



---

## Problema de las n reinas Practica 04

Analisis y diseño de algoritmos

17/01/2024

---

**Grupo:**

3CM2

**Equipo:**

Daniel Juárez Ávila  
Jared De Loera Espinosa  
Adan Torres Gutierrez  
Marco Antonio de la Cruz Diaz

## 1. Introducción

Este proyecto se centra en la implementación y optimización del algoritmo de backtracking en Python para resolver el problema de N Reinas. Se explorarán dos estrategias específicas de optimización y se realizará un análisis de su rendimiento comparativo, incluyendo gráficas que ilustren el tiempo de ejecución de diferentes versiones del código.

### 1.1. Terminología

#### 1.1.1. Backtracking

El algoritmo de backtracking (vuelta atrás) es una técnica que utiliza la recursividad para encontrar soluciones a problemas complejos dividiendo el problema y explorando todos los caminos posibles para resolver el problema computacional en cuestión. Se basa en construir soluciones de manera incremental y retrocede cuando un camino que había tomado no tiene salida para resolver el problema.

El backtracking se aplica a problemas combinatorios como el problema de las N reinas o el problema de la mochila en los que se busca una combinación específica de elementos o con ciertas reglas. En resumen, la estrategia del algoritmo se basa en seleccionar una opción del conjunto disponible en cada etapa del proceso de resolución. En caso de que esta elección no conduzca a una solución, la búsqueda retrocede al momento de la elección y prueba con otra alternativa. Una vez que se han explorado todas las posibles opciones, la búsqueda regresa a la fase anterior donde se tomó otra elección entre valores. Si no existen más puntos de elección, la búsqueda concluye. Este término fue utilizado por primera vez por el matemático D.H. Lehmer en la década de 1950.

#### 1.1.2. Heurística

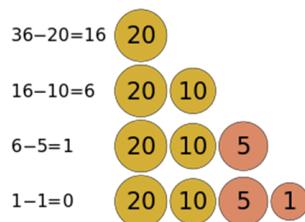
Para la informática, la heurística consiste en un método de resolución aproximado que busca obtener buenas soluciones utilizando una menor cantidad de recursos (tiempo o memoria). Los heurísticos que no garantizan una solución óptima, pero ofrecen una aproximación para modelos de una gran dimensión en contraparte de los Exactos.

#### *Criterios de uso*

- Se necesita una solución factible en poco tiempo.
- La instancia es tan grande que, aunque se pueda formular mediante IP, la resolución mediante métodos exactos no es viable.
- La instancia es tan compleja que la formulación mediante IP no es posible.
- Para algunos problemas combinatorios complejos, son más eficientes que los métodos generales exactos.

#### *Usos*

Ejemplos de heurística son los algoritmos voraces, es una estrategia de búsqueda por la cual se sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Como el problema del cambio de monedas.



### 1.1.3. Ramificación-Poda

El método de diseño de algoritmos ramificación y poda (Branch and bound) es una variante del backtracking mejorado sustancialmente. Se suele interpretar como un árbol de soluciones, donde cada rama conduce a una posible solución posterior a la actual.

La característica de esta técnica con respecto a otras anteriores (y a la que debe su nombre) es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para podar esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima.

#### *Características]*

1. La ramificación y poda realiza un recorrido sistemático en un árbol de soluciones.
2. El recorrido no tiene por qué ser necesariamente en profundidad. Tendremos una estrategia de ramificación.
3. Se tratará como un aspecto importante las técnicas de poda, para eliminar nodos que no lleven a soluciones optimas.
4. La poda se realiza estimando en cada nodo cotas del beneficio que podemos obtener a partir del mismo.

#### *Funcionamiento de la Poda*

Si la menor rama para algún árbol nodo (conjunto de candidatos) A es mayor que la rama padre para otro nodo B, entonces A debe ser descartada con seguridad de la búsqueda. Este paso es llamado poda, y usualmente es implementado manteniendo una variable global m que graba el mínimo nodo padre visto entre todas las subregiones examinadas hasta entonces. Cualquier nodo cuyo nodo hijo es mayor que m puede ser descartado. La recursión para cuando el conjunto candidato S es reducido a un solo elemento, o también cuando el nodo padre para el conjunto S coincide con el nodo hijo. De cualquier forma, cualquier elemento de S va a ser el mínimo de una función dentro de S.

