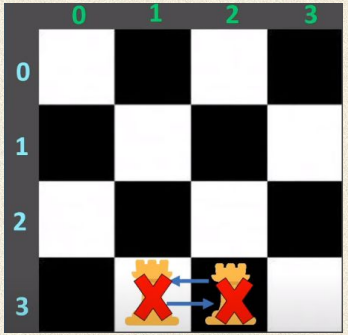


Fig.15

Colisiones en la misma diagonal

Siempre que las reinas se encuentren en diagonales, se va a cumplir la siguiente generalización.

reinas[i] = x
reinas[k] = y
|i-k| = |x-y|

reinas[0] = 1
reinas[2] = 3

0-2	1-3
|-2|	|-2|
2 =	2

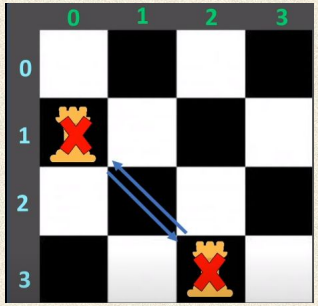


Fig.16

reinas[3] = 0
reinas[0] = 3

3-0	0-3
|3|	|-3|
3 =	3

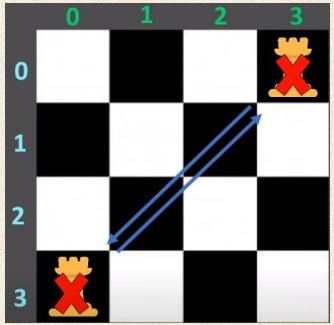


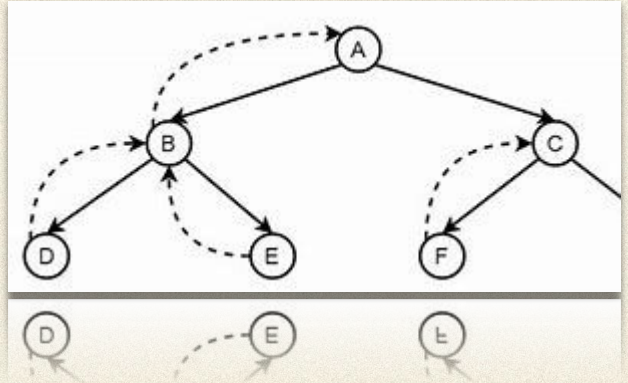
Fig.17

“Backtracking”

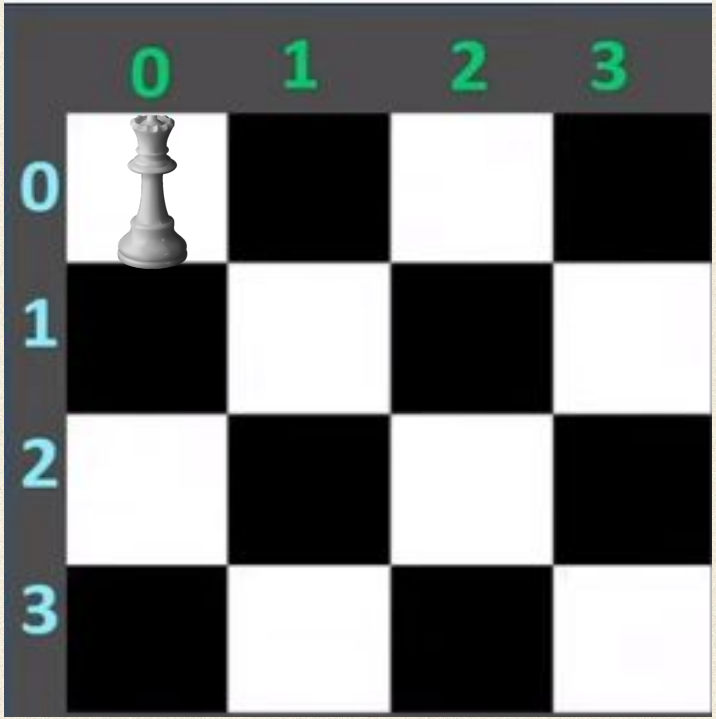
La técnica de Backtracking está muy asociada al tema de la recursividad, o más propiamente a la estructura recursiva de la mayoría de tipos de datos: listas, árboles.

