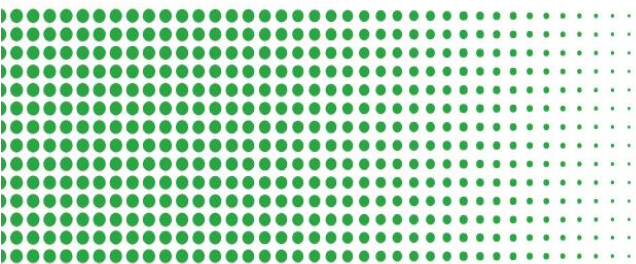


ACTIVIDAD 3

Estructura de Datos



Profesor: Emmanuel Rojas

Alumno: Dael Alexis Espino
Frías

Fecha: 05/09/2025

Descripción

El objetivo de esta actividad es que implementes y apliques tus conocimientos de recursividad y algoritmos de divide y vencerás mediante la resolución de problemas clásicos en programación. Se te proporcionará una serie de problemas que deberás resolver utilizando técnicas recursivas, recursividad indirecta y algoritmos de backtracking. Desarrollarás programas en Java para cada problema, documentando tu proceso y razonamiento.

Objetivo

- Aplicar conceptos de recursividad directa e indirecta.
- Implementar algoritmos de backtracking y divide y vencerás.
- Desarrollar la habilidad para resolver problemas complejos mediante la descomposición en subproblemas.

Requerimientos

- Java Development Kit (JDK) y un IDE como IntelliJ IDEA, Eclipse o Netbeans.
- Comprensión básica de los conceptos de recursividad y algoritmos de divide y vencerás.
- Lectura previa de los temas.

Instrucciones

1. Primer problema: Serie de Fibonacci recursiva. Implementa una función recursiva en Java que calcule el n -ésimo número en la serie de Fibonacci. Asegúrate de incluir tanto el caso base como el caso recursivo.

```
public class Fibonacci { no usages
    public static int calcular(int n) { 3 usages
        if (n == 0) return 0;
        if (n == 1) return 1;
        return calcular(n - 1) + calcular(n - 2);
    }

    public static void ejecutar(Scanner sc) { no usages
        System.out.print("Ingresa n para Fibonacci: ");
        int n = sc.nextInt();
        System.out.println("Fibonacci de " + n + " es: " + calcular(n));
    }
}
```

```
=== MENÚ DE PROBLEMAS ===
1. Serie de Fibonacci
2. Subset Sum
3. Resolver Sudoku
4. Salir
Elige una opción: 1
Ingresa n para Fibonacci: 5
Fibonacci de 5 es: 5
```

Básicamente lo que hace el código es calcular el (n) número de la serie de Fibonacci usando recursión. El usuario proporciona un número entero (n) mayor o igual a cero, que representa la posición del número de Fibonacci que desea calcular. La función devuelve el valor correspondiente en la serie. La serie de Fibonacci comienza con 0 y 1 y cada número siguiente se obtiene sumando los dos anteriores o sea que, $F(n) = F(n-1) + F(n-2)$. Los casos base son $F(0) = 0$ y $F(1) = 1$ es decir si se solicita el quinto número de Fibonacci ($n = 5$), la función devuelve 5

2. Segundo problema: Suma de subconjuntos (Subset Sum). Desarrolla un algoritmo recursivo para determinar si existe un subconjunto de un conjunto dado de enteros que sume un valor objetivo.

```
public class Subset_sum { 1 usage
    public static boolean existeSubconjunto(int[] nums, int n, int target) { 4 usages
        if (target == 0) return true;
        if (n == 0 && target != 0) return false;
        if (nums[n - 1] > target) return existeSubconjunto(nums, n - 1, target);
        return existeSubconjunto(nums, n - 1, target) ||
            existeSubconjunto(nums, n - 1, target - nums[n - 1]);
    }

    public static void ejecutar(Scanner sc) { 1 usage
        System.out.print("¿Cuántos números tendrá el conjunto? ");
        int size = sc.nextInt();
        int[] set = new int[size];

        System.out.println("Ingresa los números del conjunto:");
        for (int i = 0; i < size; i++) {
            set[i] = sc.nextInt();
        }

        System.out.print("Ingresa el valor objetivo: ");
        int target = sc.nextInt();

        if (existeSubconjunto(set, set.length, target)) {
            System.out.println("Existe un subconjunto cuya suma es " + target + "!");
        } else {
            System.out.println("No existe ningún subconjunto con esa suma.");
        }
    }
}
```

```
=== MENÚ DE PROBLEMAS ===
1. Serie de Fibonacci
2. Subset Sum
3. Resolver Sudoku
4. Salir
Elige una opción: 2
¿Cuántos números tendrá el conjunto? 5
Ingresa los números del conjunto:
1
2
3
4
5
Ingresa el valor objetivo: 15
¿Existe un subconjunto cuya suma es 15!
```

