ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA	INFORME DE PRÁCTICA DE LABORATORIO

DEPARTAMENTO:	**Ciencias de la computación**	**CARRERA:**	**Ingenieria en tecnologías de la informacion**		
ASIGNATURA:	Estructura de datos	**NIVEL:**	3	**FECHA:**	12-01-2025
DOCENTE:	Margoth Elisa Guaraca	**PRÁCTICA N°:**	3.1	**CALIFICACIÓN:**	

Análisis del Algoritmo de Backtracking a través del Problema de las N-Reinas

Angel Steven Rodriguez Chavez

RESUMEN

En esta práctica programamos una solución del problema de las N-Reinas con backtracking en Java y la conectamos a una interfaz en Swing para visualizar el resultado. El programa ubica una reina por columna y, si aparece un conflicto, deshace la última jugada y prueba otra posición. La GUI dibuja el tablero con casillas alternadas y marca las reinas con el símbolo ♛, lo que nos ayudó a comprobar rápidamente si la solución era correcta. Probamos con varios tamaños de tablero y observamos que para N=2 y N=3 no existen soluciones, mientras que para N=4 y N=8 sí. Durante el trabajo revisamos el punto exacto donde se realiza el "vuelta atrás" y medimos el impacto de las verificaciones de seguridad en filas y diagonales para reducir intentos innecesarios. Finalmente, dejamos planteado cómo recorrer todo el espacio de soluciones sin detenerse en la primera.

In this exercise, we programmed a solution to the N-Queens problem using backtracking in Java and connected it to a Swing interface to visualize the result. The program places one queen per column and, if a conflict arises, undoes the last move and tries another position. The GUI draws the board with alternating squares and marks the queens with the symbol ♛, which helped us quickly check if the solution was correct. We tested with various board sizes and observed that for N=2 and N=3 there are no solutions, while for N=4 and N=8 there are. During the work, we reviewed the exact point where the "backtracking" is performed and measured the impact of row and diagonal safety checks to reduce unnecessary attempts. Finally, we left open the question of how to traverse the entire solution space without stopping at the first one.

Palabras Claves: Backtracking; N-Queens; Java Swing

1. INTRODUCCIÓN:

El Backtracking es una técnica algorítmica que explora decisiones parciales y vuelve atrás al detectar que una rama no puede llevar a solución. En el problema de las N-Reinas, el objetivo es ubicar N reinas en un tablero N×N sin ataques mutuos. En esta práctica se aplicó backtracking para construir la solución de forma sistemática, se respetó la disciplina de laboratorio (organización del proyecto, verificación y documentación) y se incorporó una interfaz gráfica que facilita la comprobación visual de los resultados.

Backtracking is an algorithmic technique that explores partial decisions and backtracks when it detects that a branch cannot lead to a solution. In the N Queens problem, the objective is to place N queens on an N×N board without mutual attacks. In this practice, backtracking was applied to construct the solution systematically, laboratory discipline was respected (project organization, verification, and documentation), and a graphical interface was incorporated to facilitate visual verification of the results.

2. OBJETIVO(S):

2.1 Implementar en Java un solucionador de N-Reinas usando backtracking.
2.2 Diseñar una GUI con Swing que muestre el tablero y la disposición final de reinas.
2.3 Analizar el efecto de la poda (verificaciones de seguridad) y los casos con/sin solución para distintos N.

2.1 Implement an N-Queens solver in Java using backtracking.

2.2 Design a GUI with Swing that displays the board and the final arrangement of queens.
2.3 Analyze the effect of pruning (safety checks) and cases with/without a solution for different N.

3. MARCO TEÓRICO:

Backtracking realiza una búsqueda en profundidad con retroceso: ante una elección inválida se deshace y se prueban alternativas. Para N-Reinas, basta verificar amenazas hacia la izquierda de la posición candidata: misma fila, diagonal superior izquierda y diagonal inferior izquierda. Aunque el peor caso es exponencial, la poda producida por estas verificaciones reducen considerablemente el espacio explorado.

