Algorítmica Básica: Técnicas Fundamentales

Este documento complementa las Unidades 2 y 3, profundizando en patrones algorítmicos esenciales que todo programador debe dominar. Son técnicas básicas con las que construirás soluciones más complejas.

1. Intercambio de Variables (El "Swap")

El Problema: ¿Cómo intercambiar el valor de dos variables? Si tienes `a = 5` y `b = 10`, ¿cómo consigues que `a` valga 10 y `b` valga 5?

La Solución Clásica: Necesitas una "caja" auxiliar temporal para no perder uno de los valores durante el intercambio.

Algoritmo:

1. Guarda el valor de `a` en una variable temporal `aux`.
2. Asigna el valor de `b` a `a`. (Ahora `a` tiene el valor original de `b`).
3. Asigna el valor guardado en `aux` (el valor original de `a`) a `b`.

Ejemplo en Java:

```java
public class IntercambioVariables {
    public static void main(String[] args) {
        int a = 5;
        int b = 10;
        int aux; // Nuestra "caja" auxiliar

        System.out.println("Valores iniciales:");
        System.out.println("a = " + a); // 5
        System.out.println("b = " + b); // 10

        // 1. Guardamos 'a'
        aux = a;

        // 2. 'a' toma el valor de 'b'
        a = b;

        // 3. 'b' toma el valor guardado en 'aux'
        b = aux;

        System.out.println("\nValores finales:");
        System.out.println("a = " + a); // 10
        System.out.println("b = " + b); // 5
    }
}
```

---

2. Búsqueda de Elementos en un Array

Es muy común necesitar encontrar información dentro de una colección de datos (un array). Veamos las búsquedas más típicas.

2.1. Búsqueda del Máximo y Mínimo

El Problema: Dado un array de números, ¿cómo encontrar el valor más grande (máximo) o el más pequeño (mínimo)?

Algoritmo (para el máximo):

1. Supón que el primer elemento del array es el máximo (`max = array[0]`).
2. Recorre el resto del array (desde el segundo elemento).
3. En cada paso, compara el elemento actual con tu `max` actual.
4. Si el elemento actual es mayor que `max` , actualiza `max` con ese valor.
5. Al final del recorrido, `max` contendrá el valor más grande.
   *(El algoritmo para el mínimo es idéntico, pero comparando si el elemento actual es *menor* que `min`)*.

Ejemplo en Java (Búsqueda del Máximo):

```
public class BusquedaMaximo {
    public static void main(String[] args) {
        int[] numeros = {12, 5, 23, 8, 19, 3, 15};

        int maximo = numeros[0]; // 1. Suponemos que el primero es el máximo

        // 2. Recorremos desde el segundo elemento
        for (int i = 1; i < numeros.length; i++) {
            // 3. Comparamos
            if (numeros[i] > maximo) {
                // 4. Actualizamos si encontramos uno mayor
                maximo = numeros[i];
            }
        }

        // 5. Mostramos el resultado
        System.out.println("El número máximo en el array es: " + maximo); // 23
    }
}
```

2.2. Búsqueda de un Elemento Específico (Búsqueda Lineal)

El Problema: ¿Cómo saber si un valor concreto existe dentro de un array y, si existe, en qué posición se encuentra?

Algoritmo (Búsqueda Lineal):

1. Recorre el array elemento por elemento, desde el principio.
2. En cada paso, compara el elemento actual con el valor que buscas.
3. Si coinciden, ¡lo has encontrado! Puedes devolver la posición (`i`) o simplemente un indicador (`true`).
4. Si llegas al final del array sin encontrarlo, significa que no está. Puedes devolver un valor especial (como `-1` si buscabas la posición) o `false` .