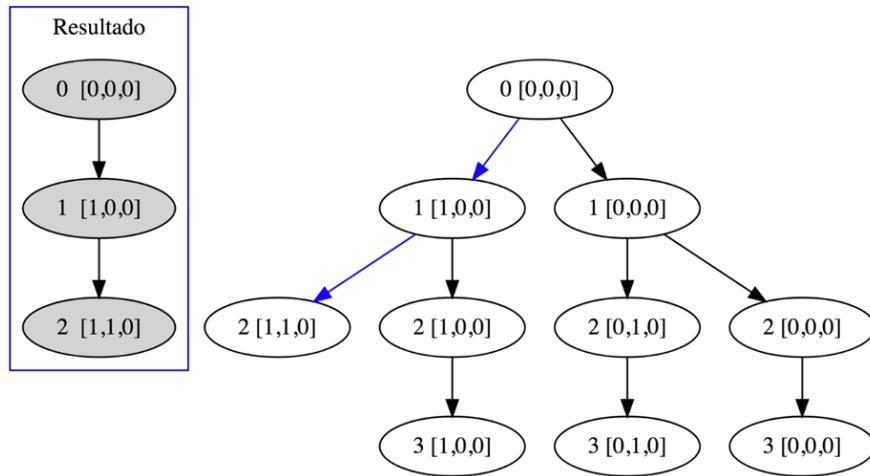
#### *Psdeudocodigo*

```
Funcion RyP {
    P = Hijos(x,k)
    while ( no vacio(P) )
        x(k) = extraer(P)
        if esFactible(x,k) y G(x,k) < optimo
            si esSolucion(x)
                Almacenar(x)
            else
                RyP(x,k+1)
}
```

- **G(x)** es la función de estimación del algoritmo.
- **P** es la pila de posibles soluciones.
- **esFactible** es la función que considera si la propuesta es válida.
- **esSolución** es la función que comprueba si se satisface el objetivo.
- **óptimo** es el valor de la función a optimizar evaluado sobre la mejor solución encontrada hasta el momento.
- **NOTA:** Usamos un menor que ( $\downarrow$ ) para los problemas de minimización y un mayor que ( $\uparrow$ ) para problemas de maximización.

**Usos**

Este método es aplicable en problemas de optimización discreta y en problemas de juegos donde podamos establecer un árbol, por ejemplo, el problema de la mochila o de la n reinas.



## 2. Desarrollo

### 2.1. Backtracking

En este algoritmo para las n-reinas se uso la idea del backtraking en la cual se regresará en pasos a fin de explorar mas posibles soluciones. Primero definimos el tamaño del tablero con el que vamos a estar trabajando, según el tamaño del tablero va a ser el numero de reinas que se tienen que acomodar en este mismo. El siguiente paso es crear una lista la cual va a estar vacía, en esta lista se van a estar guardando las posibles soluciones para ver si son validas o no. Esta lista se llena de modo que los índices nos van a representar la columna que se representa y el numero dentro de la lista nos representará la fila en la que está acomodada la reina.

Vamos a añadir un cero a la lista para representar que vamos a empezar en la posición [0,0], después de hacer esto se entra en un bucle hasta que se encuentren todas las posibles soluciones que va a ser cuando el switch sea igual a FALSE, para esto se creó una función que verifica si la posición actual es valida la cual se llama “valido”, en esta función se toma el índice actual y lo compara con las posiciones anteriores, si el valor guardado en la lista es igual a alguno anterior significa que están en una misma fila por lo cual no es una posición valida, también se verifica si la distancia horizontal es igual a la distancia vertical entre dos reinas eso significaría que se encuentran en la misma diagonal y tampoco sería una posición valida. Ahora que sabemos si la posición es valida o no, vamos a evaluar si podemos seguir aumentando la columna en la que estamos, como la lista esta ordenada desde 0 hasta n-1 sabremos si estamos al tope de la búsqueda si la columna actual al sumarle 1 es igual a n, por lo cual evaluamos si n+1 es menor a n, si se cumple esta condición y que la posición sea valida entonces podemos seguir añadiendo mas posiciones por lo cual se le añade otro 0 a la lista y se aumenta la posición de la columna actual, para así seguir recorriendo el árbol de posibilidades.