Backtracking performs a depth-first search with backtracking: when faced with an invalid choice, it backtracks and tries alternatives. For N Queens, it suffices to check threats to the left of the candidate position: same row, upper left diagonal, and lower left diagonal. Although the worst case is exponential, the pruning produced by these checks considerably reduces the space explored.

4. DESCRIPCIÓN DEL PROCEDIMIENTO/ PROCEDURE DESCRIPTION:

- Entorno: Java (JDK 8+), IDE (Eclipse/VS Code), sistema Windows.
- Estructura: matriz `int[N][N]` para representar el tablero; `1` indica una reina colocada.
- Lógica principal: método `resolverNReinas(tablero, col)` que avanza columna por columna.
- Verificación: `esSeguro(tablero, fila, col)` revisa fila izquierda y diagonales superior/inferior izquierdas.
- Impresión: `imprimirSolucion(tablero)` para consola (Q/.) y `TableroPanel` para GUI (♕ en cada celda con `1`).
- Interfaz: `InterfazNReinas` crea la ventana, solicita N y, si hay solución, pinta el tablero.
- Pruebas: ejecutar con N={2,3,4,8}; observar salida en consola y GUI.

- Environment: Java (JDK 8+), IDE (Eclipse/VS Code), Windows system.
- Structure: `int[N][N]` array to represent the board; `1` indicates a queen has been placed.
- Main logic: `resolveNQueens(board, col)` method that advances column by column.
- Verification: `esSeguro(board, row, col)` checks the left row and upper/lower left diagonals.
- Printing: `imprimirSolucion(board)` for console (Q/.) and `TableroPanel` for GUI (♕ in each cell with `1`).
- Interface: `InterfazNReinas` creates the window, requests N, and, if there is a solution, paints the board.
- Tests: run with N={2,3,4,8}; observe output in console and GUI.

5. ANÁLISIS DE RESULTADOS/ ANALYSIS OF RESULTS:

- Vuelta atrás: se realiza en `resolverNReinas(...)` al ejecutar `tablero[i][col] = 0` cuando la rama no conduce a solución; esté paso es crítico para evitar quedar atrapados en decisiones incorrectas.
- Resultados: N=2 y N=3 no tienen solución; N=4 y N=8 sí. El programa actual se detiene en la primera solución válida, lo que acelera la ejecución pero no enumera todas las posibilidades.
- Poda: las verificaciones de seguridad evitan explorar configuraciones completas inválidas, reduciendo el tiempo de búsqueda respecto de fuerza bruta.
- Extensión propuesta: para obtener todas las soluciones, registrar cada vez que `col >= N`, almacenar una copia del tablero y continuar la búsqueda sin retornar `true` de inmediato.
- Observaciones prácticas: el orden de exploración por filas impacta la solución específica que aparece primero; variar el orden puede producir una configuración distinta sin cambiar la corrección del algoritmo.

- Backtracking: performed in `resolverNReinas(...)` by executing `tablero[i][col] = 0` when the branch does not lead to a solution; this step is critical to avoid getting stuck in incorrect decisions.
- Results: N=2 and N=3 have no solution; N=4 and N=8 do. The current program stops at the first valid solution, which speeds up execution but does not enumerate all possibilities.
- Pruning: safety checks prevent exploring completely invalid configurations, reducing search time compared to brute force.
- Proposed extension: to obtain all solutions, record each time `col >= N`, store a copy of the board, and continue the search without immediately returning `true`.
- Practical observations: the order of exploration by rows impacts the specific solution that appears first; varying the order can produce a different configuration without changing the correctness of the algorithm.