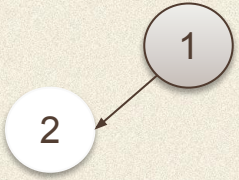
Esta técnica consiste en:

- Enumerar sistemáticamente las alternativas que existen en cada momento para dar con la solución a un problema.
- Se prueba una alternativa, guardando memoria del resto de alternativas.
- Si no damos con la solución, podemos dar marcha atrás (backtracking) y probar otra alternativa.

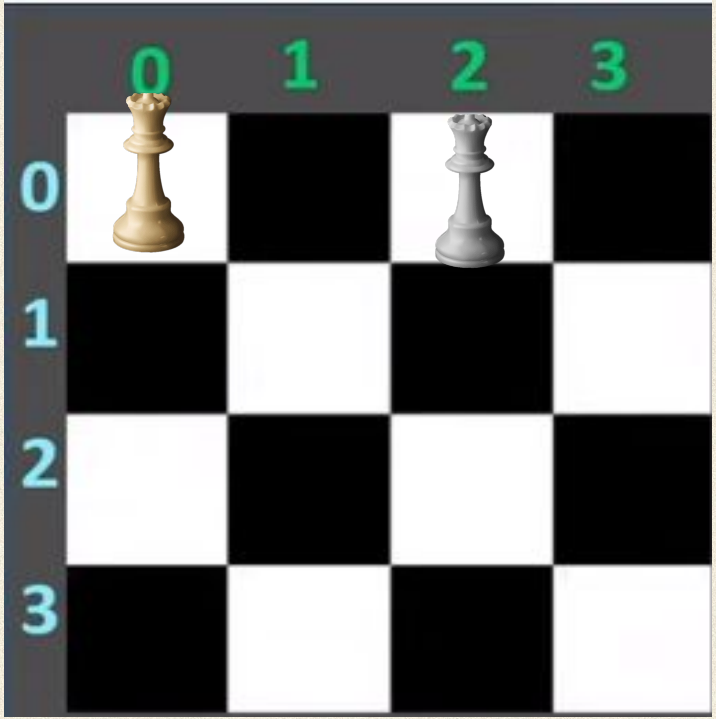


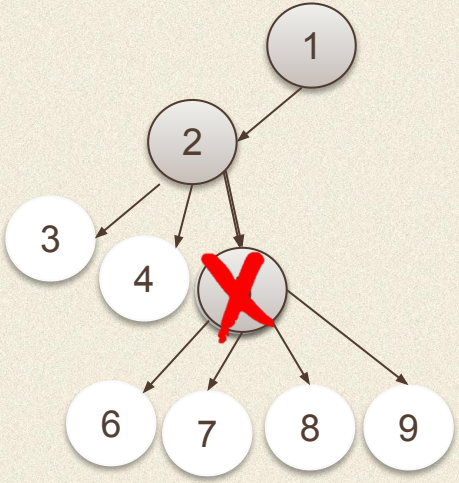
Resolución con Backtraking (Simulación)