Si ninguna de las dos condiciones se cumplen, entonces vamos a evaluar si la posición es válida y columna actual + 1 es igual a n, si esto se cumple significa que no podemos seguir incrementando mas pero las posiciones son todas validas, por lo cual esta es una solución, así que incrementamos el contador de soluciones en 1, el siguiente paso es explorar las demás soluciones, si la fila + 1 es menor a n entonces incrementamos en 1 para seguir explorando el árbol, si la fila + 1 es igual a n significa que ya exploramos todas las posiciones para esa columna por lo cual tenemos que volver atrás para verificar las demás soluciones posibles. Vamos a entrar a un bucle en el que se seguirá repitiendo siempre que la fila de la columna actual + 1 sea igual a n y que el switch sea igual a TRUE, la primera condición nos representa que la columna actual esta totalmente explorada por lo cual tenemos que volver a la anterior, dentro del bucle hay una condición que verifica si la columna actual es igual a 0 si esto se cumple significa que se recorrió todo el árbol por lo cual se ha

terminado la búsqueda por lo que el switch pasa a ser FALSE saliendo de los bucles y retornando el número de soluciones, si esta condición no se cumple significa que podemos ir más para atrás en las columnas por lo cual se elimina el valor actual de la lista y se reduce en 1 el numero de columnas para luego aumentar en 1 el numero de fila que vamos a verificar en el siguiente ciclo.

### Ejemplo n=2

Empezamos con una lista [0] que significa que estamos en la columna 0 y en la fila 0, verificamos y esta posición es valida entonces pasamos a la siguiente columna. Ahora tenemos la lista [0,0] que significa que tanto en la columna 0 como en la 1 tenemos una reina en la fila 0, esta posición no es válida ya que están en la misma fila, entonces probamos otro valor para la columna actual.

Tenemos la lista [0,1] entonces revisamos que la posición sea valida y encontramos que no lo es ya que la distancia horizontal (que sería el valor absoluto de la resta de los índices) y la distancia vertical (que sería el valor absoluto de la resta de los valores) son iguales por lo cual están en la misma diagonal, como  $1+1=n$  entonces y ya no podemos aumentarlo en 1 por lo cual vamos a volver un paso (eliminamos la columna actual) y sumarle 1 a la nueva columna actual. Ahora tenemos la lista [1] esta es una posición valida por lo cual le agregamos un 0 a la lista. Tenemos [1,0] la cual no es una posición valida ya que están en la misma diagonal, por lo cual podemos aumentar en 1 la columna actual. Tenemos la lista [1,1] verificamos y nos damos cuenta de que están en la misma fila por lo cual no es valido entonces intentamos regresar una posición para darnos cuenta de que la columna 0 ya no puede ser incrementada ya que su valor actual + 1 es igual a n por lo cual ya hemos terminado. Como el contador no se aumentó en ningún momento no existen soluciones para n=2.

### Ejemplo n=3

Empezamos con la lista [0] la cual es valida por lo cual agregamos 0, ahora tenemos [0,0] la cual no es una posición valida ya que están en la misma fila, como podemos lo incrementamos en 1, ahora con la lista [0,1] la cual no es una posición valida ya que están en la misma diagonal, por lo cual aumentamos en 1. Tenemos la lista [0,2] esta posición es valida por lo cual agregamos un 0.

En la nueva lista [0,2,0] esta posición no es valida ya que comparte fila con la columna 0, entonces como podemos lo aumentamos en 1, en [0,2,1] esta posición no es valida ya que comparte diagonal con la reina de la columna 2 entonces aumentamos en 1, en la lista [0,2,2] no es válida porque está en la misma fila que la columna 1 por lo cual regresamos hasta que se pueda incrementar en 1 la fila sin llegar a n que sería la columna 0 y se le suma 1 a la fila, en la nueva lista[1] es una posición valida por lo cual agregamos un 0.

En la nueva lista [1,0] la posición no es valida ya que comparte diagonal con la columna 0, entonces le agregamos 1 a la columna actual, en la lista [1,1] la posición no es valida ya que comparte fila con la columna 0 entonces agregamos 1 a la columna actual, en la lista [1,2] la posición no es valida ya que comparte diagonal con la columna 0, entonces como ya no se puede aumentar en 1 regresamos a la columna anterior y la aumentamos en 1. En la lista [2] la posición es valida entonces agregamos un 0, en la lista [2,0] la posición es valida entonces agregamos un 0, en la lista [2,0,0] la posición no es valida porque comparte fila con la columna 1, entonces aumentamos en 1 la fila de la columna actual, en la lista [2,0,1] la posición no es valida ya que comparte vértice con la columna 1, entonces aumenta en 1 la fila de columna actual, en la lista [2,0,2] la posición no es valida ya que comparte fila con la columna 0, como ya no puede aumentar, eliminamos el valor actual y decrementamos el número de columnas.