Ejemplo en Java (Encontrar la posición):

```
public class BusquedaLineal {
    public static void main(String[] args) {
        String[] frutas = {"Manzana", "Pera", "Naranja", "Plátano", "Fresa"};
        String frutaBuscada = "Naranja";
        int posicionEncontrada = -1; // -1 indica "no encontrado"

        // 1. Recorremos el array
        for (int i = 0; i < frutas.length; i++) {
            // 2. Comparamos (¡OJO! Strings se comparan con .equals())
```

```
            if (frutas[i].equals(frutaBuscada)) {
                // 3. ¡Encontrado! Guardamos la posición y salimos del bucle
                posicionEncontrada = i;
                break; // No hace falta seguir buscando
            }
        }

        // 4. Mostramos el resultado
        if (posicionEncontrada != -1) {
            System.out.println(frutaBuscada + " encontrada en la posición: " + posicionEncontrada); //
Naranja encontrada en la posición: 2
        } else {
            System.out.println(frutaBuscada + " no se encuentra en el array.");
        }
    }
}
```

---

3. Recorridos de Bucles: Más Allá del Básico `for`

El bucle `for` es muy potente. No solo sirve para ir de 0 hasta N. Podemos controlar el inicio, el fin, la dirección y el tamaño del "salto" en cada iteración.

3.1. Recorrido Inverso

La Técnica: ¿Y si necesitas procesar un array desde el último elemento hasta el primero? Simplemente, inicializa el contador del `for` en el último índice (`array.length - 1`), pon la condición para que continúe mientras sea mayor o igual a 0, y usa el decremento (`i--`).

Ejemplo en Java:

```
public class RecorridoInverso {
    public static void main(String[] args) {
        char[] letras = {'P', 'R', 'O', 'G', 'R', 'A', 'M', 'A'};

        System.out.println("Recorrido inverso:");
        // Inicializamos en el último índice
        for (int i = letras.length - 1; i >= 0; i--) {
            System.out.print(letras[i] + " "); // A M A R G O R P
        }
        System.out.println();
    }
}
```

3.2. Recorrido con Saltos (Intervalos)

La Técnica: ¿Necesitas procesar solo los elementos en posiciones pares? ¿O avanzar de 3 en 3? Modifica la parte del incremento/decremento del `for` . En lugar de `i++` o `i--` , usa `i += 2` , `i += 3` , `i -= 5` , etc.

Ejemplo en Java (Procesar índices pares y contar de 3 en 3):

```
public class RecorridoConSaltos {
    public static void main(String[] args) {
        int[] numeros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
        System.out.println("Elementos en índices pares:");
        // Saltamos de 2 en 2
        for (int i = 0; i < numeros.length; i += 2) {
            System.out.print(numeros[i] + " "); // 0 2 4 6 8 10
        }
        System.out.println();

        System.out.println("\nContando de 3 en 3 hasta 20:");
        // Saltamos de 3 en 3
        for (int i = 0; i <= 20; i += 3) {
            System.out.print(i + " "); // 0 3 6 9 12 15 18
        }
        System.out.println();
    }
}
```

3.3. Recorrido Parcial de un Array

La Técnica: No siempre necesitas recorrer un array completo. Puedes ajustar las condiciones de inicio y fin del bucle `for` para procesar solo una sección.

Ejemplo en Java (Sumar solo la segunda mitad del array):

```
public class RecorridoParcial {
    public static void main(String[] args) {
        int[] valores = {10, 20, 30, 40, 50, 60, 70, 80};
        int sumaSegundaMitad = 0;

        // Calculamos dónde empieza la segunda mitad
        int inicioSegundaMitad = valores.length / 2;

        System.out.println("Sumando la segunda mitad del array (índices " + inicioSegundaMitad + " a " +
(valores.length - 1) + ")");

        // Empezamos el bucle en la mitad
        for (int i = inicioSegundaMitad; i < valores.length; i++) {
            sumaSegundaMitad += valores[i];
        }

        System.out.println("La suma de la segunda mitad es: " + sumaSegundaMitad); // 50 + 60 + 70 + 80
= 260
    }
}
```

3.4. Bucles Anidados Dependientes (Patrones)

La Técnica: A veces, el número de veces que se ejecuta un bucle interior depende de la iteración actual del bucle exterior. Esto es muy útil para crear patrones, como triángulos.

