OBJETIVO

El objetivo es implementar y comparar diferentes enfoques algorítmicos para resolver el problema del recorrido del caballo en un tablero de ajedrez, utilizando las técnicas de **Backtracking** y **Branch & Bound** (B&B).

El recorrido debe visitar todas las casillas del tablero exactamente una vez, en lo que se conoce como el problema del caballo.

RESOLUCIÓN

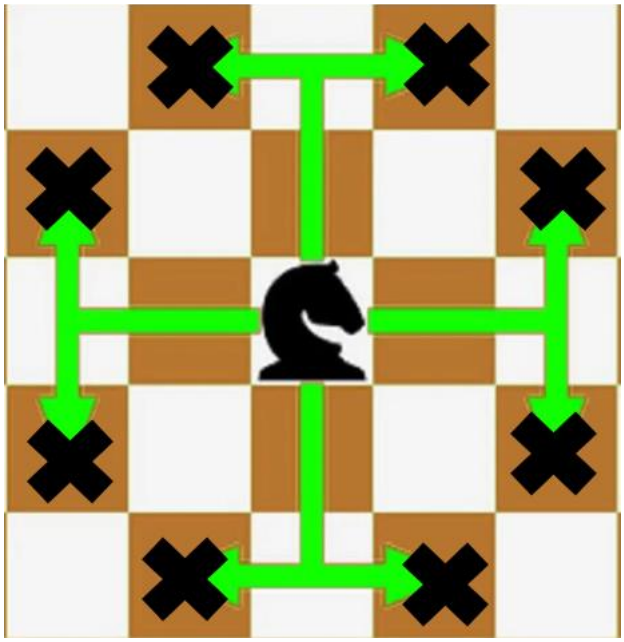
Tanto para la implementación con **Backtracking** como con **Branch & Bound**, se comienza inicializando el tablero de NxN con -1 en todas sus posiciones (el '-1' indica que el casillero aún no fue visitado).

A medida que se recorra el tablero, el -1 será reemplazado por un 1 (casillero visitado), indicando así que dicha posición ya fue recorrida una vez y el caballo no puede volver a pasar por esa posición.

Teniendo en cuenta que un caballo de ajedrez sólo puede moverse en L, sólo se podrán realizar 8 posibles movimientos desde una posición (x,y). Para esto, se generan 2 arrays que marcarán los movimientos que el caballo puede realizar en "x" y en "y".

Ejemplo "moverse dos posiciones a la izquierda sobre el eje 'x' (-2) y una posición a la derecha sobre el eje 'y'" (+1).

Movimientos posibles:



BACKTRACKING

La implementación con Backtracking consiste en una búsqueda en profundidad donde el caballo, desde una posición inicial cualquiera, va a intentar moverse a cada una de esas 8 posibilidades en forma recursiva.

Se debe tener en cuenta que sólo se puede pasar una única vez por cada casillero y que el caballo no puede salir del tablero. Es decir, sólo puede moverse en el eje “x” y en el eje “y” entre 0 y N-1.

Por lo tanto, si un camino no lleva a la solución, se vuelve para atrás y se prueba otra opción. En este caso, el casillero visitado pasaría nuevamente a no visitado (-1).

El proceso se termina cuando se recorre todo el tablero, es decir se encuentra una solución o cuando no hay más combinaciones posibles para recorrer.

Complejidad temporal

Desde cada uno de los casilleros, el caballo puede intentar realizar hasta 8 movimientos. Como el objetivo es visitar una sola vez todos los casilleros del tablero NxN, la cantidad de combinaciones posibles a recorrer dependerá de N.

El algoritmo explora **todas las combinaciones posibles** de movimientos hasta encontrar una solución válida. Por lo tanto, la complejidad temporal es exponencial debido a la cantidad de combinaciones que deben hacerse para recorrer el tablero.

$$\theta(8^{NxN})$$

Para tableros de ajedrez grandes, la cantidad de combinaciones se vuelve inmanejable con Backtracking. Para N ≥ 7 el algoritmo resulta ineficiente.