En la lista [2,1] la posición no es valida ya que comparte diagonal con la columna 1 entonces aumentamos en 1 la fila de la columna actual, en la lista [2,2] la posición no es valida ya que tiene la misma fila que la columna cero entonces regresamos, pero como regresamos a la columna 0 y su fila ya no puede incrementar significa que terminamos con un numero de 0 soluciones.

### Ejemplo n=4

Empezamos con la lista [0] la cual es válida por lo cual agregamos 0, ahora tenemos [0,0] la cual no es una posición valida ya que están en la misma fila, como podemos lo incrementamos en 1, ahora con la lista [0,1]

la cual no es una posición valida ya que están en la misma diagonal, por lo cual aumentamos en 1. Tenemos la lista [0,2] esta posición es válida por lo cual agregamos un 0.

En la nueva lista [0,2,0] esta posición no es válida ya que comparte fila con la columna 0, entonces como podemos lo aumentamos en 1, en [0,2,1] esta posición no es válida ya que comparte diagonal con la reina de la columna 2 entonces aumentamos en 1, en la lista [0,2,2] no es válida porque está en la misma fila que la columna 1 entonces incrementamos en 1 el valor de la fila actual, en la lista [0,2,3] la posición no es valida porque coincide con la diagonal de la columna 1 entonces como no podemos aumentar eliminamos el elemento de la lista y retrocedemos a la anterior columna y aumentamos su fila en 1.

En la lista [0,3] la posición es valida entonces agregamos 0, en la lista [0,3,0] la posición no es valida porque coincide con la fila de la columna 0 entonces aumentamos en 1 la fila de la columna actual, en la lista [0,3,1] la posición es valida entonces agregamos 0, en [0,3,1,0] la posición no es valida pues coincide con la fila de la columna 0, entonces aumentamos en 1 la fila de la columna actual, en la lista [0,3,1,1] la posición no es valida pues coincide con la fila de la columna 2, entonces seguimos aumentando, en la lista [0,3,1,2] la posición no es valida pues coincide con la diagonal de la columna 2, entonces seguimos aumentando, en la lista [0,3,1,3] la posición no es valida pues coincide con la fila de la columna 2, ya no podemos aumentar entonces retrocedemos y aumentamos a la nueva columna, en la lista [0,3,2] la posición no es válida porque coincide con la diagonal de la columna 1, entonces aumentamos en 1 la fila de la columna actual, en la lista [0,3,3] la posición no es valida puesto que coincide con la fila de la columna 1, como no podemos aumentar retrocedemos hasta que se pueda y aumentamos en 1 a la nueva columna.

En la lista [1] la posición es valida entonces agregamos 0, en la lista [1,0] la posición no es valida porque coincide con la diagonal de la columna 0, entonces agregamos 1 a la fila de la columna actual, en la lista [1,1] la posición no es valida porque coincide con la fila de la columna 0, entonces seguimos aumentando, en la lista [1,2] la posición no es valida porque coincide con la diagonal de la columna 0, entonces seguimos aumentando, en la lista [1,3] la posición es valida entonces agregamos 0, en la lista [1,3,0] la posición es valida entonces agregamos 0, en la lista [1,3,0,0] la posición no es valida pues coincide con la fila de la columna 2, entonces agregamos 1 a la fila de la columna actual, en la lista [1,3,0,1] la posición no es valida porque coincide con la diagonal de la columna 2, entonces seguimos agregando, en la lista [1,3,0,2] la posición es valida y hemos llegado a un límite de columnas, entonces sumamos 1 a la cantidad de soluciones, luego seguimos buscando agregándole 1 a la columna actual, en la lista [1,3,0,3] la posición no es valida porque coincide con la fila de la columna 1, como no podemos agregar retrocedemos una columna y le agregamos 1 a la nueva columna, en la lista [1,3,1] la posición no es valida porque coincide con la fila de la columna 0, entonces agregamos 1 a la fila de la columna actual, en la lista [1,3,2] la posición no es valida porque coincide con la diagonal de la columna 1, entonces seguimos agregando, en la lista [1,3,3] la posición no es valida porque coincide con la fila de la columna 2, entonces regresamos hasta que se pueda aumentar y le aumentamos 1 a la nueva columna.