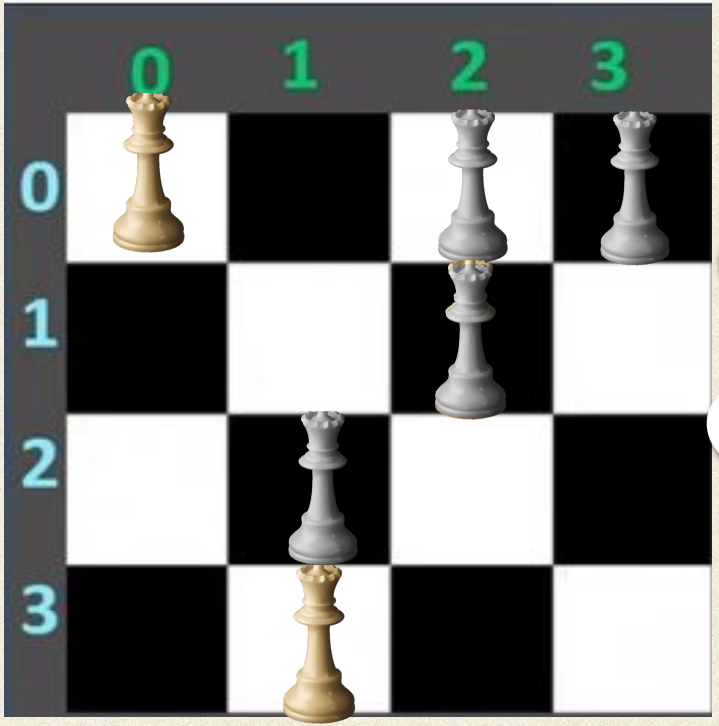


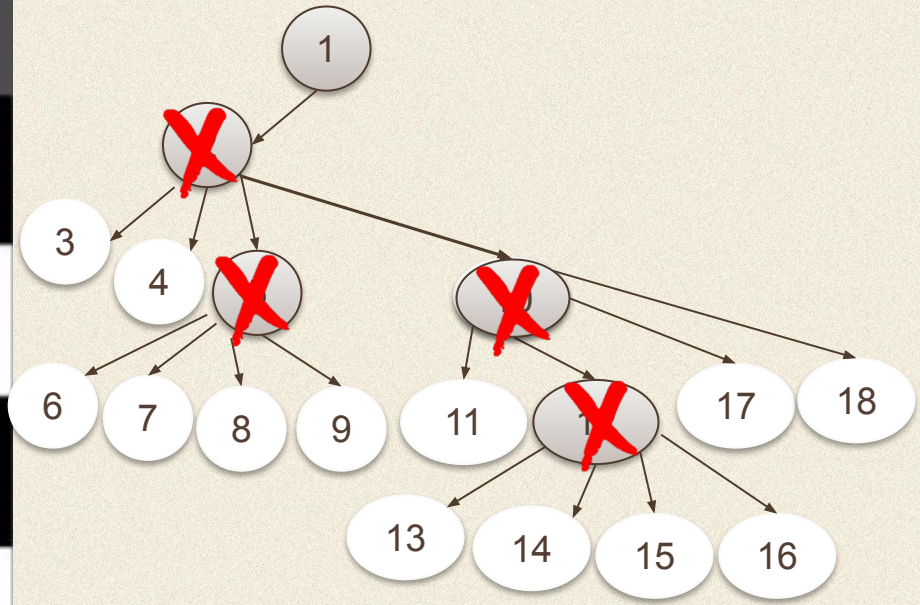


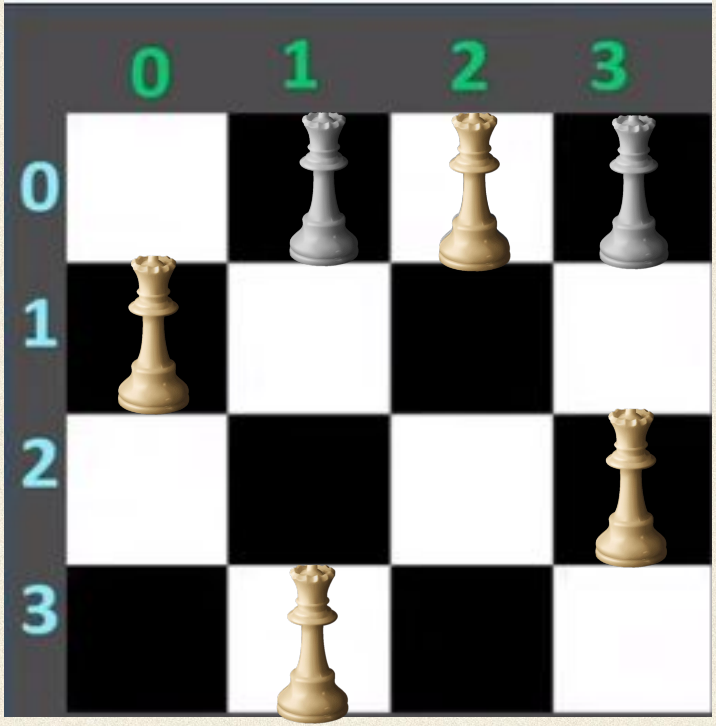
- Reina Gris : Representa su posible posición.
- Reina Amarilla: Representa la posición en donde no ocasiona ninguna colisión la reina.
- Nodo Blanco: Posición no apta para una reina.
- Nodo Color: Posición apta para una reina. (No es atacada).