BRANCH & BOUND

A diferencia de Backtracking, Branch & Bound es una técnica de optimización.

Utiliza una heurística para la toma de decisiones al momento de elegir un camino a recorrer. Se comienzan explorando los caminos más prometedores.

Siguiendo la heurística de Warnsdorff, el próximo movimiento a elegir debe ser el que nos lleve a un casillero desde el cual existe la menor cantidad de opciones posibles para el siguiente paso.

A partir de los 8 movimientos posibles que pueden realizarse, siempre y cuando el movimiento sea válido y el casillero aún no haya sido visitado, se va a calcular la cantidad de movimientos que existen desde esa posición (x,y), con dichas cantidades de movimientos posibles, se arma una lista ordenada por prioridad, de menor a mayor.

De esta forma, se empieza a recorrer por el nodo más prometedor. Esto permite reducir el número de ramas exploradas y por lo tanto, también se reducen los nodos explorados hasta llegar a la solución.

Al igual que con backtracking, se retrocede cuando el recorrido no puede continuar por un camino.

El algoritmo se repite hasta que se llega a la solución o en el peor de los casos, se terminan probando todas las posibilidades.

La complejidad temporal sería igual a la de Backtracking:

$$\theta(8^{n^2})$$

Sin embargo, en la práctica, la media de los casos tiene una complejidad temporal de

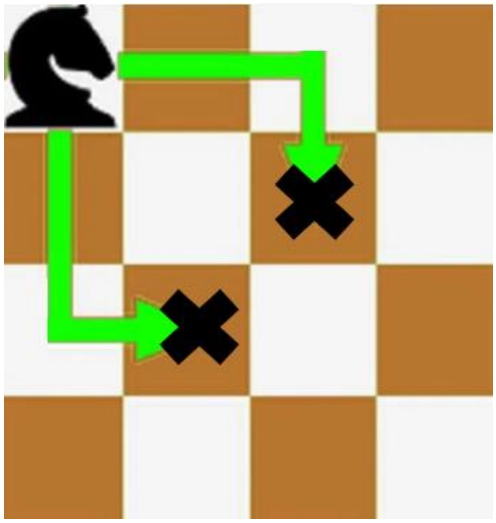
$$\theta(n^2)$$

Al incorporar la heurística de Warnsdorff y priorizarse los movimientos que conducen a casilleros con la menor cantidad de salidas futuras posibles (principio del menor grado), se evita el recorrido por ramas que no llevan a la solución por lo cual, la cantidad de nodos explorados va a ser significativamente menor a la obtenida con el algoritmo de Backtracking.

Es decir que al ramificar y podar, la complejidad temporal se reduce.

ANÁLISIS

Pruebas realizadas partiendo de la posición [0,0]



Comparación de Resultados

Para las pruebas se evaluaron las cantidades de nodos explorados y el tiempo de ejecución en milisegundos para tableros de ajedrez de distintos tamaños N tanto con el algoritmo de Backtracking como con Branch & Bound.