En la lista [2] la posición es válida, entonces agregamos 0, en la lista [2,0] la posición es valida entonces agregamos 0, en la lista [2,0,0] la posición no es válida porque coincide con la fila de la columna 1, entonces agregamos en 1 la fila de la columna actual, en la lista [2,0,1] la posición no es válida porque la posición coincide con la diagonal de la columna 1, entonces seguimos agregando, en la lista [2,0,2] la posición no es valida porque coincide con la fila de la columna 0, entonces seguimos agregando, en la lista [2,0,3] la posición es válida, entonces agregamos 0, en la lista [2,0,3,0] la posición no es valida porque coincide con la fila de la columna 1, entonces agregamos en 1 la fila de la columna actual, en la lista [2,0,3,1] la posición es valida y hemos llegado a un límite para las columnas entonces aumentamos en 1 el contador de soluciones y seguimos buscando aumentando la fila de la columna actual en 1, en la lista [2,0,3,2] la posición no es valida porque coincide con la fila de la columna 0, entonces seguimos agregando, en la lista [2,0,3,3] la posición no es válida porque coincide con la fila de la columna 2, como ya no podemos agregar retrocedemos hasta que se pueda y le agregamos 1 a la nueva columna actual, en la lista [2,1] la posición no es valida porque coincide con la diagonal de la columna 0, entonces agregamos en 1 la fila de la columna actual, en la lista [2,2] la posición no es válida porque coincide con la fila de la columna 0, entonces seguimos agregando, en la lista [2,3] la posición no es válida porque coincide con la diagonal de la columna 0, entonces retrocedemos hasta que se pueda agregar y a la nueva columna actual le agregamos 1. Si seguimos con este procedimiento para la rama [3] no encontraremos mas soluciones y al no encontrar nada y regresar al valor [3] el programa terminará.

## 2.2. Poda

Mediante el algoritmo anterior se le implemento una poda para reducir el numero de nodos que se tienen que explorar, para esto se creó una lista con los numeros del 1 al n-1 (llamada num) que serán nuestros valores sin usar. El inicio es igual añadiendo 0 a la lista principal e iniciando el bucle, al momento de verificar si la solución es valida solo se verifica la condición de si la distancia horizontal y vertical son iguales, más adelante veremos por qué.

Si la solución es valida y el numero de columna puede incrementar en 1 sin llegar a n entonces podemos pasar a revisar la siguiente columna, pero la diferencia de este algoritmo es que la siguiente columna no se llena con un 0 si no que toma el numero menor dentro de la lista de numeros por usar (num) ya que estos son los valores que no hemos usado aún, de esta forma estamos evitando poner valores que ya sabemos que son posiciones no validas debido a que estén en la misma fila. Si las dos condiciones no se cumplen, verificamos que la posición sea valida y si esto se cumple podemos inferir que como  $i + 1$  no es menor que n entonces ya estamos en la última posición por lo cual al ser valida la posición, encontramos una solución e incrementamos el contador de soluciones en 1.

Para este algoritmo es necesario una lista auxiliar por lo que veremos ahora. Entraremos a un bucle que se seguirá ejecutando siempre que el tamaño de la lista auxiliar sea diferente de 0 y que el switch sea TRUE, dentro de este bucle vamos a buscar todos los valores en la lista de posiciones sin usar y verificaremos si es mayor a la posición actual en la lista principal y cada que se encuentre uno se le añadirá a la lista auxiliar, esto es para ver si hay otra fila que podamos probar por utilizar que no hemos utilizado antes, si el tamaño de la lista auxiliar es diferente de 0 entonces el valor actual de la lista principal se va a pasar a la lista de valores sin usar, la lista principal va a añadir el valor mas pequeño de la lista auxiliar ya que este es el valor mas pequeño que no se ha usado y que es mayor al valor anterior, este mismo valor se va a eliminar de la tabla de posiciones sin usar.

Si la lista auxiliar es 0 entonces vamos a verificar si el numero de columna es igual a 0, si esto se cumple significa que ya no tenemos mas valores por explorar por lo cual el switch se vuelve FALSE y retorna el numero de soluciones acumulado hasta ahora, si esta solución no se cumple significa que podemos ir un paso atrás a verificar mas valores, por lo cual el valor actual de la lista principal se elimina y se pasa a la lista de valores sin usar y también el número de columna reduce en 1.

