

# Mini Proyecto Unidad 3

## 1. Datos Generales

<b>Asignatura</b>	Estructura de datos
<b>Ciclo</b>	3 A
<b>Unidad</b>	3
<b>Resultado de aprendizaje de la unidad</b>	Entiende los principios básicos de las operaciones en árboles, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.

## 2. Propósito:

Aplicar la técnica de búsqueda en profundidad con retroceso (backtracking) para resolver el problema clásico de las 9 reinas, fortaleciendo el razonamiento algorítmico, la validación de estados y la toma de decisiones en problemas combinatorios.

## 3. Competencias específicas:

- Diseño de algoritmos de búsqueda.
- Uso de recursividad y estructuras auxiliares.
- Análisis lógico de restricciones.
- Comunicación técnica de soluciones algorítmicas.

## 4. Equipos y herramientas

- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión "Extension Pack for Java") o IntelliJ IDEA Community.
- Sistema de control de versiones: Git; repositorio en GitHub.
- EVA/Moodle institucional: para entrega de evidencias.
- Herramientas de documentación: README Markdown, editor ofimático (Google Docs/LibreOffice/Word).

## 5. Procedimiento / Metodología

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos).

Inicio

- Presentación del problema de las 9 reinas y sus restricciones.
- Análisis grupal de por qué no es viable una solución por fuerza bruta.
- Discusión sobre el enfoque de búsqueda en profundidad con retroceso.

Desarrollo

- Representar el tablero usando un arreglo unidimensional `reinas[fila] = columna`.

- Implementar un método para verificar si una posición es segura:
  - Misma columna.
  - Diagonales.
- Implementar el método recursivo que:
  - Avanza fila por fila.
  - Intenta todas las columnas posibles.
  - Aplica backtracking cuando no es posible continuar.
- Detener el algoritmo al encontrar la primera solución válida.
- Mostrar la solución en consola de forma clara.

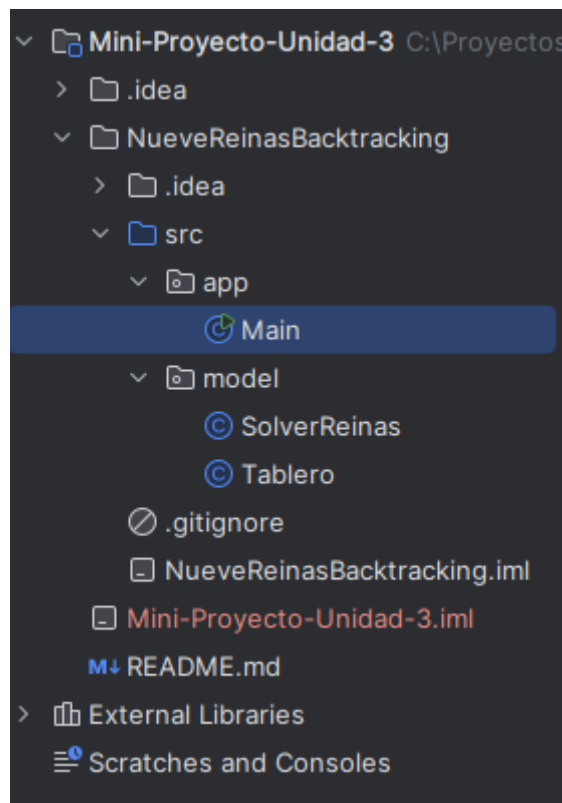
Cierre

- Análisis de la solución encontrada.
- Discusión sobre el número de intentos y retrocesos realizados.
- Retroalimentación del docente sobre la lógica del algoritmo.