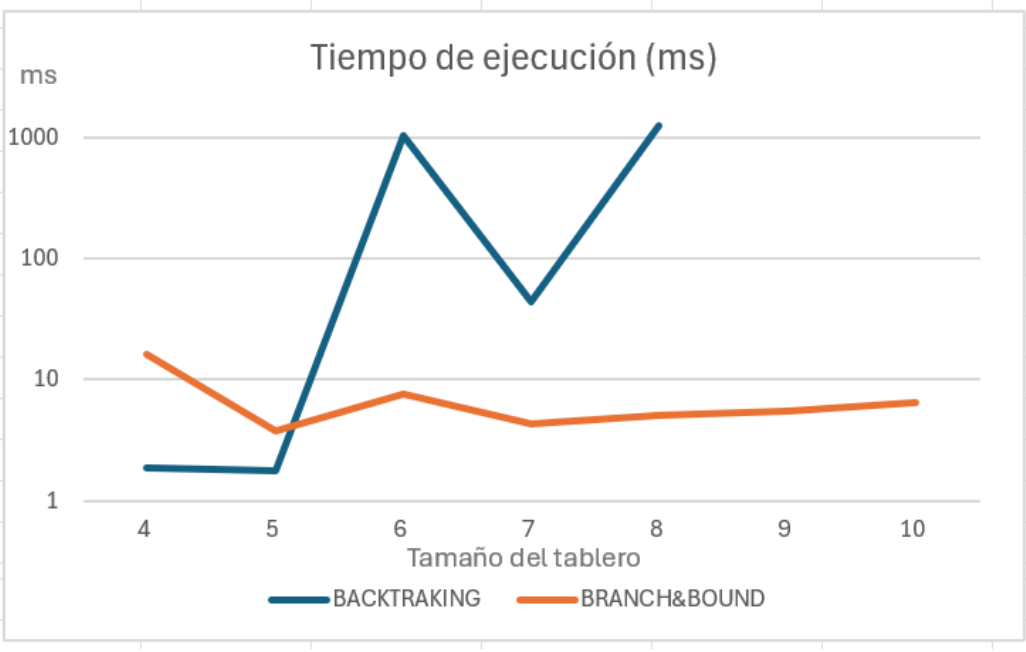
Se comenzó desde una posición inicial (0,0) para tableros con tamaño N = 4, 5, 6, 7, 8, 9,10, 13, 14 y 30.

	Backtracking		Branch & Bound	
Tablero	Nodos recorridos	Tiempo (ms)	Nodos recorridos	Tiempo (ms)
4x4	2.223	1.8522	2.223	16.2837
5x5	6.11	1.7543	25	3.7642
6x6	15.769.344	1045.7149	36	7.7082
7x7	364.451	44.1313	49	4.3542
8x8	20.716.588	1250.5969	64	5.0716
9x9	No termina	-	81	5.5212
10x10	No termina	-	100	6.4702
13x13	No termina	-	169	6.856
14x14	No termina	-	196	7.9077
30x30	No termina	-	900	12.8078

En el caso del tablero 4x4, no existe una solución para el recorrido del caballo.

Gráficos

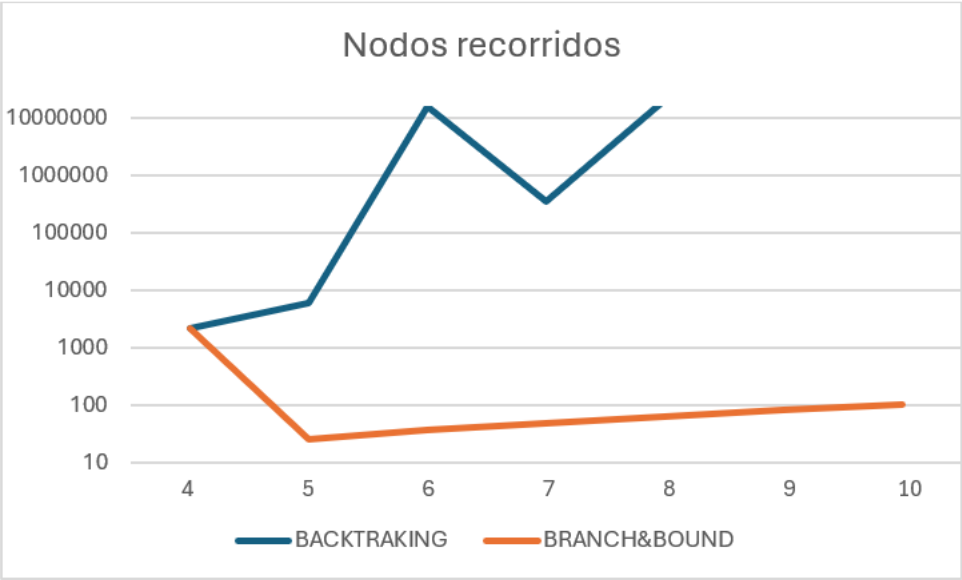
Por tiempo de ejecución:



Con backtracking el tiempo de ejecución crece exponencialmente al aumentar el tamaño del tablero y se vuelve ineficiente cuando se intenta utilizar el algoritmo para tableros con N mayor a 6.