El problema de suma de subconjuntos se resuelve mediante un algoritmo recursivo que determina si existe un subconjunto de un conjunto dado que sume un valor objetivo la clase Subset_Sum define el método existeSubconjunto que maneja dos casos base: si el objetivo es cero da verdadero y si no quedan elementos y no se alcanzó el objetivo de falso. En el caso recursivo, la función decide incluir o excluir el último elemento del conjunto, probando ambos caminos. El método ejecutar permite al usuario ingresar los números del conjunto y el valor objetivo mostrando al final si existe un subconjunto cuya suma coincide con el objetivo.

3. Tercer problema: Algoritmo de backtracking para el problema del sudoku.
Implementa un algoritmo de backtracking que resuelva un Sudoku. El programa debe llenar las celdas vacías del tablero de Sudoku dado.

```
private static boolean esValido(int[][] tablero, int fila, int col, int num) { 1 usage
    for (int x = 0; x < N; x++) {
        if (tablero[fila][x] == num) return false;
        if (tablero[x][col] == num) return false;
    }
    int startRow = fila - fila % 3;
    int startCol = col - col % 3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (tablero[startRow + i][startCol + j] == num) return false;
        }
    }
    return true;
}
```

```
private static boolean resolver(int[][] tablero) { 2 usages
    for (int fila = 0; fila < N; fila++) {
        for (int col = 0; col < N; col++) {
            if (tablero[fila][col] == 0) {
                for (int num = 1; num <= 9; num++) {
                    if (esValido(tablero, fila, col, num)) {
                        tablero[fila][col] = num;
                        if (resolver(tablero)) return true;
                        tablero[fila][col] = 0;
                    }
                }
                return false;
            }
        }
    }
    return true;
}
```

```
private static void imprimir(int[][] tablero) { 2 usages
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            System.out.print(tablero[i][j] + " ");
        }
        System.out.println();
    }
}

public static void ejecutar() { 1 usage
    int[][] tablero = {
        {0, 0, 0, 2, 6, 0, 7, 0, 1},
        {6, 8, 0, 0, 7, 0, 0, 9, 0},
        {1, 9, 0, 0, 0, 4, 5, 0, 0},
        {8, 2, 0, 1, 0, 0, 0, 4, 0},
        {0, 0, 4, 6, 0, 2, 9, 0, 0},
        {0, 5, 0, 0, 0, 3, 0, 2, 8},
        {0, 0, 9, 3, 0, 0, 0, 7, 4},
        {0, 4, 0, 0, 5, 0, 0, 3, 6},
        {7, 0, 3, 0, 1, 8, 0, 0, 0}
    };

    System.out.println("=== Sudoku sin resolver ===");
    imprimir(tablero);

    if (resolver(tablero)) {
        System.out.println("\n=== Sudoku resuelto ===");
        imprimir(tablero);
    } else {
        System.out.println("No se puede resolver el Sudoku");
    }
}
```

```
=== MENÚ DE PROBLEMAS ===
1. Serie de Fibonacci
2. Subset Sum
3. Resolver Sudoku
4. Salir
Elige una opción: 3
=== Sudoku sin resolver ===
0 0 0 2 6 0 7 0 1
6 8 0 0 7 0 0 9 0
1 9 0 0 0 4 5 0 0
8 2 0 1 0 0 0 4 0
0 0 4 6 0 2 9 0 0
0 5 0 0 0 3 0 2 8
0 0 9 3 0 0 0 7 4
0 4 0 0 5 0 0 3 6
7 0 3 0 1 8 0 0 0

=== Sudoku resuelto ===
4 3 5 2 6 9 7 8 1
6 8 2 5 7 1 4 9 3
1 9 7 8 3 4 5 6 2
8 2 6 1 9 5 3 4 7
3 7 4 6 8 2 9 1 5
9 5 1 7 4 3 6 2 8
5 1 9 3 2 6 8 7 4
2 4 8 9 5 7 1 3 6
7 6 3 4 1 8 2 5 9
```

El Sudoku se resuelve mediante el backtracking que explora todas las posibles soluciones. La clase Sudoku incluye el método `esValido` que comprueba si un número puede colocarse en una posición específica revisando la fila, la columna y el subcuadro 3x3. El método `resolver` busca celdas vacías e intenta colocar números válidos y llama recursivamente a sí mismo para continuar con el siguiente espacio. Si no hay ningún número válido, retrocede y prueba otro número, aplicando la técnica de backtracking. El método `ejecutar` inicializa un tablero sin resolver, lo imprime, llama a `resolver` y finalmente muestra el tablero resuelto mostrando cómo el código encuentra la solución paso a paso.