Ejemplo en Java (Dibujar un triángulo de asteriscos):

```
public class TrianguloAsteriscos {
    public static void main(String[] args) {
        System.out.println("Dibujando un triángulo:");

        // Bucle exterior: controla las filas (de 0 a 4, 5 filas en total)
```

```
        for (int i = 0; i < 5; i++) {
            // Bucle interior: imprime asteriscos.
            // El número de asteriscos depende de la fila actual (i).
            // Si i=0, j va de 0 a 0 (1 vez). Si i=1, j va de 0 a 1 (2 veces), etc.
            for (int j = 0; j <= i; j++) {
                System.out.print("*");
            }
            // Salto de línea al final de cada fila
            System.out.println();
        }
    }
}
```

Salida:

```
Dibujando un triángulo:
*
**
***
****
*****
```

---

4. Manipulación Básica de Strings

Los `String` son secuencias de caracteres y, aunque en Java son objetos, a menudo los tratamos algorítmicamente como si fueran arrays de caracteres. Dominar su manipulación es crucial.

4.1. Comprobación de Palíndromos (Versión Manual)

El Problema: Un palíndromo es una palabra o frase que se lee igual hacia adelante que hacia atrás (ignorando espacios y mayúsculas/minúsculas). Ejemplos: "Ana", "oso", "Se van sus naves". ¿Cómo comprobar si un `String` es un palíndromo usando solo métodos básicos?

Algoritmo:

1. **Limpiar y Normalizar:**
   - Convierte el `String` original a minúsculas (`toLowerCase()`).
   - Recorre el `String` en minúsculas carácter a carácter (`charAt()` y `length()`).
   - Construye un nuevo `String` auxiliar (`limpio`) añadiendo solo los caracteres que NO sean espacios en blanco.
2. **Compararación de caracteres:**
   - Recorre el `String` `limpio` desde el inicio hasta la mitad (`limpio.length()/2`).
   - Compara cada posición con su *homólogo* inverso `limpio.length() - 1 - i` . Se detiene si los caracteres son diferentes.

Ejemplo en Java:

```
public class ComprobarPalindromo {

    public static void main(String[] args) {
        String frase1 = "Se van sus naves";
        String frase2 = "Java Mola";
```

```
        if (esPalindromo(frase1)) {
            System.out.println("'" + frase1 + "' es un palíndromo.");
        } else {
            System.out.println("'" + frase1 + "' NO es un palíndromo.");
        }

        if (esPalindromo(frase2)) {
            System.out.println("'" + frase2 + "' es un palíndromo.");
        } else {
            System.out.println("'" + frase2 + "' NO es un palíndromo.");
        }
    }

    /**
     * Función que comprueba si un String es palíndromo manualmente.
     * @param texto El String a comprobar.
     * @return true si es palíndromo, false en caso contrario.
     */
    public static boolean esPalindromo(String texto) {

        // 1. Limpiar y Normalizar (solo quitamos espacios)
        String textoEnMinusculas = texto.toLowerCase().trim();
        String limpio = ""; // String auxiliar para construir el texto sin espacios
        boolean esPalindromo = true;
        char aux;

        for (int i = 0; i < textoEnMinusculas.length(); i++) {
            aux = textoEnMinusculas.charAt(i);
            if (aux != ' ') {
                limpio = limpio + aux; // Concatenamos si no es espacio
            }
        }

        // 2. Comparamos los caracteres (recorremos solo hasta la mitad)
        for (int i = 0; i < limpio.length()/2 && esPalindromo; i++) {
            if (limpio.charAt(i) != limpio.charAt(limpio.length() - 1 - i)) {
                esPalindromo = false;
            }
        }
        return esPalindromo;
    }
}
```

Salida:

```
'Se van sus naves' es un palíndromo.
'Java Mola' NO es un palíndromo.
```