## 6. Resultados esperados:

Como evidencia del trabajo realizado, el grupo deberá presentar:

- **Código Java funcional que resuelva el problema.**



```
package app;

import model.Tablero;
import model.SolverReinas;

public class Main { @Ismael

    public static void main(String[] args) { @Ismael

        int N = 9;

        System.out.println("Resolviendo el problema de las " + N + " Reinas...\n");

        Tablero tablero = new Tablero(N);
        SolverReinas solver = new SolverReinas(tablero);

        if (solver.resolver( fila: 0)) {
            System.out.println("Solución encontrada:\n");
            tablero.mostrar();
            System.out.println("\nIntentos realizados: " + solver.getIntentos());
            System.out.println("Retrocesos realizados: " + solver.getRetrocesos());
        } else {
            System.out.println("No se encontró una solución.");
        }
    }
}

public class SolverReinas { 3 usages @Ismael

    private Tablero tablero; 5 usages
    private int intentos; 3 usages
    private int retrocesos; 3 usages

    public SolverReinas(Tablero tablero) { 3 usages @Ismael
        this.tablero = tablero;
        this.intentos = 0;
        this.retrocesos = 0;
    }

    public boolean resolver(int fila) { 2 usages @Ismael

        if (fila == tablero.getTamaño()) {
            return true;
        }

        for (int columna = 0; columna < tablero.getTamaño(); columna++) {
            intentos++;

            if (tablero.esSeguro(fila, columna)) {
                tablero.getReinas()[fila] = columna;

                if (resolver( fila: fila + 1)) {
                    return true;
                }

                retrocesos++;
            }
        }
    }
}
```

```
package model;

public class Tablero {

    private int tamaño;
    private int[] reinas;

    public Tablero(int tamaño) {
        this.tamaño = tamaño;
        this.reinas = new int[tamaño];
    }

    public int getTamaño() { return tamaño; }

    public int[] getReinas() { return reinas; }

    /**
     * Verifica si es seguro colocar una reina en (fila, columna)
     */
    public boolean esSeguro(int fila, int columna) {
        for (int i = 0; i < fila; i++) {

            // Misma columna
            if (reinas[i] == columna) {
                return false;
            }

            // Diagonales
            if (Math.abs(i - fila) == Math.abs(reinas[i] - columna)) {
                return false;
            }
        }
    }
}
```

- Salida en consola con una solución válida.
- Capturas de pantalla de la ejecución.

```
Solución encontrada:

      a  b  c  d  e  f  g  h  i
+---+---+---+---+---+---+---+---+---+
1 | Q | . | . | . | . | . | . | . |
+---+---+---+---+---+---+---+---+---+
2 | . | . | Q | . | . | . | . | . |
+---+---+---+---+---+---+---+---+---+
3 | . | . | . | . | . | . | Q | . |
+---+---+---+---+---+---+---+---+---+
4 | . | . | . | . | . | . | . | Q |
+---+---+---+---+---+---+---+---+---+
5 | . | Q | . | . | . | . | . | . |
+---+---+---+---+---+---+---+---+---+
6 | . | . | . | Q | . | . | . | . |
+---+---+---+---+---+---+---+---+---+
7 | . | . | . | . | . | . | . | Q |
+---+---+---+---+---+---+---+---+---+
8 | . | . | . | . | . | . | Q | . |
+---+---+---+---+---+---+---+---+---+
9 | . | . | . | . | Q | . | . | . |
+---+---+---+---+---+---+---+---+---+

Intentos realizados: 333
Retrocesos realizados: 32
```

- **Breve explicación escrita del enfoque utilizado.**

El algoritmo utiliza búsqueda en profundidad con retroceso, avanzando fila por fila:

- Cada fila contiene exactamente una reina.
- En cada fila se prueban todas las columnas posibles.
- Antes de colocar una reina se validan las restricciones:
  - No compartir columna con otra reina.
  - No compartir diagonales.
- Si no es posible continuar, el algoritmo retrocede y prueba una nueva alternativa.
- El proceso se detiene al encontrar la primera solución válida.

**LINK DEL GIT:** <https://github.com/IsmaelGonz/Mini-Proyecto-Unidad-3>



## 7. Evaluación

Criterio	Excelente	Bueno	Básico	Puntaje
Implementación del algoritmo	Correcta y completa	Error menor	Parcial	4
Uso de backtracking	Correcto y justificado	Parcial	Deficiente	3
Validación de restricciones	Completa	Incompleta	Incorrecta	2
Claridad del código	Muy clara	Aceptable	Desordenada	1

## 8. Bibliografía

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] M. A. Weiss, Data Structures and Algorithm Analysis in Java, 4th ed., Pearson, 2021.

[3] D. E. Knuth, The Art of Computer Programming, Vol. 4A, Addison-Wesley, 2019.