6. GRÁFICOS O FOTOGRAFÍAS/ GRAPHICS OR PHOTOGRAPHS:

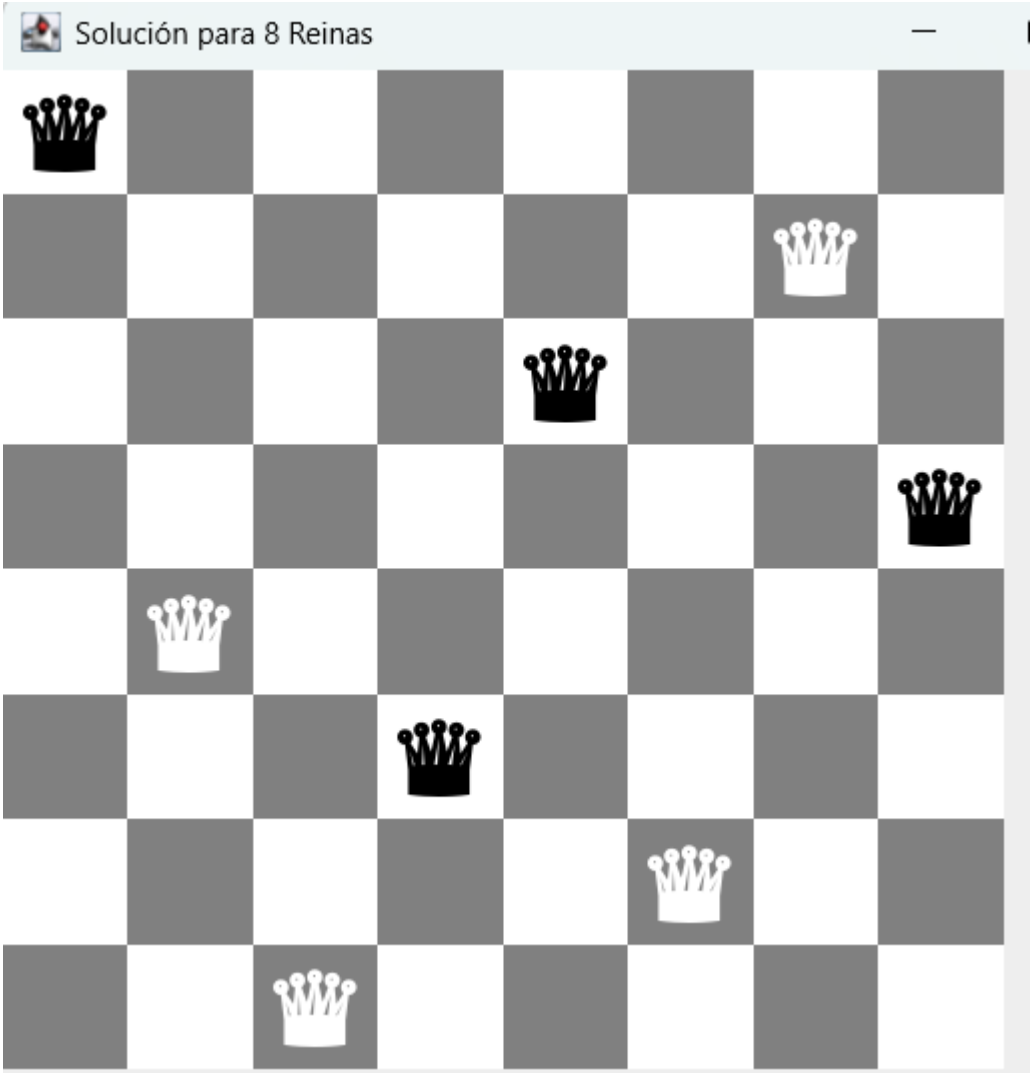

Ilustración 1 Ventana principal – solución para N=8

```java
private boolean resolverNReinas(int[][] tablero, int col) {
    boolean exito = false;
    if (col >= N) {
        exito = true;
    } else {
        int i = 0;
        while (i < N && !exito) {
            if (esSeguro(tablero, i, col)) {
                tablero[i][col] = 1;
                exito = resolverNReinas(tablero, col + 1);
                if (!exito) {
                    tablero[i][col] = 0; // BACKTRACK
                }
            }
            i++;
        }
    }
    return exito;
}
```

