
TRABAJO PRÁCTICO INTEGRADOR PROGRAMACIÓN I

Análisis de Algoritmos - Validación de IDs duplicados entre registros y formularios

Emmanuel Rivero

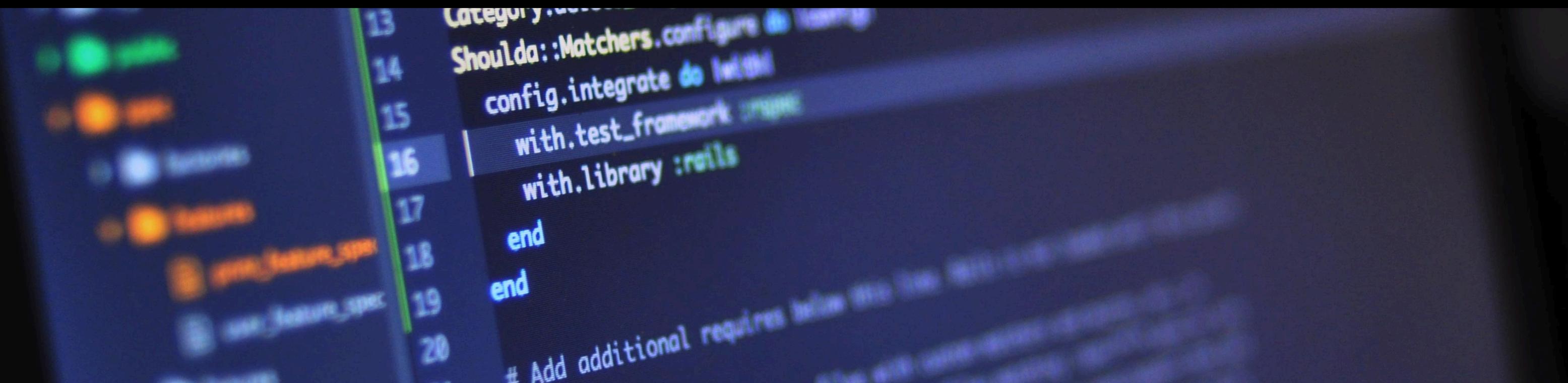
Julio Roja

```
"container">
  <div class="row">
    <div class="col-md-6 col-lg-8"> <!--
      <nav id="nav" role="navigation">
        <ul>
          <li><a href="index.html">Home</a>
          <li><a href="home-events.html">Home
          <li><a href="multi-col-menu.html">
          <li class="has-children"> <a href="#>
            <ul>
              <li><a href="tall-button-h
              <li><a href="image-logo.htm
              <li class="active"><a href=
            </ul>
          </li>
          <li class="has-children"> <a href="#>
            <ul>
              <li><a href="variable-width-
```

Porque elegimos este tema?

- Problema común en sistemas reales.
- Validación de formularios, sincronización y control de integridad.
- Permite comparar eficiencia algorítmica con datos reales.

```
16  
17     string sInput;  
18     int iLength, iN;  
19     double dblTemp;  
20     bool again = true;  
21  
22     while (again) {  
23         iN = -1;  
24         again = false;  
25         getline(cin, sInput);  
26         system("cls");  
27         stringstream(sInput) >> dblTemp;  
28         iLength = sInput.length();  
29         if (iLength < 4) {  
30             again = true;  
31             continue;  
32         } else if (sInput[iLength - 1] == '  
33             again = true;
```



Objetivos del Trabajo

- Comparar dos algoritmos para detectar duplicados.
- Analizar sus tiempos y comportamiento teórico (Big-O).
- Comprender el impacto del diseño algorítmico en la programación.



Análisis de Algoritmos

- Evalúa como se comporta un algoritmo al aumentar los datos
- Dos aspectos clave:
 - a) Eficiencia temporal
 - b) Eficiencia espacial
- Complemento de testeo tradicional.

Notación Big-O

La notación Big-O es una forma de expresar el rendimiento de un algoritmo en función del crecimiento de los datos.

- Nos indica como escalan el tiempo o el uso de memoria.
- Describe el “peor caso” de comportamiento de un algoritmo.
- Nos ayuda a elegir algoritmos mas eficientes incluso sin ejecutarlos.
- No depende del lenguaje elegido ni del entorno.



Notación Big-O



Big-O	Descripción
$O(1)$	Tiempo constante
$O(\log n)$	Tiempo logarítmico
$O(n)$	Tiempo lineal
$O(n^2)$	Tiempo Cuadrático
$O(2^n)$	Tiempo exponencial
$O(n!)$	Tiempo factorial

Problema

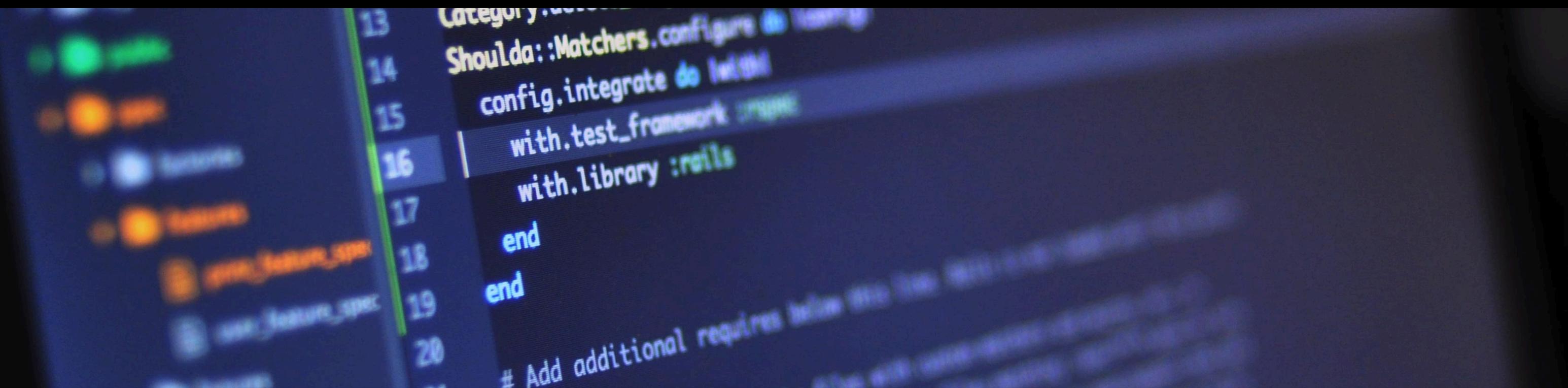
¿Cómo detectar si un ID ingresado a través de un formulario ya está en la base de datos?

- Comparar dos listas
 - ids_registro
 - ids_formulario
 - Aplicar dos estrategias algorítmicas distintas.



Ahora veamos el código en acción

- A continuación mostraremos en pantalla como funciona el programa en Python.
- Veremos ambos algoritmos, los tiempos de ejecución, y una explicación paso a paso.



Conclusiones

- La eficiencia algorítmica influye directamente en la calidad del software
- Big-O permite elegir soluciones escalables.
- Estructuras como diccionario son claves para optimizar.
- Aprendimos a pensar en algoritmos no solo por funcionalidad, sino por rendimiento.

MUCHAS GRACIAS!
MUCHAS GRACIAS!

Emmanuel Rivero
Julio Roja