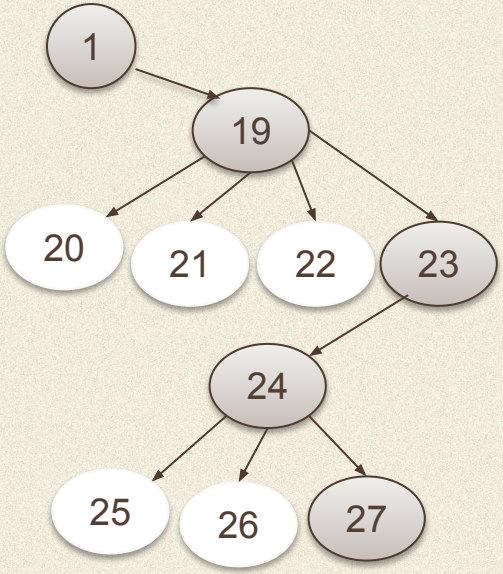

0
1
2
3
0
1
2
3


1
2
3
4
6
7
8
9

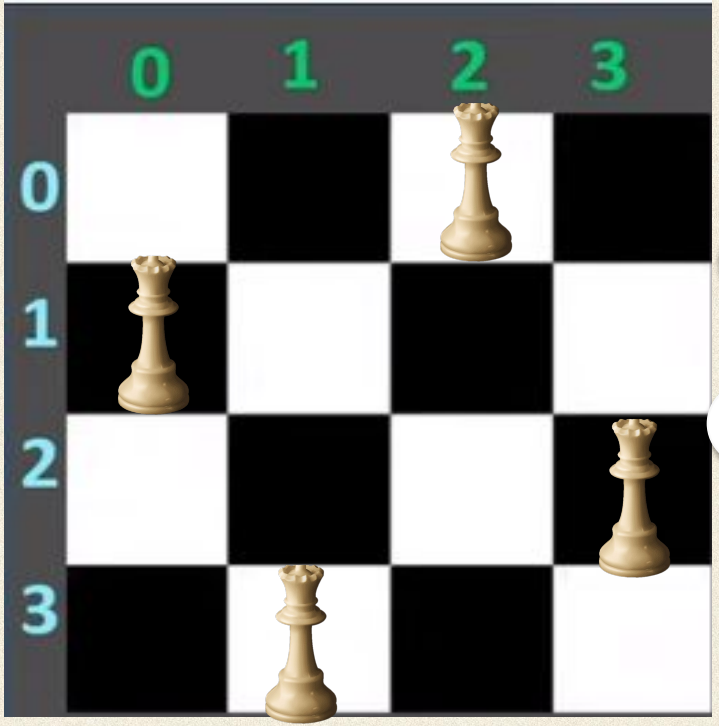

0
1
2
3
0
1
2
3

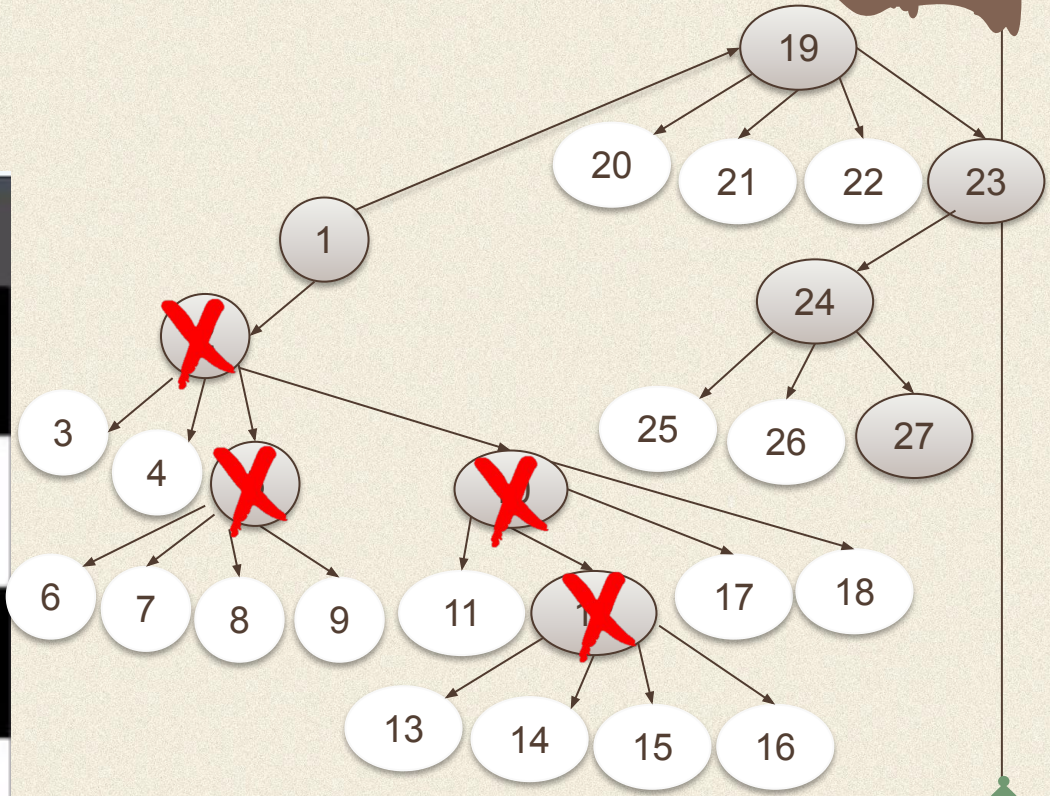

1
3
4
6
7
8
9
11
17
18
13
14
15
16


0
1
2
3
0
1
2
3


1
19
20
21
22
23
24
25
26
27

Resultado





Código de las N-Reinas

Función Principal

```
int main(int argc, char *argv[]) {

    int k=0;
    int cant;
    cout<<"Ingresar la cantidad de reinas : ";
    cin>>cant;

    int *reinas = new int[cant];

    for(int i=0;i<cant;i++){
        reinas[i]=-1;
    }

    Nreinas(reinas,cant,k);

    return 0;
}
```

Nivel del árbol

Cantidad de reinas

Array de N cantidad de reinas

Llenamos todo el array con -1, donde nos indica que no hemos colocado ninguna reina.