Ilustración 2 Método `resolverNReinas` – punto de backtracking

```java
private boolean esSeguro(int[][] tablero, int fila, int col) {
    boolean esSegura = true;
    int i, j;
    // 1. Verificar la fila hacia la izquierda
    for (i = 0; i < col && esSegura; i++) {
        if (tablero[fila][i] == 1) {
            esSegura = false;
        }
    }
    // 2. Verificar la diagonal superior izquierda
    for (i = fila, j = col; i >= 0 && j >= 0 && esSegura; i--, j--) {
        if (tablero[i][j] == 1) {
            esSegura = false;
        }
    }
    // 3. Verificar la diagonal inferior izquierda
    for (i = fila, j = col; j >= 0 && i < N && esSegura; i++, j--) {
        if (tablero[i][j] == 1) {
            esSegura = false;
        }
    }
    return esSegura;
}
```

Ilustración 3 Método `esSeguro` – verificación de fila y diagonals

```
65      public static void imprimirSolucion(int[][] tablero) {
66          for (int i = 0; i < tablero.length; i++) {
67              for (int j = 0; j < tablero.length; j++) {
68                  System.out.print(" " + (tablero[i][j] == 1 ? "Q" : ".") + " ");
69              }
70              System.out.println();
71          }
72      }
```

Ilustración 4 Impresión en consola – `imprimirSolucion

```
// Método simplificado para referencia
private boolean resolverNReinas(int[][] tablero, int col) {
  if (col >= N) return true; // Caso Base: Éxito

  for (int i = 0; i < N; i++) { // Probar todas las filas
    if (esSeguro(tablero, i, col)) {
      tablero[i][col] = 1; // Colocar Reina
      if (resolverNReinas(tablero, col + 1)) return true; //
Recursión
      tablero[i][col] = 0; // BACKTRACKING (Deshacer)
    }
  }
  return false; // No hay solución desde aquí
}
```

Se utiliza N=4 para la prueba de escritorio porque es el valor entero más pequeño que produce una solución no trivial, permitiendo demostrar el funcionamiento del Backtracking y la poda del árbol sin la explosión combinatoria

Proceso (Traza Paso a Paso)					
Paso	**Columna (col)**	**Fila (i)**	**¿Es Seguro?**	**Acción en tablero**	**Resultado / Comentario**
1	0	0	**SÍ**	[0][0] = 1	Coloca Q en (0,0). Llama recursión col=1.
2	1	0	NO	-	Ataca fila (0,0).
3	1	1	NO	-	Ataca diagonal (0,0).
4	1	2	**SÍ**	[2][1] = 1	Coloca Q en (2,1). Llama recursión col=2.
5	2	0	NO	-	Ataca fila (0,0).
6	2	1	NO	-	Ataca diagonal (2,1).
7	2	2	NO	-	Ataca fila (2,1).
8	2	3	NO	-	Ataca diagonal (2,1).
9	2	-	-	-	**FALLO**. Fin de filas. Retorna false.

10	1	2	-	[2][1] = 0	**BACKTRACKING**. Borra Q de (2,1). Avanza i.
11	1	3	**SÍ**	[3][1] = 1	Coloca Q en (3,1). Llama recursión col=2.
12	2	0	NO	-	Ataca fila (0,0).
13	2	1	**SÍ**	[1][2] = 1	Coloca Q en (1,2). Llama recursión col=3.
14	3	0	NO	-	Ataca diagonal (1,2).
15	3	1	NO	-	Ataca fila (1,2).
16	3	2	NO	-	Ataca diagonal (3,1) y (0,0).
17	3	3	NO	-	Ataca fila (3,1).
18	3	-	-	-	**FALLO**. Retorna false.
19	2	1	-	[1][2] = 0	**BACKTRACKING**. Borra Q de (1,2).
20	2	...	NO	-	Filas 2 y 3 tampoco son seguras.
21	2	-	-	-	**FALLO**. Retorna false.
22	1	3	-	[3][1] = 0	**BACKTRACKING**. Borra Q de (3,1).
23	1	-	-	-	**FALLO**. Fin de filas en col 1. Retorna false.
24	0	0	-	[0][0] = 0	**BACKTRACKING PRINCIPAL**. La primera reina en (0,0) no sirve.
25	0	1	**SÍ**	[1][0] = 1	**NUEVO CAMINO**. Coloca Q en (1,0). Llama col=1.
26	1	0	NO	-	Ataca diagonal (1,0).
27	1	1	NO	-	Ataca fila (1,0).
28	1	2	NO	-	Ataca diagonal (1,0).
29	1	3	**SÍ**	[3][1] = 1	Coloca Q en (3,1). Llama recursión col=2.
30	2	0	**SÍ**	[0][2] = 1	Coloca Q en (0,2). Llama recursión col=3.
31	3	0	NO	-	Ataca fila (0,2).