En cambio, Branch & Bound puede resolver tableros grandes en un tiempo razonable. El tiempo de ejecución crece de manera casi lineal.

Por cantidad de nodos recorridos:



Al no haber poda en backtracking, se exploran todas las posibilidades desde cada casillero hasta llegar a una solución.

Mientras más grande sea N, mayor será la cantidad de posibles combinaciones a explorar.

En cambio, Branch & Bound prioriza los movimientos más prometedores, descartando rápidamente ramas sin futuro gracias a la heurística de Warnsdorff. Esto permite reducir la cantidad de nodos a explorar.

Comparación con distintas posiciones iniciales

TABLERO 4x4 (no tiene solución)				
(x,y)	Nodos Backtracking	Tiempo (ms) Backtracking	Nodos B&B	Tiempo (ms) B&B
(0,0)	2.223	1.3052	2.223	17.561
(2,3)	1.885	1.3033	1.885	15.7522
(1,2)	1.501	2.2412	1.501	16.3201
(0,2)	1.885	1.1833	1.885	11.3683

TABLERO 6x6				
(x,y)	Nodos Backtracking	Tiempo (ms) Backtracking	Nodos B&B	Tiempo (ms) B&B
(0,0)	15769344	961.8015	36	4.6577
(3,3)	113078576	5470.0832	36	4.0375
(2,4)	77051958	3856.7687	36	4.0159
(5,5)	2373110	118.0108	36	4.7331

TABLERO 7x7				
(x,y)	Nodos Backtracking	Tiempo (ms) Backtracking	Nodos B&B	Tiempo (ms) B&B
(0,0)	364451	24.9852	49	4.2579
(3,6)	-	-	-	-
(5,5)	8580532	582.7194	49	4.2969
(2,4)	-	-	522362	297.868

TABLERO 8x8				
(x,y)	Nodos Backtracking	Tiempo (ms) Backtracking	Nodos B&B	Tiempo (ms) B&B
(0,0)	20716588	1139.0704	64	5.8922
(4,4)	40613513	2066.1458	64	5.6662
(5,6)	-	-	64	5.4981
(3,7)	-	-	64	4.5014

TABLERO 9x9				
(x,y)	Nodos Backtracking	Tiempo (ms) Backtracking	Nodos B&B	Tiempo (ms) B&B
(0,0)	-	-	81	5.455
(2,4)	-	-	81	4.76
(7,5)	-	-	90	5.8501
(1,3)	-	-	81	4.7602

Dependiendo de la posición de inicio, en la mayoría de los casos, la cantidad de nodos explorados y el tiempo de ejecución va a variar mucho más con el uso de Backtracking que con Branch & Bound.

Para Branch & Bound, la posición inicial tiene poca influencia práctica sobre el rendimiento. La heurística ajusta el recorrido de forma eficiente en todos los casos.

CONCLUSIÓN

Para tableros a partir de 7x7 no es eficiente utilizar Backtraking debido al alto tiempo de ejecución que lleva evaluar todas las posibles combinaciones de movimientos.

El número de combinaciones a recorrer crece de manera exponencial con el tamaño del tablero siendo cada vez mayor la cantidad de nodos que se deben explorar.

En cambio, Branch & Bound, prioriza los caminos más probables para recorrer todo el tablero y poda aquellos que no conducen a la solución. Esto permite recorrer una menor cantidad de nodos y llegar a la solución (si es que existe) de forma más rápida. Al reducir los caminos a explorar, se minimiza el tiempo de ejecución.

En la práctica la complejidad temporal es menor, incluso en tableros de gran tamaño debido a la optimización al momento de elegir un camino.

SI bien Banch & Bound resulta más eficiente para el problema del recorrido del caballo de Ajedrez que intenta buscar una sola solución. La técnica de Backtracking puede ser más útil si se necesitara buscar todas las soluciones posibles y no sólo una. Siempre y cuando no sea un tablero grande con N mayor a 6.