Función recursiva N reinas, que calculara todas las soluciones posibles(Posiciones,N-Reinas, Nivel de árbol)

Punto en el que no hay reinas en el tablero

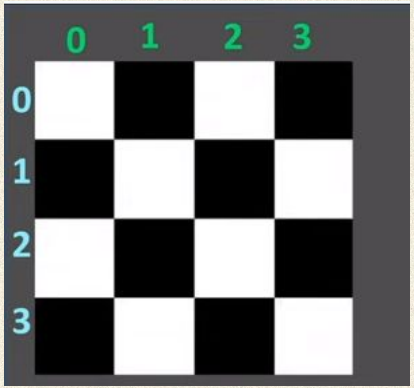
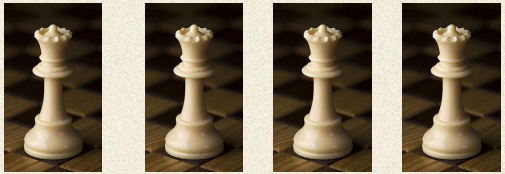


Fig.17

reinas[4]={-1,-1.-1,-1}



Función Recursiva

```
void Nreinas(int reinas[],int n, int k){
    if(k==n){
        x++;
        cout<<"Solucion "<<x<<" : ";
        for(int i=0;i<n;i++){
            cout<<reinas[i]<<" , ";
        }
        cout<<endl;
    }
    else{
        for(reinas[k]=0;reinas[k]<n;reinas[k]++){
            if(comprobar(reinas,n,k)){
                Nreinas(reinas,n,k+1);
            }
        }
    }
}
```

Nivel de árbol y N-Reinas son iguales.

