



REPRESENTACIÓN DE FIXTURES DEPORTIVOS COMO ÁRBOLES BINARIOS EN PYTHON

Implementación usando Listas Anidadas



Luciano José Cartagena









Santiago Arroquigaray



Prof. Sebastián Bruselario








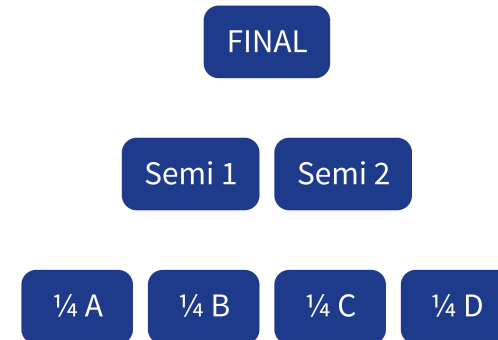
Junio 2025

1.  Problema y Motivación
2. Objetivos del Trabajo
3.  Marco Teórico: Árboles Binarios
4.  Metodología: Listas vs Clases
5.  Demostración del Sistema
6.  Análisis de Complejidad
7.  Resultados y Conclusiones









¿POR QUÉ FIXTURES COMO ÁRBOLES?

-  Los torneos de eliminación directa tienen estructura jerárquica natural
-  Cada partido depende del resultado de partidos anteriores
-  La final es la "raíz", los cuartos de final son las "hojas"
-  Necesitamos representar y simular el avance de equipos
-  Visualizar claramente la estructura del torneo








OBJETIVOS DEL PROYECTO

-  Implementar árboles binarios usando únicamente listas de Python
-  Modelar torneos deportivos de eliminación directa
-  Simular el progreso de partidos y avance de ganadores
-  Demostrar aplicaciones prácticas de estructuras de datos
-  Evaluar ventajas/desventajas vs implementaciones orientadas a objetos
-  Analizar complejidad algorítmica de las operaciones



ÁRBOLES BINARIOS: CONCEPTOS FUNDAMENTALES


-  Estructura jerárquica no lineal de nodos conectados
-  Cada nodo tiene máximo 2 hijos: izquierdo y derecho
-  Propiedades: raíz, hojas, altura, profundidad
-  Recorridos: Preorden, Inorden, Postorden
-  Aplicaciones: búsqueda, bases de datos, IA, torneos





ESTRUCTURA DE DATOS

```
[nombre_partido, hijo_izq,  
hijo_der] # Ejemplo: ["Final",  
["Semi 1", cuarto_1, cuarto_2],  
["Semi 2", cuarto_3, cuarto_4] ]
```

-  Python 3.x con recursión
- Funciones puras para manipular

COMPARACIÓN DE ENFOQUES

Ventajas Listas:

- Simplicidad conceptual
- Ideal para aprendizaje
- Sin POO requerida

Desventajas:

- Menos flexibilidad
- Sin encapsulamiento
- Escalabilidad limitada



</> DEMOSTRACIÓN EN VIVO

```
# Creación del fixture: 8 equipos, 7 partidos fixture =  
crear_torneo_8_equipos() # Estructura: Cuartos → Semifinales → Final  
imprimir_fixture(fixture) # Simulación de resultados paso a paso  
avanzar_ganador(cuarto_1, "Equipo A") avanzar_ganador(cuarto_2, "Equipo B") #  
Visualización rotada del árbol recorrido_inorden(fixture)
```



Ejecución interactiva del código



ARQUITECTURA DEL SISTEMA

+ crear_partido() → Inicializa nodos

↻ asignar_subpartidos() → Conecta partidos

👁️ imprimir_fixture() → Visualización recursiva

▶▶ avanzar_ganador() → Actualiza resultados

📍 recorridos() → Preorden, inorden, postorden

⚙️ Funciones auxiliares → altura, conteo, búsqueda



COMPLEJIDAD ALGORÍTMICA

Operación	 Tiempo	 Espacio	 Descripción
Crear partido	$O(1)$	$O(1)$	Inicialización de lista simple
Imprimir fixture	$O(n)$	$O(h)$	Recorrido completo del árbol
Recorridos	$O(n)$	$O(h)$	Preorden, inorden, postorden
Búsqueda	$O(n)$	$O(h)$	Búsqueda lineal en el árbol
Avanzar ganador	$O(1)$	$O(1)$	Actualización directa

Donde: n = número de partidos, h = altura del árbol $\approx \log_2(n)$ para torneos balanceados



RESULTADOS OBTENIDOS

- Sistema funcional completo para torneos de eliminación
- Visualización clara y comprensible del fixture
- Simulación exitosa de partidos y avance de ganadores
- Implementación de todos los recorridos estándar
- Análisis de rendimiento y complejidad
- Código documentado y reutilizable








Implementación exitosa de estructura de datos compleja usando herramientas básicas








ANÁLISIS CRÍTICO DEL ENFOQUE

VENTAJAS

-  Simplicidad conceptual
-  Ideal para aprendizaje
-  Sin POO requerida
-  Implementación directa
-  Visualización clara

DESVENTAJAS







-  Menor flexibilidad que clases
-  Sin encapsulamiento
-  Difícil escalabilidad
-  Limitado para árboles auto-balanceados
-  Menos herramientas de debugging



Recomendado para: educación, prototipos, demostraciones didácticas



CONCLUSIONES Y APRENDIZAJES

-  Los árboles binarios modelan naturalmente torneos deportivos
-  Las listas ofrecen una alternativa simple a implementaciones complejas
-  Excelente herramienta didáctica para enseñar estructuras de datos
-  Balance entre simplicidad y funcionalidad logrado exitosamente
-  Aplicación práctica de algoritmos recursivos y análisis de complejidad
-  Fundamentos sólidos para implementaciones más avanzadas



Proyecto exitoso que demuestra la elegancia de las estructuras de datos
aplicadas



¿PREGUNTAS?

♥ Gracias por su atención

✉ Luciano José Cartagena - lucianocartagena17@gmail.com

✉ Santiago Arroquigaray - arroqui192@gmail.com

🐙 Repositorio: https://github.com/Pitdog192/tp_integrador_aboles_listas

🏛️ Programación I - 2025