32	3	1	NO	-	Ataca diagonal (0,2).
33	3	2	**SÍ**	[2][3] = 1	**Coloca Q en (2,3).** Llama recursión col=4.
34	4	-	-	-	col >= 4. **¡SOLUCIÓN ENCONTRADA!** Retorna true.

Pantalla (Salida Final) 4x4			
.	.	Q	.
Q	.	.	.
.	.	.	Q
.	Q	.	.

7. DISCUSIÓN:

Los resultados concuerdan con la teoría: para N<4 no existen soluciones (excepto N=1), mientras que para N≥4 sí. La separación entre verificación de seguridad y recursión mejora la claridad del algoritmo y la posibilidad de mantenimiento.
La GUI facilita validar y comunicar resultados, además de ofrecer una herramienta visual para explicar el concepto de backtracking y el efecto de la poda. Como limitación, el rendimiento para N grandes puede ser elevado; se pueden explorar optimizaciones como heurísticas de ordenación de filas, interrupción por tiempo o enumeración con filtros de simetría para reducir duplicados.

The results agree with the theory: for N<4 there are no solutions (except N=1), while for N≥4 there are. The separation between safety verification and recursion improves the clarity of the algorithm and the possibility of maintenance.
The GUI makes it easier to validate and communicate results, as well as providing a visual tool to explain the concept of backtracking and the effect of pruning. As a limitation, performance for large N can be high; optimizations such as row sorting heuristics, timeout, or enumeration with symmetry filters
can be explored to reduce duplicates.
The results are consistent with the theory: for N<4 there are no solutions (except N=1), while for N≥4 there are. The separation between security verification and recursion improves the clarity of the algorithm and the possibility of maintenance. The GUI makes it easier to validate and communicate results, as well as providing a visual tool to explain the concept of backtracking and the effect of pruning. As a limitation, performance for large N can be high; optimizations such as row sorting heuristics, timeout, or enumeration with symmetry filters can be explored to reduce duplicates.

8. CONCLUSIONES:

- Se implementó el solucionador con backtracking y se identificó claramente el punto de vuelta atrás.
- La GUI en Swing representa el tablero y las reinas con ♛, mejorando la verificación visual de soluciones.
- Las verificaciones de seguridad podan el árbol de búsqueda y explican la ausencia/presencia de soluciones según N.
- Objetivo 1 alcanzado: el solucionador en Java funciona correctamente y aplica retroceso cuando es necesario.
- Objetivo 2 alcanzado: la interfaz en Swing muestra el tablero y la disposición final, facilitando la validación.
- Objetivo 3 alcanzado: se analizó la poda por verificaciones y se comprendieron los casos con/sin solución.
- Se recomienda, como trabajo futuro, implementar la enumeración completa de soluciones y medir tiempos para distintos N, documentando comparativas y posibles optimizaciones.

- The solver with backtracking was implemented and the backtracking point was clearly identified.
- The Swing GUI represents the board and queens with ♛, improving visual verification of solutions.
- Safety checks prune the search tree and explain the absence/presence of solutions according to N.
- Goal 1 achieved: the Java solver works correctly and applies backtracking when necessary.
- Objective 2 achieved: the Swing interface displays the board and the final layout, facilitating validation.
- Objective 3 achieved: pruning by checks was analyzed and cases with/without solutions were understood.
- As future work, it is recommended to implement the complete enumeration of solutions and measure times for different N, documenting comparisons and possible optimizations.

9. BIBLIOGRAFÍA:

- Wikipedia. Backtracking. https://en.wikipedia.org/wiki/Backtracking (consulta: 12-01-2025)
- Oracle. Creating a GUI With Swing. https://docs.oracle.com/javase/tutorial/uiswing/ (consulta: 12-01-2025)