Conocer el número de soluciones

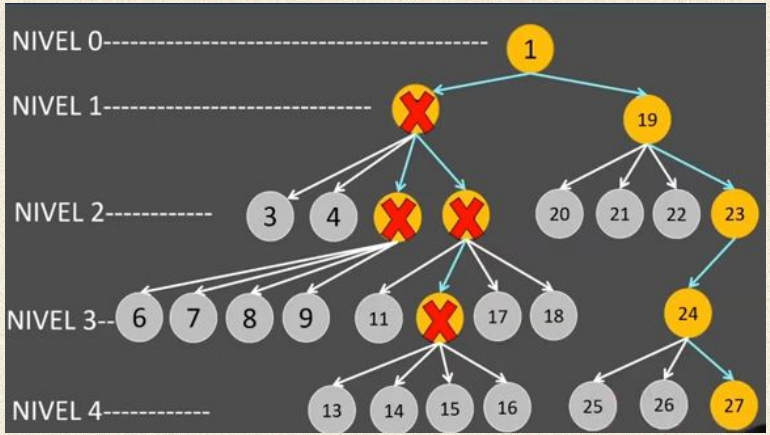
Imprimir todas las posiciones del array(numero de soluciones)

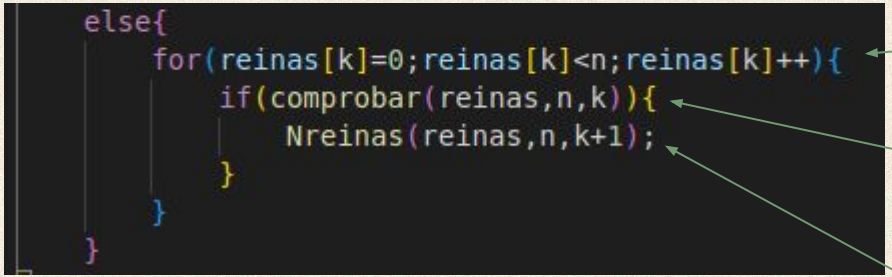
Aún quedan reinas por colocar y el algoritmo debe seguir buscando.

Entonces, tenemos un tablero de 4x4, para encontrar la primera solución tenemos el siguiente árbol con 4 niveles.

Donde k son los niveles del árbol, y cada nodo son las reinas colocadas dentro del arreglo (amarillo) de una posible solución.

Nodos grises son donde las reinas no pueden estar colocadas.





Donde k es el nivel de árbol reinas[k]

Comprobar si donde queremos colocar una nueva reina es una posición válida.

Llamada recursiva si no hay colisiones.

Expandirá el árbol de búsqueda en anchura, en este caso se expande por los costados de manera horizontal.