### Ejemplo n=2

Agregamos 0 y la lista nos queda [0] esta es una posición valida por lo cual tomaremos el valor mas pequeño sin usar, en la lista [0,1] la posición no es valida porque coincide con la diagonal de la columna 0, entonces como ya no podemos aumentar regresamos a una columna no llena y asignamos el próximo valor sin usar, en la lista [1] la posición es valida entonces agregamos el valor mínimos sin usar, en la lista [1,0] la posición no es valida entonces retrocedemos, pero como la columna 0 está vacía hemos terminado y no hemos tenido ninguna solución para n=2.

### Ejemplo n=3

Agregamos 0 y las lista nos queda [0] esta es una posición valida por lo cual tomaremos el valor mas pequeño sin usar, en la lista [0,1] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos la fila por el valor próximo más pequeño sin usar, en la lista [0,2] la posición es valida entonces agregamos el valor mínimo sin usar, en la lista [0,2,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta tener una columna que tenga la posibilidad de agregarle un valor sin usar y reemplazamos por el valor mínimo sin usar que sea mayor a la fila, en la lista [1] la posición es valida entonces agregamos el valor mínimo sin usar, [1,0] la posición no es valida porque coincide con la diagonal de la columna 0, entonces reemplazamos por el próximo valor sin usar, en la lista [1,2] la posición no es valida porque coincide con la diagonal de la columna 0, entonces como ya no podemos reemplazar, regresamos hasta que se pueda reemplazar una fila de una columna por un valor sin usar mayor a la fila, en

la lista [2] la posición es valida entonces agregamos el valor mas pequeño sin usar, en la lista [2,0] la posición es valida entonces agregamos el valor mas pequeño sin usar, en la lista [2,0,1] la posición no es valida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta que una fila tenga un valor próximo que se le pueda agregar, en la lista [2,1] la posición no es válida porque coincide con la diagonal de la columna 0, y al retroceder ya no tenemos ningún valor sin usar para reemplazar en la columna 0 , entonces terminamos con la cantidad de 0 soluciones para n=3.

#### Ejemplo n=4

Agregamos 0 y las lista nos queda [0] esta es una posición valida por lo cual tomaremos el valor más pequeño sin usar, en la lista [0,1] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos la fila por el valor próximo más pequeño sin usar, en la lista [0,2] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [0,2,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces reemplazamos por el valor próximo en la fila de la columna actual, en la lista [0,2,3] la posición no es válida porque coincide con la diagonal de la columna 1, entonces regresamos hasta encontrar una columna con una fila para reemplazar con un valor próximo, en la lista [0,3] la posición es válida, entonces agregamos el valor más pequeño de los valores sin usar, en la lista [0,3,1] la posición es válida, entonces agregamos el valor mínimo de los valores sin usar, en la lista [0,3,1,2] la posición no es válida porque coincide con la diagonal de la columna 2, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [0,3,2] la posición no es válida porque coincide con la diagonal de la columna 1 entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar.

En la lista [1] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [1,0] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos por el próximo valor sin usar, en la lista [1,2] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos con el valor próximo, en la lista [1,3] la posición es valida , entonces agregamos el valor mínimo de los valores sin usar, en la lista [1,3,0], la posición es válida, entonces agregamos el valor mínimo de los valores sin usar, en la lista [1,3,0,2] la posición es valida y hemos llegado a un límite entonces se agrega 1 al contador de soluciones y seguimos buscando soluciones, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [1,3,2] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar.

En la lista [2] la posición es valida entonces agregamos el valor mínimo sin usar, en la lista [2,0] la posición es valida entonces agregamos el valor mínimo sin usar, en la lista [2,0,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces reemplazamos por el valor próximo sin usar, en la lista [2,0,3] es una posición valida, entonces agregamos el valor mínimo sin usar, en la lista [2,0,3,1] es una posición valida y llegamos a un límite, entonces se agrega 1 al contador de soluciones y seguimos buscando, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [2,1] la posición no es válida porque coincide con la diagonal de la columna 0 entonces reemplazamos por el valor próximo sin usar, en la lista [2,3] la posición es válida, entonces tomamos el valor mínimo sin usar, en la lista [2,3,0] la posición es válida, entonces tomamos el valor mínimo sin usar, en la lista [2,3,0,1] la posición no es válida porque coincide con la diagonal de la columna 2, , entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [2,3,1] la posición es válida, entonces tomamos el valor mínimo sin usar, en la lista [2,3,1,0] la posición no es válida porque coincide con la diagonal de la columna 2, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar.