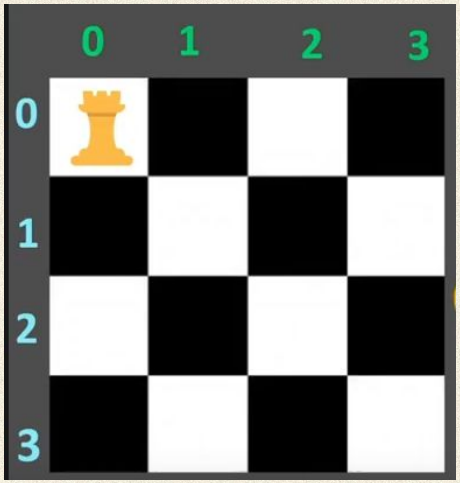


Fig.18

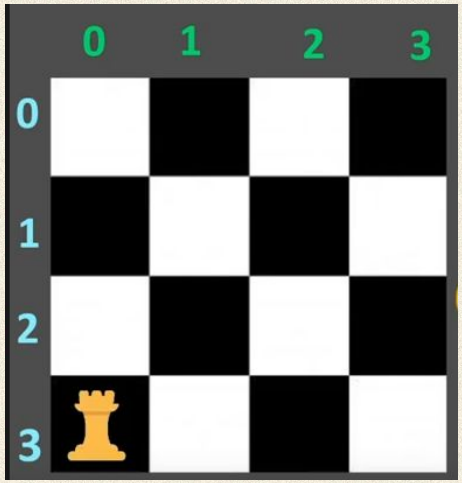
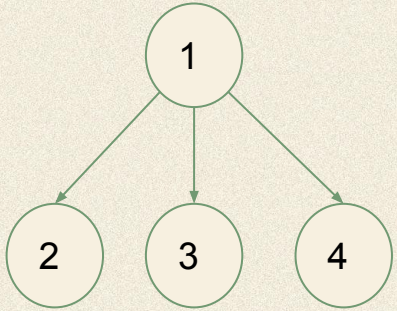


Fig.19



reinas[0] = 1
reinas[0] = 2
reinas[0] = 3
reinas[0] = 4

Donde k es el nivel de árbol reinas[k] = 3

Otra forma de moverse es en profundidad

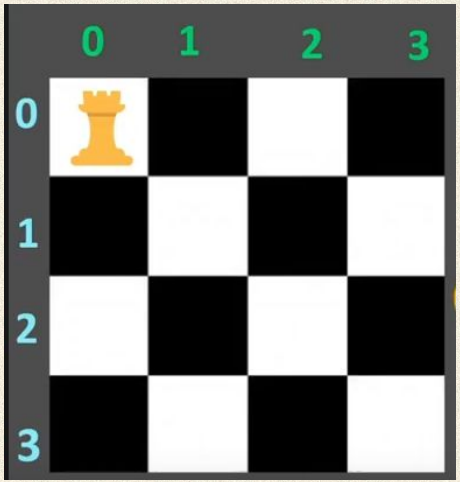


Fig.20

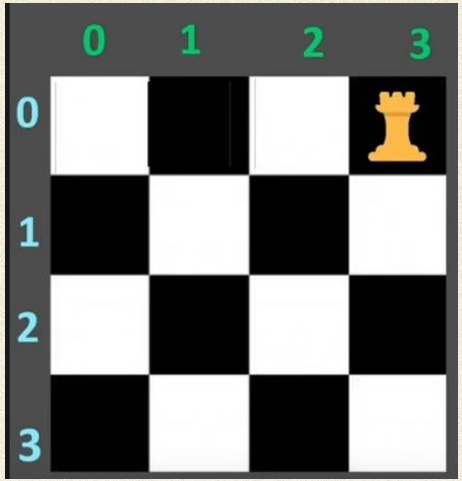
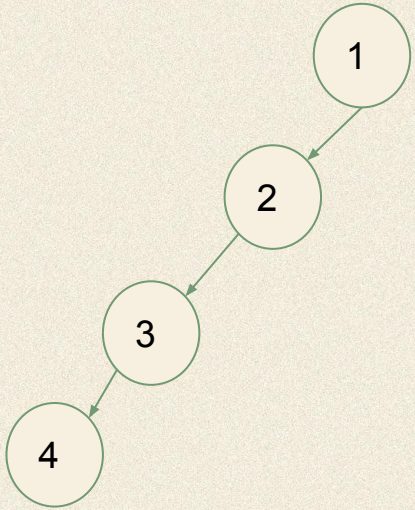


Fig.21



reinas[0] = 0
reinas[1] = 0
reinas[2] = 0
reinas[3] = 0

Función Comprobar

```
bool comprobar(int reinas[],int n, int k){
    int i;
    for(i=0;i<k;i++){
        if( (reinas[i]==reinas[k]) or (abs(k-i) == abs(reinas[k]-reinas[i]))){
            return false;
        }
    }
    return true;
}
```

No haya colisiones.

Para recorrer las k reinas que tenemos dentro del array

Número de soluciones

n	$Q(n)$
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2.680
12	14.200
13	73.712
14	365.596
15	2.279.184
16	14.772.512
17	95.815.104
18	666.090.624
19	4.968.057.848
20	39.029.188.884

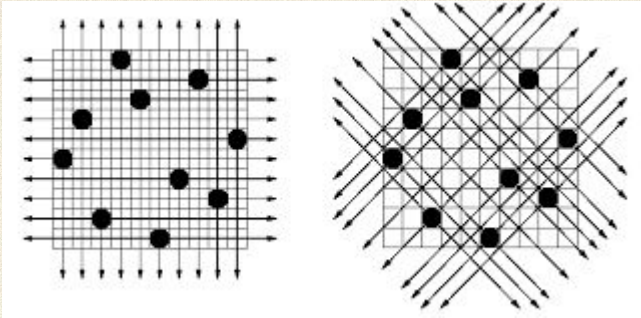
Q(n) para 4 <= n <=20

Aplicaciones.

El problema de las n-reinas es conocido usualmente como uno relacionado a un juego y también como un problema apropiado para probar algoritmos nuevos. Sin embargo, tiene otras aplicaciones, ya que se le considera como un modelo de máxima cobertura.

Algunas aplicaciones son;

- Control de Tráfico Aéreo.
- Sistema de Comunicaciones
- Programación de tareas Computacionales
- Compresión de Datos
- etc.



Otras posibles soluciones.

El curioso número S.

Si enumeramos las casillas de un tablero de ajedrez de cualquier tamaño (n x n) con los números naturales, empezando de izquierda a derecha y de arriba hacia abajo, encontramos el curioso caso del número Š, el cual siempre es una constante para un tablero dado de n x n lados y nos dice cuánto será la sumatoria de las casillas en donde se pueden ubicar las n reinas para todas las soluciones de dicho tablero. El número Š viene dado por la fórmula:

S = (n^3+n)/2

A continuación se da el ejemplo de las soluciones de un tablero de 4x4 (marcadas en negritas).

1	**2**	3	4
5	6	7	**8**
9	10	11	12
13	14	**15**	16

Numeración

1	**2**	3	4
5	6	7	**8**
9	10	11	12
13	14	**15**	16

Solución 1

1	2	**3**	4
5	6	7	8
9	10	11	**12**
13	**14**	15	16

Solución 2

S(4)=(4^3+4)/2
S(4)=34
Calculando para la primera solución
S1 = 2+8+9+15= 34
Calculando para la segunda solución
S2 = 3+5+12+14= 34

Conclusiones.

- Teniendo en cuenta lo que se ha trabajado, se ha presentado un algoritmo, donde la búsqueda sea de manera rápida y eficiente, donde sea capaz de encontrar una solución para los primeros valores de las “n reinas”. Con tan solo aumentar un poco el número de reinas, el número de soluciones aumenta considerablemente, esto pasa por que este tipo de problemas tienen complejidades de tipo O(n).

Referencias

- Spading, A., & Ananias, P. (n.d.). *“El Problema de las N-Reinas.”*

  https://www.cs.buap.mx/~zacarias/FZF/nreinas3.pdf

- *Enunciado 4 Problema de las N Reinas INDICE*. (n.d.). https://www.etsisi.upm.es/sites/default/files/asigs/arquitecturas_avanzadas/practicas/MPI/nreinas.pdf
- *Backtracking Diseño y Análisis de Algoritmos*. (n.d.). https://www.cartagena99.com/recursos/alumnos/temarios/Backtracking.pdf
- *Download Limit Exceeded*. (n.d.). Citeseerx.ist.psu.edu. Retrieved September 18, 2022, from

  https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.6147&rep=rep1&type=pdf

- Sosic, R., & Gu, J. (1990). A polynomial time algorithm for the N-Queens problem. *ACM SIGART*

  Bulletin, *1*(3), 7–11. https://doi.org/10.1145/101340.101343