En la lista [3] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [3,0] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [3,0,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces reemplazamos por el valor próximo sin usar, en la lista [3,0,2] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [3,0,2,1] la posición no es válida porque coincide con la diagonal de la columna 2, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [3,1] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [3,1,2] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [3,2]

la posición no es válida porque coincide con la diagonal de la columna 0, entonces retrocedemos pero en la columna 0 llegamos a un límite entonces la búsqueda a terminado y retornamos la cantidad de soluciones que es igual a 2 en n=4.

### 2.3. Heurística

En esta función se hace uso de la heurística de simetría, la cual nos dice que muchas de las soluciones de n reinas son simetrías de otras, por lo cual para un tablero de dimensiones n, si n es par significa que el tablero se puede partir por la mitad perfectamente, por lo cual el encontrar las soluciones para  $n/2$  y multiplicarlas por 2 nos daría el total de soluciones. En el código la implementación es la misma que con poda con la diferencia que cuando llega a la columna 0 y la fila es  $n/2$  significa que ya ha verificado la mitad de las soluciones, por lo cual el resto de las soluciones son simetrías de las ya encontradas, por lo cual se retorna el número de soluciones multiplicado por 2 para obtener el total de soluciones. En este caso el propósito del programa es encontrar el número de soluciones, por eso se implementó así, pero si el objetivo fuese retornar cada solución individual también se podría, solo se tendría retornar las soluciones acumuladas junto con sus simetrías. Claro que este método solo funciona cuando n es par, entonces para que el algoritmo funcione y sea óptimo, cuando n sea impar se mandará a llamar a la versión de poda antes definida.

#### Ejemplo n=2

Como n es par entonces vamos a resolverlo por simetría. Agregamos 0 y la lista nos queda [0] esta es una posición válida por lo cual tomaremos el valor más pequeño sin usar, en la lista [0,1] la posición no es válida porque coincide con la diagonal de la columna 0, entonces como ya no podemos aumentar regresamos a una columna no llena y asignamos el próximo valor sin usar, en la lista [1] tenemos que la columna es igual a 0 y la fila es igual a  $n/2$  por lo cual hemos terminado y retornamos las soluciones encontradas hasta ahora multiplicadas por 2, lo cual nos daría 0 soluciones para n=2.

#### Ejemplo n=3

Como n es impar entonces vamos a resolverlo por poda.

Agregamos 0 y la lista nos queda [0] esta es una posición válida por lo cual tomaremos el valor más pequeño sin usar, en la lista [0,1] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos la fila por el valor próximo más pequeño sin usar, en la lista [0,2] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [0,2,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta tener una columna que tenga la posibilidad de agregarle un valor sin usar y reemplazamos por el valor mínimo sin usar que sea mayor a la fila, en la lista [1] la posición es válida entonces agregamos el valor mínimo sin usar, [1,0] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos por el próximo valor sin usar, en la lista [1,2] la posición no es válida porque coincide con la diagonal de la columna 0, entonces como ya no podemos reemplazar, regresamos hasta que se pueda reemplazar una fila de una columna por un valor sin usar mayor a la fila, en la lista [2] la posición es válida entonces agregamos el valor más pequeño sin usar, en la lista [2,0] la posición es válida entonces agregamos el valor más pequeño sin usar, en la lista [2,0,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta que una fila tenga un valor próximo que se le pueda agregar, en la lista [2,1] la posición no es válida porque coincide con la diagonal de la columna 0, y al retroceder ya no tenemos ningún valor sin usar para reemplazar en la columna 0 , entonces terminamos con la cantidad de 0 soluciones para n=3.

#### Ejemplo n=4

Como n es par lo vamos a resolver por simetría.

Agregamos 0 y la lista nos queda [0] esta es una posición válida por lo cual tomaremos el valor más pequeño sin usar, en la lista [0,1] la posición no es válida porque coincide con la diagonal de la columna 0, entonces

reemplazamos la fila por el valor próximo más pequeño sin usar, en la lista [0,2] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [0,2,1] la posición no es válida porque coincide con la diagonal de la columna 1, entonces reemplazamos por el valor próximo en la fila de la columna actual, en la lista [0,2,3] la posición no es válida porque coincide con la diagonal de la columna 1, entonces regresamos hasta encontrar una columna con una fila para reemplazar con un valor próximo, en la lista [0,3] la posición es válida, entonces agregamos el valor más pequeño de los valores sin usar, en la lista [0,3,1] la posición es válida, entonces agregamos el valor mínimo de los valores sin usar, en la lista [0,3,1,2] la posición no es válida porque coincide con la diagonal de la columna 2, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [0,3,2] la posición no es válida porque coincide con la diagonal de la columna 1 entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar.

En la lista [1] la posición es válida entonces agregamos el valor mínimo sin usar, en la lista [1,0] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos por el próximo valor sin usar, en la lista [1,2] la posición no es válida porque coincide con la diagonal de la columna 0, entonces reemplazamos con el valor próximo, en la lista [1,3] la posición es válida , entonces agregamos el valor mínimo de los valores sin usar, en la lista [1,3,0], la posición es válida, entonces agregamos el valor mínimo de los valores sin usar, en la lista [1,3,0,2] la posición es válida y hemos llegado a un límite entonces se agrega 1 al contador de soluciones y seguimos buscando soluciones, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar, en la lista [1,3,2] la posición no es válida porque coincide con la diagonal de la columna 1, entonces retrocedemos hasta encontrar una columna con un valor próximo para reemplazar. En la lista [2] tenemos que la columna es igual a 0 y que la fila es igual a  $n/2$  entonces retornamos las soluciones encontradas hasta ahora multiplicadas por 2 dándonos una cantidad de 2 soluciones para  $n=4$ .

### Complejidades

En la implementación de Backtracking pudimos ver como crece aceleradamente el numero de operaciones mediante van avanzando, mediante un análisis de las operaciones realizadas en los ejemplos anteriores se llegó a la conclusión que por si solo el backtracking para  $n$  reinas tiene una complejidad de  $O(n!)$  debido a la naturaleza exponencial del problema. Cada reina se coloca en una fila diferente, y hay  $N$  opciones que, si bien por el mismo backtracking se puede omitir muchas opciones, en el peor de los casos se recorren la mayoría de estos.

La complejidad de la poda realizada excluye la capacidad de pasar por nodos innecesarios que contengan la misma fila, lo cual causa una mejora en el rendimiento lo cual nos da una complejidad de  $O(2^n)$  debido a que cada reina se coloca en una fila diferente, y hay  $N$  opciones para colocar la primera reina,  $N-1$  opciones para la segunda (ya que no puede estar en la misma fila ni en la misma columna que la primera),  $N-2$  opciones para la tercera, y así sucesivamente. Esto lleva a una complejidad exponencial, ya que en cada nivel del árbol se reducen las opciones. La complejidad de la heurística de simetría es más fácil de calcular sabiendo la complejidad de las dos anteriores porque si bien el mejor caso es que sea par para así solo calcular la mitad de los nodos, el peor de los casos es el que determinara la complejidad de nuestro algoritmo, que como por si mismo cuando es impar manda a llamar a la poda conserva la misma complejidad de  $O(n^2)$ .

## 2.4. Resultados

Gráfica

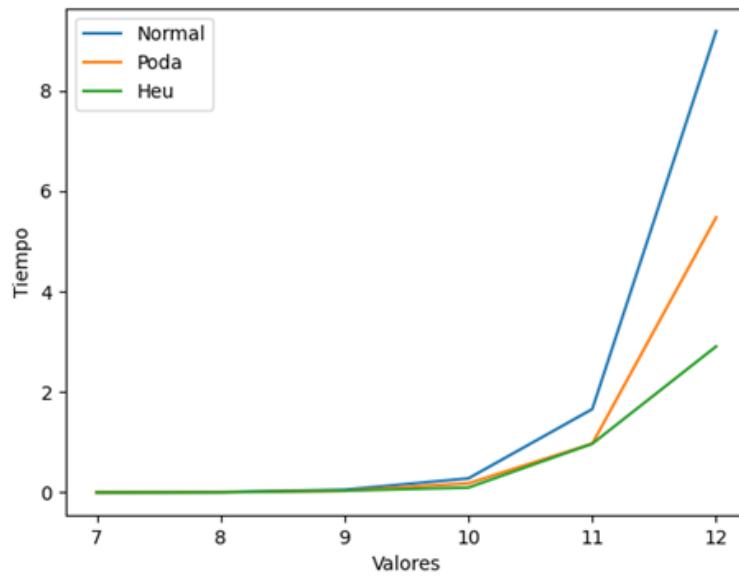


Figura 1: Resultados de la prueba de velocidad

Mediante esta grafica podemos ver como si bien en complejidad Big O no se logró bajar inmensamente la complejidad, si se pudo hacer una mejora con respecto a la complejidad temporal de este mismo. Aquí se puede apreciar como la versión sin optimización sube de manera drástica en el tiempo de ejecución, mientras que